



TESINA DE LICENCIATURA

Título: Realidad Aumentada en el Museo

Autores: Andrea M. Gallego, Leandro A. Aguilar

Director: Javier Díaz

Codirector: Ivana Harari

Carrera: Licenciatura en Informática, Licenciatura en Sistemas

Resumen

La Realidad Aumentada es un campo de investigación y desarrollo que en los últimos años se ha visto potenciado gracias al pujante crecimiento de teléfonos celulares, con cámaras de video y altas capacidades de procesamiento. Entre las disciplinas que más fuerte han apostado por la realidad aumentada se encuentra la museografía. Los museos utilizan esta nueva tecnología para ofrecer entornos más verosímiles que permitan al visitante una mayor inmersión, generando la posibilidad de que interactúen directamente con los objetos expuestos de una forma atractiva y a la vez didáctica.

El Museo de Física de la UNLP es un espacio de encuentro con la ciencia abierto a todo el público. Al tratarse de un ámbito de educación no formal, ofrece la ventaja de promover el entretenimiento, facilitando así una visión más amigable de la Física.

La Realidad Aumentada mejora la percepción del usuario y la interacción con el mundo real. Esta capacidad de la Realidad Aumentada nos motivó a desarrollar una aplicación que le permita al visitante obtener mayor información sobre los objetos en exposición dentro del museo de forma interactiva y cumpliendo con los objetivos del museo.

Palabras Claves

Dispositivos móviles, Realidad Aumentada, Museo, Android, Usabilidad

Conclusiones

Se ha logrado desarrollar una aplicación de Realidad Aumentada en el ámbito del Museo de Física. La cual es parametrizable y extensible a varios museos.

Se ha analizado la usabilidad de la aplicación mediante un test a partir del cual se obtuvieron resultados satisfactorios sobre el desempeño del sistema.

Trabajos Realizados

Se aplicó la tecnología de Realidad Aumentada en un contexto científico, educativo y cultural como el Museo de Física de la UNLP.

Se desarrolló una aplicación móvil para el Sistema Operativo Android.

Como complemento y con el objetivo de realizar una aplicación parametrizable, también se desarrolló una aplicación Web.

Se llevó a cabo un análisis de usabilidad de la aplicación móvil en el contexto del Museo.

Trabajos Futuros

Incorporar nuevas funcionalidades en la aplicación móvil.

Refactorización general del código.

Mejorar el administrador Web.

Agradecimientos

A mi familia, principalmente mis padres, Yoli y Eduardo, gracias por el apoyo incondicional, el esfuerzo y la confianza en todos estos años.

A Andrés, mi compañero de vida, gracias por tu profundo amor y apoyo, por ser mi motor y pilar de cada día... TE AMO.

Y a todas las personas que con su apoyo ayudaron para que la finalización de este trabajo sea posible, a amigos y compañeros.

Andrea.-

Quiero agradecer a todas las personas que me han ayudado a ser una mejor persona, sin ellas no hubiera llegado a este momento tan especial de mi vida. A mis padres especialmente, a toda mi familia, amigos y compañeros.

Leandro.-

Queremos por último también agradecer a todo el personal del Museo de Física cuya colaboración ha sido importantísima para ver hecho realidad nuestro trabajo. Para nosotros fue una experiencia muy enriquecedora, no solo por todo lo aprendido y desarrollado sino por haber realizado un aporte importante a una institución tan prestigiosa como el Museo de Física.

Andrea y Leandro.-

Índice general

CAPÍTULO 1. INTRODUCCIÓN.....	7
1.1 Introducción a la tesis.....	7
1.2 El Museo	7
1.3 La Propuesta.....	8
1.4 Objetivos iniciales.....	9
1.5 Estructura del proyecto.....	9
CAPÍTULO 2. DISPOSITIVOS MÓVILES	11
2.1 Introducción.....	11
2.2 Arquitectura de sistemas móviles	12
2.2.1 Operadora	12
2.2.2 Red móvil	12
2.2.3 Dispositivo.....	13
2.2.4 Plataforma	13
2.2.5 Sistemas Operativos	14
2.2.6 Framework de aplicaciones.....	17
2.2.7 Aplicaciones	18
2.2.8 Servicios.....	18
2.3 Evolución de los dispositivos.....	18
2.3.1 Primera generación	19
2.3.2 Segunda generación	19
2.3.3 Tercera generación.....	20
2.3.4 Cuarta generación.....	21
2.4 Desarrollo en aplicaciones móviles	22
2.5 Tipos de aplicaciones móviles	23
2.5.1 SMS.....	23
2.5.2 Sitios web para móviles.....	24
2.5.3 Widget Web móviles.....	25
2.5.4 Aplicaciones Web móviles	25
2.5.5 Aplicaciones Nativas.....	26
2.5.6 Juegos.....	27
2.5.7 Comparación de tipos de aplicaciones	27
CAPÍTULO 3. REALIDAD AUMENTADA.....	28
3.1 Introducción.....	28
3.2 Aplicaciones	30
3.2.1 Medicina.....	30
3.2.2 Fabricación y reparación	31
3.2.3 Anotación y visualización.....	31
3.2.4 Planificación de trayectorias Robot	31
3.2.5 Aplicaciones Militares	32
3.2.6 Aplicaciones Actuales.....	32
3.3 Diagrama de una aplicación de realidad aumentada	33
3.3.1 Captura de la escena real	33
3.3.2.1 Tracking mediante dispositivos físicos.....	34
3.3.2.2 Tracking basado en visión	35
3.3.2.3 Tracking híbrido.....	38
3.3.3 Generador de la escena virtual	38

3.3.4 Combinación del mundo virtual y la escena real	40
3.4 Realidad aumentada en dispositivos móviles	40
3.5 Códigos QR.....	41
3.6 Códigos QR y Realidad Aumentada.....	43
3.7 Realidad aumentada en los museos	43
3.7.1 Código QR en áreas de exposición	44
3.7.2 Códigos QR en áreas de reserva	44
3.7.3 Códigos QR en difusión.....	45
3.7.4 Códigos QR en la página web del museo	45
3.7.5 Códigos QR en la Biblioteca del Museo	45
3.7.6 Códigos QR en eventos	45
3.8 Recomendaciones de uso de códigos QR	45
3.9 Conclusión.....	46
CAPÍTULO 4. ANDROID	48
4.1 Introducción.....	48
4.2 Historia	48
4.3 Versiones de Android	49
4.4 Distribución de las versiones.....	51
4.5 Características	52
4.6 Arquitectura	53
4.7 Fundamentos de la aplicación.....	55
4.8 Componentes de Aplicación	56
4.8.1 Activación de componentes.....	57
4.8.2 Procesos	57
4.8.3 Ciclo de vida de una actividad.....	58
4.9 Entorno de desarrollo	61
4.9.1 Instalación del SDK	61
4.9.2 Herramientas de desarrollo	64
4.10 Estructura de un proyecto Android	66
CAPÍTULO 5. USABILIDAD	70
5.1 Introducción.....	70
5.2 Fundamentos en HCI	71
5.3 Clasificación de las Interfaces del usuario.....	73
5.4 Independencia del Diálogo.....	76
5.5 Componentes de la Interfaz del Usuario	77
5.6 Ciclo de Vida de la Interfaz del Usuario.....	77
5.7 Métricas de Evaluación de una Interfaz: Principios de Nielsen	80
5.8 Interfaces Móviles	82
5.8.1 Contexto.....	82
5.8.2 Arquitectura de la información	82
5.8.3 El diseño visual.....	83
5.9 Diseño de aplicaciones Android	86
5.9.1 Soportar múltiples pantallas	86
5.9.2 Principios de Diseño.....	92
5.10 Etapa de Evaluación de la Interfaz del Usuario	94
5.10.1 Técnicas de evaluación de Interfaz de usuario.....	94
5.10.2 Nuevas tendencias en HCI.....	95
CAPÍTULO 6. ESPECIFICACIÓN DEL PROTOTIPO	97
6.1 Conceptos de la aplicación.....	97
6.2 Requerimientos del cliente móvil.....	97
6.3 Requerimientos del administrador web	98

6.4 Casos de uso	99
6.4.1 Casos de uso de la aplicación móvil.....	99
6.4.2 Casos de uso del administrador web.....	99
6.4.3 Descripción de los casos de uso de la aplicación móvil	100
6.4.4 Descripción de los casos de uso del administrador web	101
6.5 Arquitectura del sistema	103
6.6 Diagrama de clases.....	103
6.6.1 Descripción de las clases	106
CAPÍTULO 7. SISTEMA WEB.....	109
7.1 Arquitectura	109
7.2 Framework Spring	109
7.2.1 Inyección de dependencias	109
7.2.2 AOP	110
7.3 Spring MVC	110
7.4 Spring Security	111
7.5 Hibernate	112
7.6 JQuery.....	112
7.7 Creación del proyecto	112
7.8 Creación de las clases del modelo.....	113
7.9 Creación del controlador	114
7.10 Creación de los servicios.....	116
7.11 Comunicación móvil/servidor.....	118
7.12 Configuración de la aplicación.....	119
7.12.1 Web.xml.....	119
7.12.2 applicationContext.xml	119
7.12.3 applicationContext-security.xml	120
7.12.4 jdbc.properties.....	121
CAPÍTULO 8. IMPLEMENTACIÓN DE LA APLICACIÓN MÓVIL.....	123
8.1 Prototipo de la interfaz	123
8.2 Implementación del prototipo	136
8.2.1 Creación del proyecto.....	136
8.2.2 Estructura del proyecto.....	137
8.2.3 Componentes de la aplicación	138
8.2.4 Comunicación con el servidor	151
8.2.5 Implementación de los componentes de la aplicación.....	160
8.2.6 Internacionalización.....	218
CAPÍTULO 9. ANÁLISIS DE USABILIDAD	221
9.1 Armado de Ambiente	221
9.2 Objetivo	221
9.3 Perfiles de usuarios.....	221
9.4 Encuesta	222
9.5 Análisis de los resultados	223
9.6 Conclusiones.....	223
CAPÍTULO 10. CONCLUSIONES.....	225
10.1 Cumplimiento de los objetivos iniciales	225
10.2 Resultados obtenidos.....	225
10.3 Evaluación del proyecto	226
10.4 Trabajo futuro	226
CAPÍTULO 11. BIBLIOGRAFÍA.....	227

Capítulo 1. Introducción

1.1 Introducción a la tesis

La realidad aumentada (a partir de ahora referenciada como RA) es un campo de investigación y desarrollo que en los últimos años se ha visto potenciado gracias al pujante crecimiento de teléfonos celulares, con cámaras de video y altas capacidades de procesamiento. En un mundo cada vez más globalizado donde las distancias se vuelven más cortas, y donde las necesidades de comunicación son más indispensables, la información con su manejo y acceso apropiados, pasa a ser un valor e instrumento respectivamente de gran utilidad para el desarrollo de las sociedades.

La realidad aumentada es un paradigma de interacción que agrega información sintética a la realidad. La realidad no se suprime. El usuario ve la realidad “aumentada” o enriquecida con objetos 3D sintéticos o información textual.

Según Ronald Azuma la RA:

- Combina mundo real y mundo virtual
- Es interactivo en tiempo real
- Se registra en 3 dimensiones

Por este motivo, un sistema de RA necesitará un medio para capturar la imagen del mundo real (una cámara de video o una webcam), una máquina capaz de crear imágenes sintéticas, y de procesar la imagen real añadiendo esta información (un procesador y software específico para esto) y un medio para proyectar la imagen final (una pantalla).

Hasta hace unos años, algo así hubiera sido realmente costoso, sobre todo teniendo en cuenta que no deben ser dispositivos muy grandes para poder transportarse a todas horas. Sin embargo, hoy en día, casi todos nosotros disponemos de todas estas tecnologías en un aparato realmente pequeño y que suele acompañarnos a todas horas: el teléfono móvil.

Tanto los proyectos ya existentes como los que están en desarrollo o las meras ideas acerca de posibles sistemas de RA son más que interesantes y plantean numerosas posibilidades. Además, demuestran que la RA es una tecnología con un potencial sobresaliente y que seguramente, jugará un papel muy importante en nuestras vidas cotidianas en un futuro no muy lejano.

Entre las disciplinas que más fuerte han apostado por la realidad aumentada se encuentra la museografía. Los museos utilizan esta nueva tecnología para ofrecer entornos más verosímiles que permitan al visitante una mayor inmersión, generando la posibilidad de que interactúen directamente con los objetos expuestos de una forma atractiva y a la vez didáctica. El mundo virtual de la RA lleva al museo, lo que el mundo real no puede, gracias a la capacidad de insertar objetos virtuales en un espacio real. Mediante una aplicación RA es posible aumentar la colección de los objetos que se pueden observar en el Museo. Las posibilidades de animación virtual o la inclusión de etiquetas flotantes ofrecen también un importante recurso para añadir información adicional que supera los carteles tradicionales.

No hay dudas que implementar un sistema de RA puede ofrecer al Museo de Física múltiples posibilidades de enriquecer la experiencia de visita al museo.

1.2 El Museo

El Museo de Física de la UNLP alberga una colección de más de 2.000 instrumentos utilizados para la enseñanza de la Física en las universidades de principios del siglo XX.

Debido a que el Museo está diseñado de manera didáctica y pedagógica por un equipo de profesionales de diferentes disciplinas, ofrece una atractiva propuesta para todos aquellos interesados en acercarse al conocimiento de la Física.

El Museo cuenta con un sistema de visitas guiadas que está dirigido al público en general y a grupos de nivel preescolar, escolar, terciario, universitario y con capacidades diferentes.

El objetivo del sistema de visitas es mostrar otros enfoques de los temas que se desarrollan en las clases convencionales. Al tratarse de un ámbito de educación no formal, ofrece la ventaja de promover el entretenimiento, facilitando así una visión más amigable de la Física.

El Museo se encuentra organizado en vitrinas, donde cada una expone objetos de categorías diferentes. Dichos objetos no poseen ningún cartel o información que los identifique y que explique su funcionamiento permitiendo al visitante recorrer y conocer el Museo a través de sus propios medios. Los visitantes que llegaban al Museo solo contaban con la ayuda de los guías para poder aumentar su conocimiento sobre los distintos objetos.

Por otra parte, el Museo no dispone de ningún sistema de información que permita administrar la información relacionada a los objetos en exposición. Por lo tanto, era de suma importancia para el Museo disponer de un sistema que no sólo le permita administrar la información de los objetos si no también que le dé al visitante la posibilidad de recorrer el museo sin la necesidad de solicitar un guía.

1.3 La Propuesta

La Realidad Aumentada mejora la percepción del usuario y la interacción con el mundo real. Los objetos virtuales muestran la información que el usuario no puede detectar directamente con sus propios sentidos. La información transmitida por los objetos virtuales ayuda a los usuarios a realizar tareas del mundo real.

El desarrollo y consumo de aplicaciones móviles ha experimentado un rápido crecimiento en los últimos años debido al desarrollo de las plataformas abiertas, como Android y al posicionamiento en el mercado de los celulares con acceso a Internet. Las aplicaciones móviles han superado a la navegación Web y las computadoras van perdiendo terreno frente a los smartphones y las tablets.

Por tal motivo, nos pareció muy apropiado realizar una aplicación con Realidad Aumentada para llevarla a cabo en un entorno científico, educativo y cultural como el Museo de Física y además ofrecer al visitante una experiencia más didáctica e interactiva.

Para ello realizaremos una aplicación web que permite a los usuarios del museo administrar toda la información relacionada a los objetos como imágenes, audios y videos. Dicha información estará alojada en un servidor de base de datos, la cual será diseñada especialmente para este proyecto.

Luego, mediante una aplicación móvil le permitirá al visitante investigar sobre el Museo y acceder a la información que se dispone de los objetos que se exponen en él. El visitante podrá interactuar con la aplicación de una manera más activa participando de cuestionarios, juegos o completando encuestas. Este concepto de participación activa se corresponde con la definición de RA dada por Ronald Azuma que se refiere a la interactividad en tiempo real. Aumentando la realidad no solo se pretende mostrar información sino que se intenta producir una retroalimentación con el participante de la experiencia.

1.4 Objetivos iniciales

Los objetivos iniciales de esta tesina son:

El objetivo general: es aplicar tecnología de RA en un contexto científico, educativo y cultural como el Museo de Física de la U.N.L.P. Generar una aplicación parametrizable que se pueda extender a más de un museo.

Entre los objetivos específicos:

Investigar sobre el desarrollo de aplicaciones móviles y la Realidad Aumentada, describiendo aplicaciones de RA existentes en distintos ámbitos y analizando los problemas encontrados en el desarrollo de estos sistemas.

Investigar aplicaciones de Realidad Aumentada en el museo, teniendo en cuenta el contexto, los recursos necesarios, el entorno y los usuarios que interactúan.

Desarrollar una aplicación de Realidad Aumentada para el Museo de Física de la U.N.L.P con el objetivo de ofrecer al visitante un entorno atractivo que le posibilite una mayor inmersión en el mismo, analizando todo el proceso que ello implica.

Investigar sobre los aspectos de usabilidad tanto en las aplicaciones móviles como en las aplicaciones de Realidad Aumentada.

Analizar la usabilidad de la aplicación desarrollada y observar el impacto y percepción de los usuarios. Para ellos se desarrollarán pruebas de campo, donde se pondrá a prueba el producto ante un grupo de visitantes

1.5 Estructura del proyecto.

La elaboración del presente informe está dividida en 3 secciones principales:

Una Investigación teórica, que cubrimos en los capítulos 1 a 5.

El Desarrollo tecnológico, que describimos en los capítulos 6 a 8.

El análisis de usabilidad y conclusiones, que corresponde a los capítulos 9 a 10.

En la Investigación teórica recorrimos los fundamentos teóricos de los temas a desarrollar necesarios para la implementación del prototipo propuesto inicialmente.

En el **Capítulo 2** introducimos los Dispositivos móviles, comenzamos con una introducción a la evolución de los teléfonos móviles seguido de una descripción general de su arquitectura. Describimos también detalles de plataformas actuales y particularidades de desarrollo.

En el **Capítulo 3** explicamos el concepto de Realidad Aumentada, detallando los pasos para generar un sistema de Realidad Aumentada, para luego concluir con la popularidad de la RA para dispositivos móviles.

En el **Capítulo 4** describimos la plataforma Android. Detallamos las principales características de este sistema operativo móvil, su diseño, arquitectura y funcionamiento. Introducimos la plataforma de desarrollo, herramientas de desarrollo, anatomía de las aplicaciones y sus principales componentes.

En el **Capítulo 5** se explican conceptos de usabilidad, fundamentos, métodos y métricas de evaluación, principios de diseño y etapas de evaluación.

Con respecto a la segunda sección de este trabajo, el Desarrollo tecnológico, se describe completamente la implementación tanto del lado del servidor de la aplicación, como de la aplicación cliente para el móvil.

En el **Capítulo 6**, describimos la Especificación del prototipo. Definimos la arquitectura y componentes de la aplicación, especificando a través de casos de uso la funcionalidad a implementar y llegamos al modelo de datos a utilizar.

En el **Capítulo 7**, nos referimos a la Implementación del Administrador Web, presentamos el Framework Spring y Hibernate. Ahondamos en detalles de implementación.

En el **Capítulo 8**, hacemos lo propio con a la Implementación del Cliente Móvil. Implementamos la parte cliente del prototipo describiendo la plataforma Android y decisiones de diseño que se tomaron a lo largo del desarrollo.

Con respecto a la tercera sección de este trabajo, de Análisis de usabilidad y conclusión, se hace una revisión de los resultados obtenidos a partir de las encuestas respondidas por los usuarios de la aplicación móvil y se muestran las conclusiones generales.

En el **Capítulo 9** analizamos los resultados surgidos a partir de una encuesta de usabilidad respondida por los usuarios de la aplicación móvil.

Por último en el **Capítulo 10** se muestran los objetivos iniciales, las conclusiones y los trabajos a futuro.

Capítulo 2. Dispositivos móviles

En el capítulo anterior hemos hecho una breve introducción al trabajo de la tesina. En el presente capítulo explicaremos el concepto y evolución de los dispositivos móviles.

2.1 Introducción

Al mirar hacia atrás y observar el desarrollo tecnológico que ha experimentado la Humanidad desde mediados del siglo XX hasta hoy, no cabe duda de que más que un avance se ha producido una verdadera revolución. El descubrimiento de la informática, su aplicación paulatina en todo tipo de áreas de conocimiento así como su introducción en el común de la población a través de todo tipo de componentes ha cambiado nuestra sociedad de una manera notable.

La computadora es uno de los inventos que mejor sintetiza esta nueva situación tecnológica. Aparecieron primero como enormes y costosas máquinas que solamente estaban disponibles en importantes universidades o centros de investigación. Luego, con la aparición de nuevas técnicas de fabricación como los circuitos integrados, su tamaño, sus capacidades, y sobre todo precio, variaron de tal forma que se convirtieron en un producto de consumo masivo. La aparición de Internet, y sobre todo su apertura al público general, determinaron de forma inequívoca la importancia de las computadoras en la vida social, laboral o académica de cualquier persona hasta el día de hoy.

Junto con la aparición de internet como medio de uso masivo, a mediados de la década de 1990, también se produjo la reinención de la telefonía móvil. Y de esa combinación nació la posibilidad de tener la información accesible en cualquier momento y lugar. Entonces el teléfono móvil dejó de ser un dispositivo para solo hacer llamadas o enviar SMS sino también que empezó a servir para tener acceso a internet, tomar fotografías, videos y poseer una variedad de aplicaciones.

Un *dispositivo móvil* es un término general que describe una amplísima familia de aparatos electrónicos surgidos en los últimos años, de reducido tamaño, que ofrecen alguna capacidad de procesamiento y almacenamiento de datos y que están orientados a una función concreta o varias de ellas: desde los teléfonos móviles más evolucionados (los llamados smartphones), a ordenadores portátiles, cámaras digitales, reproductores de música o consolas de videojuegos. [1]

La mayoría de estos aparatos cuentan con un sistema operativo de mayor o menor complejidad, que permite realizar las tareas de gestión de memoria y control de hardware necesarias. En el caso de los ordenadores portátiles, con tanta o incluso mayor capacidad que los de sobremesa, los sistemas operativos habituales son perfectamente compatibles y funcionan sin diferencias. Sin embargo, en dispositivos móviles como tablets y pda's entre otros, es preciso diseñar nuevos sistemas operativos adaptados específicamente a sus características: restricciones de memoria y procesamiento, consumo mínimo de energía o gran estabilidad en su funcionamiento, entre otros.

La industria móvil comenzó casi al mismo tiempo que la Web, pero tomó años para que se considere como un medio masivo. El móvil puede hacer todo lo que los otros medios de comunicación pueden hacer. Podemos leer textos, publicarlos, reproducir grabaciones, ver películas, escuchar la radio, ver televisión, acceder a internet, etc. Algunas de las características que diferencian a los dispositivos móviles del resto de los medios de comunicación son:

- Es el primer medio de comunicación verdaderamente personal
- Tienen la capacidad de enviar y recibir información en todo momento, incluso cuando está inactivo.
- Lo llevamos con nosotros a todos lados
- Nos permiten crear y publicar fotos, videos e información y compartir nuestras experiencias en las redes sociales, casi en tiempo real. [2]

La infraestructura móvil crecerá a un elevado ritmo los próximos años, con más innovación y proliferación de dispositivos móviles. Especialmente de dispositivos de gama alta tales como el iPhone y los que poseen la plataforma Android, los cuales reducirán la brecha entre la movilidad y la informática.

2.2 Arquitectura de sistemas móviles

Se puede ver un sistema móvil como un sistema de capas, donde cada capa depende de la anterior para funcionar. [2]

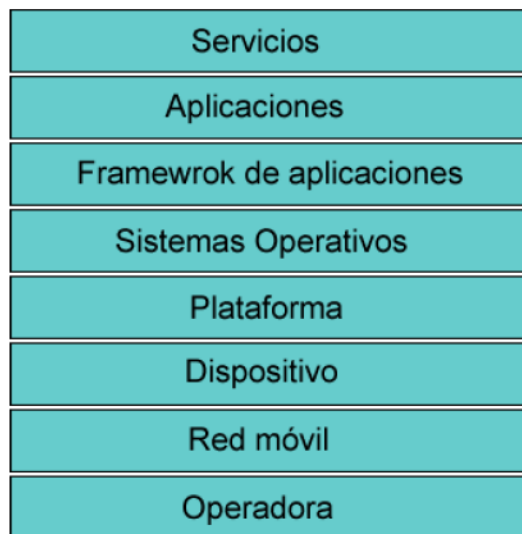


Figura 2.1. Arquitectura de sistemas móviles

Estas capas serán descriptas a continuación en las siguientes secciones.

2.2.1 Operadora

Los operadores están en la base. Ellos se refieren a los operadores de redes móviles o las compañías de teléfonos celulares. Los operadores son los que esencialmente hacen trabajar todo el sistema móvil: instalan torres de comunicación, gestionan la red celular, hacen que los servicios (como Internet) estén disponibles para los abonados de móviles. [2] En argentina ejemplos de operadoras son Claro, Movistar, Nextel y Personal.

2.2.2 Red móvil

Las operadoras operan las redes inalámbricas. La tecnología celular es, en un modo simplificado, una radio que recibe señal de una antena. El tipo de radio y antena determina la capacidad de la red y los servicios que pueden estar disponibles en él.

La mayoría de las redes alrededor del mundo usan el estándar GSM, utilizando GPRS o GPRS EDGE para 2G y TMS o HSDPA para 3G. También tenemos CDMA (Code Division Multiple Access o código de división múltiple de acceso) y su híbrido 2.5G CDMA2000 que aunque ofrece una cobertura mayor, limitan la cantidad y capacidad de servicios que pueden transferirse sobre ella. Por lo tanto, en países como Estados Unidos o China donde la gente está más esparcida es mejor utilizar CDMA. [2]. Utiliza menos torres dando a los suscriptores menos servicios.

A continuación detallamos las principales diferencias entre GSM y CDMA [3] [4]:

GSM funciona con 4 bandas que comúnmente se encuentran de 900MHz y 1.800MHz en Europa y Asia, y de 850MHz y 1.900MHz en Estados Unidos y América Latina. El GSM permite ocho llamadas simultáneas en la misma frecuencia. Con este método la frecuencia es dividida en múltiples canales que son unidos como si fuera un solo flujo de información. Esta tecnología permite a varios usuarios compartir el mismo canal al mismo tiempo. Además en la red GSM, hay un número límite fijo de usuarios que pueden usar una frecuencia al mismo tiempo. La calidad de la llamada no se degrada, pero es mucho más fácil que las frecuencias estén ocupadas y se tenga la necesidad de marcar más de una vez, cosa que no es tan común en las redes CDMA. La red CDMA trabaja de forma totalmente diferente. Ésta envía la información a través del canal, después que el canal está digitalizado. Muchas llamadas se envían una sobre la otra a través de todo el canal. A cada llamada se le asigna su propio código de frecuencia para mantener la señal distinta. Con este método se ofrece mucha más seguridad que la red GSM. Esta es la razón por la cual muchos expertos prefieren la red CDMA.

2.2.3 Dispositivo

Los dispositivos son los teléfonos celulares. Los smartphone, aunque todavía no son una porción mayoritaria del mercado, cada vez son más populares. [2] Son utilizados diariamente por millones de personas para comunicarse, navegar por internet o enviar mails entre otros usos. Según una estadística del año 2011 en Australia, el Reino Unido, Suecia, Noruega, Arabia Saudita y los Emiratos Árabes Unidos, más del 50% de su población poseen smartphones. También Estados Unidos, Nueva Zelanda, Dinamarca, Irlanda, Los Países Bajos, España y Suiza ahora tienen más de un 40% de penetración en el mercado de los smartphones.

En Estados Unidos el 80% de los dueños de smartphones dicen que ellos no salen de su casa si no llevan su dispositivo móvil y 1/3 dice que dejarían primero su TV antes que su smartphone. [5]

2.2.4 Plataforma

La plataforma móvil es la capa que existe para proporcionar acceso a los dispositivos. Para ejecutar cualquier software o servicio en cada dispositivo se necesita una plataforma, o un lenguaje de programación con el cual se escribe todo el software. Como todas las plataformas de software, éstas se dividen en 3 categorías: Licenciadas, propietarias y open source o de código libre. [2]

Licenciado

Las plataformas se venden a los fabricantes de dispositivos. La idea es crear una plataforma común o una interfaz de programación (API) que trabaje similarmente a través de múltiples dispositivos con el menor esfuerzo posible requerido para adaptarse a las diferencias entre dichos dispositivos. Aunque la realidad indica que no sucede tan así ya que a veces las diferencias suelen ser amplias.

Las plataformas licenciadas son:

Java Micro Edition (Java ME)

Antiguamente conocido como J2ME [6], Java ME es la plataforma de software más predominante. Es un subconjunto licenciado de la plataforma JAVA y provee una colección de API para el desarrollo en teléfonos móviles.

Binary Runtime Environment for Wireless (BREW)

BREW [7] es una plataforma licenciada creada por Qualcomm para dispositivos móviles, y usada mayoritariamente en el mercado de Estados Unidos. Es una plataforma independiente de la interfaz que puede correr una variedad de frameworks de aplicación como C/C++, Java, and Flash Lite.

Windows Mobile

Windows Mobile [8] es una versión compacta del sistema operativo Windows combinado con una suite de aplicaciones para dispositivos móviles que está basada en la API WIN32 de Microsoft.

LiMo

LiMo [9] es una plataforma basada en Linux creada por la fundación LiMo. LiMo incluye SDKs para crear aplicaciones JAVA, nativas o web móviles usando el WebKit browser framework.

Propietario

Las plataformas propietarias están diseñadas y desarrolladas por los fabricantes de dispositivos para ser usadas en sus dispositivos exclusivamente. Entre estas se incluyen:

Palm

Palm utiliza tres diferentes plataformas propietarias. Su primer y más reconocida es la plataforma Palm OS basado en el lenguaje C/C++, la que fue inicialmente desarrollada para su línea de Palm Pilot. Con la aparición teléfonos inteligentes, Palm creó webOS, que es una plataforma basada en Linux y su ejecución se hace totalmente en el marco del navegador WebKit [10].

BlackBerry

BlackBerry [11] mantiene su propia plataforma propietaria basada en Java, que se utiliza exclusivamente en sus dispositivos.

iPhone

Apple utiliza una versión compacta de Mac OS X como plataforma para su línea de dispositivos iPhone, iPad e iPod, que ciertamente está basada en Unix. Actualmente se encuentra en la versión 4G. Provee todas las herramientas y recursos para construir aplicaciones nativas con un buen soporte para multimedia y gráficos [12].

Código libre

Plataformas de código abierto, son las plataformas móviles que están disponibles gratuitamente para los usuarios que pueden descargar, modificar y editar. Estas plataformas son las más recientes en móviles, pero aún así cada vez están ganando más confianza con los fabricantes de dispositivos y desarrolladores. Una de estas plataformas es Android, desarrollada por la Open Handset Alliance, encabezada por Google. La Alianza busca desarrollar una plataforma de código abierto para móviles basado en el lenguaje de programación Java. Describiremos en detalle la plataforma Android más adelante en el capítulo 4.

2.2.5 Sistemas Operativos

Antes sucedía que si un dispositivo móvil corría un sistema operativo, lo más probable es que se considerara un smartphone. Pero ahora es cada vez más amplio el conjunto de dispositivos que soportan sistemas operativos.

Los sistemas operativos suelen tener servicios básicos o conjuntos de herramientas que permiten a las aplicaciones que se comuniquen entre sí y compartir datos o servicios. [2]

Aunque no todos los teléfonos tienen sistemas operativos, los siguientes son algunos de los más comunes:

Symbian

Symbian OS es un sistema operativo de código abierto diseñado para dispositivos móviles, con bibliotecas asociadas, frameworks de interfaz de usuario e implementaciones de referencia.

Symbian es un sistema operativo para dispositivos móviles desarrollado por Psion, Nokia, Motorola y Ericsson. El principal objetivo de estas compañías era el de crear un nuevo y compartido sistema operativo que estuviera perfectamente adaptado a los teléfonos móviles del momento, y fuese además capaz de competir con Palm OS y Windows Mobile. La primera versión de Symbian, basada en el sistema EPOC de Psion, se lanzó en 1998.

El acuerdo bajo el cual se desarrolló Symbian es bastante simple: Symbian Ltd. desarrolla el sistema operativo Symbian, que incluye el microkernel, los controladores, el middleware y una considerable pila de protocolos de comunicación e interfaces de usuario muy básicas. Los desarrolladores que obtienen la licencia correspondiente para trabajar con Symbian implementan sus propias interfaces de usuario y conjuntos de aplicaciones según las necesidades de sus propios dispositivos. Esto permitió a Symbian posicionarse como un sistema operativo muy flexible, que tenía en cuenta los requisitos de la mayoría de los dispositivos fabricados y, al mismo tiempo, permitía un alto grado de diferenciación. [13]

Windows Mobile

Windows Mobile es un sistema operativo diseñado por Microsoft y orientado a una gran variedad de dispositivos móviles. En realidad, Windows Mobile representa una particularización de otro gran sistema de Microsoft llamado Windows CE.

A principios de la década de los 90, cuando comenzaron a aparecer los primeros dispositivos móviles, Microsoft tomó la decisión de crear un sistema operativo capaz de hacer frente al entonces recientemente lanzado por Apple, el sistema Newton MessagePad. Fruto de esta iniciativa surgió Pegasus, cuyo nombre comercial definitivo fue Windows Compact Edition, o Windows CE [14].

El objetivo principal que buscaba Microsoft era que el nuevo sistema fuera lo suficientemente flexible y adaptable para poder ser utilizados en un amplio abanico de dispositivos, cuyo única característica común es la de ser de reducido tamaño y tener, por tanto, una limitación obvia en sus recursos. Las características principales con las que cuenta Windows CE son las siguientes:

- Es un sistema modular, lo que permite que cada fabricante pueda seleccionar aquellas partes que le benefician más para su dispositivo.
- Contempla una considerable gama de recursos hardware: teclado, cámara, pantalla táctil, etc.
- Tiene un tamaño en memoria relativamente pequeño y bajo coste computacional.
- Es capaz de trabajar con distintas familias de procesadores de 32 bits.
- Permite interactuar con otros dispositivos móviles.

Un aspecto distintivo de Windows CE con respecto a otros productos desarrollados por Microsoft es que un elevado número de sus componentes se ofrece a los fabricantes y desarrolladores a

través del propio código fuente. Esto les permite poder adaptar el sistema a sus dispositivos específicos. Aquellos componentes básicos de Windows CE que no necesitan ningún tipo de adaptación siguen siendo ofrecidos únicamente como código binario.

Palm OS

Palm OS es el sistema operativo desarrollado inicialmente por Palm, Inc. Palm utiliza tres diferentes plataformas propietarias. Su primer y más reconocida es la plataforma Palm OS basado en el lenguaje C/C++, la que fue inicialmente desarrollada para su línea de Palm Pilot. Con la aparición de teléfonos inteligentes, Palm creó webOS, que es una plataforma basada en Linux y su ejecución se hace totalmente en el marco del navegador WebKit. [2]

Linux

El código abierto de Linux se está utilizando cada vez más como un sistema operativo para teléfonos inteligentes, incluyendo RAZR2 de Motorola. [2] Linux fue inicialmente desarrollado por Linus Torvalds y actualmente pueden ser encontradas en 5-6 % de los teléfonos alrededor del mundo. Linux es una base para una serie de plataformas desarrolladas por diferentes vendedores como Motorola, TrollTech. Linux también se usado en Google (Android), NEC, Panasonic, Phillips entre varios más. Las ventajas de un Linux embebido por sobre otros sistemas operativos embebidos incluyen que no haya derechos de autor y el pago de licencias [15].

Mac OS X

Una versión especializada de Mac OS X es el sistema operativo iOS utilizado en el iPhone de Apple y el iPod touch.

La interfaz de usuario de iOS está basada en el concepto de manipulación directa, usando gestos multitáctiles. Los elementos de control consisten de deslizadores, interruptores y botones. La respuesta a las órdenes del usuario es inmediata y provee de una interfaz fluida. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo del contexto de la interfaz. Se utilizan acelerómetros internos para hacer que algunas aplicaciones respondan a sacudir el dispositivo (por ejemplo, para el comando deshacer) o rotarlo en tres dimensiones (un resultado común es cambiar de modo vertical al apaisado u horizontal).

iOS cuenta con cuatro capas de abstracción: la capa del núcleo del sistema operativo, la capa de "Servicios Principales", la capa de "Medios" y la capa de "Cocoa Touch". Provee todas las herramientas y recursos para construir aplicaciones nativas con un buen soporte para multimedia y gráficos.

iOS no permite Adobe Flash ni Java. En cambio, usa HTML5 como una alternativa a Flash. [16]

Android

Android ejecuta su propio sistema operativo de código abierto, que puede ser personalizado por los operadores y fabricantes de dispositivos.

Android constituye una pila de software pensada especialmente para dispositivos móviles y que incluye tanto un sistema operativo, como middleware y diversas aplicaciones de usuario. Representa la primera incursión seria de Google en el mercado móvil y nace con la pretensión de extender su filosofía a dicho sector.

Todas las aplicaciones para Android se programan en lenguaje Java y son ejecutadas en una máquina virtual especialmente diseñada para esta plataforma, que ha sido bautizada con el nombre de Dalvik.

La licencia de distribución elegida para Android ha sido Apache 2.0 [17], lo que lo convierte en software de libre distribución. A los desarrolladores se les proporciona de forma gratuita un SDK y

la opción de un plug-in para el entorno de desarrollo Eclipse que incluyen todas las API necesarias para la creación de aplicaciones, así como un emulador integrado para su ejecución. Existe además disponible una amplia documentación de respaldo para este SDK. [18].

Con Android se busca reunir en una misma plataforma todos los elementos necesarios que permitan al desarrollador controlar y aprovechar al máximo cualquier funcionalidad ofrecida por un dispositivo móvil (llamadas, mensajes de texto, cámara, agenda de contactos, conexión Wi-Fi, Bluetooth, aplicaciones ofimáticas, videojuegos, etc.), así como poder crear aplicaciones que sean verdaderamente portables, reutilizables y de rápido desarrollo. En otras palabras, Android quiere mejorar y estandarizar el desarrollo de aplicaciones para cualquier dispositivo móvil.

2.2.6 Framework de aplicaciones

La primer capa a la que puede acceder el desarrollador es el framework de aplicaciones o API liberada por alguna de las compañías mencionadas.

El framework de aplicaciones a menudo se ejecuta sobre los sistemas operativos, comparten los servicios básicos, tales como las comunicaciones, mensajes, gráficos, localización, seguridad, autenticación, entre otros. [2]

Java

Las aplicaciones escritas en el framework de Java ME con frecuencia pueden ser ejecutadas en la mayoría de los dispositivos basados en Java, pero dada la diversidad de tamaño de pantalla de los dispositivos y el poder del procesador, el despliegue a través de diversos dispositivos puede ser un desafío.

La mayoría de las aplicaciones Java se compran y se distribuyen a través del operador, pero también puede ser descargado e instalado a través de un cable o de forma inalámbrica.

S60

La plataforma S60, anteriormente conocida como la serie 60, es la plataforma de aplicaciones para dispositivos que ejecuta el sistema operativo Symbian. S60 se asocia a menudo con los dispositivos Nokia -Nokia posee la plataforma-, pero también se ejecuta en varios dispositivos que no son de Nokia. S60 es un framework de código abierto. Aplicaciones de S60 pueden ser creadas en Java, Framework Symbian C ++, o incluso Flash Lite.

BREW

Las aplicaciones escritas en el framework BREW pueden ser ejecutadas en la mayoría de los dispositivos basados en BREW, con un poco menos de adaptación a dispositivos que otros frameworks.

Sin embargo las aplicaciones BREW deben seguir un costoso y largo proceso de certificación y puede solo ser distribuido a través de un operador.

Flash Lite

Adobe Flash Lite es un Framework de aplicaciones que usa Flash Lite y el framework ActionScript para crear aplicaciones basadas en vectores. Las aplicaciones de Flash Lite pueden ser ejecutadas dentro del Flash Lite Player, el cual está disponible en mucho dispositivos alrededor del mundo. Flash Lite es una prometedora y poderosa plataforma, pero ha habido alguna dificultad para obtenerla en los dispositivos.

Windows Mobile

Aplicaciones escritas usando la API de Win32 pueden ser desplegadas a través de la mayoría de dispositivos basados en Windows. Como java, las aplicaciones de Windows Mobile puede ser descargadas e instaladas a través de un cable conectado a la computadora o de forma inalámbrica.

Cocoa Touch

Cocoa Touch es la API usada para crear aplicaciones nativas para iPhone y iPod Couch. Las aplicaciones de Cocoa Touch deben ser submiteadas y certificadas por Apple antes de ser incluidas en el App Store. Una vez que están en el App Store, las aplicaciones pueden ser compradas, descargas e instaladas mediante un cable conectado a la computador o de forma inalámbrica.

Android SDK

El Android SDK permite a los desarrolladores crear aplicaciones nativas para cualquier dispositivo que ejecute la plataforma Android. Al usar el Android SDK los desarrolladores pueden escribir aplicaciones en C/C++ o usar la máquina virtual de Java incluida en el OS que permite la creación de aplicaciones con Java.

Web Runtimes (WRTs)

Nokia, Opera, and Yahoo! Proveen varios Web Runtimes or WRTs. Estos son miniframeworks basados en estándares web para crear widget móviles. Tanto los WRTs de Opera y Nokia cumplen con las especificaciones recomendadas por la W3C para widget móviles.

WebKit

Con la introducción de webOS por parte de Palm, una plataforma móvil basada en WebKit, y dada su predominio como navegador móvil incluido en plataformas móviles como iPhone, Android y S60, y que la vasta mayoría de aplicaciones móviles son escritas específicamente para WebKit, se puede decir que ahora nos podemos referir a Webkit como una Framework móvil en su propio derecho.

2.2.7 Aplicaciones

Los framework de aplicación se utilizan para crear aplicaciones, como juegos, un navegador web, una cámara o un reproductor de multimedia. Aunque los frameworks son bien estandarizados, los dispositivos no lo son. El mayor desafío de la implementación de aplicaciones es conocer los atributos del dispositivo y capacidades específicas. Por ejemplo, si se está creando una aplicación que utiliza el framework de aplicación Java ME, es necesario saber con qué versión de Java ME es compatible el dispositivo, las dimensiones de la pantalla, la potencia del procesador, la capacidad gráfica, el número de botones que tiene y cómo los botones están orientados, etc.[2]

2.2.8 Servicios

Finalmente, llegamos a la última capa de la arquitectura: servicios. Los servicios incluyen tareas tales como el acceso a Internet, el envío de un mensaje de texto, o ser capaz de obtener una ubicación, básicamente, todo lo que el usuario está tratando de hacer. [2]

2.3 Evolución de los dispositivos

La tecnología móvil ha ido a través de varias evoluciones para llegar hasta donde esta actualmente. En la industria se refiere con frecuencia a estas evoluciones como “generaciones” o simplemente “G”, lo cual se refiere a la madurez y capacidades de la red celular actual. [2]

2.3.1 Primera generación

La primera generación o 1G es llamada la era del ladrillo (1973-1988). Estas fueron normas de redes móviles, analógicas, que se introdujeron en los años 80 y continuó hasta que fue reemplazado por los teléfonos móviles digitales 2G. Una de las normas fue NMT (Nordic Mobile Telephone), usada en los países nórdicos, Europa del Este y Rusia. Otra fue AMPS (Advanced Mobile Phone System) utilizada en los Estados Unidos. Antecedentes de la tecnología 1G es el teléfono móvil de radio. Los teléfonos celulares se hicieron populares y recibieron la demanda del público durante el período 1983 a 1989. Los dispositivos eran instalados generalmente en coches, pues los primeros modelos de los dispositivos de primera generación no fueron diseñados para ser portátiles.



Figura 2.2: Motorola DynaTAC 8000X

2.3.2 Segunda generación

Durante la década de 1990 (1988- 1998), la tecnología sobre la que los teléfonos celulares funcionaban, fue llamada 2G. Estados Unidos y Europa utilizaban en ese momento proveedores digitales de telefonía celular y redes. Teléfonos celulares 2G tenían una red más rápida que la que funciona en las señales de radio.

2G sustituyó a la red de las frecuencias analógicas y finalmente, la adaptación de las redes modernas hicieron obsoletas las frecuencias analógicas. Los teléfonos celulares 2G eran más pequeños, con un peso aproximado de 100 a 200 gramos. Estos fueron de mano y eran portátiles. Normalmente no podían transferir datos, como correo electrónico o software, que no sean datos digitales de voz, y otros datos básicos auxiliares, como fecha y hora. Sin embargo, aparece el servicio SMS como una forma de transmisión de datos para algunas de las normas.

Por todos los avances que involucraron a los dispositivos móviles, sus baterías, y su procesamiento, la popularidad del teléfono celular creció rápidamente.



Figura 2.3: Nokia

2.3.3 Tercera generación

La tercera generación (1998-2008) permitió que los teléfonos no sólo sean capaces de hacer llamadas, enviar mensajes de texto y jugar al juego de la viborita sino que también pudieran reproducir música, tomar fotos o navegar por internet.

La tecnología 3G es la última de las de las generaciones en comunicaciones móviles. Está diseñada para que dispositivos móviles que incluyen una experiencia multimedia verdadera, con características de mayor velocidad de transferencia pasen a dar cabida a aplicaciones como las basadas en Web. Los servicios asociados con 3G proporcionan la posibilidad de transferir tanto voz como datos, que incluyen la descarga de información, intercambio de correo electrónico, mensajería instantánea y video en línea. Las tecnologías de 3G son una respuesta a la especificación de la Unión Internacional de Telecomunicaciones en el año 2000.

Las redes 3G tienen una velocidad de transferencia potencial de hasta 3 Mbps. En comparación, el teléfono más rápido de 2G podía alcanzar hasta 144 kbps. Los teléfonos 3G, son como mini-ordenadores portátiles y puede adaptarse a aplicaciones de banda ancha como la videoconferencia, streaming de vídeo desde Web, y enviar y recibir de inmediato mensajes de correo electrónico con archivos adjuntos.



Figura 2.4: Motorola RAZR

2.3.4 Cuarta generación

La cuarta generación o era del Smartphone ocurrió al mismo tiempo que la tercera y la quinta generación. Se extiende desde principios del año 2002 hasta el presente.

Dentro de los dispositivos móviles, un smartphone (cuya traducción en español sería “teléfono inteligente”) es una evolución del teléfono móvil tradicional que cuenta con ciertas características y prestaciones que lo acercan más a un ordenador personal que a un teléfono tradicional.

Entre dichas características, se puede encontrar una mejora en la capacidad de proceso y almacenamiento de datos, conexión a Internet mediante Wi-Fi, pantalla táctil, acelerómetro, posicionador geográfico, teclado QWERTY y diversas aplicaciones de usuario como navegador web, cliente de correo, aplicaciones ofimáticas, reproductores de vídeo y audio, etc. incluyendo la posibilidad de descargar e instalar otras nuevas.

A pesar de estas importantes mejoras con respecto a sus predecesores móviles, el reducido tamaño de los smartphones conlleva inexorablemente limitaciones de hardware que los mantienen claramente diferenciados de los ordenadores convencionales. Estas limitaciones se reflejan principalmente en pantallas más pequeñas, menor capacidad del procesador, restricciones de memoria RAM y memoria persistente, y necesidad de adaptar el consumo de energía a la capacidad de una pequeña batería.

Estas limitaciones obligan a tener muy presente la capacidad real del dispositivo a la hora de desarrollar su software, ya sean aplicaciones de usuario o el propio sistema operativo.



Figura 2.5: Los primeros smartphone provenían de empresas como Nokia, Handspring, y Research in Motion

2.4 Desarrollo en aplicaciones móviles

El desarrollo de aplicaciones móviles conlleva una variedad de consideraciones de acuerdo al propósito y escenario para el que van a ser utilizadas. Hace algunos años se tenía la creencia que desarrollar aplicaciones móviles era igual a desarrollar una aplicación tradicional pero en “pequeño”.

El desarrollo de aplicaciones móviles difiere del desarrollo de software tradicional en muchos aspectos, lo que provoca que las metodologías usadas para estos entornos también difieran de las del software clásico. Esto es porque el software móvil tiene que satisfacer una serie de requerimientos y condicionantes especiales que lo hace más complejo [19]:

- **Canal radio:** consideraciones tales como la disponibilidad, las desconexiones, la variabilidad del ancho de banda, la heterogeneidad de redes o los riesgos de seguridad deben tenerse especialmente en cuenta en este entorno de comunicaciones móviles.
- **Movilidad:** aquí influyen consideraciones como la migración de direcciones, alta latencia debido a cambio de estación base o la gestión de la información dependiente de localización. Sobre esta última, de hecho, se pueden implementar un sinnúmero de aplicaciones, pero la información de contexto asociada resulta muchas veces incompleta y varía frecuentemente.
- **Portabilidad:** la característica portabilidad de los dispositivos implica una serie de limitaciones físicas directamente relacionadas con el factor de forma de los mismos, como el tamaño de las pantallas, o del teclado.
- **Fragmentación de la industria:** la existencia de una considerable variedad de estándares, protocolos y tecnologías de red diferentes, añaden complejidad al escenario del desarrollo móvil.
- **Capacidades limitadas de los terminales:** incluimos factores como la baja potencia de cálculo o gráfica, los riesgos en la integridad de datos, las interfaces de usuario poco funcionales en muchos aspectos, la baja capacidad de almacenamiento, la duración de las baterías o la dificultad para el uso de periféricos en movilidad. Factores todos que, por otro lado, están evolucionando en

la dirección de la convergencia de los ultra portátiles (netbooks) con los dispositivos inteligentes, constituyendo cada vez menos un elemento diferencial.

- **Usabilidad:** las necesidades específicas de amplios y variados grupos de usuarios, combinados con la diversidad de plataformas tecnológicas y dispositivos, hacen que el diseño para todos se convierta en un requisito que genera una complejidad creciente difícil de acotar.
- **Time-to-market:** en un sector con un dinamismo propio, dentro de una industria en pleno cambio, los requisitos que se imponen en términos de tiempo de lanzamiento son muy estrictos y añaden complejidades en la gestión de los procesos de desarrollo.

El diseño de sistemas de software móvil es, por tanto, bastante más complejo que el tradicional [20], forzando a los investigadores a reconsiderar el uso de las metodologías actuales de desarrollo de software. El uso de metodologías ágiles es el medio más apropiado para el desarrollo de tecnología en móviles, aunque las características especiales de los terminales y de las redes de telefonía móvil demandan algunos ajustes sobre las actuales metodologías ágiles. Ha pasado ya mucho tiempo desde el estreno de plataformas abiertas para el desarrollo de aplicaciones móviles como Java o Symbian. La disponibilidad de herramientas y la facilidad para construir y comercializar software para el móvil ha hecho que su número se dispare. A eso se ha añadido, tal como se menciona más arriba, la aparición de las metodologías ágiles adaptadas para el desarrollo de aplicaciones para el escritorio. Los móviles son un medio diferente y se rige por un conjunto diferentes de reglas. Grandes productos móviles son capaces de adaptarse e incluso ir más allá de la estrategia tradicional para identificar nuevas soluciones y una forma única para hacer frente a los retos y los beneficios que el medio móvil tiene para ofrecer [21].

2.5 Tipos de aplicaciones móviles

Como hemos mencionado anteriormente, el ecosistema móvil es grande y complejo. Se tiene un número de opciones en los que el medio se utiliza para hacer frente a los objetivos, cada uno con sus pros y sus contras. Algunos desarrollos se apresuran a crear aplicaciones funcionales, pero accesible a menos usuarios. Otros se dirigen a un mercado más grande, pero son mucho más complejos y costosos. [2]

Cuando se enfrenta el desarrollo de un nuevo sistema móvil, se tiene que decidir en qué tipo de contexto la aplicación va a presentar su contenido o información. En otras palabras, qué tipo de aplicación es el más adecuado a nuestro problema o necesidad. A continuación se tratan los diferentes tipos de aplicaciones móviles y se definen cada una de estas opciones: sms, Sitios web para móviles, Widget Web móviles, Aplicaciones Web móviles, aplicaciones nativas y juegos.

2.5.1 SMS

La aplicación móvil más básica que podemos crear son las de tipo SMS. A pesar de que podría parecer extraño considerar las aplicaciones de mensajes de texto, son sin embargo un diseño con experiencia. Dada la ubicuidad de los dispositivos que soportan SMS, estas aplicaciones pueden ser herramientas útiles cuando se integra con otros tipos de aplicaciones móviles.

Normalmente, el usuario envía una palabra clave a un código corto de pocos dígitos que resulta en la devolución de información o un enlace a contenidos de calidad.

Los usos más comunes de las aplicaciones SMS son la descarga de contenido móvil, tal como tonos de llamada e imágenes o para interactuar con bienes y servicios reales. Los mensajes SMS también pueden utilizarse para compra de tiempo en un estacionamiento o zona de pago.

Las ventajas de las aplicaciones SMS son:

- Trabajan en cualquier dispositivo móvil casi instantáneamente.
- Son útiles para el envío de alertas a tiempo al usuario.
- Se pueden incorporar en cualquier Web o de aplicación móvil.
- Pueden ser fáciles de configurar y administrar.

Las desventajas de las aplicaciones SMS son:

- Están limitados a 160 caracteres.
- Ofrecen una experiencia limitada basada en texto.
- Las implementaciones pueden resultar costosas.

2.5.2 Sitios web para móviles

Un sitio web móvil es un sitio web diseñado específicamente para dispositivos móviles, que no debe confundirse con la visualización de un sitio hecho para los navegadores de escritorio en un navegador móvil.

Sitios web para móviles se caracterizan por su sencilla arquitectura "drill-down" o la simple presentación de enlaces de navegación que lo llevará a una página de un nivel más profundo.

Sitios web para móviles suelen tener un diseño sencillo y suelen ser de naturaleza informativa, ofrecen pocos elementos interactivos que se pueden esperar de un sitio de escritorio.

A pesar de que los sitios web para móviles son bastante fáciles de crear, no se muestran consistentemente a través de múltiples navegadores móviles. La web móvil ha ido en aumento en el uso en los últimos años en la mayoría de los principales mercados, pero la experiencia limitada ofrece pocos incentivos para el usuario. Muchos comparan la web móvil a una versión de 10 años atrás de la Web: lento, caro para el uso y no hay mucho que ver. Cuando se introdujeron mejores navegadores móviles a las plataformas de dispositivos como el iPhone y Android, la calidad de sitios web para móviles comenzó a mejorar de manera espectacular, y con ello, la mejora de su uso. Por ejemplo, en sólo un año, el mercado de EE.UU. pasó de estar apenas en los cinco principales consumidores de la web móvil al número uno, en gran parte debido al impacto del iPhone.

Las ventajas de sitios web para móviles son los siguientes:

- Son fáciles de crear, mantener y publicar.
- Se pueden utilizar las mismas herramientas y técnicas que se utilizan para sitios de escritorio.
- Casi todos los dispositivos móviles pueden acceder a sitios web móviles.

Las desventajas de los sitios web para móviles son los siguientes:

- Pueden ser difíciles de soportar múltiples dispositivos.
- Ofrecen a los usuarios una experiencia limitada.

- La mayoría de sitios web móviles son simplemente contenido de escritorio formateado para dispositivos móviles.
- Se pueden cargar las páginas lentamente, debido a la latencia de la red

2.5.3 Widget Web móviles

En gran parte en respuesta a la mala experiencia proporcionada por la Web móvil a través de los años, ha existido un creciente movimiento para establecer Widget. Un Widget es un componente de interfaz de usuario con un bloque independiente de código HTML que es ejecutado por el usuario de una manera particular.

Básicamente, los Widgets Web para móviles son pequeñas aplicaciones Web que no se puede ejecutar por sí mismos. Puede crear atractivas experiencias de usuario que aprovechan las funciones del dispositivo y, en muchos casos, requieren ejecutarse mientras el dispositivo está en línea.

Las ventajas de Widgets Web para móviles son:

- Son fáciles de crear, a partir de HTML, CSS y JavaScript
- Pueden ser fácil de implementar a través de múltiples dispositivos.
- Ofrecen una experiencia de usuario mejorada y un diseño más rico, y algunos no requieren conexión.

Las desventajas de Widgets Web para móviles son:

- Por lo general requieren una plataforma de Widgets compatibles para ser instalado en el dispositivo.
- Generalmente no se puede ejecutar en cualquier navegador Web para móviles.
- En algunos casos, se requiere el aprendizaje adicional técnicas propias de Widgets.

2.5.4 Aplicaciones Web móviles

Aplicaciones web para móviles son las aplicaciones móviles que no necesitan ser instalados o compilados en el dispositivo destino. El uso de XHTML, CSS y JavaScript, son capaces de proporcionar una experiencia de aplicación para el usuario final mientras se ejecuta en cualquier navegador web para móviles. Las aplicaciones web permiten a los usuarios interactuar con el contenido en tiempo real, donde un solo clic o toque realiza una acción dentro de la vista actual.

Poco después de la explosión de la Web 2.0, las aplicaciones web como Facebook, Flickr y Google Reader golpeó a los navegadores de escritorio, y hubo una discusión de cómo llevar las mismas aplicaciones web para dispositivos móviles. El movimiento Web 2.0 trajo los principios de diseño centrados en el usuario de escritorio a la web y fueron muy necesarios en el espacio web móvil.

El reto, como siempre, fue la fragmentación del dispositivo. Los navegadores móviles fueron años detrás de los navegadores de escritorio, haciendo casi imposible para un dispositivo móvil hacer una experiencia comparable. Si bien el apoyo de XHTML se había convertido bastante común a través de dispositivos, la prestación de CSS2 fue muy inconsistente y el soporte para JavaScript, DHTML y Ajax era completamente inexistente.

Con la introducción del iPhone, se ha producido un gran cambio en todos los ámbitos. Con el uso de WebKit, el iPhone podía renderizar aplicaciones web no optimizadas para dispositivos móviles

como perfectamente usables, incluyendo DHTML. De esta manera rápidamente creció la creación de aplicaciones web para móviles optimizadas sobre todo para el iPhone.

Las ventajas de las aplicaciones web móviles son las siguientes:

- Son fáciles de crear, con base HTML, CSS y JavaScript.
- Son fáciles de implementar a través de múltiples dispositivos.
- Ofrecen una mejor experiencia de usuario y un diseño rico, aprovechando las características del dispositivo y el uso sin conexión.
- El contenido es accesible a cualquier navegador web para móviles.

Las desventajas de las aplicaciones web móviles son las siguientes:

- La experiencia óptima podría no estar disponible en todos los teléfonos.
- Puede ser un reto (aunque no imposible) soportar múltiples dispositivos.
- No siempre compatible con las funciones nativas de la aplicación, al igual que el modo sin conexión, la búsqueda de ubicación, el acceso al sistema de archivos, la cámara, etc.

2.5.5 Aplicaciones Nativas

Estas aplicaciones en realidad debería llamarse "aplicaciones de la plataforma", ya que tienen que ser desarrolladas y compiladas para cada plataforma móvil.

Estas aplicaciones nativas o de la plataforma se construyen específicamente para los dispositivos que ejecutan la plataforma en cuestión. El más común de todas las plataformas es Java ME (J2ME). En teoría, un dispositivo escrito como un MIDlet Java ME debería funcionar en la gran mayoría de los teléfonos vendidos en todo el mundo. La realidad es que incluso una aplicación escrita en un MIDlet Java ME todavía requiere algunos ajustes y pruebas para cada dispositivo que se implementa.

La creación de una aplicación nativa significa decidir qué dispositivos se van a soportar, tener un medio de pruebas y certificación, y un método para distribuir la aplicación a los usuarios. La gran mayoría de aplicaciones de la plataforma están certificadas, se venden y se distribuyen a través de un portal del operador o una tienda de aplicaciones. Es posible crear una aplicación Java ME MIDlet y publicar de forma gratuita en la Web, pero se hace rara vez.

Dado que las aplicaciones de la plataforma se posicionan en la parte superior de la capa de la plataforma se puede aprovechar la mayoría de las características del dispositivo, trabajar en línea o sin conexión, acceder a la ubicación y al sistema de archivos, utilizar la cámara del dispositivo, etc. De ahí la necesidad de una certificación antes de que la aplicación se distribuya, para asegurarse de que nadie distribuya una aplicación que roba datos personales de un usuario o maliciosamente utiliza el dispositivo para propagar virus.

Las ventajas de las aplicaciones nativas son:

- Ofrecen una mejor experiencia de usuario, ofreciendo un diseño rico y aprovechando las características del dispositivo dando la posibilidad de utilizarla sin conexión.
- Son relativamente fáciles de desarrollar para una sola plataforma.
- Se puede cobrar por las aplicaciones.

Las desventajas de las aplicaciones nativas son:

- No pueden ser fácilmente portadas a otras plataformas móviles.
- El desarrollo, prueba y soporte de múltiples plataformas de dispositivos es muy costoso.

- Se requiere la certificación y la distribución de un tercero.
- Requiere que comparta los ingresos con uno o varios terceros.

2.5.6 Juegos

Los juegos son los tipos de aplicaciones más populares para dispositivos móviles.

Técnicamente, los juegos son en realidad aplicaciones nativas que utilizan la SDK de la plataforma para crear experiencias de inmersión. Pero los tratamos de forma diferente de las aplicaciones nativas por dos razones: no pueden ser fácilmente duplicada como aplicaciones Web, y la migración a otras plataformas móviles es un poco más fácil que las típicas aplicaciones nativas.

Los juegos son relativamente fáciles de portar, es que la mayor parte de un juego está en los gráficos y de hecho utiliza muy poco de la API dispositivo. La mecánica del juego es lo único que tiene que adaptarse a las diversas plataformas.

Las ventajas de aplicaciones de juegos son los siguientes:

- Proporcionan una manera simple y fácil de crear una experiencia de inmersión.
- Pueden ser portado a múltiples dispositivos con relativa facilidad.

Las desventajas de aplicaciones de juegos son las siguientes:

- El desarrollo puede ser costoso para un juego original.
- No puede ser portado a la Web móvil.

2.5.7 Comparación de tipos de aplicaciones

En resumen, para ayudar a comparar y contrastar cuál de estos tipos de aplicaciones móviles es mejor para un determinado producto móvil, se presentan en la siguiente tabla.

	Dispositivo de soporte	Complejidad	Experiencia de usuario	Lenguaje	Soporte offline	Características del dispositivo
SMS	Todos	Simple	Limitada	Ninguno	No	Ninguna
Sitios Web	Todos	Simple	Limitada	HTML	No	Ninguna
Widget Web	Algunos	Media	Buena	HTML	Limitado	Limitada
Aplicaciones Web	Algunos	Media	Buena	HTML, CSS, Javascript	Limitado	Limitada
Aplicaciones Nativas	Todos	Alta	Excelente	Varios	Sí	Sí
Juegos	Todos	Alta	Excelente	Varios	Sí	Sí

Tabla 2.1: Comparación de tipos de aplicaciones

Capítulo 3. Realidad Aumentada

En el capítulo anterior hemos descrito el concepto de dispositivo móvil, su evolución y características. En el presente capítulo explicaremos el concepto de realidad aumentada, su implementación y ejemplos de su uso en la actualidad.

3.1 Introducción

En el año 1992 Tom Caudell crea el término Realidad Aumentada (RA) para describir una pantalla que usarían los técnicos electricistas de Boeing que mezclaba gráficos virtuales con la realidad física. Este sistema les permitiría aumentar la eficiencia de su trabajo al facilitarles de alguna forma la operativa sobre las tareas a realizar.

La RA es una variación de la realidad virtual (RV). Las tecnologías de la RV sumergen completamente a un usuario dentro de un entorno sintético. Mientras se está inmerso, el usuario no puede ver el mundo real que le rodea. Por el contrario, la RA permite al usuario ver el mundo real, con objetos virtuales superpuestos o compuesto con el mundo real. Por lo tanto, RA suplementa la realidad, en lugar de reemplazarla por completo. Lo ideal sería que al usuario le parezca que los objetos virtuales y reales coexisten en el mismo espacio.

La Realidad Aumentada mejora la percepción del usuario y la interacción con el mundo real. Los objetos virtuales muestran la información que el usuario no puede detectar directamente con sus propios sentidos. La información transmitida por los objetos virtuales ayuda a los usuarios a realizar tareas del mundo real. [22]

Siguiendo la definición del autor Ronald Azuma [22] [23] un sistema de RA tiene 3 requerimientos:

- Combina la realidad con información sintética
- Los objetos virtuales están registrados en el mundo real
- Es interactivo en tiempo real

Desde un punto de vista más amplio, la RA es una aplicación interactiva que combina la realidad con información sintética -tal como imágenes 3D, sonidos, videos, texto, sensaciones táctiles- en tiempo real y de acuerdo al punto de vista del usuario.

La información virtual tiene que estar vinculada espacialmente al mundo real, es decir, un objeto virtual siempre debe aparecer en cierta ubicación relativa a un objeto real. La visualización de la escena aumentada (mundo real + sintético) debe hacerse de manera coherente para el usuario. Se denomina registro de imágenes (registration) al proceso de tener diferentes datos en relación a un único sistema de coordenadas. Por este motivo, se necesita saber en todo momento la posición del usuario, tracking, con respecto al mundo real y de esta manera la mezcla de información real y sintética podrá registrarse.

Se denomina Tracking [24] al proceso de seguimiento de la posición y orientación de un objeto. Puede realizarse tracking del usuario o el tracking de objetos reales, por ejemplo, en el caso de una aplicación de medicina de entrenamiento para cirugía en la que el participante sostiene un bisturí.

A diferencia de las aplicaciones de RV, las aplicaciones de RA generalmente necesitan la movilidad del usuario, incluso hacia ambientes externos- en inglés se denominan aplicaciones outdoor. En

dichas aplicaciones de realidad aumentada puede ser necesaria conocer la posición global del participante utilizando dispositivos como GPS y brújulas digitales.

Dependiendo del dispositivo de visualización usado las aplicaciones de realidad aumentada pueden basarse en[22]:

- Gafas de video see-through
- Gafas de óptica see-through
- Proyector
- Monitor
- Dispositivos móviles o HandHeld



Figura 3.1 - Gafas de óptica see-through



Figura 3.2 - Gafas de video see-through



Figura 3.3 - Proyector



Figura 3.4 - Monitor



Figura 3.5 - Dispositivos Handheld

3.2 Aplicaciones

En Azuma (1997) [22] pueden encontrarse ejemplos de los primeros dominios de aplicación típicos de la realidad aumentada como medicina, fabricación y reparación, anotación y visualización, Planificación de trayectorias Robot y aplicaciones militares.

3.2.1 Medicina

Los médicos podrían usar la realidad aumentada para la visualización y entrenamiento para la formación en cirugías. Puede ser posible recoger en conjuntos de datos 3D de un paciente en tiempo real, usando sensores no invasivos como la resonancia magnética (MRI), tomografía computarizada (TC) o imágenes de ultrasonido. Estos conjuntos de datos podrían ser traducidos y combinados en tiempo real para ver el paciente real. En efecto, esto le daría a un médico la "visión de rayos X" dentro de un paciente.



Figura 3.1: Feto Virtual dentro del vientre de una paciente embarazada.

3.2.2 Fabricación y reparación

Las instrucciones que tienen que ver con pasos para fabricar o reparar un objeto determinado podrían ser más fáciles de entender si estuvieran disponibles, no como manuales con texto e imágenes, sino más bien como dibujos 3D superpuestos en el objeto real, mostrando paso a paso las tareas que hay que hacer y cómo hacerlas.

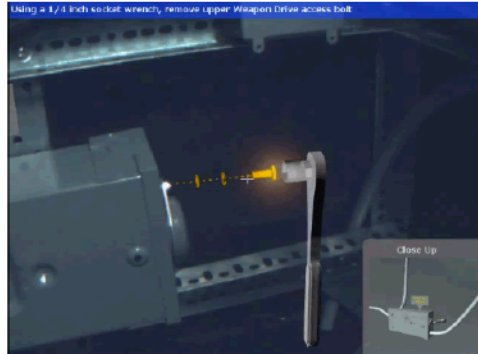


Figura 3.2: Mecánicos Militares: <http://www.youtube.com/watch?v=mn-zvymiSvk>

3.2.3 Anotación y visualización

La RA podría ser utilizada para poner notas de objetos y entornos con información pública o privada. Por ejemplo, si estuviera disponible una base de datos que contiene información sobre la estructura de un edificio, la aplicación RA podría dar a los arquitectos una "visión de rayos X" del interior de un edificio, mostrando dónde están las tuberías, conexiones eléctricas y las estructuras de soporte que se encuentran dentro de las paredes.

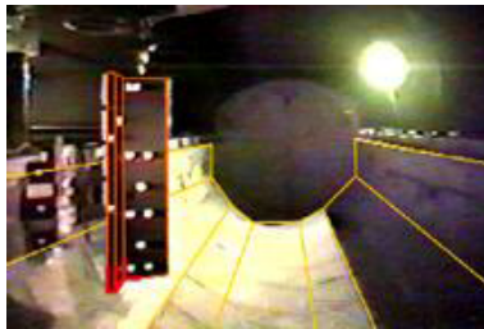


Figura 3.3: Líneas virtuales ayudan a la visualización de la geometría, como se ve en órbita.
(Courtesy David Drascic and Paul Milgram, U. Toronto.)

3.2.4 Planificación de trayectorias Robot

La teleoperación de un robot es a menudo una tarea difícil, especialmente cuando el robot está lejos, con largos retrasos en el enlace de comunicación. Bajo esta circunstancia, en lugar de controlar directamente el robot, puede ser preferible controlar una versión virtual del robot. El usuario planea y especifica las acciones del robot mediante la manipulación de la versión virtual, en tiempo real. Los resultados se muestran directamente en el mundo real. Una vez que el plan está probado y determinado, entonces el usuario le dice al robot real que ejecute el plan especificado.

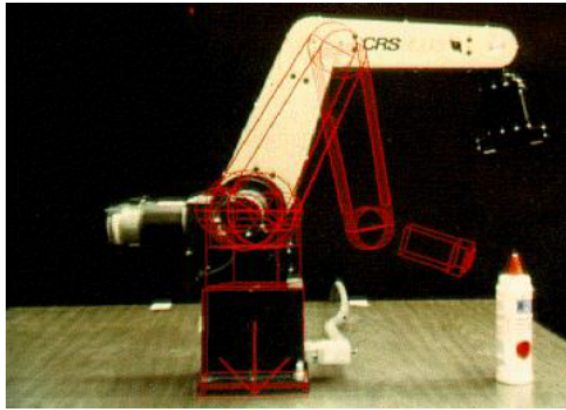


Figura 3.4: Líneas virtuales muestran un movimiento previsto de un brazo de robot (Courtesy David Drascic and Paul Milgram, U. Toronto.)

3.2.5 Aplicaciones Militares

Aumentar el campo de visión de los pilotos de combate con información de navegación, instrucciones, mapas y ubicaciones enemigas ha sido una de las aplicaciones más práctica de realidad aumentada.



Figura 3.5: Visión de piloto de combate

3.2.6 Aplicaciones Actuales

La aparición de la librería de código abierto ARToolkit [25] en el año 1999 dio lugar a un gran crecimiento en el desarrollo de aplicaciones de realidad aumentada. En Azuma (2001) [23] presenta una actualización de la publicación anterior donde distingue tres grandes áreas de nuevas aplicaciones clasificadas en: aplicaciones móviles, aplicaciones colaborativas y aplicaciones comerciales, que desde la fecha han ido en auge creciente.

Mientras que las primeras aplicaciones de realidad aumentada se desarrollaban en un entorno conocido y controlado, el interés en aplicación de realidad aumentada para un ambiente externo (outdoor) fue creciendo.

Dentro de las aplicaciones móviles pueden encontrarse aplicaciones para asistencia a la navegación, la recuperación de información geográfica, aplicaciones de arquitectura, museística, juegos, etc. Dentro de las aplicaciones colaborativas pueden encontrarse aplicaciones de diseño,

entretenimiento y educación. En los últimos diez años hubo una explosión de aplicaciones comerciales - publicitarias- que hacen uso de la realidad aumentada. En particular, pueden encontrarse muchos ejemplos de aplicaciones web donde el usuario interactúa con una escena virtual por medio de un marcador impreso.

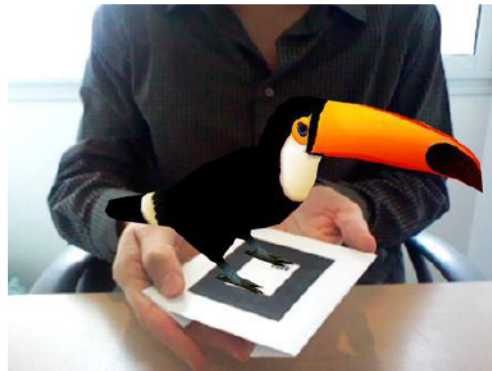


Figura 3.6: RA utilizando marcadores

3.3 Diagrama de una aplicación de realidad aumentada

Una aplicación de realidad aumentada tiene tres partes principales que son:

- Tracking
- Recuperación de información
- Salida de información

La figura 3.7 muestra un diagrama de las diferentes partes involucradas en una aplicación de realidad aumentada visual:

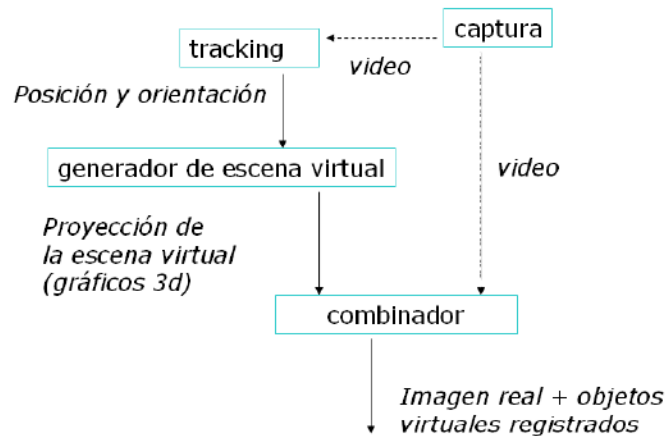


Figura 3.7: Diagrama de las partes de una aplicación de realidad aumentada

3.3.1 Captura de la escena real

Una de las tareas más significativas en todo sistema de RA es la de identificar la escena que se quiere aumentar. En el caso de los sistemas que utilicen reconocimiento visual, es indispensable disponer de algún mecanismo que permita tomar la escena para luego ser procesada. Entonces generalmente se inicia con el registro de las señales del mundo real (generalmente video, aunque

bien pudiera ser audio). Los dispositivos de captura de imágenes son dispositivos físicos que recogen la realidad para luego ésta poder ser aumentada. Básicamente, estos dispositivos se agrupan en dos grupos:

- **Dispositivos see-through:** estos dispositivos realizan simultáneamente la tarea de capturar la escena real como la de mostrar la información aumentada al usuario. Estos dispositivos acostumbran a trabajar en tiempo real, haciéndolos no sólo más costosos en presupuesto sino también en complejidad. Dentro de este grupo encontramos aquellos dispositivos conocidos como head-mounted. Sabemos que estos dispositivos see-through llevan años siendo utilizados, por ejemplo, en los Head Up Displays (HUDs) utilizados por los aviones de combate para visualizar información sobre altura, velocidad, identificación de objetivos, y otros datos sin necesidad de retirar la vista de la zona frontal de la cabina.
- **Dispositivos video-through:** dentro de este grupo encontramos los dispositivos donde el módulo que realiza la captura de video es independiente al módulo de visualización. En este se encontrarían las cámaras de video o los terminales móviles.

3.3.2 Tracking

En aplicaciones de RV y RA se debe realizar el seguimiento o tracking del participante [26], para determinar su posición y orientación en el mundo virtual (en aplicaciones de realidad virtual) o en el mundo real (en el caso de aplicaciones de realidad aumentada).

El tracking es proceso de seguimiento de un objeto en movimiento, es decir, la estimación de la posición y orientación del mismo en cada instante.

En una aplicación de realidad aumentada se necesita el tracking del participante para conocer la matriz de transformaciones geométricas y así realizar el registro de imágenes sintéticas y reales.

En la mayoría de aplicaciones de realidad aumentada se trata de realizar un tracking de la cámara que captura la escena. Aunque también puede tratarse del tracking de la cabeza del usuario o de algún objeto manipulado por este.

El tracking completo estima los 6 parámetros o grados de libertad (DOF). 3 para la traslación en cada eje T_x, T_y, T_z ; y 3 para los ángulos de rotación con respecto a cada eje $f^{\hat{E}}_x, f^{\hat{E}}_y, f^{\hat{E}}_z$.

Según la aplicación puede interesar conocer:

- La posición y orientación de la cámara en un sistema de coordenadas global
- La posición y orientación de la cámara en relación a un objeto de la escena real en cuestión
- La posición y orientación de la cámara en relación a la posición y orientación del cuadro de video anterior

El tracking en una aplicación de realidad aumentada puede hacerse:

- Mediante dispositivos físicos específicos.
- Mediante el análisis de la imagen capturada, denominado tracking basado en visión
- Tracking híbrido, que combina las salidas de dispositivos físicos con el análisis de la imagen

3.3.2.1 Tracking mediante dispositivos físicos

Según Zhou (2008) [27], el tracking basado puramente en sensores fue utilizado en las primeras aplicaciones de realidad aumentada, encontrándose muy pocas publicaciones recientes que no utilicen combinaciones con tracking basado en visión.

Según la tecnología el tracking puede realizarse utilizando sensores mecánicos, magnéticos, ultrasónicos, inerciales u ópticos. Cada uno de estos dispositivos presenta sus ventajas y desventajas.

Actualmente, pueden encontrarse en el mercado teléfonos móviles con dispositivos integrados cuyo uso resulta muy apropiado para aplicaciones de realidad aumentada móvil. A continuación se detallan los dispositivos más comunes que pueden usarse para tracking en una aplicación de realidad aumentada en ambiente externo:

- **GPS (Global Positional System):** nos da la latitud y longitud en el sistema de coordenadas global que puede usarse para consultar un Sistema de Información Geográfica (SIG). El GPS brinda los 3 DOF de la traslación. Para mejorar la exactitud del valor que brinda puede usarse lo que se denomina GPS diferencial, donde una estación base de la cual se conoce su localización con exactitud computa y transmite el error introducido, por ejemplo por los efectos atmosféricos, el cual es utilizado por el receptor GPS para corregir su posición. Para obtener los 6 DOF se necesita combinarlo con otro dispositivo que brinde los 3 DOF de la rotación como una brújula.

- **Brújula digital:** al igual que una brújula convencional brinda la orientación en un sistema global (3 DOF). Las brújulas que se encuentran en los teléfonos móviles son denominadas brújulas de estado sólido, generalmente construidas mediante sensores de campo magnético que envían señales a un microprocesador.

Generalmente se combina con el GPS para obtener los 6 DOF del movimiento del dispositivo. La ventaja con respecto al uso de sensores inerciales es que brindan un resultado con error constante, el cual puede precalibrarse durante la instalación del sistema.

- **Sensores inerciales:** los acelerómetros y giroscopios permiten conocer aceleración y velocidad de rotación, a partir de las cuales puede conocerse los 6 DOF de la pose. Las ventajas de este tipo de dispositivos radica en su rapidez y su buena respuesta a cambios bruscos. Sin embargo, su principal desventaja suele ser la acumulación de errores debido al ruido, y por esto cada cierto tiempo deben recalibrarse. Algunos modelos de teléfonos móviles de última generación poseen integrados acelerómetros y giroscopios con tecnología MEMS (Micro Electro Mechanical Systems).

3.3.2.2 Tracking basado en visión

En lugar de utilizar dispositivos físicos específicos para tracking, la posición y orientación de la cámara en la escena real puede estimarse analizando el video capturado por una cámara utilizando técnicas de visión por computador. En resumen se trata de buscar en la imagen elementos de la escena real que permitan deducir la posición y orientación de la cámara en relación a una cierta referencia, o de forma equivalente la posición y orientación de los objetos en relación a la cámara. Según la aplicación puede tratarse de una cámara integrada al participante en movimiento, o puede tratarse de una cámara fija en una escena con objetos en movimiento.

En dispositivos de visualización video see-through (aquí incluimos también al monitor y los dispositivos hand-held), el video capturado por la cámara se usa simultáneamente:

- como el fondo (video background) de la escena que ve el usuario en la pantalla
- para realizar el tracking

Existen diferentes estrategias usadas para tracking basado en visión, que pueden clasificarse en dos grandes ramas:

- Tracking de marcadores
- Tracking basado en características naturales

El tracking de marcadores fue la primera estrategia utilizada y consiste en introducir uno o más marcadores conocidos en la escena real para superponer objetos virtuales o utilizarlos como interfaz tangible. Esta estrategia tuvo mucho auge desde la publicación de Kato y Billinghurst (1999) [28], según Zhou (2008) [27] uno de los artículos más referenciados en realidad aumentada y la disponibilidad del código abierto de su desarrollo ARToolkit [25]. Es muy usada actualmente, hecho fácilmente comprobable navegando en la web donde pueden encontrarse infinidad de aplicaciones de publicidad.

Dado que puede resultar molesto o indeseable el agregar marcadores visibles a la escena, recientemente la investigación se derivó a encontrar otros algoritmos de tracking basado en visión que exploren la presencia de características naturales en la misma (puntos, líneas, bordes, texturas) evitando la introducción de marcadores. Zhou (2008) [27] afirma que hasta la fecha de su publicación esta ha sido el área más activa en la investigación de tracking basado en visión, y se podría afirmar que actualmente lo sigue siendo.

Entre los algoritmos que exploran las características naturales de la escena existe la siguiente gran división:

- Tracking con conocimiento de la escena
- Tracking sin conocimiento de la escena

En las siguientes secciones se detalla cada uno de los enfoques de tracking basado en visión.

Tracking de marcadores

Un marcador, denominado en inglés “fiducial markers”, es una imagen 2D impresa con un formato específico conocido por la aplicación de tracking.

En la figura 3.8 pueden verse diferentes tipos de marcadores de los que existen lectores de código abierto.

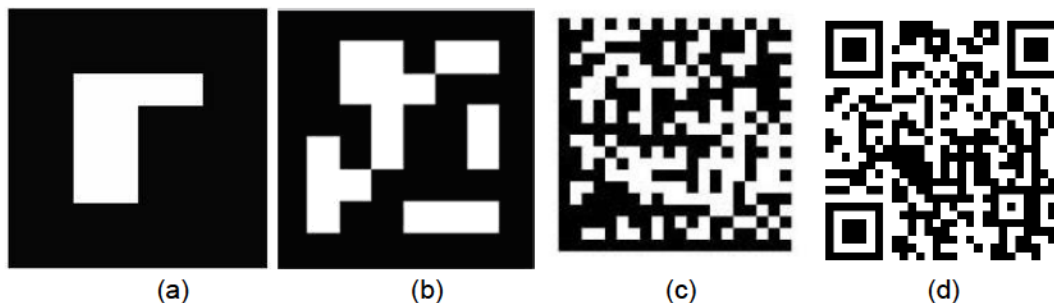


Figura 3.8. Diferentes tipos de marcadores: template (a); ID-Marker (b); DataMatrix (c), QRCode (d)

Los marcadores template son muy conocidos dado que son los utilizados por la librería de realidad aumentada ARToolkit[25], que fue la primera librería que popularizó las aplicaciones de realidad aumentada. El formato es un cuadrado negro y dentro un cuadrado blanco que tiene dentro una imagen asimétrica en negro.

Los marcadores ID-marker codifican un número de 9-bits (hasta 512 diferentes) en un patrón de 6 x 6, repitiendo los 9 bits 4 veces completando los 36 bits. Una variante de estos marcadores son los denominados BCH (Bose, Ray-Chaudhuri, Hocquenghem), los cuales son más robustos que los anteriormente descritos, ya que usa un algoritmo avanzado de chequeos de redundancia cíclica (CRC) que permite restaurar marcadores dañados. Se incrementa el número de marcadores disponibles a un total de 4096. Este tipo de marcadores o una variante de los mismos son los utilizados por las librerías de realidad aumentada ARTag [29] y ARToolkitPlus [30].

Los marcadores DataMatrix y marcadores QRCode no fueron diseñados específicamente para realidad aumentada, sino que su propósito inicial es codificar una serie de caracteres ASCII. Uno de los usos más comunes es la codificación de una URL de forma que una aplicación al leerlos y decodificarlos, pueda derivar al sitio web codificado. Por esto, su uso principal se asocia a los hipervínculos y no a realidad aumentada.

Mientras que los DataMatrix pueden almacenar hasta 2335 caracteres, los QR Code almacenan 4296 caracteres. Una diferencia entre ellos radica que el QRCode incluye además símbolos japoneses. Ambos códigos son abiertos y pueden descargarse de forma gratuita aplicaciones lectoras para los diversos teléfonos celulares del mercado. Desde un punto de vista amplio algunos autores los consideran una forma de realidad aumentada. Además, existen algunas aplicaciones de realidad aumentada que utilizan dichos códigos no solo con el propósito mencionado sino como marcadores para tracking.

Según la aplicación podrá tratarse de una cámara fija con un marcador en movimiento, o de un marcador fijo y una cámara en movimiento. En cualquiera de los casos el cálculo no varía ya que se trata de una posición relativa entre la cámara y el marcador.

Básicamente el algoritmo de tracking basado en marcadores consta de dos pasos principales:

- Detección e identificación de marcadores
- Estimación de la matriz de transformaciones geométricas de cada marcador detectado

Tracking basado en características naturales con conocimiento de la escena

Con el objetivo de evitar invadir una escena con marcadores surgieron las técnicas de tracking basado en características naturales, es decir basadas en la localización de características como puntos, líneas, bordes o texturas.

Dentro de los diferentes enfoques basados en características naturales se pueden distinguir:

- Métodos basados en bordes
- Métodos basados en texturas

Los dos enfoques anteriores relacionan puntos de cada cuadro de video con el cuadro anterior, y dado que el punto de vista de la cámara varía ligeramente de un cuadro a otro, se dice que son técnicas de correspondencia de desplazamiento de cámara corto o en inglés “short-baseline-matching”.

Un enfoque diferente también basado en características naturales son los denominados:

- Métodos basados en detección

A diferencia de los enfoques anteriores que exploran las relaciones entre cuadros vecinos, este tipo de técnicas relaciona las características de un cierto cuadro con cuadros almacenados en una base de datos que corresponden a puntos de vista diferentes. Estas son técnicas se denominan de correspondencia de desplazamiento de cámara ancho, o en inglés “wide-baseline-matching”.

Generalmente todos estos métodos tienen en común que se debe tener algún tipo de conocimiento del objeto o escena real. Por esto se denominan métodos basados en el modelo, o en inglés “model-based”.

Tracking sin conocimiento de la escena

Los enfoques descritos en la sección anterior son enfoques basados en el conocimiento del modelo. Un problema de gran interés reciente en aplicaciones de realidad aumentada es realizar el tracking en escenas desconocidas total o parcialmente. Este es un problema más complejo ya que no solo debe estimarse la posición de la cámara sino que también debe realizarse un mapa del ambiente desconocido.

Los algoritmos utilizados para tracking en escenas desconocidas se basan en un algoritmo denominado SLAM (Simultaneous Localisation and Mapping) que fue desarrollado por Thrun (2004) [31] en la comunidad de robótica. Este algoritmo no sólo deduce la estructura del ambiente sino que al mismo tiempo establece una correlación de la misma con la posición y orientación de la cámara.

3.3.2.3 Tracking híbrido

En algunas aplicaciones de realidad aumentada las técnicas de visión por computador no proveen una solución de tracking robusta, por esto se han desarrollado métodos de tracking híbrido que consisten en combinar las salidas de dispositivos físicos y el análisis de video. Este tipo de tracking resulta efectivo para aplicaciones que requieren estimar la posición de la cámara con respecto a una escena estática, pero no se aplica al tracking de objetos en movimiento con una cámara estática.

Por un lado, las técnicas de tracking basado en visión son estables a largo plazo. Sin embargo, pueden resultar lentas y además los movimientos bruscos suelen causar fallas en el tracking con el consecuente consumo de tiempo en la recuperación. Por otra parte, el tracking basado en sensores, en particular el tracking inercial ofrece características complementarias: es rápido y robusto y puede usarse para predecir el movimiento cuando ocurren cambios bruscos. A partir de la aceleración y la velocidad de rotación puede estimarse la pose, pero su desventaja radica en que tienden a ir a la deriva debido a acumulación de ruido.

3.3.3 Generador de la escena virtual

En el proceso de formación de imágenes, reales o sintéticas, existe:

- una escena compuesta por diversos objetos (en diferentes posiciones y orientaciones)
- una cámara, con una cierta posición y orientación en la escena
- una imagen resultante de proyectar la escena de acuerdo a la cámara

Cuando se habla de posición y orientación de un objeto o de una cámara debe existir un sistema de referencia en base al cual se expresan. En realidad aumentada, al hablar de imágenes

registradas significa que tanto las imágenes sintéticas como el mundo real estén en referencia al mismo sistema de coordenadas.

En adelante nos referiremos a una serie de sistemas de coordenadas que se detallan a continuación.

- **Sistema de coordenadas local:** Es un sistema de coordenadas 3D utilizado para referenciar los puntos de un objeto. Para entenderlo mejor puede pensarse que al modelar un objeto se utilizará un sistema de coordenadas local al objeto, situado en algún punto del mismo. Por ejemplo si se modela un árbol seguramente lo haremos situando un sistema de coordenadas centrado en el árbol o en la base del mismo, con un eje alineado con el tronco.

- **Sistema de coordenadas mundo:** La posición y orientación de los objetos de una escena suele expresarse en relación a un sistema de coordenadas 3D situado en algún lugar del mundo. Al modelar una escena completa se utiliza un sistema de coordenadas único situado en algún lugar de la escena, denominado sistema de coordenadas mundo. Todos los objetos que integran una escena se posicionan, orientan y escalan en relación al mismo sistema de coordenadas mundo.

- **Sistema de coordenadas cámara:** El sistema de coordenadas cámara 3D tiene el origen en el centro óptico de la cámara. El eje de proyección de la cámara pasa por el centro óptico y es perpendicular al plano de formación de la imagen. El eje Z del sistema de coordenadas cámara está alineado con el eje de proyección, con Z+ hacia donde se ubica la escena. El eje Y tiene dirección vertical y el eje X dirección horizontal.

El sistema de coordenadas mundo 3D puede expresarse en relación al sistema de coordenadas cámara 3D mediante transformaciones geométricas. De forma equivalente, el sistema de coordenadas cámara puede expresarse en relación al sistema de coordenadas mundo.

- **Sistema de coordenadas de la imagen:** Los puntos de la imagen se expresan en relación a un sistema de coordenadas 2D con los ejes alineados con los bordes de la imagen.

Dado un punto P_m expresado en coordenadas mundo, se expresa en coordenadas cámara mediante transformaciones geométricas, y a continuación se proyecta para obtener el punto 2D de la imagen. Estas sucesivas transformaciones pueden expresarse como multiplicaciones de matrices sucesivas.

En una aplicación de realidad aumentada se tiene una escena sintética modelada en un cierto sistema de coordenadas mundo. Existe un participante que visualiza la escena desde un punto de vista, el de la cámara con la que captura la escena real. A medida que el participante se mueve en el mundo real se debe producir un cambio en el punto de vista del mundo virtual de acuerdo al cambio de posición y orientación del participante. Por esto, para poder combinar los objetos virtuales con la realidad de forma coherente (imágenes registradas) se debe conocer en cada momento la relación entre el sistema de coordenadas mundo y el sistema de coordenadas cámara. Esto significa conocer la matriz de transformaciones geométricas del sistema de coordenadas del mundo con respecto a la cámara, o de forma equivalente la matriz de transformaciones de la cámara en relación al sistema de coordenadas mundo.

Para poder obtener la imagen sintética debe conocerse además la matriz de proyección. A diferencia de la matriz de transformaciones geométricas la matriz de proyección no cambia mientras la cámara se mueve dado que está basada en características intrínsecas de la misma. Por esto se calcula una sola vez con un procedimiento a priori denominado calibración.

Conocidas la matriz de transformaciones geométricas y la matriz de proyección puede obtenerse la proyección de la escena virtual aplicando operaciones de transformación como rotación, traslación y escalado.

3.3.4 Combinación del mundo virtual y la escena real

En los casos en los que existe una visión directa del mundo real la combinación se realiza directamente en el ojo del participante. En otro caso, se realiza un video resultado de la escena capturada y la escena sintética proyectada.

3.4 Realidad aumentada en dispositivos móviles

Las primeras aplicaciones de realidad aumentada móvil, y para ambientes externos (outdoor), utilizaban una mochila para cargar el conjunto de hardware incluyendo la PC y las fuentes de alimentación, y un visor en la cabeza o HMD (head mounted display). Las aplicaciones de realidad aumentada móvil siguen el modelo de dispositivo sostenido con la mano (hand-held) utilizando una tablet PC, una PDA o un smartphone o teléfono celular de última generación. Estos dispositivos cuentan con cámara integrada y opcionalmente dispositivos para tracking integrados como GPS, acelerómetro, giroscopio, brújula.

En general, los dispositivos móviles como PDAs y teléfonos móviles tienen unas características diferentes a los PC en relación al hardware y software. Dichas características influyen en el desarrollo de cualquier algoritmo para este tipo de dispositivos, particularmente de realidad aumentada.

La tecnología cambia muy rápido y recientemente hubo un gran salto en los teléfonos celulares. Entre los avances más notorios relacionados con el campo de realidad aumentada se pueden enumerar el aumento de velocidad de la CPU y la inclusión de procesador gráfico.

Entre los sistemas operativos más importantes se enumeran Symbian, Windows Mobile, Android, iOS (para Iphone). Pese a que las plataformas anteriores son programables son mutuamente incompatibles, lo que hace el diseño de software para varias plataformas más dificultoso. Aún entre diferentes modelos de dispositivos que soportan el mismo sistema operativo puede haber pequeñas incompatibilidades del hardware de bajo nivel que requiere recompilación para cada modelo. Actualmente puede encontrarse una amplia variedad de aplicaciones, incluso de realidad aumentada, desarrolladas tanto para Android como para iOS, los dos sistemas operativos más dominantes.

El tracking en dispositivos "hand-held" fuerza ciertas restricciones no presentes en otras configuraciones basadas en PC. Los sensores externos generalmente no son posibles y además de su alto coste, los dispositivos móviles son pequeños y no tienen la interfaz necesaria para conectarlos. Actualmente existen móviles con GPS y sensores como acelerómetros y giroscopios incorporados.

El software de tracking tiene que diseñarse específicamente para correr bajo estas plataformas restringidas. El tracking basado en cámaras es una buena opción para estos dispositivos ya que es una alternativa que da resultados buenos con bajo coste.

El tracking de marcadores es una de las estrategias más usadas ya que es robusta y computacionalmente eficiente. Existen librerías de tracking de marcadores de código abierto para dispositivos móviles.

Por otra parte el tracking basado en características naturales resulta más atractivo dado que no se necesita la invasión de marcadores. Entre los algoritmos de tracking basado en características naturales, donde se tiene conocimiento de la escena, se encuentran la implementación de un algoritmo de tracking basado en SIFT presentada por Wagner [31], el cual permite el seguimiento de un objeto plano texturado con una imagen cualquiera.

3.5 Códigos QR

Un código de barras QR (Quick Response Barcode) es un sistema para almacenar información en una matriz de puntos o un código de barras bidimensional creado por la compañía japonesa Denso-Wave en 1994; se caracterizan por los tres cuadrados que se encuentran en las esquinas y que permiten detectar la posición del código al lector. La sigla "QR" se derivó de la frase inglesa "Quick Response" pues el creador aspiraba a que el código permitiera que su contenido se leyera a alta velocidad. Los códigos QR son muy comunes en Japón y de hecho son el código bidimensional más popular en ese país. Aunque inicialmente se usó para registrar repuestos en el área de la fabricación de vehículos, hoy, los códigos QR se usan para administración de inventarios en una gran variedad de industrias. Recientemente, la inclusión de software que lee códigos QR en teléfonos móviles, ha permitido nuevos usos orientados al consumidor, que se manifiestan en comodidades como el dejar de tener que introducir datos de forma manual en los teléfonos. Las direcciones y los URLs se están volviendo cada vez más comunes en revistas y anuncios. El agregado de códigos QR en tarjetas de presentación también se está haciendo común, simplificando en gran medida la tarea de introducir detalles individuales de un nuevo cliente en la agenda de un teléfono móvil.

Si bien los códigos de barras convencionales son capaces de almacenar un máximo de 20 dígitos, el código QR es capaz de manejar cientos de veces más información. Un código QR es capaz de manejar todo tipo de datos, tales como caracteres numéricos y alfabéticos, Kanji, Kana Hiragana [32], símbolos, binarios y códigos de control. Hasta 7.089 caracteres se pueden codificar en un solo símbolo.

Un código QR lleva la información tanto horizontales como verticales, es capaz de codificar la misma cantidad de datos en aproximadamente una décima parte del espacio de un código de barras tradicionales. (Para un tamaño de impresión más pequeños, existe Micro QR Code.) [33]

QR Code tiene la capacidad de corrección de errores. Los datos pueden ser restaurados, incluso si el símbolo es parcialmente sucio o dañado. Un máximo de 30% de las claves se puede restaurar. Ejemplo de dos claves que pueden ser restauradas:



Figura 3.9: Ejemplo de códigos deteriorados

El código QR es capaz de ser leído en 360 grados (omni-direccional), es de lectura de alta velocidad. Un código QR lleva a cabo esta tarea a través de patrones de detección de posición situado en las tres esquinas del símbolo. Estos patrones de detección de posición dan garantía de lectura estable de alta velocidad, eludiendo los efectos negativos de las interferencias de fondo. Esto hace al código QR un gran candidato para la identificación de objetos del mundo real dado su capacidad de lectura y supervivencia al deterioro que el exterior le pueda provocar.



Figura 3.10: Estructura de un código QR

Al ser un estándar abierto hay muchos generadores de QR Code gratuitos y en software libre. Pero aunque la creación de estos códigos es sencilla y sin costes, es en el sistema de lectura donde se encuentra una de las principales ventajas. Existen terminales específicos pero no son necesarios para poder descifrar un código QR pues basta con un móvil con cámara de fotos para decodificarlos. Ni siquiera es preciso que el terminal sea de última generación para que funcione y la mayor parte de los móviles actuales de los principales fabricantes permiten ya esta opción. En España los móviles aún no suelen llevar integrado de serie los lectores de QR, como ya ocurre en otros países, pero basta con añadir un software gratuito, que se puede descargar directamente desde Internet, para poder leer estos códigos. Leer un código QR es tan sencillo como enfocarlos con la cámara del móvil. Con gran rapidez se decodifica la información que aparece escrita en nuestro terminal. El sistema sorprende por la facilidad y velocidad de lectura, convirtiéndolo así en una opción versátil y fácil de usar. El QR Code presenta además numerosas ventajas:

- Este código tiene un patrón de localización, aunque se encuentren “mal” colocados los sistemas de lectura detectan la orientación, lo que permite que puedan ser leídos en cualquier posición

(360°). Al no ser la ubicación del código esencial para su correcta lectura, como sí ocurre con los códigos de barras, se reduce el margen de errores y se simplifica el trabajo.



Figura 3.11: Patrón de localización de un código QR

- Los QR code poseen una alta capacidad para restaurar información y cuatro niveles diferentes de corrección de errores que el usuario puede elegir en el momento de generarlo. Hasta un 30% de los datos pueden ser recuperables si parte del código se ha alterado o se ha perdido.
- Soportan cualquier tipo de lenguaje y diversos tipos de caracteres.
- Sólo precisan una décima parte del espacio que un código de barras requeriría. El código QR se encuentra además estandarizado, y fue aprobado como estándar ISO internacional (ISO/IEC18004) en el año 2000.

3.6 Códigos QR y Realidad Aumentada

El uso de códigos QR para aplicaciones de realidad aumentada ha sido motivo de disensos en el pasado, pero desde hace un tiempo el concepto se ha ampliado y ahora podemos decir que la realidad aumentada significa aumentar la realidad y eso significa darle un valor agregado a la realidad que a simple vista no podemos percibir. Además, la realidad aumentada permite personalización y la medición de resultados; y le permite al receptor de esa información interactuar con el contenido que se está ofreciendo. Es una herramienta interactiva que por medio del reconocimiento de patrones nos permite agregar información digital en un entorno real. Así como se han visto ejemplos de tablas periódicas digitales donde cada elemento está representado por un código QR u otro ejemplo donde en un libro se pegan códigos QR que representan enlaces a comentarios hechos por la gente sobre este libro, estos son ejemplos claros de esta interacción con el receptor o usuario. Por lo tanto podemos justificar de manera fehaciente, como esta interacción produce una retroalimentación o un intercambio de información que conlleva a aumentar aún más realidad.

3.7 Realidad aumentada en los museos

Las posibilidades y las aplicaciones que se le puede dar a los códigos QR en los museos son variadas. La facilidad de su uso, los avances tecnológicos y la difusión de la telefonía móvil y el bajo coste que supone su utilización, hace de este sistema una herramienta de utilidad para aportar información e interactuar con los usuarios de un museo.

Su reducido tamaño facilita la integración en la museografía o en las publicaciones del museo sin distorsionar la imagen general y aportando un elemento de valor.

Sus aplicaciones son variadas, tanto de cara al visitante como para la organización interna del trabajo.

3.7.1 Código QR en áreas de exposición

A la hora de programar un área expositiva no siempre resulta sencillo encontrar un equilibrio entre ofrecer demasiada información, que puede abrumar a algunos visitantes, o aportar poca, que puede resultar escasa a un público más especializado o interesado. Ante ello, numerosos museos han optado por opciones intermedias, mostrando una información básica a través de cartelas y ampliando la misma mediante sistemas diversos.

El código QR puede actuar como uno de estos elementos para que los usuarios interesados obtengan más datos sobre aquellas piezas que sean de su interés. Así, un código de este tipo puede aportar información en formato texto de manera directa, pero también enlazar a una página web o a un archivo de audio o vídeo. Hay que tener en cuenta que cada vez son más los terminales móviles con conexión a internet bien a través de 3G o a través de Wi-Fi, que puede ser proporcionado por el propio museo. Los códigos QR pueden albergar una URL que redirija al usuario a la web del museo, a la página con información del objeto o una galería virtual con información extra. Es una forma de facilitar el acceso al visitante a una mayor cantidad de información y de darle a conocer el espacio en Internet donde podrá obtener más datos de la colección. Se incentiva así la interacción entre el usuario y los objetos de la exposición, motivando a los visitantes a la participación activa para descubrir una información que además puede almacenar en su propio terminal y guardar como referencia.

Otras utilidades en zonas expositivas:

- Descarga directa al teléfono de aplicaciones que puedan ser de utilidad a los usuarios en su visita al museo: mapas de orientación, realidad aumentada, itinerarios, información complementaria, etc.
- Juegos de pistas en el museo a través de códigos QR, una iniciativa útil para motivar a los nativos digitales.
- Geolocalización del lugar donde la pieza fue hallada, creada, etc.
- Ayuda a las personas con visibilidad reducida: los códigos QR pueden conducir a archivos en mp3 con la información en formato audio. La combinación de códigos QR con braille o con móviles equipados con sistema de voz está dando buenos resultados en el campo de la accesibilidad.

3.7.2 Códigos QR en áreas de reserva

Los códigos QR se convierten en un aliado del personal técnico del museo a la hora de gestionar el almacenaje de las piezas. De hecho, una de las primeras y más extendidas utilidades de estos códigos ha sido su uso en procesos de inventario. Hasta ahora venía siendo habitual la utilización de códigos de barras para la organización de las áreas de reserva de los museos, pero el código QR aporta nuevos elementos de valor que hacen más recomendable su uso. En primer lugar permiten incluir mayor cantidad de información, con lo que más allá de una simple referencia topográfica los códigos QR pueden contener además los principales datos y características del objeto. Los requerimientos especiales que puede tener una pieza a la hora de su almacenamiento o transporte también pueden estar también incluidos en su QR Code, con lo que en el mismo almacén y solamente con un teléfono móvil se pueden conocer las precauciones a tener en cuenta antes de movilizar un bien.

El código QR de las piezas puede además vincular con una URL con información del objeto, facilitando el trabajo en el área de reserva y el acceso inmediato a todos los datos de la colección. El menor índice de errores es sin duda otro elemento de peso a la hora de decantarse por la utilización de este sistema, así como por la facilidad de su lectura, ya que pueden ser decodificados por el personal del museo con sus propios terminales telefónicos.

En almacenes visitables, donde no existen carteles con información para el público, el uso de códigos QR puede aportar referencias informativas a los visitantes de manera poco intrusiva y sin necesidad de crear una museografía expositiva en un espacio de trabajo.

3.7.3 Códigos QR en difusión

La utilización de códigos QR en cartelería, trípticos, folletos publicitarios, publicaciones de la institución, anuncios de prensa o de televisión, etc., añade un elemento multimedia a los productos creados por el museo y permite ampliar de forma sencilla la información que estos aportan. Por ejemplo, la vinculación a sitios web específicos a los que sólo se entre a través de una URL inscrita en el código da valor al anuncio, folleto o publicación e incentiva el interés del usuario por estos medios.

3.7.4 Códigos QR en la página web del museo

La utilización de códigos QR en la web del museo facilita descargar directamente en el móvil, datos útiles para el público durante la visita. Horarios o servicios que presta el centro, itinerarios por la exposición, obras destacadas, aplicaciones, etc. Son numerosas las posibilidades existentes y el hecho de que el usuario sólo deba escanear el código para tener toda la información en su terminal, hace más sencillo el proceso de descarga y de posterior utilización en su visita a la institución.

3.7.5 Códigos QR en la Biblioteca del Museo

Al igual que en el almacén, la utilización de los códigos QR en la biblioteca del museo aumenta las posibilidades frente al uso de códigos de barras por su capacidad para albergar mayor cantidad de información y más variada. Se pueden utilizar:

- Localización topográfica de libros.
- Referencia bibliográfica completa de la obra.
- Rapidez en las búsquedas de libros, al no tener que anotar referencias que pueden guardarse en el móvil.
- Enlaces a web con más información sobre la obra, etc.

3.7.6 Códigos QR en eventos

Conferencias, seminarios, presentaciones que se realicen en el museo y todo tipo de eventos en los que los asistentes vayan identificados, son otras posibilidades para el uso de los códigos QR. Además de los datos habituales que suelen aparecer en las tarjetas identificativas de los asistentes, se puede usar un código QR como complemento. Si lo desean, los asistentes podrían además compartir sus datos de contacto a través del código QR de su identificación y almacenarlos en la agenda del móvil. El museo también puede usar el código QR para el control de accesos.

3.8 Recomendaciones de uso de códigos QR

A la hora de utilizar los códigos QR hay que tener en cuenta una serie de precauciones para facilitar su lectura:

- Los QR precisan de un mínimo de calidad para poder ser leídos adecuadamente. Los códigos extremadamente pequeños (menores de 1x1 cm. de lado) o con una calidad de impresión muy baja no darán buenos resultados, sobre todo si contienen gran cantidad de información.
- Todos los códigos precisan de una pequeña área circundante, en blanco o sin ruido, de al menos 0,5 cm., detalle que tendrá que ser tenido en cuenta si en el diseño aparece junto a otros elementos.
- El tamaño del QR está en función de la cantidad de datos que lo integren, a mayor información contenida menor posibilidad de reducción en el tamaño. Si, por necesidades de diseño, se requiere que todos los QR tengan unas dimensiones uniformes es recomendable que la densidad de datos que contenga no sea muy desproporcionada de unos respecto a otros. Usar generadores que permitan codificar en tamaños establecidos y tengan limitación de caracteres facilita la estandarización en las medidas de los códigos.
- La capacidad del QR para reparar errores condiciona el tamaño del mismo, un QR con una capacidad de corrección del 7% se podrá usar a menor tamaño que otro con una capacidad de reparación de errores del 30%. La institución tendrá que primar una u otra posibilidad en función de sus necesidades y de la utilidad que le vaya a dar.
- Los brillos dificultan la lectura, con lo que hay que tener la precaución de no imprimirlos en formatos excesivamente satinados. Si se colocan bajo vidrio u otra superficie reflectante habrá que verificar que la reflexión especular de la iluminación no impida la decodificación.
- La ubicación en sitios con sombras muy duras puede dificultar igualmente la lectura.
- En zonas de iluminación tenue los lectores móviles presentan mayores dificultades para leer los códigos de pequeño tamaño (menores de 2x2 cm. de lado, aproximadamente). En espacios con muy baja iluminación se recomienda usar códigos de tamaño medio.

Además de todo esto, si el contenido del código es una URL es recomendable que la web a la que enlaza sea accesible y esté adaptada para poder ser visualizada correctamente con navegadores móviles.

3.9 Conclusión

Los códigos QR son hoy por hoy un camino para la experimentación por parte de los museos. La facilidad de su uso y el bajo coste que supone su implantación hacen que sea recomendable explorar todas las posibilidades que ofrecen y evaluar su utilidad, tanto en la gestión interna de la institución como en la respuesta del público a la hora de interactuar con el museo a través de estos medios.

La popularización que está adquiriendo este tipo de tecnología y el hecho de que la mayor parte de los visitantes del museo lleven un teléfono móvil en su bolsillo, hacen del código QR una apuesta al futuro.

Al ser una herramienta no muy popularizada en nuestro país y cuyo uso se limita a los sectores más relacionados con las nuevas tecnologías, el museo ha de asumir un papel didáctico a la hora de presentar a sus visitantes esta posibilidad, facilitando el conocimiento del sistema y los medios necesarios para que pueda usarlo.

Capítulo 4. Android

En el capítulo anterior hemos descrito el concepto de realidad aumentada. En el presente capítulo detallaremos la plataforma Android. Se hará una reseña de su historia y orígenes para luego explicar sus principales características y diferentes versiones.

4.1 Introducción

Android es una plataforma completa de código abierto diseñado para dispositivos móviles. Es promovido por Google y propiedad de la Open Handset Alliance [34]. El objetivo de la alianza es "acelerar la innovación en los consumidores móviles y ofrecer una experiencia móvil más rica, más barata y mejor." Android es el vehículo para hacerlo.

Como tal, Android está revolucionando el espacio móvil. Por primera vez, es una verdadera plataforma abierta que separa el hardware del software que se ejecuta en él. Esto permite un número mucho más grande de dispositivos para ejecutar las mismas aplicaciones y crea un ecosistema mucho más rico para desarrolladores y consumidores. [35]

Android es una pila de software para dispositivos móviles que incluye un sistema operativo, middleware y aplicaciones. El SDK de Android proporciona las herramientas y APIs necesarias para empezar a desarrollar aplicaciones en la plataforma Android usando el lenguaje de programación Java. Estas aplicaciones son ejecutadas en una máquina virtual diseñada para esta plataforma, que ha sido bautizada con el nombre de Dalvik. Esta está optimizada para requerir poca memoria y diseñada para permitir ejecutar varias instancias de la máquina virtual simultáneamente, delegando en el sistema operativo subyacente el soporte de aislamiento de procesos, gestión de memoria e hilos.

Android se distribuye bajo una Licencia Apache [17], versión 2, es una licencia de software libre creada por la Apache Software Foundation. La Licencia Apache permite al usuario del software la libertad de usarlo para cualquier propósito, distribuirlo, modificarlo y distribuir versiones modificadas de ese software. La Licencia Apache requiere la conservación del aviso de copyright y el disclaimer, pero no es una licencia copyleft, ya que no requiere la redistribución del código fuente cuando se distribuyen versiones modificadas.

4.2 Historia

Android fue desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005. El 5 de noviembre de 2007 se fundó la Open Handset Alliance (OHA) con el fin de desarrollar estándares abiertos para dispositivos móviles. Fue liderada por Google con otros 34 miembros entre los que se incluyen fabricantes de dispositivos móviles, desarrolladores de aplicaciones, algunos operadores de comunicaciones y fabricantes de chips. El mismo día la OHA dio a conocer Android, una plataforma de código abierto para móviles basada en el sistema operativo Linux.

Una versión beta del SDK fue lanzada para desarrolladores el 12 de noviembre de 2007. El primer teléfono comercialmente disponible con Android fue el HTC Dream (T-Mobile G1), el cual utilizó Android 1.0.

En el año 2009 se produce una proliferación de dispositivos basados en Android. Se liberaron las nuevas versiones del sistema operativo: Cupcake (1,5), Donut (1,6), y Eclair (2,0 y 2,1). Son más de 20 dispositivos que ejecutan Android.

En 2010, Android es la segunda plataforma de teléfonos inteligentes, después de Blackberry, más vendida. Froyo (Android 2.2) se libera y son más de 60 dispositivos que lo ejecutan. [35]




En el año 2011 fue anunciado y liberado Android 4.0 (Ice Cream Sandwich) y finalmente en octubre de este año se liberó Android 4.1 (Jelly Bean).

4.3 Versiones de Android

El siguiente cuadro presenta las diferentes versiones de Android hasta la fecha.

Versión	Nivel de API	Características
1.0	1	<p>La primera versión utilizada por el T-Mobile lanzado en Estados Unidos en octubre de 2008 contenía las siguientes características:</p> <ul style="list-style-type: none"> • Ventana de notificación desplegable. Fue el primer sistema operativo que implementó una plataforma de notificaciones por medio de mensajes y alertas. • Menú especial para agrupar las aplicaciones y programas del teléfono. • Integración de Gmail, aunque con funciones muy limitadas. • Se introdujo la tienda Android Market.
1.1	2	<p>Esta primera actualización llegó en febrero de 2009 y si bien no fue un cambio revolucionario, se solucionaron varios errores. Lo más notable fue la mejora y sencillez para encontrar y ejecutar cualquier actualización que el sistema operativo del teléfono necesitara.</p>
1.5 Cupcake	3	<p>Fue liberada en Abril de 2009 y se introdujeron las siguientes características:</p> <ul style="list-style-type: none"> • Grabación de video • Soporte para Bluetooth • Teclado virtual tipo QWERTY • Soporte para Widgets • Mejoras en el portapapeles para realizar acciones como "copiar y pegar" texto • Grabación de vídeo y reproducción



<p>1.6 Donut</p> 	<p>4</p>	<p>Fue liberada en septiembre de 2009 y no se caracterizó por mostrar grandes cambios visuales, pero sí por sus nuevas y útiles mejoras:</p> <ul style="list-style-type: none"> ● Soporte para múltiples pantallas ● Quick Search Box, una caja de búsqueda en la pantalla de inicio que permite buscar entre distintas fuentes (los contactos, el historial del navegador, Google, etc.). Con autocompletado y capacidad de aprendizaje. ● Mejorada la velocidad de la cámara. ● Posibilidad de conectarse a redes VPN, 802.1x. ● Nueva pantalla para controlar la batería, que permite comprobar qué aplicaciones y servicios son los que más consumen. Desde esta pantalla se puede también parar o desinstalar estas aplicaciones ● Las aplicaciones de Android Market aparecen ahora ordenadas por categorías (Aplicaciones, Juegos y Descargas). ● Nuevo motor de texto a voz.
<p>2.0, 2.0.1, 2.1 Eclair</p> 	<p>5, 6, 7</p>	<p>En octubre de 2009 se libera la versión 2.0, en diciembre de ese mismo año la versión 2.0.1 y en enero de 2010 la versión 2.1.</p> <ul style="list-style-type: none"> ● Rediseño de la interfaz del navegador, contando ahora con soporte para distintas características de HTML5. ● Soporte nativo de flash para la cámara, zoom digital, modo escena, balance de blanco, efectos de color y modo macro. ● Mejoras en el teclado virtual. ● Soporte para nuevos tamaños y resoluciones de pantalla. ● Contactos rápidos. ● Bluetooth 2.1 ● Mejoras en Google Maps, que pasaba a ser multitáctil y soportar capas. ● Soporte de Microsoft Exchange. ● Mejoras en el calendario. ● Reconocimiento de voz ● Mejoras en la duración de la batería.
<p>2.2 Froyo</p> 	<p>8</p>	<p>Disponible desde finales de Junio del 2010, Froyo introdujo las siguientes novedades:</p> <ul style="list-style-type: none"> ● Actualizaciones automáticas para aplicaciones. ● Soporte WiFi IEEE 802.11n ● Soporte para Radio FM. ● Soporte Flash 10.1 y Adobe AIR 2.5 ● Soporte de la API gráfica OpenGL 2.0 ● Posibilidad de asignar un color de LED en el TrackBall para diferentes eventos del terminal. ● Creación de un compilador JIT que mejora entre 2 y 5 veces en Rendimiento frente a Eclair. ● Tethering por USB y hotspot WiFi ● Incorporación del mismo motor de Javascript V8 de Chrome.





<p>2.3, 2.3.3 Gingerbread</p> 	<p>9, 10</p>	<p>La versión 2.3 fue liberada en diciembre 2010 y la versión 2.3.3 en febrero de 2011.</p> <ul style="list-style-type: none"> • Mejor control en copiar y pegar • Teclado multi-táctil rediseñado • Soporte nativo para múltiples cámaras • Soporte nativo para sensores como giroscopios y barómetros • Soporte nativo para telefonía VoIP
<p>3.0, 3.1, 3.2 Honeycomb</p> 	<p>11, 12, 13</p>	<p>Fue liberada en mayo de 2011 y fue optimizada especialmente para tablets y dispositivos con tamaños grandes de pantalla.</p> <ul style="list-style-type: none"> • Se rediseño la pantalla de inicio y colocación de widgets • Multitarea mejorada • Un nuevo paradigma para el diseño de aplicación • No hay necesidad de botones dedicados y físicos para el regreso, home, menú y buscar.
<p>4.0.3, 4.0.4 Ice Cream Sandwich</p> 	<p>15</p>	<p>Fue liberada a finales de 2011 y fue diseñado para todas las plataformas (smartphones, tablets).</p> <ul style="list-style-type: none"> • Nuevo tipo de letra llamada Roboto • Pantalla principal con imágenes 3D • Barras de estado • Widgets redimensionables • Desbloqueo con reconocimiento facial • Mejora de reconocimiento de voz • Se busca potenciar el uso de NFC con una nueva característica para transferencia de datos entre dos teléfonos con solo tocarlos. • Nuevo calendario y aplicaciones de correo
<p>4.1 Jelly Bean</p> 	<p>16</p>	<p>Fue liberada en octubre del 2012.</p> <ul style="list-style-type: none"> • Se añade un nuevo sistema de notificaciones expandibles. • Se incorpora un nuevo sistema de gestión de datos entre la CPU y la GPU llamado Project Butter haciendo las transiciones entre aplicaciones o entre ventanas mucho más rápidas, suaves y optimizadas. • Los iconos de los escritorios se desplazan automáticamente para dejar sitio a nuevas incorporaciones. • Mejor gestor de la aplicación de la cámara de fotos • Teclado más rápido y con mejor predicción • Google Now, el asistente de voz nativo de Android

Tabla 4.1: Versiones de Android

4.4 Distribución de las versiones

Como desarrolladores de aplicaciones tendremos que asegurarnos de que el nivel de API elegido para la aplicación permita que ésta se ejecute en tantos dispositivos como sea posible. El nivel de API determinará qué dispositivos puede y no puede ejecutar la aplicación.

Por lo tanto, hay que tratar de elegir un nivel de API que sea lo más bajo posible y además, tener en cuenta la distribución de las versiones de Android en los dispositivos reales que hay hasta el momento.

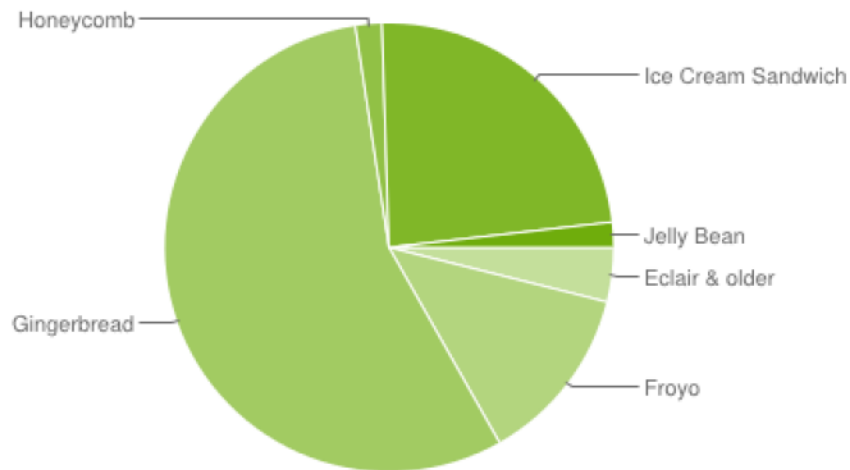


Figura 4.1: Distribuciones de versiones 2012 [36]

Versión	Nombre	Distribución
1.5	Cupcake	0.1%
1.6	Donut	0.4%
2.1	Eclair	3.4%
2.2	Froyo	12.9%
2.3 - 2.3.2	Gingerbread	0.3%
2.3.3 - 2.3.7		55.5%
3.1	Honeycomb	0.4%
3.2		1.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	23.7%
4.1	Jelly Bean	1.8%

Tabla 4.2: Porcentaje de distribución de cada versión Android

Se puede notar que no hay una gran cantidad de usuarios de Android 1.5 y 1.6. También se puede notar que no es una gran cantidad de usuarios que tienen la última versión de Android 4.1, pero el número de usuarios con la versión 2.x son la mayoría.

Analizando estos porcentajes se podría concluir que las versiones que se podrían elegir para que la aplicación llegue a la mayor cantidad de usuarios, sería de la 1.6 a 2.2.

4.5 Características

Las principales características de la plataforma son:

- **Application framework** que permite la reutilización y sustitución de componentes
- **Dalvik virtual machine** optimizada para dispositivos móviles
- **Navegador Integrado** basado en el motor de código abierto WebKit
- **Gráficos optimizados** biblioteca de gráficos 2D y biblioteca de gráficos 3D basada en las especificaciones de la OpenGL ES 1.0
- **SQLite** para el almacenamiento de datos
- **Soporte multimedia** para audio, videos y formatos de imagen (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **Conectividad** GSM, Bluetooth, EDGE, 3G y WiFi
- **Soporte para hardware adicional** cámara de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, sensores de proximidad y de presión, termómetro, aceleración 2D y 3D.
- **Entorno de desarrollo** incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software. El entorno de desarrollo integrado es Eclipse usando el plugin de Herramientas de Desarrollo de Android (ADT Plugin).

4.6 Arquitectura

El sistema operativo Android está compuesto por varias capas. Cada capa tiene sus propias características y propósitos. En la siguiente figura se pueden ver cada una de las capas [35]:

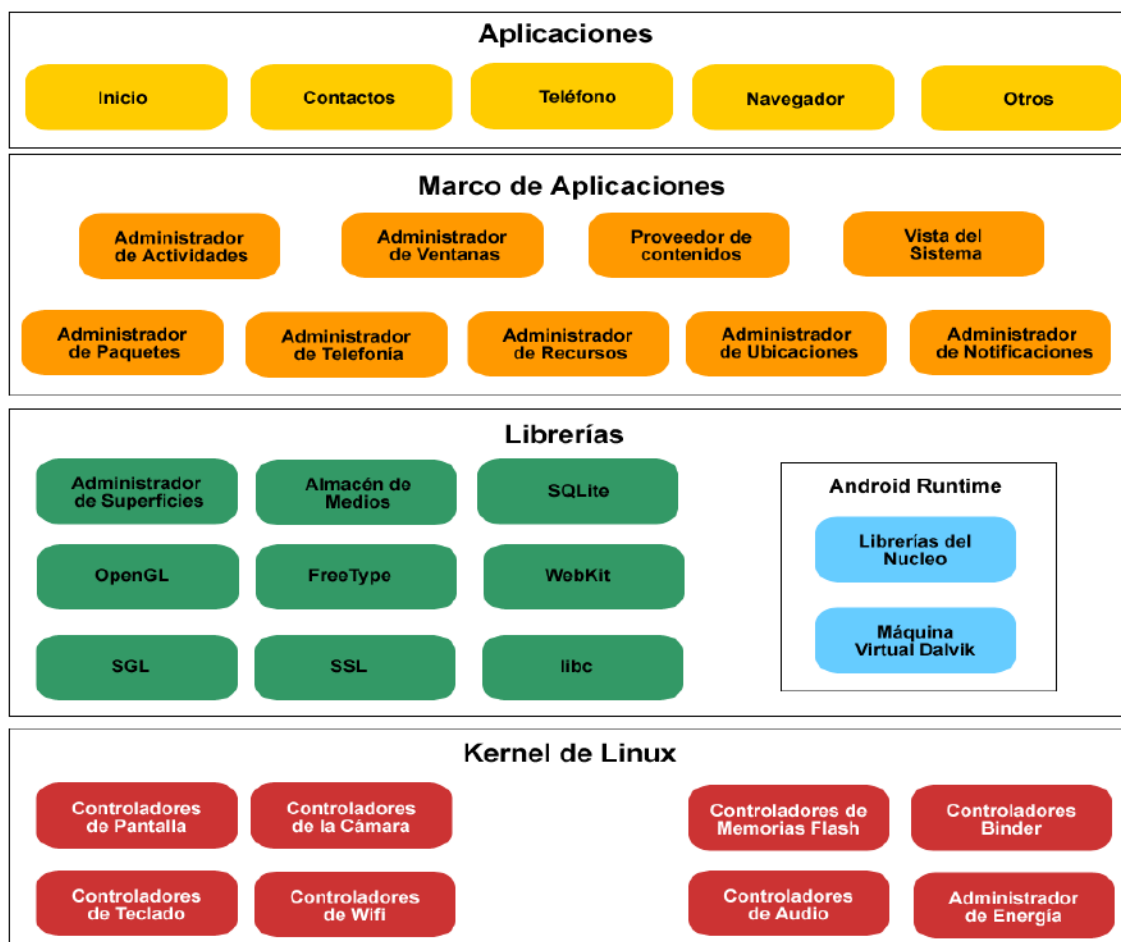


Figura 4.2: Arquitectura del Sistema de Android

Kernel de Linux

Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software. Todas las aplicaciones de Android se ejecutan como procesos separados de Linux con permisos otorgados por el sistema Linux.

Librerías

Android incluye un conjunto de librerías de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android. Algunas de ellas son:

- Webkit: Un rápido motor de renderizado web usado por Safari, Chrome y otros navegadores
- SQLite: Funciones de base de datos SQL
- Apache Harmony: Una implementación de código abierto de Java
- OpenGL: Bibliotecas de gráficos 3D
- OpenSSL: La capa de seguridad

Android Runtime

Android incluye un conjunto de librerías base que proporcionan la mayor parte de las funciones disponibles en las librerías base del lenguaje Java.

Dalvik es una máquina virtual especialmente diseñada para Android. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato .dex por la herramienta incluida "dx". A continuación se muestra una figura donde se compara el proceso de compilación entre Java estándar y Android.

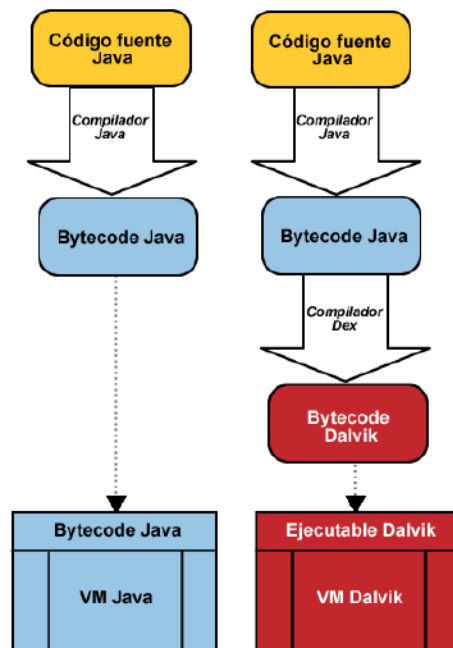


Figura 4.3: Proceso de compilación de Java y Android

Marco de aplicaciones

El marco de aplicación es un ambiente rico que ofrece numerosos servicios que ayudan al desarrollador de la aplicación a hacer su trabajo. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario. Entre los componentes tenemos:

- Proveedores de contenido que permiten a las aplicaciones acceder a datos de otras aplicaciones (como los contactos), o para compartir sus propios datos.
- Sistema de vistas: proporciona un conjunto de elementos para construir interfaces de usuario
- Un administrador de recursos para facilitar el acceso a los recursos como string, gráficos, layout.
- Un Administrador de notificaciones que permite a todas las aplicaciones mostrar alertas personalizadas en la barra de estado.
- Un gestor de actividad que gestiona el ciclo de vida de las aplicaciones y proporciona una navegación común.

Aplicaciones

Por último, están las aplicaciones donde se encuentran las preinstaladas tales como un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y las aplicaciones que los desarrolladores distribuyen y que se puede descargar desde los distintos mercados de Android.

4.7 Fundamentos de la aplicación

Las aplicaciones de Android están escritas en el lenguaje de programación Java. Las herramientas del SDK de Android compilan el código, junto con los datos y archivos de recursos en un paquete de Android, un archivo con sufijo .apk. Todo el código en un único archivo .apk es considerada como una aplicación y es el archivo que Android utiliza para instalar la aplicación. Un archivo .apk tiene básicamente tres componentes principales:

- **Dalvik ejecutable:** Es el código fuente de Java compilado a un ejecutable Dalvik. Éste es el código que ejecuta la aplicación.
- **Recursos:** Los recursos son todos los que no es código. La aplicación puede contener un conjunto de imágenes y archivos de audio/vídeo, así como numerosos archivos XML que describen el diseño, paquetes de idiomas, etc.
- **Librerías nativas:** Opcionalmente, la aplicación puede incluir algún código nativo, como librerías C/C++.

El sistema operativo Android es un sistema multi-usuario de Linux en el que cada aplicación es un usuario diferente. Por defecto, el sistema asigna a cada aplicación un único ID de usuario de Linux (el ID es utilizado sólo por el sistema y desconocido por la aplicación). El sistema establece permisos para todos los archivos en una aplicación para que sólo el ID de usuario asignado a esa aplicación puede acceder a ellos.

Cada proceso tiene su propia máquina virtual (VM), por lo que el código de una aplicación se ejecuta en forma aislada de otras aplicaciones.

Por defecto, cada aplicación se ejecuta en su propio proceso de Linux. Android inicia el proceso cuando alguno de los componentes de la aplicación debe ser ejecutado, luego se cierra el proceso cuando ya no se necesita o cuando el sistema debe recuperar la memoria para otras aplicaciones.

De esta manera, el sistema Android aplica el *principio de mínimo privilegio*. Es decir, cada aplicación, por defecto, sólo tiene acceso a los componentes que necesita para hacer su trabajo y nada más. Esto crea un ambiente muy seguro en el que una aplicación no puede acceder a partes del sistema para el cual no se le da permiso.

Sin embargo, hay maneras para que una aplicación pueda compartir datos con otras aplicaciones: Es posible disponer de dos aplicaciones que comparten el mismo ID de usuario de Linux, en cuyo caso se podrá acceder a los demás archivos. Para ahorrar recursos del sistema, las aplicaciones con el mismo ID de usuario también puede hacer arreglos para ejecutar en el mismo proceso Linux y compartir la misma máquina virtual (Las aplicaciones deben estar firmadas con el mismo certificado).

Una aplicación puede solicitar permiso para acceder a los datos del dispositivo, tales como los contactos del usuario, los mensajes SMS, el almacenamiento (tarjeta SD), cámara, Bluetooth, entre otros. Todos los permisos de la aplicación debe ser concedida por el usuario durante la instalación. [37]

4.8 Componentes de Aplicación

Veremos los distintos tipos de componentes de software con los que podremos construir una aplicación Android [37]:

Actividad (Activity)

Las actividades representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana en cualquier otro lenguaje visual.

Una actividad es implementada como una subclase de Activity.

Servicio (Service)

Los servicios son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son exactamente iguales a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, mostrar notificaciones, o incluso mostrar elementos visuales (Activities) si se necesita en algún momento la interacción con del usuario.

Un servicio es implementado como una subclase de Service.

Proveedor de Contenido (Content Provider)

Un proveedor de contenidos es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de nuestra aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra, a través de los proveedores de contenido que se hayan definido.

Un proveedor de contenidos es implementado como una subclase de ContentProvider

Receptor de eventos (Broadcast Receiver)

Un receptor de eventos es un componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema (por ejemplo: "Batería baja", "SMS recibido", "Tarjeta SD insertada", etc) o por otras aplicaciones (cualquier aplicación puede generar mensajes (Intents, en terminología Android) broadcast, es decir, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo).

Un receptor de eventos es implementado como una subclase de BroadcastReceiver

4.8.1 Activación de componentes

Intents

Tres de los cuatro tipos de componentes: actividades, servicios y receptores de eventos se activan mediante un mensaje asincrónico llamado Intent.

Un intent es el elemento básico de comunicación entre los distintos componentes Android. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un intent se puede activar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc. [37]

Un *objeto intent* es un paquete de información. Contiene información de interés para el componente que recibe la intención (tal como la acción a realizar y los datos necesarios) más información de interés para el sistema Android (tales como la categoría del componente que debe manejar la intención y las instrucciones de cómo poner en marcha una actividad destino). El objeto intent se compone de tres elementos principales: acción, categoría y datos.

- **Acción:** es una cadena de texto que indica la acción a realizar. Por ejemplo, android.intent.action.VIEW.
- **Categoría:** Son meta datos referidos al Intent, por ejemplo, android.intent.category.LAUNCHER indica que el Intent puede iniciar una aplicación si así es requerido.
- **Datos:** los cuales están definidos en forma de objeto URI o texto plano. [38]

Los Intents se pueden dividir en dos grupos:

- **Intents explícitos:** se puede llamar a un componente directamente a través de su nombre.
- **Intents implícitos:** se pide al sistema evaluar los componentes registrados para una cierta intención. Los intents implícitos a menudo se utilizan para activar componentes en otras aplicaciones.

Hay una clase relacionada a los Intents llamada IntentFilter. Para informar al sistema que intenciones implícitas son capaces de manejar las actividades, servicios y receptores de eventos, pueden declarar uno o más Intents Filters (Filtros). Cada filtro describe una capacidad del componente, un conjunto de intenciones que el componente está dispuesto a recibir.

Content Resolver

El proveedor de contenidos no se activa mediante un Intent, sino que debemos obtener una referencia a un *Content Resolver*, a través del cual se realizan todas las acciones necesarias sobre el proveedor de contenidos tales como consultar, actualizar, insertar o eliminar.

4.8.2 Procesos

Una aplicación Android se ejecuta en su propio proceso de Linux. Este proceso es creado para la aplicación se ejecuta y permanecerá hasta que la aplicación deje de trabajar o el sistema reclame recursos para otras aplicaciones.

Una característica fundamental de Android es que el ciclo de vida de una aplicación no está controlado por la misma aplicación sino que lo determina el sistema a partir de una combinación de estados, como pueden ser qué aplicaciones están funcionando, qué prioridad tienen para el usuario y cuánta memoria queda disponible en el sistema. De esta manera, Android coloca cada proceso en una jerarquía de importancia basada en los componentes en ejecución y en el estado de los mismos.

Procesos de menor importancia se eliminan primero, luego los siguientes en importancia, y así sucesivamente. Hay cinco niveles en la jerarquía. El orden de importancia es el siguiente [39]:

1 Procesos Activos (Foreground Process): Aquellos procesos necesarios para lo que el usuario está haciendo. Por lo general habrá muy pocos procesos de este tipo corriendo a la vez en el sistema y son aquellos que se eliminarán como última opción. Un proceso se considera en primer plano si cumple alguna de las siguientes condiciones:

- Se encuentra activa una actividad con la que el usuario está interactuando (el método `onResume()` ha sido llamado).
- Se encuentra activo un servicio que está destinado a la actividad con la que el usuario está interactuando o está ejecutando algunos de los métodos de su ciclo de vida (`OnCreate()`, `OnStart()`, o `OnDestroy()`).
- Se encuentra activo un `BroadcastReceiver` que está ejecutando su método `OnReceive()`.

2 Procesos Visibles: es aquel que contiene una Actividad que es visible al usuario mediante la pantalla pero no en primer plano (esta pausada). Este proceso solo se eliminará en caso de que sea necesario mantener ejecutándose los procesos en primer plano.

3 Procesos de servicio: es aquel que contiene un servicio que ha sido inicializado. No son directamente visibles al usuario, pero si realizan tareas percibidas por este. El sistema sigue en funcionamiento a menos que no haya suficiente memoria para retenerlos junto con todos los procesos de primer plano y visibles.

4 Procesos en segundo plano (Background Process): Un proceso que mantiene una actividad que no es actualmente visible para el usuario (el método de la actividad `OnStop()` ha sido llamado). Estos procesos no tienen un impacto directo en la experiencia del usuario, y el sistema puede matar en cualquier momento para recuperar memoria para los procesos de primer plano, visibles o procesos de servicio.

5 Procesos Vacíos: Es un proceso que no aloja ningún tipo de componente. Su razón de ser es el de tener una caché disponible para la próxima aplicación que lance el usuario.

4.8.3 Ciclo de vida de una actividad

La actividad es un componente fundamental de las aplicaciones. El ciclo de vida de una actividad comienza con su instanciación y termina con su destrucción e incluye muchos estados durante ese tiempo. Cuando un actividad cambia de estado, el método correspondiente al ciclo de vida se llama, notificando a la actividad correspondiente del cambio de estado inminente y permitiéndole ejecutar código con el fin de adaptarse a ese cambio.

En la imagen puede verse un esquema con los estados en los que puede estar una actividad y qué transiciones se pueden dar entre ellos. A continuación se detalla cada uno de ellos:

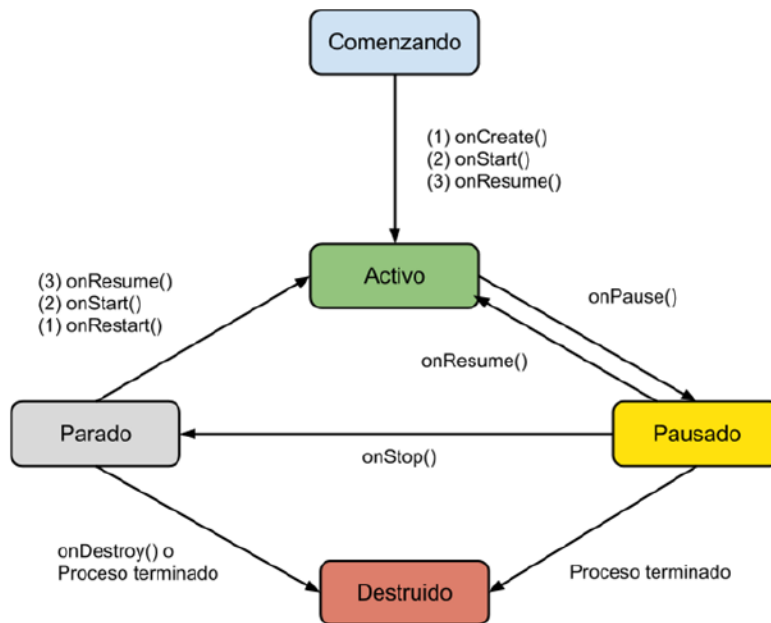


Figura 4.4: Estados de una Actividad

- **Activa (Running):** Es la primera en la pila de ejecución, el usuario ve la actividad y puede interactuar con ella.
- **Pausada (Paused):** Cuando el dispositivo va a entrar en reposo, o una actividad aún es visible, pero parcialmente oculta por una nueva, que no es de tamaño completo o transparente, la actividad se considera pausada. Las actividades en pausa todavía están vivas, es decir, mantienen toda la información del estado, y permanecen unidos al administrador de ventanas. En este caso, la actividad puede ser cerrada por el sistema si necesita liberar recursos para la nueva actividad.
- **Parada (Stopped):** Ha pasado a segundo plano y está completamente oculta por la nueva actividad, en ese caso el sistema también puede optar por cerrarla si necesita liberar recursos.
- **Destruída (Destroyed):** ya no está disponible, se han liberado todos sus recursos y en caso de ser llamada, necesitaría comenzar un nuevo ciclo de vida.

Métodos para la gestión del ciclo de vida

De la figura 4.5 se puede determinar lo siguiente:

- **onCreate(), onDestroy():** abarcan todo el ciclo de vida. Cada uno de estos métodos representan el principio y el fin de la actividad.
- **onStart(), onStop():** representan la parte visible del ciclo de vida. Desde onStart() hasta onStop(), la actividad será visible para el usuario, aunque es posible que no tenga el foco de acción por existir otras actividades superpuestas con las que el usuario está interactuando. Pueden ser llamados múltiples veces.
- **onResume(), onPause():** determinan la parte útil del ciclo de vida. Desde onResume() hasta onPause(), la actividad no sólo es visible, sino que además tiene el foco de la acción y el usuario puede interactuar con ella.

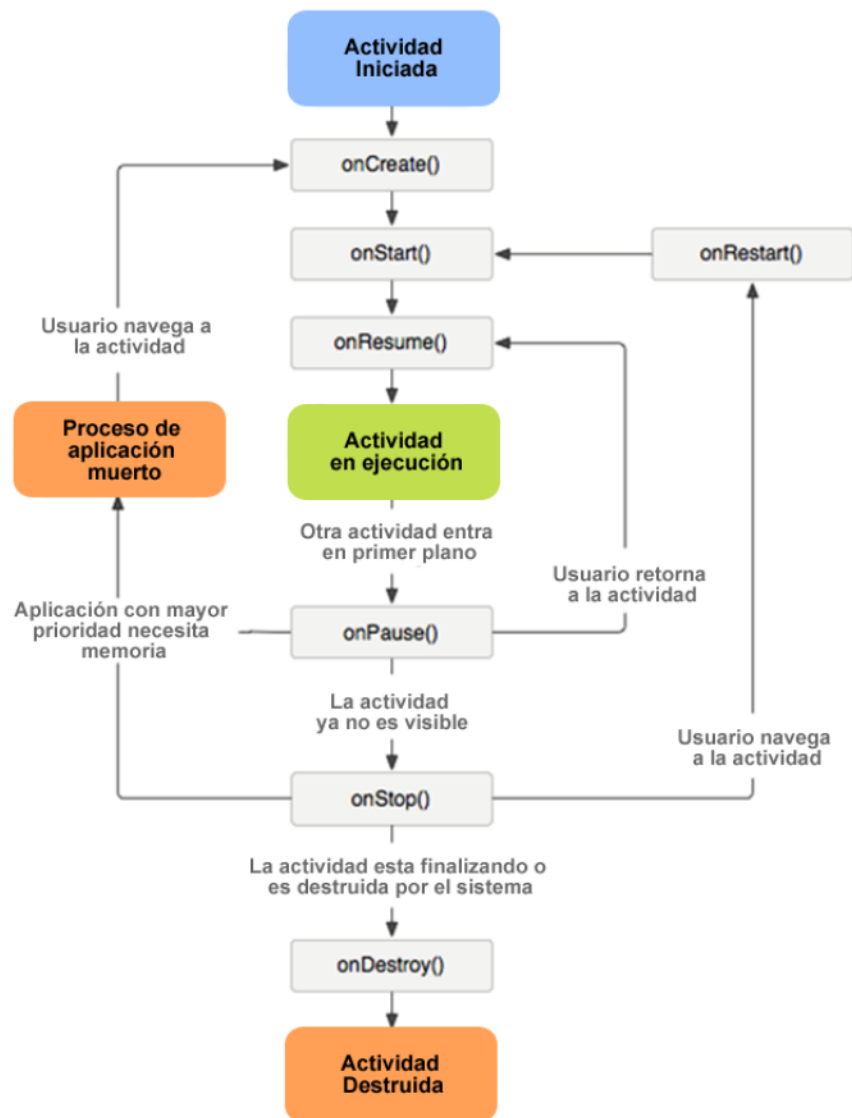


Figura 4.5: Ciclo de vida de una actividad

A continuación se detallan los métodos más relevantes del ciclo de vida:

- **onCreate():** Se llama al crear la actividad. Es donde se prepara la interfaz gráfica de la pantalla. Tras esta función, el proceso sobre el que se ejecuta la Actividad no puede ser destruido por el sistema. El siguiente método que se llama es onStart().
- **onRestart():** Se llama cuando una actividad que se había parado vuelve a estar activa, justo antes de que comience de nuevo. Tras ella se llama a onStart() y su proceso no puede ser destruido ni durante ni luego de su ejecución.
- **onStart():** Se ejecuta justo antes de que la aplicación sea visible al usuario. El siguiente método de ciclo de vida llamado será onStop() u onResume(), dependiendo de la situación.
- **onResume():** Se ejecuta en el momento en que la actividad se encuentra en la parte superior de la pila, justo antes de que el usuario pueda interactuar con ella. El siguiente método será onPause().
- **onPause():** Se llama cuando la actividad va a ser ocultada por otra. En este método debemos aprovechar para liberar todo aquello que consume recursos o guardar datos de manera

persistente. No podrá contener tareas lentas ya que hasta que no termine este método no podrá ejecutarse el `onResume()` de la nueva actividad. El método siguiente será `onResume()` u `onStop()`.

- **onStop():** Se ejecuta cuando la actividad se hace invisible al usuario. Puede ser porque otra actividad la oculte y entonces el siguiente método será `onRestart()` o porque la actividad haya sido destruida, llamado a continuación a `onDestroy()`.
- **onDestroy():** Se llama antes de destruir la actividad. Durante la destrucción de la actividad se perderán todos los datos asociados a ella por lo que en este método podrá ser utilizado para controlar la persistencia de datos. Se llamará cuando se ejecuta el método de finalización `finish()` sobre la actividad o porque el sistema elimina la actividad para conseguir más recursos.

4.9 Entorno de desarrollo

Android tiene un completo SDK (depurador, APIs, bibliotecas, un emulador, documentación, código de ejemplo y tutoriales) para facilitar el desarrollo de nuevas aplicaciones. Actualmente, la plataforma de desarrollo está disponible para Linux, Mac y Windows.

Con Android se busca reunir en una misma plataforma todos los elementos necesarios que permitan al desarrollador controlar y aprovechar al máximo cualquier funcionalidad ofrecida por un dispositivo móvil (llamadas, mensajes de texto, cámara, agenda de contactos, conexión Wi-Fi, Bluetooth, videojuegos, etc.), así como poder crear aplicaciones que sean verdaderamente portables, reutilizables y de rápido desarrollo. En otras palabras, Android quiere mejorar y estandarizar el desarrollo de aplicaciones para cualquier dispositivo móvil y, por ende, acabar con la fragmentación existente hoy en día.

Android ofrece un plugin para Eclipse que extiende la funcionalidad de éste y facilita el desarrollo de aplicaciones para Android. Además, ofrece las herramientas que utiliza este plugin como scripts de Ant para que puedan ser utilizados también desde otros entornos de desarrollo. Entre las funcionalidades de este plugin se encuentra:

- Emulador de Android. Permite elegir entre distintos terminales móviles y la versión del sistema operativo.
- El acceso a herramientas de desarrollo de Android como tomar capturas de pantalla, la redirección de puertos, la posibilidad de depurar con puntos de parada o ver el estado de los procesos corriendo en el sistema).
- Asistentes para la creación rápida de aplicaciones Android.
- Editores de código para los distintos archivos de configuración (XML) que facilitan su comprensión y desarrollo.
- Interfaces gráficas que permiten el desarrollo de componentes visualmente.

4.9.1 Instalación del SDK

A continuación se describen los pasos necesarios para disponer en la PC del entorno y las herramientas necesarias para comenzar a programar aplicaciones Android. [40]

Paso 1: Descarga e instalación de Eclipse

Utilizamos Eclipse Indigo la versión Eclipse IDE for Java Developers. Se puede utilizar el Eclipse 3.5 (Galileo) o superior.

Paso 2: Descargar el SDK de Android

El SDK de la plataforma Android se puede descargar desde <http://developer.android.com/sdk/index.html>.

Una vez descargado hay que descomprimir el zip en cualquier ubicación. Como sugerencia, no instalarlo en “Archivos de Programas”, si no por ejemplo en C:\Android, ya que hay problemas con los espacios cuando hay que referenciarla desde eclipse.

Paso 3: Descargar el plugin Android para Eclipse

Google pone a disposición de los desarrolladores un plugin para Eclipse llamado *Android Development Tools* (ADT) que facilita en gran medida el desarrollo de aplicaciones para la plataforma, por lo cual es recomendable instalarlo. Descargarlo accediendo al menú *Help > Install new software...* e indicar la url de descarga <https://dl-ssl.google.com/android/eclipse/> y seguir los pasos de instalación.

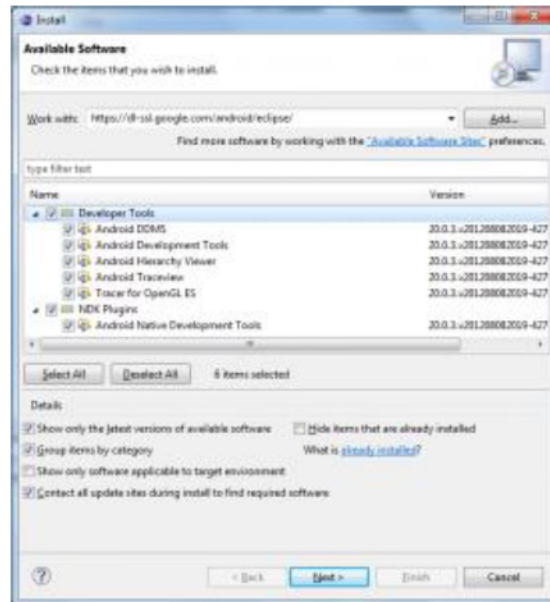


Figura 4.6: Instalación del plugin ADT

Paso 4: Configurar el plugin ADT.

En la ventana de configuración de Eclipse, se debe acceder a la sección de Android e indicar la ruta en la que se ha instalado el SDK (Paso 2)

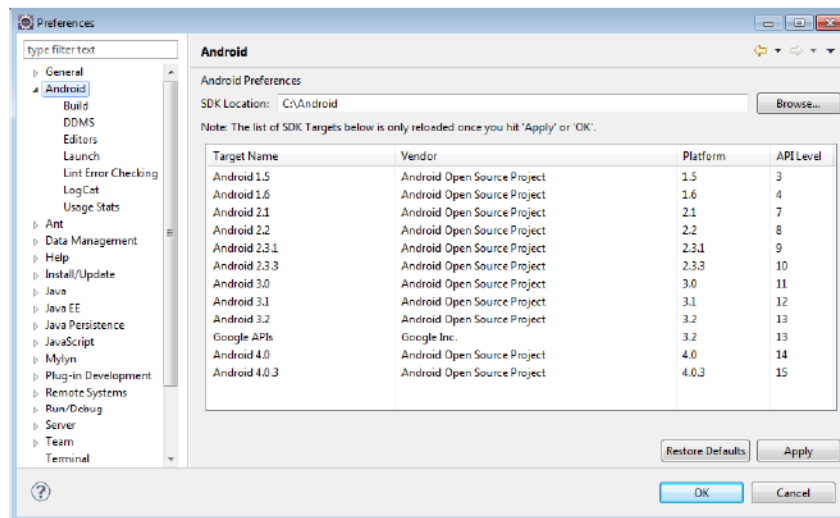


Figura 4.7: Localización del SDK

Paso 5: Agregar Plataformas y otros componentes

Además del SDK de Android comentado en el paso 2, también debemos descargar los llamados *SDK Targets* de Android, que son las librerías necesarias para desarrollar en cada una de las versiones concretas de Android. Así, si queremos desarrollar por ejemplo para Android 1.6 tendremos que descargar su *target* correspondiente. Para ello, desde Eclipse debemos acceder al menú “*Window / Android SDK and AVD Manager*”, y en la sección *Available Packages* seleccionar e instalar todos los paquetes deseados.

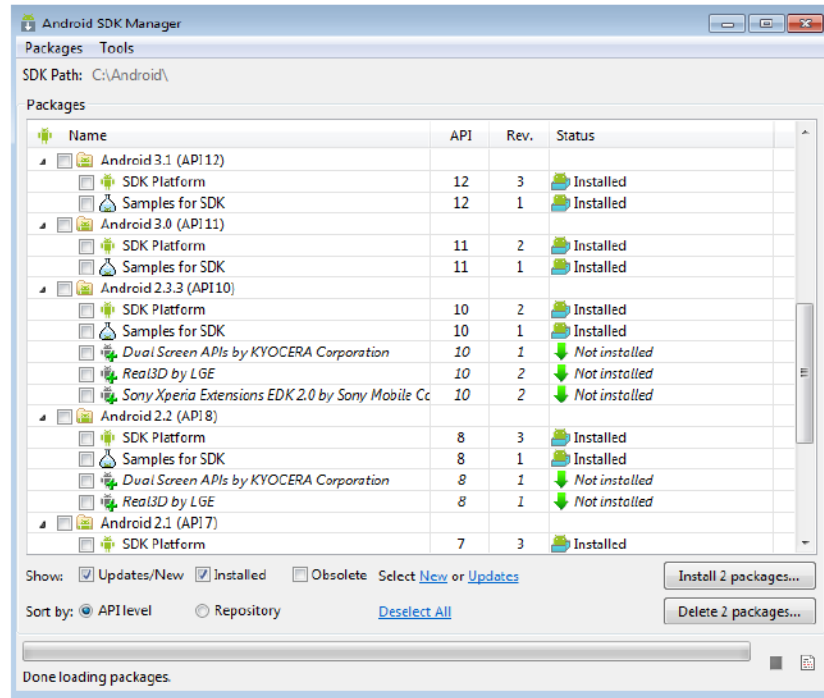


Figura 4.8: Android SDK Manager

Paso 6. Configurar un AVD.

Para probar y depurar aplicaciones Android podemos configurar un emulador o dispositivo virtual (*Android Virtual Device*, o AVD). Para ello, volveremos a acceder al *AVD Manager*, y en la sección *Virtual Devices* podremos añadir tantos AVD como se necesiten (por ejemplo, configurados para distintas versiones de Android).

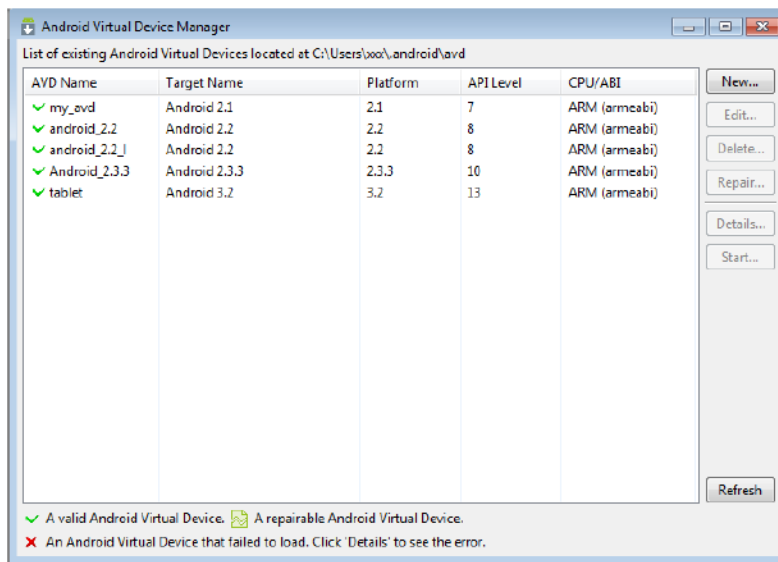


Figura 4.9: AVD Manager

Para configurar el AVD tan sólo tendremos que indicar un nombre descriptivo, el target de Android que utilizará, y las características de hardware del dispositivo virtual, como por ejemplo su resolución de pantalla, el tamaño de la tarjeta SD, o la disponibilidad de GPS.

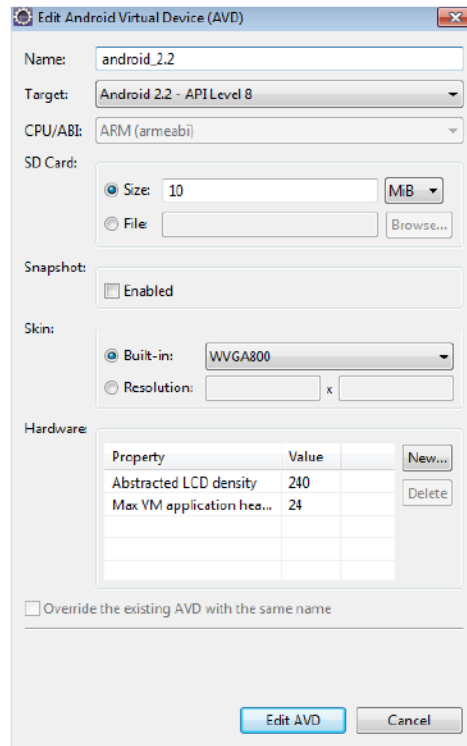


Figura 4.10: Configuración del AVD

Y con este paso se tiene el ambiente preparados para crear un proyecto Android.

4.9.2 Herramientas de desarrollo

Android Virtual Device (AVD)

El SDK de Android incluye un emulador de un dispositivo móvil, un dispositivo móvil virtual que se ejecuta en su computadora. El emulador permite desarrollar y probar aplicaciones Android sin necesidad de utilizar un dispositivo físico.

El emulador es una implementación de la máquina virtual Dalvik, lo que lo convierte en una plataforma válida para el funcionamiento de aplicaciones de Android como cualquier teléfono Android.

El emulador de Android imita todas las características de hardware y software de un dispositivo móvil típico, excepto que no pueden hacer llamadas de teléfono reales. Se ofrece una variedad de teclas de navegación y de control, que se puede "pulsar" con el ratón o el teclado para generar eventos de la aplicación. También proporciona una pantalla en la que aparece la aplicación, junto con todas las otras aplicaciones Android activas. [41]

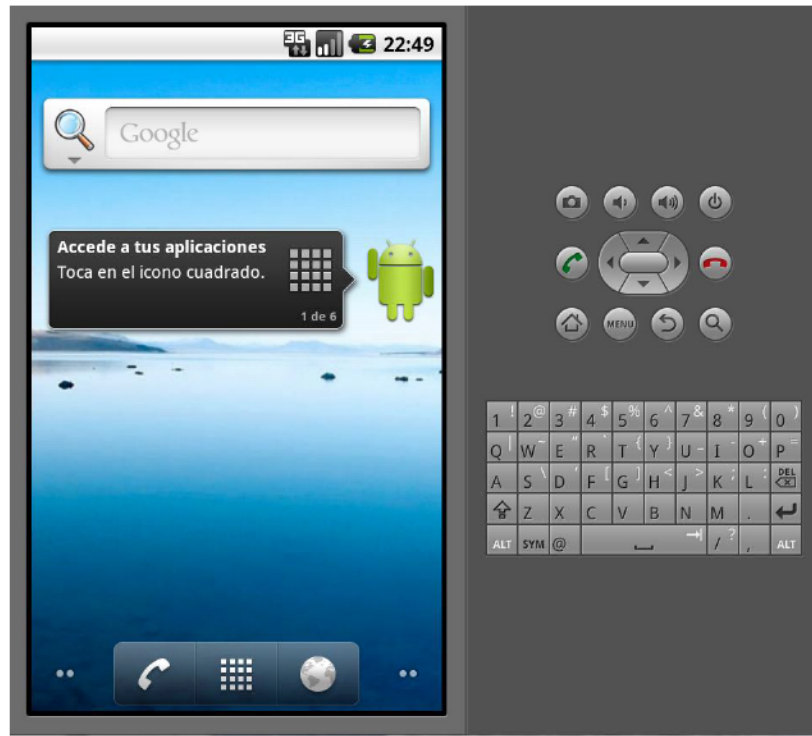


Figura 4.11: Emulador Android

El emulador también incluye una variedad de capacidades de depuración, como una consola desde la que se puede acceder al log del núcleo, simular interrupciones (tales como la llegada de mensajes SMS o llamadas telefónicas), y simular efectos de latencia y abandonos en la red de datos.

Dalvik Debug Monitor Server (DDMS)

Esta herramienta es un monitor de depuración de la máquina virtual Dalvik. Esta aplicación proporciona servicios de redireccionamiento de puertos, captura de pantallas, captura y listado de información en el dispositivo, hacer logcats, ver los procesos y la información del estado de radio, simular llamadas entrantes o SMS, e incluso simular una posición de ubicación [42]. Todas estas características convierten al DDMS en una herramienta muy útil para cualquier desarrollador.

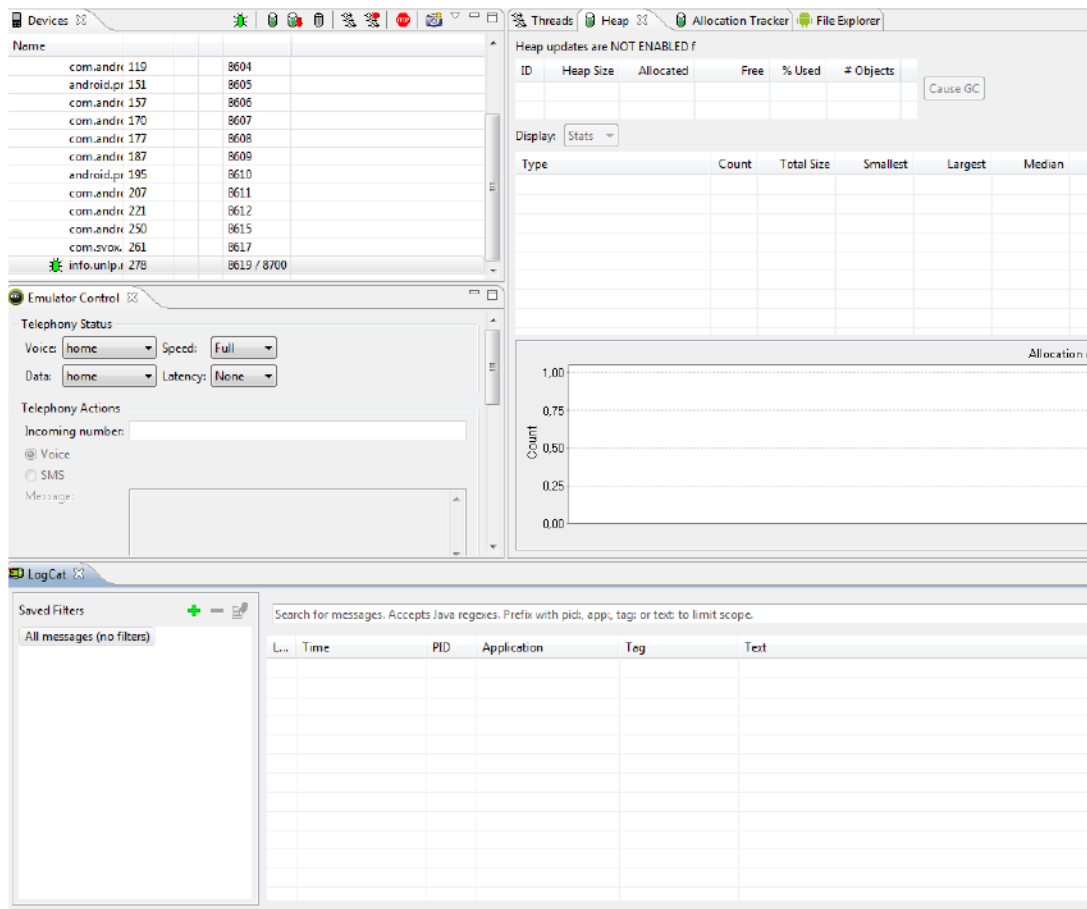


Figura 4.12: Perspectiva DDMS en Eclipse

Android Debug Bridge (ADB)

Android Debug Bridge (ADB) es una herramienta de línea de comandos versátil que le permite comunicarse con una instancia de un emulador o dispositivo conectado con tecnología Android. Se trata de un programa cliente-servidor que incluye tres componentes:

- Cliente, que se ejecuta en la máquina de desarrollo. Puede invocar a un cliente de una consola mediante el comando adb.
- Servidor, que se ejecuta como un proceso en segundo plano en el equipo de desarrollo. El servidor gestiona la comunicación entre el cliente y el daemon adb que se ejecuta en un emulador o dispositivo.
- Daemon, que se ejecuta como un proceso en segundo plano en cada instancia del emulador o dispositivo. [43]

La herramienta adb se puede encontrar en el directorio donde se instala el sdk:
<SDK>/plataforma-tools/

4.10 Estructura de un proyecto Android

Cuando creamos un nuevo proyecto Android en Eclipse se genera automáticamente la estructura de carpetas necesaria para poder generar posteriormente la aplicación.

En la siguiente imagen vemos los elementos creados inicialmente para un nuevo proyecto Android:

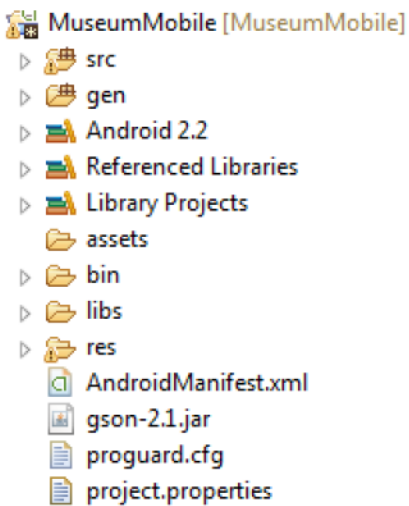


Figura 4.13: Estructura de un proyecto Android

Carpeta /src/

Contiene todo el código fuente de la aplicación, código de la interfaz gráfica, clases auxiliares, etc. Inicialmente, Eclipse creará por nosotros el código básico de la pantalla (*Activity*) principal de la aplicación, siempre bajo la estructura del paquete java definido, por ejemplo info.unlp.museum.

Carpeta /res/

Contiene todos los archivos de recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, etc. Los diferentes tipos de recursos se deberán distribuir entre las siguientes carpetas:

- /res/drawable/. Contienen las imágenes de la aplicación. Se puede dividir en /drawable-ldpi, /drawable-mdpi y /drawable-hdpi para utilizar diferentes recursos dependiendo de la resolución del dispositivo.
- /res/layout/. Contienen los archivos de definición de las diferentes pantallas de la interfaz gráfica. Se puede dividir en /layout y /layout-land para definir distintos layouts dependiendo de la orientación del dispositivo.
- /res/anim/. Contiene la definición de las animaciones utilizadas por la aplicación.
- /res/menu/. Contiene la definición de los menús de la aplicación.
- /res/values/. Contiene otros recursos de la aplicación como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), etc.
- /res/xml/. Contiene los archivos XML utilizados por la aplicación.
- /res/raw/. Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyen en el resto de carpetas de recursos.

Como ejemplo, para un proyecto nuevo Android, se crean los siguientes recursos para la aplicación:

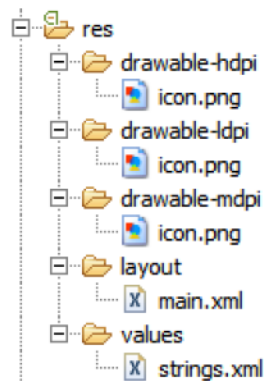


Figura 4.14: Recursos de una aplicación

Carpeta /gen/

Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. Cada vez que generamos nuestro proyecto se genera una serie de archivos fuente en java dirigidos al control de los recursos de la aplicación.

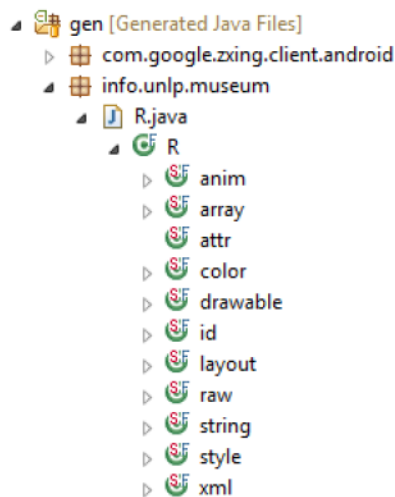


Figura 4.15: Carpeta gen

El más importante es R.java. Esta clase R contendrá en todo momento una serie de constantes con los ID de todos los recursos de la aplicación incluidos en la carpeta /res/, de forma que podamos acceder fácilmente a estos recursos desde nuestro código a través de este dato. Así, por ejemplo, la constante R.drawable.icon contendrá el ID de la imagen "icon.png" contenida en la carpeta /res/drawable/.

Carpeta /assets/

Contiene todos los demás archivos auxiliares necesarios para la aplicación (y que se incluirán en su propio paquete), como por ejemplo archivos de configuración, de datos, etc.

La diferencia entre los recursos incluidos en la carpeta /res/raw/ y los incluidos en la carpeta /assets/es que para los primeros se generará un ID en la clase R y se deberá acceder a ellos con los diferentes métodos de acceso a recursos. Para los segundos sin embargo no se generarán ID y se podrá acceder a ellos por su ruta como a cualquier otro archivo del sistema. Usaremos uno u otro según las necesidades de nuestra aplicación.

Archivo AndroidManifest.xml

El manifiesto hace una serie de cosas, además de declarar los componentes de la aplicación, tales como:

- Identificar los permisos de usuario de la aplicación requiere, tales como acceso a Internet o acceso de lectura a los contactos del usuario.
- Declarar el nivel de API mínimo requerido por la aplicación
- Declarar las características de hardware y software utilizado o requerido por la aplicación, tales como una cámara, los servicios de bluetooth, o una pantalla multitáctil.
- API de la aplicación que debe estar vinculada (aparte de la API de Android), tales como la API de Google Maps.

Capítulo 5. Usabilidad

En el capítulo anterior hemos descrito la plataforma Android. En el presente capítulo explicaremos el concepto de usabilidad y su aplicabilidad especialmente a los dispositivos móviles.

5.1 Introducción

Según la ISO/IEC 9241-11, la usabilidad es “el nivel con el que un producto se adapta a las necesidades del usuario y puede ser utilizado por el mismo para lograr ciertas metas con efectividad, eficiencia y satisfacción en un contexto específico de uso” [44]

Millones de personas utilizan hoy en día recursos informáticos tales como computadoras, redes de computadoras, smartphones, para solventar problemas de comunicación, de trabajo o por cuestiones cotidianas.

A medida que el número de usuarios se amplía cada día, incorporando personas de formaciones y culturas dispares, aumenta a la par la demanda de los mismos de adquirir sistemas de software con mayor nivel de asistencia y facilidad de uso.

Sistemas interactivos que provean un estilo de comunicación más simple, que sean cada vez más fáciles de utilizar, de instalar, de aprender, sin requerir ningún tipo de entrenamiento específico en Informática. Además, se pretende que el software brinde mecanismos de interacción más inteligentes y que se adapte al usuario en forma eficiente.

El usuario necesita ver al sistema interactivo que está utilizando como una herramienta capaz de ayudarlo a resolver sus problemas, a concretar sus intenciones. Pero esto no siempre es así. Es muy común que se le suman innumerables problemas más al utilizar el software, debiendo solucionar fallos o errores producidos cuyas causas no son claras, aprender cuestiones técnicas imprevistas, incurrir a ciertas artimañas para sortear obstáculos innecesarios, adivinar cómo realizar determinada tarea recurriendo a la técnica de prueba y error.

Aunque la componente funcional esté correcta y completamente desarrollada, existen un sin número de problemas de interacción que pueden empañar el acceso y productividad de la misma, provocando en el peor de los casos, la modificación general del software o hasta su final desuso.

Estos problemas de interacción residen en una componente del software denominada “Interfaz del Usuario”, que se encarga de innumerables cuestiones como la entrada y salida de la información, visualización de los datos, soporte, manejo y control del diálogo con el usuario final, entre otros.

La interfaz del usuario o componente de diálogo es una parte del software cuyo diseño afecta el nivel de productividad del sistema general, incide en el grado de satisfacción que pueda percibir el usuario, por lo tanto es determinante para la elección, utilidad y evaluación final del software.

Como la interfaz del usuario se encuentra dentro del sistema de software, su diseño e implementación está en manos del especialista informático, no puede delegarse a un diseñador visual o a cualquier otro profesional.

Es obligación del desarrollador de sistemas atender tanto cuestiones funcionales como de interacción, dando el mismo nivel de importancia a cada una de estas partes y responsabilizándose de su integración, manejo y control de ambas.

Es necesario entender que el desarrollo de la interfaz del usuario forma parte del proceso de la Ingeniería del software. Cómo es la única parte del sistema que interactúa directamente con los usuarios, es fundamental efectuar un estudio completo del comportamiento humano.

Se debe incluir como requerimientos indispensables en la generación del software parámetros tales como ergonomía, facilidad de uso, amigabilidad, simpleza (en términos de reducir el esfuerzo mental del usuario de llevar a cabo en la máquina la tarea en mente o pensada), flexibilidad, naturalidad (en el sentido que el usuario a través de la pantalla vea reflejado o modelado su realidad).

La interfaz del usuario debe ser diseñada teniendo en cuenta principios de diseño propios que no coinciden con la componente de aplicación. Debe ser planteada bajo la consideración de factores específicos que inciden en su gestación, tales como factores humanos principalmente, perfil de los usuarios, elementos del entorno, hardware disponible.

Su diseño parte desde un profundo conocimiento del o los usuarios finales del software y su contexto, más que desde los términos computacionales y algoritmos de programación.

Debido a que el proceso de diseño está centrado en el usuario, se trabaja con un nivel de incertidumbre y ambigüedad elevados, por lo que hace que el ciclo de vida de la interfaz sea especial y deba ser tratado en forma independiente.

La componente de interfaz, se construye con la práctica, necesita ser diseñada, evaluada varias veces hasta alcanzar los requerimientos de los usuarios.

Se generan versiones prototípicas de la interfaz que son puestas a prueba aunque la misma esté incompleta y aunque aún no tenga acoplada la componente funcional.

5.2 Fundamentos en HCI

La Interacción Hombre-Computadora, comúnmente referida con la abreviatura HCI, es el intercambio observable de información, datos y acciones entre un humano y la computadora, y viceversa. Está constituido por el diálogo, la conversación, la comunicación que pueda fluir en ambas direcciones, entre un usuario y el sistema de software que está utilizando. [45]

La "Interfaz del Usuario", en cambio es el medio por el cual, la interacción hombre-máquina es establecida, manejada y controlada. Conformar la parte del software y hardware que permite, que ese intercambio de información y sus distintas secuencias o hilos de diálogo, se produzcan. Estos dos términos, interacción hombre-máquina e interfaz del usuario, están muy entrelazados en el proceso de desarrollo y en estos casos se los usa como sinónimos. [45]

En muchos contextos ambos conceptos se refieren a las entradas del usuario final, su procesamiento localizado de las mismas y la presentación de las salidas o respuestas. La transformación funcional o algorítmica de esas entradas a esas salidas pertenece a la componente computacional o de aplicación.

Desde una concepción más amplia, se puede definir al HCI, también como una disciplina. Se la considera un área dentro de las Ciencias de Computación que se encarga del diseño, evaluación e implementación de sistemas de computación interactivos, para el uso humano, incluyendo el estudio de todos los fenómenos concernientes a ello.

Analizando más en profundidad esta última definición, los alcances de esta nueva disciplina son muy difusos y, pueden influir en ella factores muy heterogéneos, desde aspectos tecnológicos vinculados al hardware a aspectos sociales o psicológicos del usuario. Inciden además, cuestiones diversas como facilidad de uso, productividad, eficacia para expresar distintas posibilidades, capacidad de adaptación de acuerdo a distintos usuarios, desde niños a personas de mayor edad, pasando por distintas culturas y características sociales.

Como es un área que se centra fundamentalmente en la interacción entre humanos y máquinas, se puede plantear muchas situaciones diferentes. Cuando se refiere a "humano", puede considerarse desde un usuario particular, un grupo de usuarios, una organización o corporación, o hasta el mundo entero, como es el caso de las aplicaciones para Internet. Cuando se refiere a "máquina", puede incluirse desde una simple PC, una Workstation, una Intranet o hasta una supercomputadora con máquinas computacionales embebidas.

Además, el HCI se convierte en un área interdisciplinaria, que puede nutrirse de nociones provenientes de la Psicología, por la aplicación de teorías de procesos cognitivos y el análisis empírico del comportamiento del usuario; de la Sociología y Antropología, por el estudio de la interrelación entre la tecnología, el trabajo y las organizaciones; del Diseño Industrial, por el estudio y diseño de productos de interacción; del Diseño Visual, por el uso de las formas, colores y otros paradigmas visuales y sus diferentes aplicaciones; entre otros.

Otro punto que no hay que olvidar, son los aspectos relacionados con la Tecnología e Ingeniería, vinculados a la capacidad gráfica de las pantallas y dispositivos, para que el usuario ingrese información, ya sea por teclado, ratón, pantalla digital, voz, teléfono, entre otros medios.

Entonces, mientras exista el diálogo entre una máquina y un humano, surgirán innumerables factores relacionados con el HCI, que incluirán todos los aspectos para el diseño y construcción de interfaces del usuario adecuadas.

A continuación, se mostrará una figura que resume las interrelaciones entre los diferentes tópicos que aborda el área de HCI:

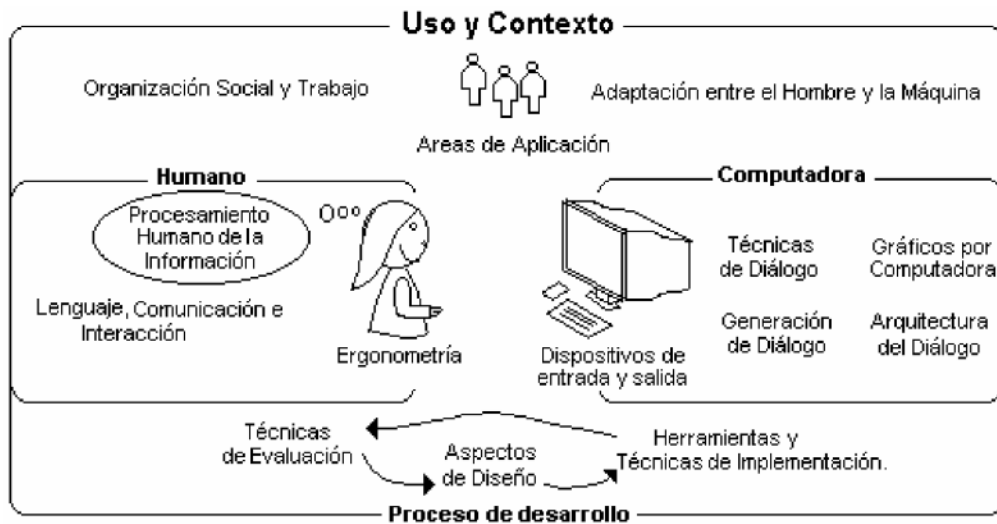


Figura 5.1: Interrelaciones entre diferentes tópicos del HCI

Los sistemas de computación existen dentro de un medio social, organizacional y de trabajo. Dentro de este contexto, se encuentran las aplicaciones. Incorporar computadoras en el trabajo implica un proceso de integración.

Además del uso y contexto social de las computadoras, del lado humano debemos considerar el procesamiento humano de la información, la comunicación - el lenguaje - y las características físicas del usuario -ergonomía-.

Desde el punto de vista de la computadora, una variedad de tecnologías han sido desarrolladas para soportar la interacción con humanos, como los dispositivos de entrada y salida.

Diálogos complejos o interfaces del usuario con características de avanzada, pueden llevar a consideraciones de arquitecturas de sistemas, necesarios para soportar cuestiones como tiempos de respuesta, interconectividad de redes, utilización de base de datos, acceso a servicios específicos, entre otros.

Finalmente, hay un proceso de desarrollo que incorpora el diseño del diálogo, herramientas de programación y técnicas de evaluación, cuyos resultados afectan las etapas anteriores, retroalimentándolas, en un proceso de continua mejoría.

5.3 Clasificación de las Interfaces del usuario

Con la llegada de los sistemas de tiempo compartido en la década del 70', el dispositivo de interfaz disponible era el teclado. Esto provocaba el desarrollo de un determinado estilo de interfaz que era totalmente textual, denominado "Interfaz orientada a Comandos".

Un procesador de comandos es un sistema al cual se le ingresa una cadena de caracteres que luego será analizado para determinar la función apropiada de la aplicación a invocar.

Este es el caso de la interfaz del usuario más elemental, se caracteriza por ser fácil de desarrollar, se interactúa únicamente con la consola y teclado, las pantallas y las salidas del sistema son tradicionales, con impresiones a cadena de caracteres.

La misma, presenta ciertas limitaciones, puesto que la entrada desde el teclado provoca mayor tasa de errores, requiere que el usuario recuerde el conjunto de posibles entradas legales, y

brinda una visión de la aplicación, del estilo “verbo-objeto” que no resulta amigable, ya que el usuario siente que es la aplicación la que tiene el control.

Luego, a medida que se incrementó la complejidad del hardware, aparecieron las “Interfaces orientadas a Menús”, que se caracteriza por presentar un conjunto de opciones, que pueden ser seleccionadas por los usuarios. Con estas interfaces se permitió abstraer la interfaz de la aplicación, pues fuerza al desarrollador a considerar el espacio de comandos y acciones como una entidad independiente.

Se caracteriza por ser fácil de usar y de implementar, la prefieren la mayoría de los usuarios inexpertos, sin entrenamiento previo. Estudios sobre factores humanos demuestran que el proceso de lectura y elección de las opciones del menú, resulta ser más sencillo que la invocación a comandos, que requiere recordar la sintaxis de los mismos. Además, alienta a la navegación y exploración del sistema.

Pero también, se le atribuyen ciertos inconvenientes como, por ejemplo, en los casos de presentar listas de opciones demasiado extensas, que resulta una técnica engorrosa, o cuando hay demasiados menús anidados, que puede provocar pérdida del contexto.

Tanto las “Interfaces orientadas a Comandos”, como las “Interfaces orientadas a Menús”, utilizan al texto como el único medio de representación y de interacción, empleando un diálogo secuencial, por lo tanto están encuadradas en lo que se denomina “Interfaces Textuales”.

A mediados de la década de los 80', debido a los grandes avances tecnológicos, la aparición de pantallas con mayor definición, dispositivos de interacción gráficos, como por ejemplo, el lápiz óptico o el ratón, comenzaron desarrollarse para los sistemas de software, un estilo de interfaz más poderoso que el textual.

Así, surgieron las “Interfaces Gráficas” -GUI- o “Interfaces Visuales”, que se caracterizaron por la utilización de recursos visuales para la representación de los objetos y por permitir la manipulación directa de los mismos, mediante un diálogo asíncrono.

Una GUI por lo general parte del sistema operativo de un equipo que se caracteriza por WIMPs y WYSIWYG [46]. WIMP (Windows icons, menus, pointing devices) es una abreviación de los conceptos de ventanas, iconos, menú y dispositivos de interfaz tales como mouse o trackball. Designa de un modo genérico el primer modelo interactivo desarrollado por el centro de investigación PARC (Palo Alto Research Center) para interactuar con los ordenadores a través de las interfaces gráficas de usuario. Las ventanas, los iconos y los menús, son elementos interactivos, que pertenecen a la parte simbólico-lingüística de la interfaz. El mouse pertenece al lado de interfaz humana o física del interfaz gráfico. Juntos constituyen el paradigma más potente y eficiente alcanzado hasta el momento para interactuar con los ordenadores personales. Aunque en la actualidad se intenta superar este paradigma desde la combinación de dispositivos de interfaz humano novedosos (pantallas táctiles) en combinación con nuevas metáforas visuales, el paradigma WIMP es el que está actualmente vigente en el grueso de los ordenadores personales que operan en la actualidad y dispositivos interactivos introducidos en el mercado.

WYSIWYG es el acrónimo de What You See Is What You Get (en inglés, "lo que ves es lo que obtienes"), se refiere a la capacidad de imprimir exactamente lo que ves en la pantalla.

Normalmente, lo que uno ve en la pantalla del ordenador y lo que uno obtiene en una impresión tienen naturalezas tecnológicas muy diferentes. En los mismo términos, conseguir que una cosa que se ve en la pantalla y algo que es impreso a través de una impresora sean próximos, entraña un conjunto de problemas técnicos nada fácil de lidiar.

Algunas cuestiones problemáticas relacionadas con el diseño WYSIWYG en el interfaz podrían ser las siguientes:

1. Escalabilidad: Consistiría en hacer coincidir y coordinar la representación de las escalas del objeto en pantalla de modo que se acerque a las expectativas del sujeto en la vida real.

2. Similitud: Consiste en acercar en forma, color y textura los signos que se recrean en pantalla y los signos que posteriormente serán impresos en el medio, por ejemplo la representación de imágenes o tipografía en pantalla y su reproducción final.

3. Modelado: La simulación de modelos tridimensionales, y sus posibles resultados están dentro de las problemáticas que este principio de diseño debe contemplar dentro de un sistema interactivo

Actualmente el principio WYSIWYG es el que ha modelado el espacio metafórico de la interfaz, haciéndonos creer que lo que vemos es tan real como lo que obtendremos posteriormente.

Este tipo de interfaces, además de incrementar el poder representativo, también aumentó los costos y la complejidad para manejar este tipo de representaciones.

Como una clase específica de las interfaces visuales, se encuentran las "Interfaces Icónicas", que utilizan como medio de interacción y representación visual, exclusivamente al icono. Se puede mencionar, que el icono es una imagen, una figura, pero que tiene un significado o semántica subyacente. El mismo está determinado por una imagen que debe ser significativa y fácilmente reconocible por la comunidad de usuarios.

También, se puede mencionar otro tipo de interfaces que son clasificadas como interfaces inteligentes, puesto que permiten que el comportamiento de la interfaz se acerque aún más al usuario, proveyendo capacidad de razonamiento, de adquisición y aplicación de conocimiento y de comunicación de ideas.

Dentro de las interfaces inteligentes, se pueden citar a las "Interfaces con signos de Adaptación", "Interfaces Evolutivas" e "Interfaces con Inferencia", que presentan características disímiles respecto a sus objetivos particulares, como a su cualidad de "inteligentes".

Las "Interfaces con signos de Adaptación", brindan diferentes modos de interacción que se pueden seleccionar automáticamente de acuerdo al tipo de usuario en cuestión. Son sensibles a los perfiles individuales de los usuarios y a sus estilos de interacción.

Las "Interfaces Evolutivas", tienen la propiedad de cambiar y evolucionar con el tiempo junto con el grado de perfeccionamiento que el usuario particular va adquiriendo con el sistema. Pueden acompañar la evolución o el crecimiento que presenta el usuario ante el uso del sistema, con nuevas ayudas, mensajes más específicos, un estilo de interacción más ágil, entre otras cuestiones.

Las "Interfaces con Inferencia", tienen la capacidad de captar secuencias de acciones que el usuario repite con frecuencia. Una vez registrado esa costumbre y ante la próxima iniciativa del usuario de realizarla nuevamente, el sistema se le adelanta y brinda la posibilidad de completar la secuencia de acciones en forma automática.

La automatización de este tipo de interfaces, se llevaría a cabo mediante el reconocimiento y almacenamiento de patrones observados en el comportamiento de cada usuario. Esta adquisición de información es transparente para el usuario, constituye una base de conocimiento especializado que, junto con parámetros o criterios de evaluación y de los métodos apropiados para aplicarlo, constituyen un marco apropiado para que la interfaz provea signos de inteligencia, simplificando la labor del usuario.

También, se encuentran las "Interfaces Web" o interfaces presentes en los sitios de la World Wide Web, que debido al espectro inmensurable de usuarios al que está dirigida, debe tener consideraciones de diseño y desarrollo especiales.

Por último, se pueden citar otras interfaces importantes, como las "Interfaces Accesibles", "Interfaces para Groupware", "Interfaces puramente Conversacionales" y las "Interfaces Móviles".

Las “Interfaces Accesibles”, son aquellas que respetan las normas del diseño universal, para que pueda ser accedida por cualquier usuario, independientemente de sus condiciones físicas o mentales.

Las “Interfaces para Groupware”, se caracterizan por interactuar con un grupo de usuarios que tendrán objetivos comunes, recursos a compartir, un ambiente virtual de reunión. La misma deberá tener facultades para solventar la coordinación del grupo, como aspectos de colaboración y comunicación entre los miembros del mismo.

Las “Interfaces puramente Conversacionales”, se montan sobre los sistemas telefónicos y se caracterizan por no contar con una pantalla. El único medio de salida es la voz y la entrada, puede ser a través de la voz, como a través de los botones del dispositivo telefónico.

Las “Interfaces Móviles”, son aquellas presentes en los dispositivos inalámbricos como celulares y tablets, que los usuarios utilizan para acceder a la aplicación.

Cuestiones de interrupción, distracción, visualización de una pantalla muy reducida, el apremio por acceder a la información en ese momento y en ese lugar, hacen que la interfaz presente características de diseño muy especiales.

Además de los tipos de interfaces mencionados anteriormente se encuentran: las interfaces hápticas, la cual está basada en el tacto, usando los movimientos del usuario como entrada y el sentido del tacto como salida (retroalimentación tanto táctil y kinestésica) [47]; las interfaces modales, donde el usuario se mueve entre diferentes estados o modos y en cada modo, la entrada del usuario se interpreta de una manera diferente [48]; la de Windows Surface, una interfaz diseñada para tablets y comercializada por Microsoft; entre otras.

5.4 Independencia del Diálogo

En los estudios iniciales sobre el desarrollo de sistemas interactivos se mezclaba fuertemente el diálogo con el software computacional, causando que el diálogo se convirtiera en una componente cada vez más resistente a cambios a medida que el diseño del sistema iba en progreso. Que el diálogo se encuentre inmerso y oculto dentro de la componente computacional atenta contra el desarrollo de interfaces del usuario de alta calidad puesto que la misma requiere de un ciclo iterativo de diseño y evaluación donde las modificaciones son una constante.

La solución emergió bajo el concepto de “Independencia del Diálogo” [Ehrich y Hartson 1981] que se basa en una definición formal para la comunicación entre la interfaz del usuario y el programa computacional.

En la práctica, esto significa que la componente computacional se desentiende de todo lo que abarca el aspecto de la interfaz, como su estilo de interacción, si provee un lenguaje de comandos para comunicarse con el usuario final o si está basada en menús o formularios.

La independencia del diálogo es crucial tanto para modificar fácilmente la interfaz en el proceso iterativo de refinamiento como para su mantenimiento.

Sin independencia del diálogo, la forma en que se lo estructura y los detalles de cómo el mismo se comunica con el usuario final está dirigido y controlado por los requerimientos computacionales del sistema de aplicación. Los conocimientos específicos sobre el diálogo y decisiones sobre estilos de interacción se entremezclan con la componente computacional, provocando no sólo la resistencia a cambios ya mencionada y necesaria en la construcción de la interfaz, sino que dificulta además la consideración de factores humanos en el desarrollo de la misma.

Entonces, presenta dos tipos de inconvenientes, uno a nivel relacionado con las Ciencias de Computación y otros a nivel de factores humanos. Respecto al primer tipo de problemas, se refiere a que cada rutina computacional necesita de un conjunto de valores de entrada validados

para su posterior proceso, desentendiéndose de la forma en que esas entradas fueron validadas, cómo fueron ingresados por el usuario, ni desde qué dispositivo. Necesita focalizar su interés en el procesamiento de la información no en la conversación con el usuario.

Los problemas a nivel de los factores humanos es diseñar el diálogo de una manera que la computadora se adapte a los requerimientos del usuario, convirtiéndose en una herramienta que realiza las tareas al mismo. Desde esta perspectiva el diálogo debe conjugar la simplicidad de interacción con la optimización en el proceso de validación, aceptando los valores de entradas verificándolos de acuerdo a determinados criterios, proveyendo feedback y ayudas para guiar al usuario a evitar el ingreso de entradas erróneas y a considerar solamente aquellas que pertenezcan al conjunto de valores permitidos.

5.5 Componentes de la Interfaz del Usuario

La mayoría de los modelos existentes son descripciones estructurales que especifican la manera que la interfaz del usuario se relaciona con el sistema de aplicación, no son modelos descriptivos de la propia interacción hombre-computadora. Entre estos modelos se encuentra el que se desarrolló en R. Seeheim, más tarde modificado por Albert Green, en el año 1985. Este modelo se muestra en la siguiente figura:

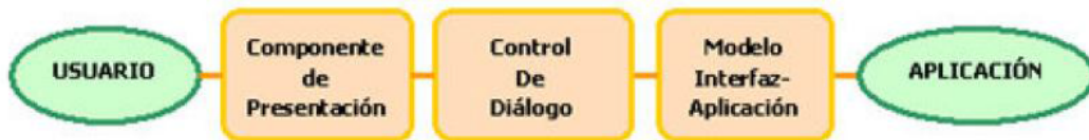


Figura 5.2: Modelo Interacción Hombre-Máquina

La componente de presentación es la responsable de la visualización de la interfaz, especifica las pantallas que se presentarán al usuario y el estilo de interacción. Los colores, ubicaciones de la pantalla, figuras, menús son conceptos relacionados con esta componente.

La componente de control de diálogo está a cargo del manejo de la secuencia de eventos y del control de las técnicas de interacción provistas. Define la estructura, secuencia, el proceso del intercambio de información entre el usuario y la aplicación.

Finalmente, la componente denominada Modelo de Interfaz-Aplicación es la representación de la aplicación desde el punto de vista de la interfaz. Contiene tanto una visión de la aplicación desde la interfaz como una visión de la interfaz desde la aplicación. Abstrae, por un lado, la forma en que el usuario consulta y provee información a la aplicación y, por otro, independiza las acciones de la aplicación de las implementaciones particulares. Especifica la relación entre los eventos de la interfaz y las funcionalidades de la aplicación. La comunicación entre ambas partes del software se realiza vía llamados a procedimientos y mediante el uso de estructuras de datos descriptas en un nivel de abstracción independiente de la implementación.

5.6 Ciclo de Vida de la Interfaz del Usuario

Algunos desarrolladores de software posponen la construcción de la interfaz solamente a la etapa de implementación, pero como se explicó anteriormente, existen descripciones, detalles, consideraciones presentes en la interfaz que intervienen y afectan a todo el proceso de ingeniería del sistema y que deben ser encaradas desde el principio.

Tanto la interfaz del usuario como la parte de cómputos se desarrollan en etapas que consisten de una fase de especificación, diseño, implementación y de un proceso de evaluación. Pero, la forma en que se manifiestan estas etapas en la interfaz difiere radicalmente con la de la aplicación.

Mientras que la componente de cómputos requiere que cada etapa de desarrollo culmine correcta y completamente antes que comience la siguiente, la interfaz del usuario necesita que las mismas se realicen en forma simultánea y concurrente, y que estén en constante estado de evaluación y modificación.

La diferencia reside, en que en el diseño centrado en el usuario, el usuario juega un rol preponderante en todas las etapas de desarrollo, desde la etapa inicial donde se lo modela hasta participar en evaluaciones del producto, o en sugerir aspectos de diseño.

El ciclo de vida de la UI requiere de un proceso iterativo de refinamientos sucesivos, donde las etapas se solapan, se adelantan y se regeneran, constituyendo lo que se denomina "Ciclo de Vida Prototípico", a diferencias del ciclo de vida tradicional en cascada.

En el ciclo de vida prototípico, las etapas iniciales de requerimiento y diseño se desarrollan en forma incompleta y se implementan parcialmente generando lo que se denomina "prototipos del sistema".

Como el prototipo comienza mostrando la confección parcial del producto, se lo utiliza para proveer una visión temprana del sistema y así se lo puede poner a prueba anticipadamente frente a usuarios. Esto permitirá obtener información más concreta sobre las necesidades del usuario, preferencias, modos de trabajo, errores que comete, dificultades que presenta y demás cuestiones observables que completarán y corregirán los requerimientos y diseños iniciales.

A medida que al prototipo se lo corrige y se le van agregando en forma gradual nuevos elementos de interfaz, comportamiento, servicios y hasta componentes funcionales de la aplicación, se van generando nuevas versiones prototípicas de mejor calidad. Incluso, en el caso de los prototipos evolutivos, se llega a la versión final del software, con la aprobación ya garantizada de los usuarios.

Todo este proceso se desarrolla mediante un esquema metodológico denominado "Prototipación", que consiste en la generación sistemática de versiones prototípicas del software, que se van evaluando mediante la participación activa de los usuarios y que se van mejorando progresivamente.

El ciclo de vida culmina cuando las evaluaciones de las versiones avanzadas del prototipo van arrojando resultados positivos, es decir, que el usuario o demás roles evaluadores, aceptan el diseño expresado y simulado en el prototipo.

A continuación, se ilustra el ciclo de vida prototípico de la interfaz, teniendo en cuenta una prototipación evolutiva en donde la última versión prototípica converge al sistema de software completo, totalmente culminado y aprobado. Se muestran cómo ocurren las diferentes etapas de desarrollo a través del tiempo:



Figura 5.3: Ciclo de vida prototípico de la interfaz

Como se muestra en el gráfico, ni bien se obtienen algunas especificaciones iniciales de lo que se pretende de la interfaz, constituyendo una etapa de requerimientos o de análisis incompleta, se comienza con el diseño de la misma.

Estos bosquejos de partes parciales de la interfaz, son implementados en forma de prototipos para ser evaluados ante el usuario, y poder así obtener más información de él frente a material concreto.

Con la información obtenida de las evaluaciones, se completa y corrige aún más las etapas de análisis y diseño. Por ello que se establecen en forma simultánea.

Una vez que se desarrolle el prototipo final evolutivo, donde se incluye toda la componente de diálogo y la funcional, y los tests del mismo arrojen resultados positivos, culminarían en forma conjunta todas las etapas cruciales de desarrollo.

La etapa de entrenamiento comienza desde que las versiones prototípicas iniciales, por lo que el usuario comienza a conocer, probar, usar el producto y familiarizarse con él desde el principio. Se puede dejar un poco más de tiempo para el entrenamiento una vez que el producto haya sido terminado.

Las etapas de conversión e instalación se refieren a la puesta a punto del producto sobre las máquinas que se destinan para la utilización del software. En el caso que se mejora o se reemplaza un producto de software anterior, hay un proceso de transferencia, adaptación de datos e información que constituyen la etapa de conversión.

Los autores tales como Joelle Coutaz y Len Bass, como por otra parte Deborah Hix y J.Hartson, muestran esquemas del ciclo de vida de una interfaz en forma de espiral y de estrella respectivamente, como se ilustra a continuación:



Figura 5.4: Ciclo de vida de una interfaz en forma de espiral y de estrella

Ambos diagramas expresan la filosofía prototípica, ya que en el caso de la espiral, cada giro de la misma está constituido por el diseño, desarrollo y evaluación de una versión prototípica. En el caso de la estrella, muestra al proceso de evaluación como eje central, a partir del cual se refinan las demás etapas de desarrollo.

5.7 Métricas de Evaluación de una Interfaz: Principios de Nielsen

Un principio sería una solución posible a un problema de diseño que ayuda a definir cómo debe mostrarse y comportarse un sistema, lo que mejora elementos de la interfaz. Conseguimos así que se proporcione a los usuarios lo necesario para interactuar exitosamente y que se presente la información de manera que se facilite su entendimiento.

Cuando hablamos de principios del diseño de interfaces de usuario, se ha generalizado la utilización del término heurístico para referirse a directrices que establecen requisitos que debe cumplir el diseño con el fin de facilitar su comprensión y uso por parte del usuario final. [49]

En este punto, se considerará determinados principios de diseño, desarrollados en primer término por Jacob Nielsen, respecto del diálogo correcto que debe proveerse en una interfaz del usuario. Sin embargo, diversos autores como Norman (1983), Heckel (1984), Mayhew (1992), Preece, Rogers y Sharp (2002), Tognazzini (2003), Shneiderman (2005); han propuesto en diversos momentos, y con la intención de dar un enfoque universal, guías y listados de principios que ofrecen recomendaciones desde diferentes perspectivas del diseño de experiencias de usuario (como interfaz, interacción, contexto o tareas). [49]

Los principios de Nielsen inicialmente fueron estipulados para interfaces textuales, pero sirven de base para el diseño preliminar de cualquier otro tipo de interfaz, desde la más sencilla a la más compleja. Estos principios, constan de 10 normas, las cuales se detallarán a continuación:

1 - Diálogo simple y natural: Este punto estipula la forma en que la interacción debe llevarse a cabo. El diálogo presente en los prompts, en los títulos, en los contenidos de las pantallas, en la solicitud de las entradas como en la muestra de los resultados. Se debe evitar abuso de abreviaturas, uso excesivo de mayúsculas, evitar errores de tipeo y distribuir bien la información.

2 - Emplear lenguaje del usuario: con este principio, se procura emplear en el sistema un lenguaje familiar al usuario y no familiar al programador del software, por lo tanto se pide evitar lenguaje técnico, lenguaje extranjero, truncamiento de palabras, entre otros.

3 - Minimizar el uso de la memoria del usuario: En este punto, se pretende evitar que el usuario esfuerce su memoria para interactuar con el sistema. Evitar que recurra a carpetas manuales para recordar algún código o tenga que recurrir a asistencia de un tercero. Esto implicaría un fracaso para el desarrollador de la interfaz.

4 - Consistencia: La consistencia es un punto clave para ofrecer confiabilidad y seguridad al sistema. Se refiere a que tanto el diálogo, el aspecto visual, el aspecto terminológico, el comportamiento del sistema se presenten ante el usuario en forma homogénea y consistente. Sin ambigüedades.

Los usuarios no deberían tener que adivinar si diferentes palabras, situaciones o acciones significan lo mismo en distintos contextos, o si un mismo término adquiere diferentes significados a lo largo del sistema. Todas estas situaciones hacen que la interfaz se considere inconsistente y de alguna forma hace que el usuario se desconcierte y no sepa qué actitud tomar.

5 - Feedback: El sistema debería mantener al usuario informado de lo que está sucediendo, más que nada cuando éste manipula los objetos directamente sobre la pantalla. Una manera de lograr esto es utilizando la técnica llamada feedback.

El proveer feedback la interfaz da una respuesta gráfica o textual en la pantalla frente a una acción del usuario, indica el efecto o reacción producida por la acción del mismo. Por lo que se deberían brindar mensajes informativos, dar mensajes de confirmación, dar información sobre el estado del sistema, entre otros.

6 - Salidas evidentes: Este principio apunta a que el usuario en todo momento tenga al alcance una opción de salida de emergencia que esté fácilmente identificable y accesible.

Esta posibilidad que le brindaría el sistema de salir del estado en que se encuentra, brinda la sensación de seguridad al usuario, pues le permitiría dejar cuanto antes alguna situación no deseada.

7 - Mensajes de error: Eventualmente, cuando los usuarios realizan alguna combinación errónea de funciones provistas por el sistema, éste suele responder con mensajes de error.

Los mensajes de error constituyen un tipo de interacción que fluye de la máquina al usuario. Es el feedback del sistema ante la presencia de un error. Estos mensajes de error deben ser expresados en un lenguaje natural, su contenido debe indicar cuál es el problema, el lugar donde se encuentra en forma precisa y clara, y sugerir alguna solución alternativa.

Aquí, hay que considerar que el usuario llegó a un punto crítico del sistema, a un punto de inseguridad, de incertidumbre por no saber bien qué hizo bien, qué hizo mal, en qué afecta, el error cometido, a los datos y demás.

El primer cuestionamiento que el desarrollador del sistema se debe hacer, es por qué se permitió que el usuario llegue a ese punto. Quizás ante la presencia de mensajes aclaratorios, buena información, más control, el error se hubiera evitado. En principio, se debería prevenir la presencia de errores de gravedad.

Ahora bien, el segundo cuestionamiento que se debe hacer es que si el usuario se encuentra en ese punto crítico, de qué mejor forma se lo puede ayudar para salir de él, lo más eficientemente posible.

8 - Prevención de errores: A veces, es preferible que en vez de proveer buenos mensajes de error, se brinde un diseño más cuidadoso que prevenga problemas para que directamente éstos no ocurran, así la persona no se encuentra en situaciones inesperadas.

En este principio, se estipula la necesidad de evitar que el usuario llegue a una instancia de error. Como desarrolladores del sistema, se debe saber cuáles son las partes más críticas del software, con mayor posibilidad de error.

Generalmente, las mismas se encuentran en la carga de datos, en las transacciones muy relevantes como eliminaciones, movimientos.

9 - Atajos: Aquí, se pide que el sistema muestre aspectos de adaptación, que considere la posibilidad de dialogar con usuarios que pueden tener distinto grado de conocimiento, formación y experiencia en el uso de aplicaciones.

Por ello, la interfaz debería proveer de alternativas de manejo para que resulte cómodo y amigable tanto para usuarios novatos como para los muy experimentados, proveyendo de esta manera mecanismos de adaptación, por ejemplo atajos o macros.

10 - Ayudas: Las ayudas son componentes de asistencia muy importantes para el usuario, y esto debe ser tenido en cuenta por el desarrollador que muchas veces no ofrece siempre ayudas, o las construye a la ligera sin prestar demasiada atención a las mismas.

Existe mucho análisis en cómo deben ser las ayudas de un software, más aún porque es generalmente texto que debe ser interpretado fácilmente. Cuando un usuario ingresa a una Ayuda, lo hace porque duda, desconoce, quiere consultar, o teme por realizar alguna acción errónea. Tiene la presión de entrar a la Ayuda, buscar, ubicarse, leer, interpretar.

Por lo tanto, un mal diseño de las mismas puede llegar a entorpecer, y dificultar este proceso, más que esclarecer o asistirlo realmente.

5.8 Interfaces Móviles

Cuando queremos crear experiencias móviles, es difícil crear una gran experiencia, sin tener estos tres ingredientes: **el contexto, la arquitectura de información y el diseño visual.** [2]

5.8.1 Contexto

El contexto es probablemente el concepto más utilizado en móvil. El contexto lo podemos definir de dos maneras:

- El contexto que permite entender mejor a una persona, un lugar, una cosa, una situación mediante la adición de información al mismo. Aumenta la experiencia y el conocimiento del entorno en una manera significativa.
- El contexto que es el modo, medio o entorno en el que llevamos a cabo una tarea. La ubicación actual, contexto físico, el estado de ánimo de la persona, etc. Cada entorno dictará cómo puedo acceder a la información y por lo tanto, cómo obtener valor de la misma.

Los dispositivos móviles, a diferencia de cualquier otro medio, presentan una buena oportunidad para crear experiencias contextuales y significativas.

Para ello necesitamos considerar en nuestra aplicación diferentes contextos en los que se pueden encontrar los usuarios:

- ¿Quiénes son los usuarios? ¿Qué sabemos de ellos? ¿Qué tipo de comportamiento podemos asumir o predecir?
- ¿Qué está pasando? ¿Cuáles son las circunstancias en las que mejor se absorbe el contenido que se va a presentar?
- ¿Cuándo van a interactuar? ¿Cuando están en casa y tiene grandes cantidades de tiempo? ¿En el trabajo, donde tienen períodos cortos de atención? ¿Durante los períodos de inactividad, en la espera de un tren?
- ¿Dónde están? ¿Están en un espacio público o un espacio privado? ¿Están dentro o afuera? ¿Es de día o es de noche?
- ¿Por qué van a utilizar nuestra aplicación? ¿Qué valor tendrá que ganar de su contenido o de los servicios en su situación actual?
- ¿Cómo están utilizando sus dispositivos móviles? ¿Se sostiene en la mano o en el bolsillo? ¿Cómo se lo mantiene? ¿Abierto o cerrado? ¿Vertical u horizontal?

La intención es definir las circunstancias, o la retórica, de la forma en que nos comunicamos y entender las ideas. El contexto es lo que hace al móvil un medio poderoso.

5.8.2 Arquitectura de la información

La arquitectura de la información (también conocido como IA), es el fundamento de un producto móvil. Un producto bien diseñado, con buen diseño visual puede fallar debido a una arquitectura de información deficiente.

La arquitectura de la información móvil no sólo define cómo su información será estructurada, sino también cómo la gente va a interactuar con ella. Esto se hace especialmente difícil si tenemos en cuenta que los diferentes dispositivos tienen diferentes capacidades y modos de interacción. Por ejemplo, un dispositivo puede ser táctil o puede utilizar el pad direccional para navegar a la ubicación deseada.

Además, la arquitectura de información hace hincapié en cómo abordar el contexto. En otras palabras, una buena arquitectura de la información móvil se basa en los contextos de los diversos usuarios.

Similar a cómo la tecnología móvil tiene muchas facetas, también las tiene la arquitectura de información, ya que se utiliza a menudo como un término general para describir varias disciplinas únicas, incluyendo las siguientes:

- **Arquitectura de la información:** La organización de los datos dentro de un espacio informativo. En otras palabras, cómo el usuario obtendrá la información o realizará las tareas dentro de un sitio web o aplicación.
- **Diseño de interacción:** El diseño de cómo el usuario puede participar con la información presente, ya sea en forma directa o indirecta, es decir, cómo el usuario interactúa con el sitio web de la aplicación para crear una experiencia más significativa y lograr sus objetivos.
- **Diseño de la información:** La organización de la información o cómo el usuario va a evaluar el significado y la dirección dada la información que se le presentó.
- **Diseño de la navegación:** Las palabras usadas para describir los espacios de información, las etiquetas o desencadenadores se utilizan para informar a los usuarios lo que algo es y para establecer la expectativa de lo que encontrarán.
- **Diseño de la interfaz:** El diseño de los paradigmas visuales utilizados para crear una acción o comprensión.

Como se puede ver, la arquitectura de información compone el núcleo de la experiencia del usuario.

El papel de un arquitecto de información móvil sería la interpretación del contenido de una página web de escritorio al contexto móvil. ¿Utiliza la misma estructura, o las secciones? ¿Se presenta la misma información? ¿cómo debería ser priorizada? ¿De qué manera el usuario se desplaza a otras áreas? ¿Utiliza los mismos paradigmas visuales y de interacción, o es necesario inventar otros nuevos?

También debe diseñar la arquitectura de la información móvil para hacer frente al contexto móvil. Teniendo en cuenta que muchos dispositivos pueden detectar la ubicación actual, que es uno de los tipos más inmediatos de contexto.

5.8.3 El diseño visual

Los usuarios buscan ver varias capturas de pantalla, leer los comentarios de los usuarios y juzgar el producto basado en la calidad de su icono y de las capturas de pantalla antes de comprar.

Para realizar un buen diseño se debe comenzar con proporcionar la mejor experiencia posible y adaptar esta experiencia para el mercado que más se quiere. Iterar, ajustar y afinar hasta hacerlo bien. No se debe ver obstaculizado por las limitaciones de la tecnología. Frases como "mínimo común denominador" no puede ser parte del vocabulario del diseñador. Tratar de crear un diseño móvil en el contexto de las limitaciones del dispositivo no es donde debe comenzar, sino que es donde debería terminar.

A continuación se detallan los elementos a tener en cuenta en el diseño móvil:

1 - Contexto

El contexto es fundamental para la experiencia móvil. Como diseñador, es su trabajo para asegurarse de que el usuario puede encontrar la manera de abordar utilizando el contexto de su aplicación. Las respuestas a las preguntas mencionadas anteriormente afectará en gran medida el curso del diseño. Tratar a estas preguntas como una lista de verificación para el diseño de principio a fin. Ellos pueden proporcionar no sólo una gran inspiración para los desafíos de diseño, sino la justificación de las decisiones de diseño más tarde.

2 - Mensaje

Otro elemento de diseño es el mensaje, o lo que se está tratando de decir acerca del sitio web o aplicación. El mensaje es la impresión general mental que crea de forma explícita a través del diseño visual.

El enfoque en el diseño define el mensaje y crea expectativas. Un diseño escaso, minimalista, con un montón de espacio en blanco le dirá al usuario que se enfoque en el contenido. Un diseño "pesado" con el uso de colores oscuros y un montón de gráficos le dirá al usuario que espere algo más inmersivo.

3 - Look and Feel

Es la apariencia basada en un diseño tangible, algo que claramente se puede describir y mostrar a los usuarios. Para establecer un aspecto podemos basarnos en "patrones de diseño", o soluciones documentadas a problemas de diseño, a veces conocidos como guías de estilo. En los grandes proyectos de telefonía móvil o en empresas con varios diseñadores, una guía de estilo o patrones de diseño es fundamental para mantener la consistencia en el look and feel y la reducción de la necesidad de justificar cada decisión de diseño.

4 - Layout

El layout es un elemento importante en el diseño, porque es la forma visual en la que el usuario va a procesar la página.

En el diseño del móvil, el elemento de contenido principal con el que hacer frente es la navegación. Si se está diseñando un sitio o aplicación, es necesario proporcionar a los usuarios métodos para realizar las tareas, navegar a otras páginas o leer e interactuar con el contenido. Esto puede variar, dependiendo de los dispositivos que soportan.

Otra consideración del layout es como su diseño escala a medida que cambia la orientación del dispositivo, por ejemplo, si el dispositivo se gira de modo vertical a horizontal y viceversa. Normalmente, se describe como fijo (un número determinado de píxeles de ancho) o fluido (que tiene la capacidad de escalar al tamaño completo de la pantalla independientemente de la orientación del dispositivo).

5 - Color

El obstáculo más común que encontramos en cuanto al color es la pantalla de los móviles, que vienen con un número diferente de colores o profundidad de bits, es decir, el número de bits (dígitos binarios) que se utilizan para representar el color de un solo píxel en una imagen de mapa de bits. Cuando los diseños complejos se muestran en diferentes dispositivos móviles, la profundidad del color limitada en un dispositivo puede producir bandas o posterización no deseado en la imagen.

Otra consideración a tener en cuenta con respecto al color es que las personas responden a los colores de manera diferente. Es bastante conocido que diferentes colores producen diferentes emociones en las personas. Pensando en las emociones que provocan los colores en las personas es un aspecto importante en el diseño de móviles, que es un medio tan personal. Además, se debe tener en cuenta que diferentes colores pueden tener significados distintos en

diferentes culturas. En algunos casos, el color que se utiliza puede tener significados opuestos en diferentes culturas. Esto es algo a considerar cuando se piensa en la implementación de aplicaciones móviles en los países con el mayor número de dispositivos móviles, como China o India.

La definición de paletas de colores puede ser útil para mantener un uso constante del color en el diseño móvil. Paletas de colores suelen consistir en un número predefinido de colores a utilizar en todo el diseño.

6 - Tipografía

Al parecer, existen dos enfoques básicos de cómo el tipo se representa en las pantallas de los móviles: el uso de subpíxeles basados en pantallas o con una densidad de píxeles mayor o píxeles por pulgada (ppi). Un subpixel es la división de cada píxel en un color rojo, verde y azul (o RGB) la unidad a un nivel microscópico, lo que permite un mayor nivel de suavizado de líneas para cada carácter de la fuente. La adición de estos subpíxeles RGB permite al ojo ver mayores variaciones de gris, creación de antialiasing y texto nítido.

El segundo enfoque consiste en utilizar una gran densidad de píxeles o píxeles por pulgada. A menudo nos referimos a las pantallas, ya sea por sus dimensiones físicas (Pantalla de 15,4 pulgadas) o sus dimensiones en píxeles o resolución (Resolución de 1440x900 píxeles). La densidad de píxeles se determina dividiendo la anchura del área de visualización en píxeles por la anchura de la zona de visualización en pulgadas.

Afortunadamente, los dispositivos móviles de hoy en día tienen algunas opciones más que un solo tipo de letra, pero las opciones son aún bastante limitadas. Esencialmente, tienen algunas variaciones de fuentes serif, sans-serif y monospace, y dependiendo de la plataforma, tal vez algunas fuentes personalizadas.

El papel más importante de la tipografía en el diseño del móvil es proporcionar al usuario una legibilidad excelente, o la capacidad de seguir con claridad las líneas de texto con el ojo y no desorientarse. Esto se puede hacer siguiendo estas seis simples reglas:

- **Usar un tipo de letra de alto contraste:** Recordar que los dispositivos móviles se utilizan generalmente fuera. Tener un tipo de letra de alto contraste en relación con el fondo aumentará la visibilidad y la legibilidad.
- **Utilizar el tipo de letra correcto:** El tipo de letra que utiliza le dice al usuario qué esperar. Por ejemplo, un tipo de letra sans-serif es común en la navegación o en áreas compactas, mientras que los tipos de letra serif son muy útiles para las áreas de contenido de larga duración o densas.
- **Proporcionar un espaciado de línea adecuado:** Aumentar el espacio para evitar que los usuarios se pierdan.
- **Dejar un espacio a la derecha y a la izquierda de cada línea, no llenar la pantalla:** por lo general entre tres y cuatro caracteres de ancho.
- **Títulos:** Usar títulos para indicar al usuario lo que está por venir. Utilizando diferentes tipos de letra, color y énfasis en los títulos ayudan a crear páginas de lectura.
- **Utilizar párrafos cortos:** mantener los párrafos cortos, con no más de dos o tres frases por párrafo.

7 - Gráficos

La forma más común de los gráficos utilizados en el diseño móvil son los iconos. La iconografía es útil para comunicar ideas y acciones a los usuarios en un espacio visual restringido.

Fotos e imágenes se utilizan para agregar significado a los contenidos, a menudo mostrando una representación visual de un concepto o para agregar significado a un diseño. La utilización de fotos e imágenes no es tan común en el diseño móvil, debido a que las imágenes tienen una altura y una anchura definidas y tienen que hacerse a escala al tamaño del dispositivo adecuado. El uso de gráficos para añadir significado a un diseño puede ser útil, pero puede encontrarse con

problemas con respecto a la forma en que la imagen se mostrará en una interfaz de usuario flexible, por ejemplo, cuando la orientación del dispositivo cambia.

8 - Multiplicidad de dispositivos móviles

Dispositivos con distinto tamaño de display, con diferentes objetos de interacción admisibles y modos de interacción.

5.9 Diseño de aplicaciones Android

Una de las cosas más importantes en el desarrollo de Android es sin duda el diseño de la interfaz que atraerá al usuario a usar la aplicación. Si bien, la funcionalidad no deja de ser importante, en el mundo móvil se observa el patrón constante de que los usuarios no sólo buscan que la aplicación haga lo que tenga que hacer sino que además las pantallas con las que interactúan sean llamativas, bonitas y usables.

Para los diseñadores, Android es el referente a tomar en cuenta cuando se habla de diseño de aplicaciones. No se puede ignorar que Android tiene actualmente la mayor parte de la cuota de mercado de los smartphones y que su uso se está extendiendo hacia todo tipo de dispositivos electrónicos. En poco tiempo, la plataforma Android de Google ha crecido rápidamente y las distintas marcas lo han empezado a notar.

Sin embargo, los múltiples dispositivos con Android que podemos encontrar en el mercado hacen sentir que el diseño para esta plataforma es una cuestión difícil de dominar.

Afortunadamente, Android está comenzando a abordar los problemas de tener varios dispositivos y tamaños de pantalla y los fabricantes de dispositivos están adoptando lentamente los estándares que eventualmente reducirá la complejidad en el tema de diseño.

A continuación se abordarán diferentes cuestiones que se debe tener en cuenta al momento de diseñar la interfaz de usuario de las aplicaciones.

5.9.1 Soportar múltiples pantallas

Android se ejecuta en una variedad de dispositivos que ofrecen diferentes tamaños de pantalla y densidades. Para las aplicaciones, el sistema Android ofrece un entorno de desarrollo consistente en múltiples dispositivos y la mayor parte del trabajo es para ajustar la interfaz de usuario de cada aplicación a la pantalla en la que se muestra. Al mismo tiempo, Android proporciona una API que le permite controlar la interfaz de usuario de la aplicación para tamaños de pantalla y densidades específicos, a fin de optimizar el diseño de la interfaz de usuario para configuraciones de pantalla diferentes. Por ejemplo, es posible que se desee una interfaz de usuario para las tabletas que es diferente de la interfaz de usuario para los teléfonos.

Se debe optimizar la aplicación para diferentes tamaños de pantalla y densidades, de este modo, se maximiza la experiencia del usuario para todos los dispositivos y los usuarios creen que su aplicación fue diseñada en realidad para su dispositivo, en lugar de simplemente estirarse para adaptarse a la pantalla de sus dispositivos.

Android permite crear una aplicación que se muestra adecuadamente y proporciona una experiencia de usuario optimizada en todas las configuraciones de pantalla, utilizando un solo archivo apk. [50]

Términos y conceptos

Tamaño de la pantalla: Tamaño físico real, medida de la diagonal de la pantalla. Para simplificar, Android clasifica las pantallas en cuatro tamaños generales: pequeño (small), normal, grande (large) y extra grande (extra large).

Densidad de la pantalla: La cantidad de píxeles en un área física de la pantalla, normalmente conocida como dpi (puntos por pulgada). Por ejemplo, una densidad "baja" de pantalla tiene menos píxeles dentro de un área física determinada, en comparación con la densidad de una pantalla "normal" o "alta".

Para simplificar, Android clasifica las densidades de pantalla en cuatro densidades generales: baja (LDPI), media (MDPI), alta (HDPI) y extra alta (XHDPI).

Esto es importante porque hay que realizar todos los recursos gráficos para las diferentes densidades. Por lo menos, hay que realizar un conjunto de recursos para MDPI y HDPI para cualquier aplicación Android.

Orientación: La orientación de la pantalla desde el punto de vista del usuario. Esta es horizontal o vertical, lo que significa que la relación de aspecto de la pantalla es de ancho o alto, respectivamente. Hay que tener en cuenta que no sólo los diferentes dispositivos operan en diferentes orientaciones por defecto, si no que también puede cambiar la orientación en tiempo de ejecución cuando el usuario gira el dispositivo.

Resolución: El número total de píxeles físicos sobre una pantalla. Al añadir soporte para múltiples pantallas al realizar las aplicaciones no se trabaja directamente con la resolución, sino sólo con el tamaño de pantalla y la densidad, según lo especificado anteriormente.

Píxeles independiente de la densidad (dp): Esta es una unidad de pixel virtual que se utiliza en la definición de un diseño de interfaz de usuario con el fin de expresar las dimensiones del diseño o la posición de una manera independiente de la densidad. Los píxeles independientes de la densidad equivale a un píxel físico en una pantalla de 160 DPI, que es la densidad de referencia asumida por el sistema como la densidad "media" de la pantalla. En tiempo de ejecución, el sistema maneja de manera transparente cualquier ampliación de las unidades de DP según sea necesario, en base a la densidad real de la pantalla que se esté utilizando. La conversión de las unidades de DP a los píxeles de la pantalla es simple: los píxeles = DP * (DPI / 160). Por ejemplo, en una pantalla de 240 DPI, un DP es igual a 1,5 píxeles físicos. Se debe utilizar siempre DP como unidad para definir los diseños de la interfaz de usuario para asegurarse de que se mostrará correctamente en pantallas con diferentes densidades.

Tamaños de pantallas soportadas

La línea de tiempo de las pantallas soportadas en los dispositivos Android comienza por el T-Mobile G1, el primer dispositivo Android disponible en el mercado que tiene una pantalla HVGA de 320x480 píxeles.

HVGA significa VGA de medio tamaño que es el tamaño de pantalla estándar para los smartphones de hoy en día. El iPhone 3Gs, 3G y 2G utilizan la misma configuración.

320x480 se considera un tamaño "normal" de pantalla en Android. Cuando hablemos de tamaño "extra grande" se piensa en las pantallas de las tablets. Sin embargo, hay que tomar en cuenta que los smartphones más populares de hoy en día tienen WVGA (es decir, un VGA más grande) 800x480 píxeles HD de pantalla. Por lo tanto, lo que es "normal" está cambiando rápidamente. Por ahora, vamos a tomar como referente que la mayoría de los smartphones con Android tienen pantallas grandes.

	Baja Densidad (120), ldpi	Mediana Densidad (160), mdpi	Alta Densidad (240), hdpi	Densidad Extra Alta (xhdpi)
Pantalla Pequeña	QVGA (240x320)		480x640	
Pantalla Normal	WQVGA400 (240x400) WQVGA (240x432)	HVGA (320x480)	WVGA (480x800) WVGA854 (480x854) 600x1024	640x960
Pantalla Grande	WVGA800 (480x800) WVGA854 (400x854)	WVGA800 (480x800) WVGA854 (480x854) 600x1024		
Pantalla Extra Grande	1024x600	WXGA (1280x800) 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1563 2560x1536 2560x1600

Tabla 5.1: La tabla muestra las diferentes configuraciones de pantalla disponibles en el emulador del Android SDK. [50]

La variedad de tamaños de pantalla puede ser un reto para los diseñadores que están tratando de crear un tamaño único para todos los diseños de layout. Para resolver esto, se ha encontrado que el mejor enfoque consiste en diseñar un conjunto de layouts para 320x533 píxeles físicos y luego introducir diseños personalizados para los otros tamaños de pantalla.

Cada tamaño y densidad general abarca una gama de tamaños de pantalla y densidades reales. Android hace que las diferencias que puede existir entre los dispositivos dentro de una misma categoría, sean abstractas a las aplicaciones, por lo que se puede proporcionar la interfaz de usuario diseñada para los tamaños y densidades generales y dejar que el sistema se encargue de los ajustes finales que sean necesarios. La siguiente figura 5.5 ilustra cómo diferentes tamaños y densidades se clasifican en grupos de tamaños y densidades diferentes.

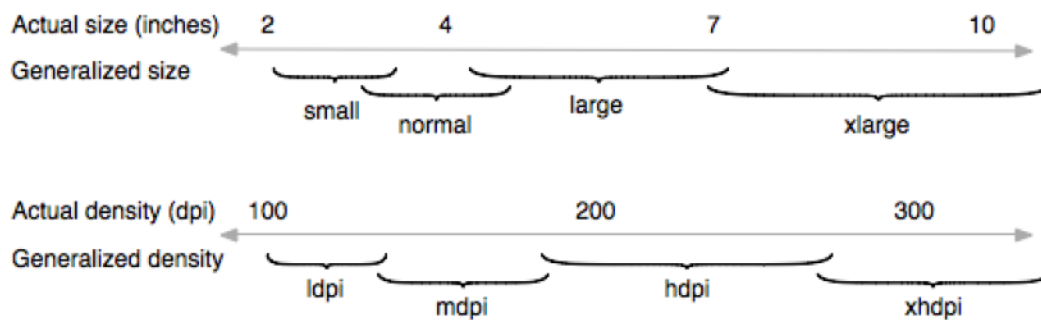


Figura 5.5: Tamaños y Densidades

A medida que se diseña la interfaz de usuario para diferentes tamaños de pantalla, se puede ver que cada diseño requiere una cantidad mínima de espacio. Así, cada tamaño general de pantalla tiene una resolución mínima asociada que está definido por el sistema. Este tamaño mínimo se encuentra en unidad DP. Esta unidad es la que se debe utilizar en la definición del diseño, ya que permite despreocuparse por los cambios en la densidad de la pantalla.

- Pantallas extra grande deben ser de por lo menos 960dp x 720dp
- Pantallas grandes deben ser de por lo menos 640dp x 480dp
- Pantallas normales deber ser de por lo menos 470dp x 320dp
- Pantallas pequeñas deber ser de por lo menos 426dp x 320dp

Para optimizar la interfaz de usuario de la aplicación para los diferentes tamaños de pantalla y densidades, se puede proporcionar recursos alternativos para cualquiera de los tamaños y

densidades generales. Por lo general, se debe proporcionar diseños alternativos para diferentes tamaños de pantalla e imágenes alternativas para diferentes densidades de pantalla. En tiempo de ejecución, el sistema utiliza los recursos adecuados para la aplicación, en función del tamaño o la densidad general de la pantalla del dispositivo actual.

No es necesario proporcionar recursos alternativos para cada combinación de tamaño y densidad de la pantalla. El sistema proporciona sólidas funciones de compatibilidad que pueden renderizar la aplicación en cualquier pantalla de dispositivo, siempre que se ha implementado la interfaz de usuario utilizando buenas prácticas de diseño.

Independencia de la densidad

La aplicación logra la "independencia de la densidad" cuando se conserva el tamaño físico (desde el punto de vista del usuario) de los elementos de la interfaz de usuario cuando se muestra en las pantallas con diferentes densidades.

Mantener la independencia de densidad es importante porque, sin ella, un elemento de interfaz de usuario (por ejemplo un botón) aparece físicamente más grande en una pantalla de baja densidad y menor en una pantalla de alta densidad. Estos cambios relacionados con la densidad de tamaño pueden causar problemas de diseño y usabilidad en la aplicación. Las siguientes figuras muestran la diferencia entre una aplicación cuando no proporciona independencia de la densidad y cuando lo hace, respectivamente.

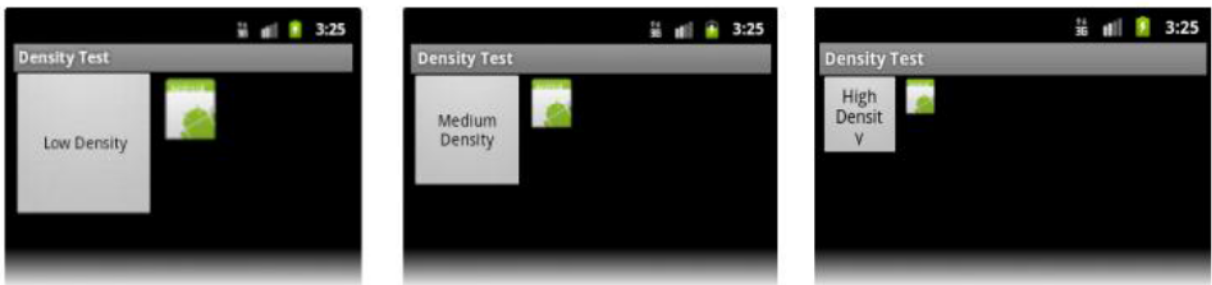


Figura 5.6: Ejemplo de aplicación sin el soporte de diferentes densidades, como se muestra en pantallas de baja, media y alta densidad.

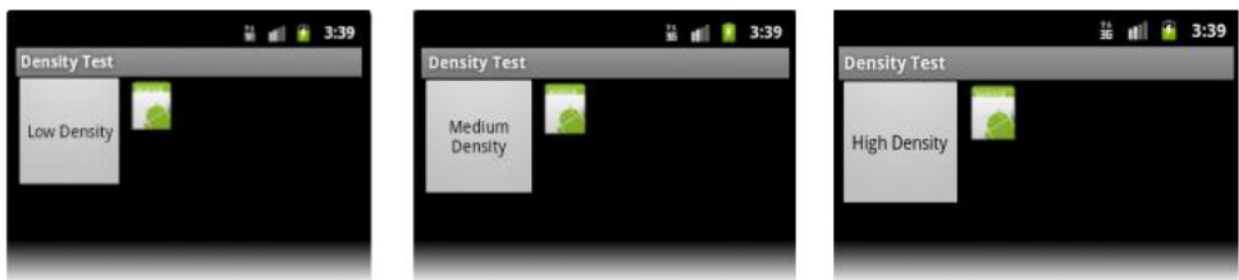


Figura 5.7: Ejemplo de aplicación con un buen soporte para diferentes densidades como se muestra en las pantallas de baja, media y alta densidad

Android ayuda a la aplicación a lograr la independencia de densidad de dos maneras:

- El sistema escala dp unidades, según corresponda para la densidad actual de la pantalla.
- El sistema escala los recursos dibujables al tamaño adecuado, basado en la densidad actual de la pantalla, si es necesario.

En la mayoría de los casos, puede garantizar la independencia de la densidad de la aplicación, simplemente especificando todos los valores de la dimensión de diseño en la densidad de píxeles independientes (dp unidades) o con "wrap_content", según corresponda. Entonces, el sistema

escala el mapa de bits dibujable con el fin de mostrar en el tamaño adecuado, basado en el factor de escala apropiado para la densidad de la pantalla actual.

Sin embargo, la escala de mapa de bits puede dar lugar a mapas de bits borrosos o pixelados, lo que se puede notar en las imágenes anteriores. Para evitar esto, se debe proporcionar recursos alternativos de mapa de bits para diferentes densidades.

Cómo soportar múltiples pantallas

La base de apoyo de Android para múltiples pantallas es su capacidad para gestionar el renderizado de layout e imágenes de la aplicación de manera apropiada a la configuración de pantalla actual. Sin embargo, también es necesario realizar:

- **Declarar explícitamente en el manifiesto los tamaños de pantalla con los que la aplicación es compatible:** Al declarar los tamaños de pantalla que la aplicación soporta, se puede asegurar que sólo los dispositivos con pantallas soportadas puede descargar la aplicación. Para declarar los tamaños de pantalla compatibles con la aplicación, se debe incluir el elemento `<supports-screens>` en el archivo de manifiesto.

- **Proporcionar diferentes diseños para diferentes tamaños de pantalla:** De forma predeterminada, Android cambia el tamaño del diseño de la aplicación para adaptarse a la pantalla del dispositivo actual. En la mayoría de los casos, esto funciona bien. En otros casos, la interfaz de usuario puede no lucir tan bien y puede ser que necesite ajustes para diferentes tamaños de pantalla. Por ejemplo, en una pantalla más grande, es posible que desee ajustar la posición y el tamaño de algunos elementos para aprovechar el espacio adicional de pantalla, o en una pantalla más pequeña, puede que tenga que ajustar el tamaño para que todo entre en la pantalla.

Los calificadores de configuración que se puede utilizar para proporcionar el tamaño de los recursos específicos son `small`, `normal`, `large` y `xlarge`.

Desde Android 3.2 (nivel de la API 13), los grupos de calificadores mencionados están en desuso y en su lugar se debe utilizar como calificador `sw<N>dp` para definir el ancho mínimo requerido por los recursos disponibles. Por ejemplo, si el diseño de una tableta requiere por lo menos 600dp del ancho de la pantalla, se debe colocar en `layout-sw600dp`.

- **Proporcionar diferentes mapas de bits para densidades de pantalla diferentes:** De forma predeterminada, Android escala los mapas de bits (archivos `.png`, `.jpg` y `.gif`) y Nine-Patch (archivos `.9.png`) para que se rendericen en el tamaño físico apropiado a cada dispositivo. Por ejemplo, si la aplicación proporciona mapa de bits sólo para la línea de base, la densidad de pantalla media (MDPI), entonces el sistema los escala hacia arriba para una pantalla de alta densidad, y los escala hacia abajo, cuando es una pantalla de baja densidad. Esta escala puede causar defectos. Para asegurarse de que los mapas de bits se vean lo mejor posible, se debe incluir versiones alternativas en diferentes resoluciones para diferentes densidades de pantalla. Los calificadores de configuración que se puede utilizar para determinadas densidad son `ldpi` (bajo), `mdpi` (medio), `hdpi` (alto), y `xhdpi` (muy alta).

En tiempo de ejecución, el sistema garantiza una visualización óptima en la pantalla para cualquier recurso dado con el siguiente procedimiento:

1) El sistema utiliza el recurso alternativo adecuado.

Basándose en el tamaño y la densidad de la pantalla actual, el sistema utiliza cualquier recurso de tamaño y densidad específica proporcionada en la aplicación. Por ejemplo, si el dispositivo tiene una pantalla de alta densidad y la aplicación requiere un recurso dibujable, el sistema busca un directorio de recursos dibujable que mejor se adapte a la configuración del dispositivo.

Dependiendo de los recursos alternativos disponibles, un directorio de recursos con el calificador hdpi (drawable-hdpi/) podría ser la mejor opción, por lo que el sistema utiliza el recurso dibujable de este directorio.

2) Si no hay recursos disponibles que se corresponda, el sistema utiliza el recurso por defecto y los escala hacia arriba o abajo según sea necesario para que coincida con el tamaño y la densidad de la pantalla actual.

El "default" de los recursos son aquellos que no están etiquetados con un calificador de configuración. Por ejemplo, los recursos en drawable/ son los recursos predeterminados que puede dibujar. El sistema asume que los recursos predeterminados están diseñados para el tamaño de la pantalla y densidad base, que es un tamaño de pantalla normal y una densidad media. Como tal, el sistema escala hacia arriba los recursos por defecto para pantallas de alta densidad y hacia abajo para pantallas de baja densidad, según corresponda.

Sin embargo, cuando el sistema está en busca de un recurso de densidad específica y no la encuentra en el directorio de la densidad específica, no siempre se utilizan los recursos predeterminados. El sistema podrá optar por un recurso de las otras densidades con el fin de proporcionar mejores resultados cuando se escala.

Uso de los calificadores de configuración

Android soporta varios calificadores de configuración que permiten controlar la forma en que el sistema selecciona los recursos alternativos basados en las características de la pantalla del dispositivo actual. Un calificador de configuración es una cadena que se puede añadir a un directorio de recursos en el proyecto Android y especifica la configuración para que los recursos en el interior se han diseñado.

Para utilizar un calificador de configuración:

1) Crear un nuevo directorio en el directorio res/ del proyecto con nombre con el formato:

`<resources_name> - <qualifier>`

`<resources_name>` es el nombre del recurso estándar (drawable o layout).

`<qualifier>` es un calificativo de configuración como por ej small, normal, ldpi, land, etc. Se debe especificar la configuración de pantalla para la que estos recursos se van a utilizar. Se puede utilizar más de un `<qualifier>` a la vez, simplemente separando cada clasificación con un guión.

2) Guardar la configuración adecuada de los recursos específicos en este nuevo directorio. Los archivos de recursos se debe llamar exactamente igual que los archivos de recursos predeterminados.

Imágenes alternativas

Casi todas las aplicaciones deben tener recursos alternativos para las diferentes densidades de pantalla, ya que casi todas las aplicaciones tiene un icono de aplicación y el icono se tiene que ver bien en todas las densidades de la pantalla. Del mismo modo, si se incluyen otras imágenes en la aplicación (por ejemplo, para los iconos de menú u otros gráficos de la aplicación), debe proporcionar versiones alternativas para diferentes densidades.

Para crear alternativas de imágenes para diferentes densidades, se debe seguir la relación de escala 3:4:6:8 entre las cuatro densidades generales. Por ejemplo, si se tiene una imagen que es de 48x48 píxeles para la pantalla de densidad media todos los diferentes tamaños deben ser:

- 36x36 para baja densidad
- 48x48 para densidad media
- 72x72 para alta densidad
- 96x96 para muy alta densidad

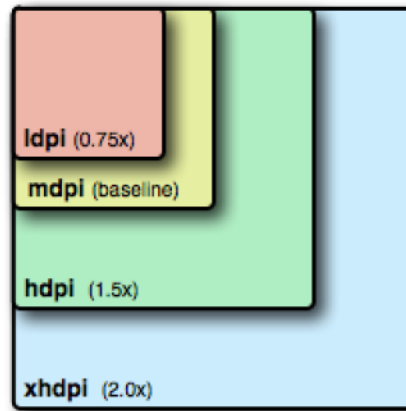


Figura 5.8: Tamaños relativos de imágenes que soportan cada densidad.

Buenas Prácticas

El objetivo de soportar múltiples pantallas es crear una aplicación que pueda funcionar correctamente y se vea bien en cualquiera de las configuraciones de pantalla generales soportadas por Android. Esta sección se va a proporcionar algunos consejos y una visión general de las técnicas que ayudan a asegurar que las aplicaciones escalan correctamente para configuraciones de pantalla diferentes.

He aquí una lista rápida de cómo se puede asegurar que la aplicación se muestre correctamente en pantallas diferentes:

1) Utilizar `wrap_content`, `fill_parent` o unidades `dp` al especificar las dimensiones en un archivo layout XML.

Al definir el `android:layout_width` y `android:layout_height` utilizando `wrap_content` y `fill_parent` o `DP` garantiza que a la vista se le da un tamaño apropiado en la pantalla del dispositivo actual.

Por ejemplo, una vista con un `layout_width = "100dp"` mide 100 píxeles de ancho en la pantalla de densidad media y escala hasta 150 píxeles de ancho en la pantalla de alta densidad, de modo que la vista ocupa aproximadamente el mismo espacio físico en la pantalla. Del mismo modo, se debe preferir `sp` (píxeles independiente de escala) para definir tamaños de texto.

2) No utilizar valores en píxeles en el código de la aplicación.

Por motivos de rendimiento y mantener el código más simple, Android utiliza píxeles como la unidad estándar para expresar los valores de dimensiones o coordenada. Esto significa que las dimensiones de una vista se expresan siempre en el código usando píxeles, pero siempre sobre la base de la densidad actual de la pantalla.

3) No utilizar `AbsoluteLayout` (es obsoleto)

4) Suministrar mapas de bits alternativos para densidades de pantalla diferentes

5.9.2 Principios de Diseño

A continuación se va a detallar principios de diseños que fueron desarrollados para mantener la experiencia de los usuarios de Android.

1. Atraer al usuario

1.1 Sorprender al usuario: Una superficie hermosa, una animación cuidadosamente colocada, o un efecto de sonido muy oportuno es una experiencia placentera. Efectos sutiles contribuyen a una sensación de falta de esfuerzo.

1.2 Utilizar objetos reales en lugar de botones y menús: Permitir que la gente manipule directamente objetos en su aplicación. Se reduce el esfuerzo cognitivo necesario para realizar una tarea.

1.3 Permitir al usuario realizar sus propias configuraciones: A la gente le encanta añadir toques personales, ya que les ayuda a sentirse como en casa y tener el control. No solo hay que proporcionar parámetros por defecto, sino también tener en cuenta personalizaciones opcionales y divertidas que no sean contrarias a las tareas primarias.

1.4 Recordar preferencias del usuario: Conocer las preferencias de los usuarios con el tiempo. En lugar de pedirle al usuario que tome las mismas decisiones una y otra vez, colocar las opciones anteriores a su alcance. Por ejemplo, al momento de realizar una búsqueda, que tenga a su alcance aquellas búsquedas que ha realizado con frecuencia.

2. Simplificar la experiencia del usuario

2.1 Ser breve: Usar frases cortas con palabras sencillas. Es probable que los usuario omita el texto si es largos. Por ejemplo, para la alertas o cuadros de confirmaciones, es importante que el texto sea claro y conciso.

2.2 Preferir las imágenes antes que las palabras: Considerar el uso de imágenes para explicar ideas. Reciben la atención de los usuarios y pueden ser mucho más eficiente que las palabras. Por ejemplo, añadir una foto a un contacto facilita la búsqueda, así como también, la identificación de la persona cuando esta llamando.

2.3 Decidir por el usuario pero permitir deshacer: Tomar la mejor respuesta y actuar en lugar de preguntar primero al usuario. Demasiadas opciones y decisiones hace al usuario infeliz. En caso de que la respuesta sea equivocada, permitir "deshacer".

2.4 Sólo mostrar lo que el usuario necesita cuando lo necesita: Los usuarios tienden a confundirse cuando ven demasiado a la vez. Por lo cual, es necesario dividir las tareas y la información en bloques pequeños y digeribles. Ocultar opciones que no son esenciales en este momento y enseñar a los usuarios a medida que avanzan.

2.5 El usuario debe saber donde está: El usuario debe estar confiado que conoce el camino. Hacer lugares en la aplicación distintivos y utilizar las transiciones para mostrar las relaciones entre las pantallas. Proporcionar comentarios sobre las tareas en curso.

2.6 No perder las cosas del usuario: Guardar lo que el usuario se tomó el trabajo de crear y permitirle que pueda acceder desde cualquier lugar.

2.7 Si tiene el mismo aspecto, la función debe ser la misma: Ayudar a los usuarios a discernir las diferencias funcionales, haciéndolos visualmente distintos en lugar de sutiles. Evitar modos que se parezcan y que actúan de manera diferente.

2.8 Sólo interrumpir al usuario si es importante: Los usuarios quieren mantener la concentración, y a menos que sea crítico y sensible al tiempo, una interrupción puede ser agotadora y frustrante.

3. Mejorar la experiencia del usuario

3.1 Dar trucos que funcionan en todas partes: Los usuarios se sienten bien cuando realizan las cosas por sí mismos. Realizar la aplicación fácil de aprender mediante el aprovechamiento de patrones visuales y la memoria muscular de otras aplicaciones de Android. Por ejemplo, el gesto de deslizar puede ser un atajo de navegación buena.

3.2 Asistir al usuario: Hay que ser amable en la forma de hacer que los usuarios hagan las correcciones. Ellos quieren sentirse inteligentes cuando utilizan su aplicación. Si algo va mal, dar instrucciones claras de recuperación, pero ahorrarles los detalles técnicos. Si se puede arreglar detrás de escena, incluso mejor.

3.3 Facilitar la tarea al usuario: Dividir las tareas complejas en pasos más pequeños que pueden llevarse a cabo fácilmente. Dar información sobre las acciones.

3.4 Realizar el trabajo pesado por el usuario: Hacer que los novatos se sienten como expertos porque les permite hacer cosas que nunca pensaron que podían.

3.5 Hacer las cosas importantes rápido: No todas las acciones son iguales. Decidir qué es lo más importante en la aplicación, que sea fácil de encontrar y rápido de usar, al igual que el botón del obturador de una cámara o el botón de pausa en el reproductor de música.

5.10 Etapa de Evaluación de la Interfaz del Usuario

5.10.1 Técnicas de evaluación de Interfaz de usuario

Las técnicas de evaluación son utilizadas para cualquier etapa de desarrollo de la Interfaz del Usuario. Además, depende del grado de acceso e identificación de los usuarios potenciales y se trabajan con muestras que deben ser representativas a la comunidad de usuarios a la que está dirigido el desarrollo.

Existe una gran variedad de métodos para evaluar la usabilidad de los sistemas software tradicionales que son utilizados en diversos contextos por personas con diferentes objetivos y necesidades.

Las técnicas pueden ser:

- Según tipo de participante:
 - Técnicas de Indagación
 - Técnicas de Inspección

- Según lugar donde se lleva a cabo:
 - Técnicas de campo
 - Técnicas de laboratorio
 - Técnicas de mercado

Las técnicas de indagación son las más utilizadas en la etapa de requerimientos y en caso de tests de mercado.

5.10.2 Nuevas tendencias en HCI

Tanto la Ingeniería del Software como la Interacción Hombre-Computadora pretenden aportar métodos y herramientas para aumentar la calidad de los productos software. Los puntos de vista ofrecidos por cada una de estas disciplinas son distintos. La Ingeniería del Software aporta procesos, métodos, notaciones y herramientas. Mientras, la HCI ofrece otros métodos e intereses que se centran en el proceso, en el producto y en la identificación de dónde, cómo y por quién es utilizado el producto software. Es decir, HCI propone un Diseño Centrado en el Usuario. [51]

Esta preocupación hacia el usuario final del producto software ha llevado a la HCI a proponer nuevas formas de interacción atrayendo a usuarios que rechazan las monótonas técnicas de interacción tradicionales, lo que ha propiciado la aparición de sistemas que permiten una interacción más natural.

Sistemas de Realidad Virtual, Realidad Aumentada y Sistemas Ubicuos son ejemplos de sistemas que proponen nuevos paradigmas de interacción.

De cara a la evaluación de la usabilidad en sistemas de Realidad Aumentada encontramos una serie de características que los diferencian de los sistemas tradicionales y que deben ser tenidos en cuenta a la hora de realizar una evaluación:

- **Características del entorno físico:** las nuevas formas de interaccionar hacen importante el entorno físico en el que se utiliza la aplicación.
- **Características del evaluador:** el número de evaluadores puede ser mayor que en el caso de la evaluación de un sistema tradicional, debido a la aparición de nuevos roles. El evaluador puede intervenir durante el test introduciendo matices y realizando comentarios sin afectar a las sensaciones del usuario, ya que en estos sistemas el usuario no está inmerso al cien por cien en el mismo.
- **Características del usuario:** que el usuario no esté familiarizado con este nuevo tipo de interfaces puede afectar negativamente a los resultados obtenidos tras una prueba de usabilidad. Por lo que se debe conseguir alcanzar un punto intermedio entre usuarios expertos y novatos.

Como cualquier producto software, un sistema de realidad aumentada, para ser útil, tiene que ser fácil de utilizar, ofrecer una interfaz atractiva (interfaz amigable) y debe permitir conseguir los objetivos de la manera más rápida posible y todo esto sin consumir una gran cantidad de recursos.

La facilidad de uso está ligada a la satisfacción del usuario, al igual que el sistema ofrezca una interfaz atractiva. Por otro lado, la consecución rápida de objetivos está ligada a la efectividad.

Además, encontramos que un consumo bajo de recursos es sinónimo de eficiencia.

Todo esto hace que podamos afirmar que la definición de usabilidad, propuesta en la ISO 9241-11, es aplicable a este tipo de sistemas.

Una vez visto esto, podemos proceder a contestar a la pregunta: ¿Son válidos los métodos de evaluación de usabilidad tradicionales para evaluar la usabilidad en Sistemas de Realidad Virtual, Realidad Aumentada y Ubicuos?

Cualquier producto software usable debe ser diseñado incorporando características y atributos para beneficiar a los usuarios en un contexto de uso determinado. Por lo que podemos pensar que si diseñamos un producto software, perteneciente a cualquiera de los tres tipos de sistemas,

incorporando características y atributos que beneficien a los usuarios de la aplicación en su contexto de uso, tendremos una aplicación usable y podremos utilizar para evaluar la usabilidad los métodos anteriormente propuestos. Pero este proceso por sí solo no asegura que un producto sea efectivo, eficiente y satisfactorio en su uso. Por lo que además de realizar un buen diseño es necesario medir el funcionamiento y la satisfacción de los usuarios trabajando con el producto. La descripción de los procesos de evaluación de aplicaciones de realidad virtual, realidad aumentada o sistemas ubicuos no se diferencia en gran medida de la evaluación de sistemas tradicionales. Sin embargo, las particularidades de estos sistemas introducen nuevos elementos que deben tenerse en cuenta.

Capítulo 6. Especificación del prototipo

En el capítulo anterior hemos hablado sobre el concepto de usabilidad. En este capítulo detallaremos principalmente los requerimientos de tanto el prototipo móvil como el web, también se describirán los casos de uso y se mostrarán los diagramas de clase.

6.1 Conceptos de la aplicación

El objetivo de este proyecto es desarrollar una aplicación para la plataforma Android que funcione como un complemento a la ayuda brindada por las guías del museo a los visitantes. Sería como una guía virtual con la cual los visitantes pueden realizar las siguientes actividades:

- Consultar información sobre un museo en particular.
- Completar una encuesta
- Consultar información relativa a un objeto del museo. Se puede acceder a la descripción del objeto, ver imágenes y videos, escuchar un audio que representa a la descripción del objeto, jugar a un juego de tipo “trivia” y a otro de tipo “unir con flechas”.

La arquitectura de la aplicación es distribuida. Básicamente el cliente móvil recupera la información del servidor web, excepto en el caso de la encuesta, que se envía el resultado. Dentro del modelo de datos podemos identificar varias entidades para representar los objetos del museo y sus materiales, para las encuestas, museos, categorías, etc.

La realidad aumentada está implementada mediante un lector de códigos QR que es utilizado para identificar el QR asociado a cada objeto y para luego traer toda la información relativa a dicho objeto.

La decisión del método de realidad aumentada elegido fue tomada a partir de las siguientes restricciones:

Los objetos dentro del museo se encuentran dentro de una vitrina con poca iluminación. Además, se encuentran superpuestos unos con otros y los mismos se rotan constantemente con otros elementos que se encuentran en un depósito, cambiando de lugar de exposición.

Es por este motivo que resultaba difícil aplicar RA basada en características naturales o basadas en dispositivos físicos tales como el GPS, ya que además nos encontramos en un ambiente “indoor”.

Es por ello, que elegimos RA basada en marcadores. Utilizamos el marcador QR, debido a que la información se genera dinámicamente y teníamos que encontrar un código que nos permita identificar fácilmente cada uno de los objetos que se exponen dentro del museo y que nos provea todas las herramientas necesarias para poder crearlos y leerlos fácilmente.

6.2 Requerimientos del cliente móvil

Funcionales:

D	Nombre	Descripción
RFM_01	Consultar objeto	Se debe recuperar y mostrar la información relativa a un objeto. Esta incluye imágenes, videos, texto y audio.

RFM_02	Consultar información de categoría	Se debe mostrar información sobre la categoría a la cual pertenece el objeto
RFM_03	Consultar información del museo y ver mapa	Se debe mostrar la descripción del museo y la imagen del mapa
RFM_04	Consultar sugerencia de objetos	Se debe poder recibir una sugerencia de algunos objetos de otras categorías para que el visitante explore.
RFM_05	Recuperar y enviar encuesta	Se debe completar la encuesta y enviar el resultado
RFM_06	Recuperar trivia	Se debe mostrar la trivia
RFM_07	Recuperar unir con flechas	Se debe mostrar el juego unir con flechas
RFM_08	Consultar la lista de museos	Se debe poder mostrar en pantalla la lista inicial de museos

No funcionales:

ID	Nombre	Descripción
RNFM_01	Tiempo de respuesta	El sistema debe optimizar el tiempo de respuesta de conexión para el intercambio de datos con el servidor
RNFM_02	Aspectos de Interfaz gráfica	La interfaz grafica debe ser amigable, intuitiva y fácil de usar.
RNFM_03	Tolerancia a fallos	El sistema debe poder recuperarse ante un error para evitar el cierre forzoso del mismo.

6.3 Requerimientos del administrador web

Funcionales:

ID	Nombre	Descripción
RFW_01	Administrar objetos	El sistema debe permitir consultar, crear, modificar y eliminar objetos
RFW_02	Administrar materiales	El sistema debe permitir consultar, crear, modificar y eliminar materiales
RFW_03	Administrar usuarios	El sistema debe permitir consultar, crear, modificar y eliminar usuarios
RFW_04	Administrar museos	El sistema debe permitir consultar, crear, modificar y eliminar museos
RFW_05	Administrar categorías	El sistema debe permitir consultar, crear, modificar y eliminar categorías
RFW_06	Administrar encuesta	El sistema debe permitir consultar, crear, modificar y eliminar la encuesta definida para cada museo
RFW_07	Administrar juego	El sistema debe permitir consultar, crear, modificar y eliminar el juego definido para cada objeto
RFW_08	Administrar información de un museo	El sistema debe permitir ingresar, modificar y eliminar la información de un museo y poder elegir un mapa y un cono representativo.
RFW_09	Modificar contraseña	El sistema debe permitir la modificación de la contraseña de un usuario.
RFW_10	Iniciar sesión en el sistema	El sistema debe autorizar y autenticar el acceso de un usuario a la aplicación

No funcionales:

ID	Nombre	Descripción
RNFW_01	Interfaz simple	Debe contar con una interfaz simple que no requiera un alto grado de capacitación para su utilización
RNFW_02	Capacidad de respuesta	El sistema debe responder de manera rápida a los requerimientos tanto desde el administrador web como del cliente móvil.

6.4 Casos de uso

Los casos de uso sirven para describir en forma natural la funcionalidad de un sistema. Es utilizado para realizar la especificación de requisitos del sistema. Permiten capturar y definir los requisitos que deben definir una aplicación.

6.4.1 Casos de uso de la aplicación móvil

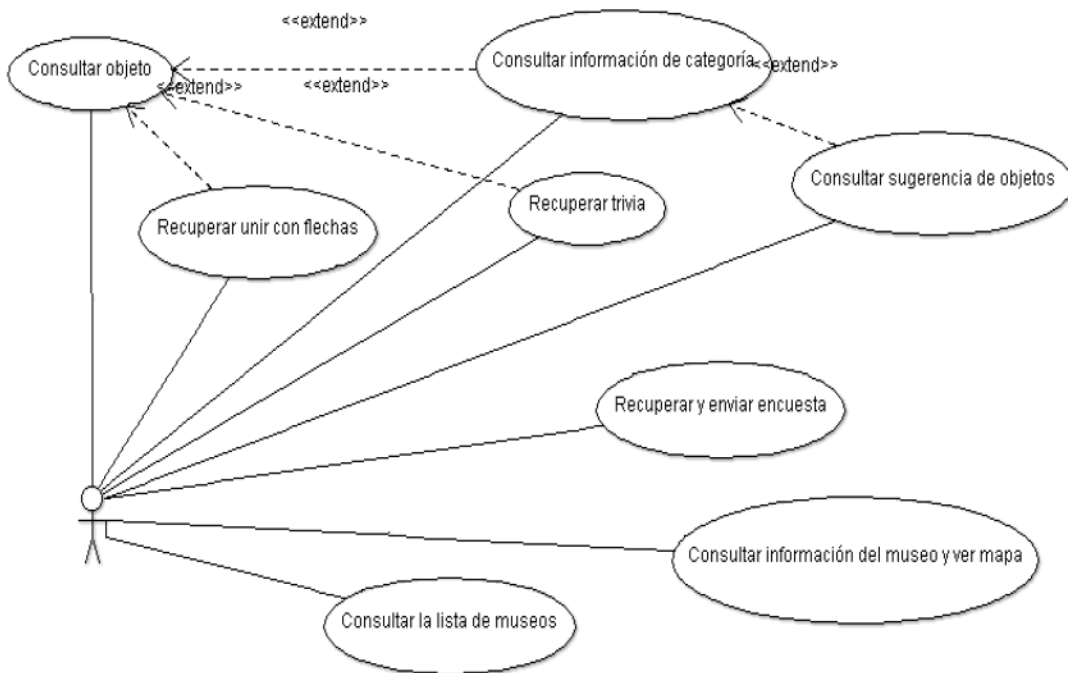


Figura 6.1: Casos de uso de la aplicación móvil

6.4.2 Casos de uso del administrador web

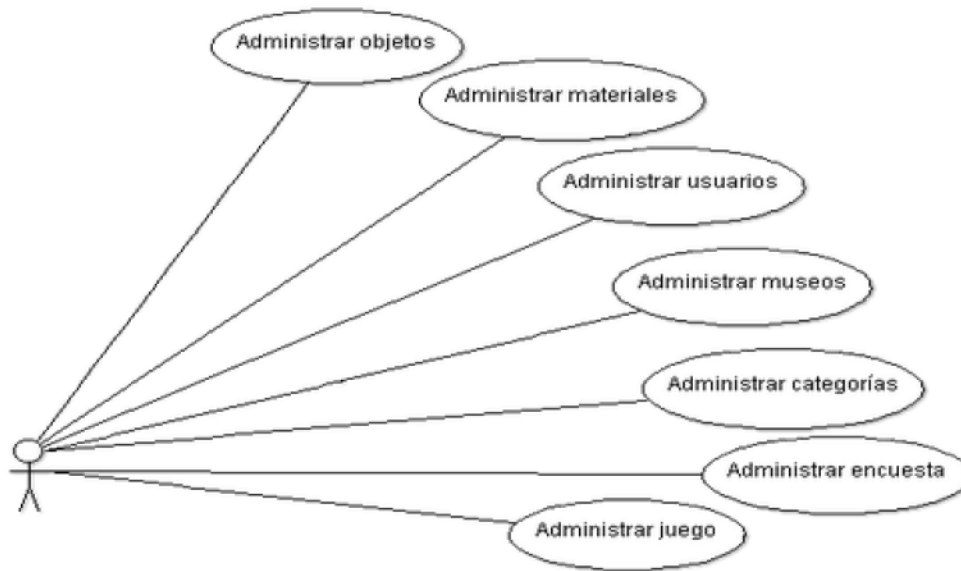


Figura 6.1: Casos de uso de la aplicación móvil

6.4.3 Descripción de los casos de uso de la aplicación móvil

Nombre	Consultar objeto
Req. que satisface	RFM_01
Actor	Usuario
Descripción	A partir de la lectura del código QR la aplicación debe mostrar en distintas solapas diversa información: una solapa es para mostrar un listado de imágenes de ese objeto; otra es para mostrar una lista de videos de ese objeto; otra es para mostrar un texto descriptivo sobre el objeto; otra es para mostrar un audio que representa la descripción del objeto hablada; y por último, la solapa que muestra una opción de juegos.
Precondiciones	El id de objeto existe en la base de datos

Nombre	Consultar información de categoría
Req. que satisface	RFM_02
Actor	Usuario
Descripción	El usuario puede acceder a la información sobre la categoría a la cual pertenece el objeto que decidió explorar.
Precondiciones	El objeto pertenece a una categoría

Nombre	Consultar información del museo y ver mapa
Req. que satisface	RFM_03
Actor	Usuario
Descripción	Desde la pantalla principal el usuario debe poder acceder a la información sobre el museo y ver un mapa representativo sobre este.

Precondiciones	
Nombre	Consultar sugerencia de objetos
Req. que satisface	RFM_04
Actor	Usuario
Descripción	A partir de la lectura de la información sobre la categoría a la que pertenece el objeto explorado (ver req. RFM_2) el usuario debe poder obtener algunas sugerencias de otros objetos que existan en el museo. Es a modo descriptivo, ya que el usuario deberá buscar ese objeto físicamente y luego explorarlo para acceder a su información.
Precondiciones	

Nombre	Recuperar y enviar encuesta
Req. que satisface	RFM_05
Actor	Usuario
Descripción	Desde la pantalla principal el usuario debe poder completar la encuesta del museo y enviar los resultados a modo de feedback. Para ello primero la aplicación móvil debe recuperar el modelo de encuesta desde el servidor web.
Precondiciones	

Nombre	Recuperar trivía
Req. que satisface	RFM_06
Actor	Usuario
Descripción	El usuario debe poder jugar una "trivía". Debe responder las preguntas y obtener un puntaje por ello. El modelo de la trivía debe ser recuperado del servidor. El juego corresponde a un objeto en particular y se debe encontrar en la solapa de "juegos" del objeto explorado.
Precondiciones	

Nombre	Recuperar unir con flechas
Req. que satisface	RFM_07
Actor	Usuario
Descripción	El usuario debe poder jugar a un juego del tipo unir con flechas, en el cual se unen conceptos relacionados a un objeto. Cuando todas los ítems se han correspondido se le avisa al usuario. El juego corresponde a un objeto en particular y se debe encontrar en la solapa de "juegos" del objeto explorado.
Precondiciones	

Nombre	Consultar la lista de museos
Req. que satisface	RFM_08
Actor	Usuario
Descripción	Cuando se inicia la aplicación móvil se debe mostrar al usuario una lista de museos disponibles.
Precondiciones	

6.4.4 Descripción de los casos de uso del administrador web

Nombre	Administrar objetos
Req. que satisface	RFW_01

Actor	Usuario con rol de carga de datos
Descripción	El usuario debe poder crear, modificar y eliminar objetos. Para cada objeto se debe poder determinar a qué categoría pertenece. Solo se permite la eliminación física si el objeto no posee otros objetos asociados.
Precondiciones	El usuario está autenticado

Nombre	Administrar materiales
Req. que satisface	RFW_02
Actor	Usuario con rol de carga de datos
Descripción	El usuario debe poder crear, modificar y eliminar para los distintos objetos, materiales de los siguientes tipos: imágenes, video, audio y texto. Solo se permite un archivo de audio y un archivo de texto.
Precondiciones	El usuario está autenticado

Nombre	Administrar usuarios
Req. que satisface	RFW_03
Actor	Usuario con rol Administrador
Descripción	El usuario debe poder crear, modificar y eliminar un usuario con rol carga de datos y elegir el museo al cual pertenece. La eliminación es lógica, se debe marcar como inactivo.
Precondiciones	El usuario está autenticado. El usuario que se quiere ingresar no existe en la base de datos.

Nombre	Administrar museos
Req. que satisface	RFW_04
Actor	Usuario con rol Administrador
Descripción	El usuario debe poder crear, modificar y eliminar un museo con su descripción
Precondiciones	

Nombre	Administrar categorías
Req. que satisface	RFW_05
Actor	Usuario con rol de carga de datos
Descripción	El usuario debe poder crear, modificar y eliminar categorías. Solo se me permite la eliminación física si la categoría no posee objetos asociados.
Precondiciones	

Nombre	Administrar encuesta
Req. que satisface	RFW_06
Actor	Usuario con rol de carga de datos
Descripción	El usuario debe poder crear un modelo de encuesta, modificar sus items o eliminarlos también.
Precondiciones	

Nombre	Administrar juego
Req. que satisface	RFW_07
Actor	Usuario con rol de carga de datos
Descripción	El usuario debe para cada objeto poder crear un modelo de juego de tipo trivia y modificar o eliminar sus ítems, debe poder crear también un juego del tipo unir con flechas con 4 pareja de ítems relacionados. Se deben así también poder modificar y eliminar esos ítems.
Precondiciones	

Nombre	Administrar información de un museo
Req. que satisface	RFW_08
Actor	Usuario con rol de carga de datos
Descripción	El usuario debe poder crear, modificar y eliminar la información relativa a un museo. Esta incluye la descripción y la imagen del mapa y el icono representativo.
Precondiciones	

Nombre	Modificar contraseña
Req. que satisface	RFW_09
Actor	Usuario con rol de carga de datos
Descripción	El usuario debe poder modificar su contraseña
Precondiciones	

Nombre	Iniciar sesión en el sistema
Req. que satisface	RFW_10
Actor	Usuario con rol de carga de datos
Descripción	El usuario debe poder ingresar al sistema. El sistema debe validar que el usuario existe y que la clave es correcta
Precondiciones	

6.5 Arquitectura del sistema

La arquitectura del sistema es distribuida. El cliente móvil funciona sobre dispositivos Android que posean la versión 2.2 en adelante. Todo el intercambio de información entre la aplicación móvil y el servidor se realiza utilizando el protocolo HTTP y la tecnología Json [52]. Para realizar este intercambio se debe disponer de una conexión a Internet. Si bien es mejor que el teléfono donde se ejecuta el cliente móvil esté conectado a internet, el rendimiento bajo la red de tecnología móvil 3G es bastante rápido.

El servidor siempre está a la espera de peticiones por parte del cliente móvil a las cuales satisface con respuesta que brindan información al usuario. El servidor está implementado utilizando la versión 8.4 de PostgreSQL [53].

6.6 Diagrama de clases

Se muestra a continuación el diagrama de clases del modelo, el cual representa la solución que se encontró para resolver el dominio del problema.

Para simplificar el diagrama se evito poner la clase EntidadBase y la cual solo sirve para dar el atributo de nombre "id" necesario para la persistencia mediante *Hibernate*. Todas las clases que son raíces de jerarquías y las que no forman parte de ninguna jerarquía extienden de EntidadBase excepto la clase QRData.

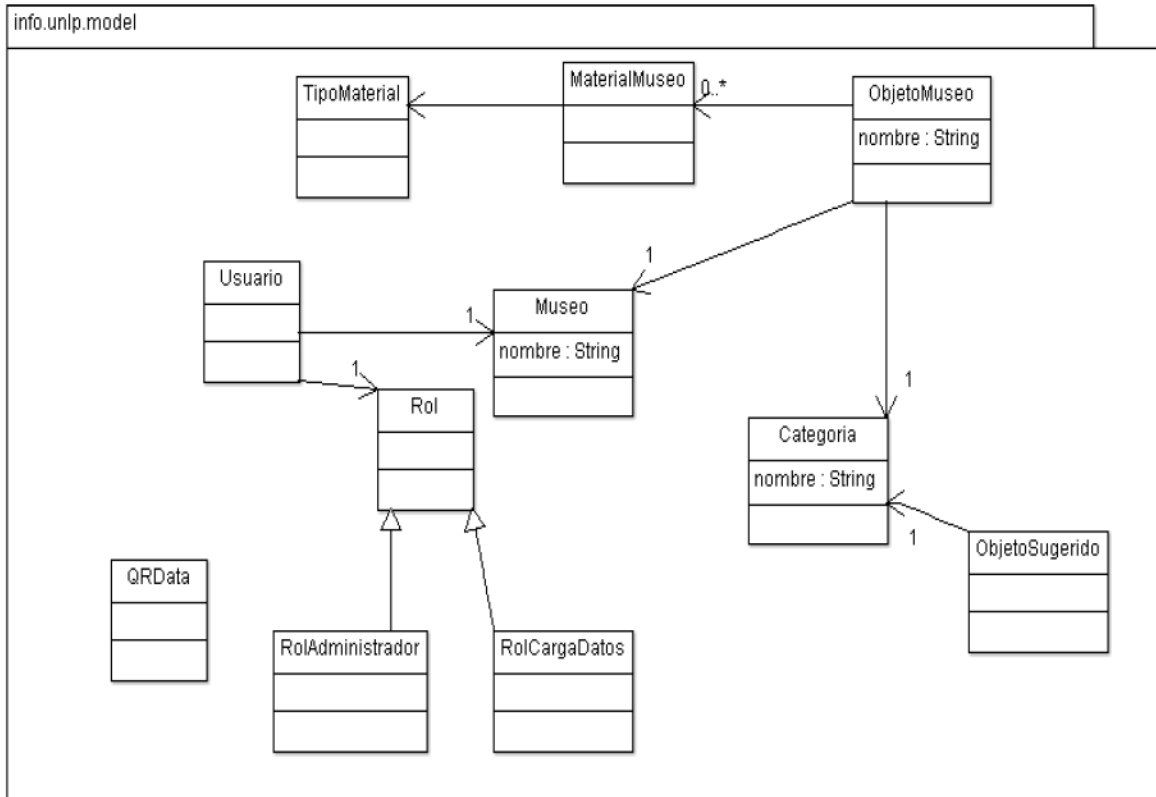


Figura 6.2: Diagrama de clases info.unlp.model

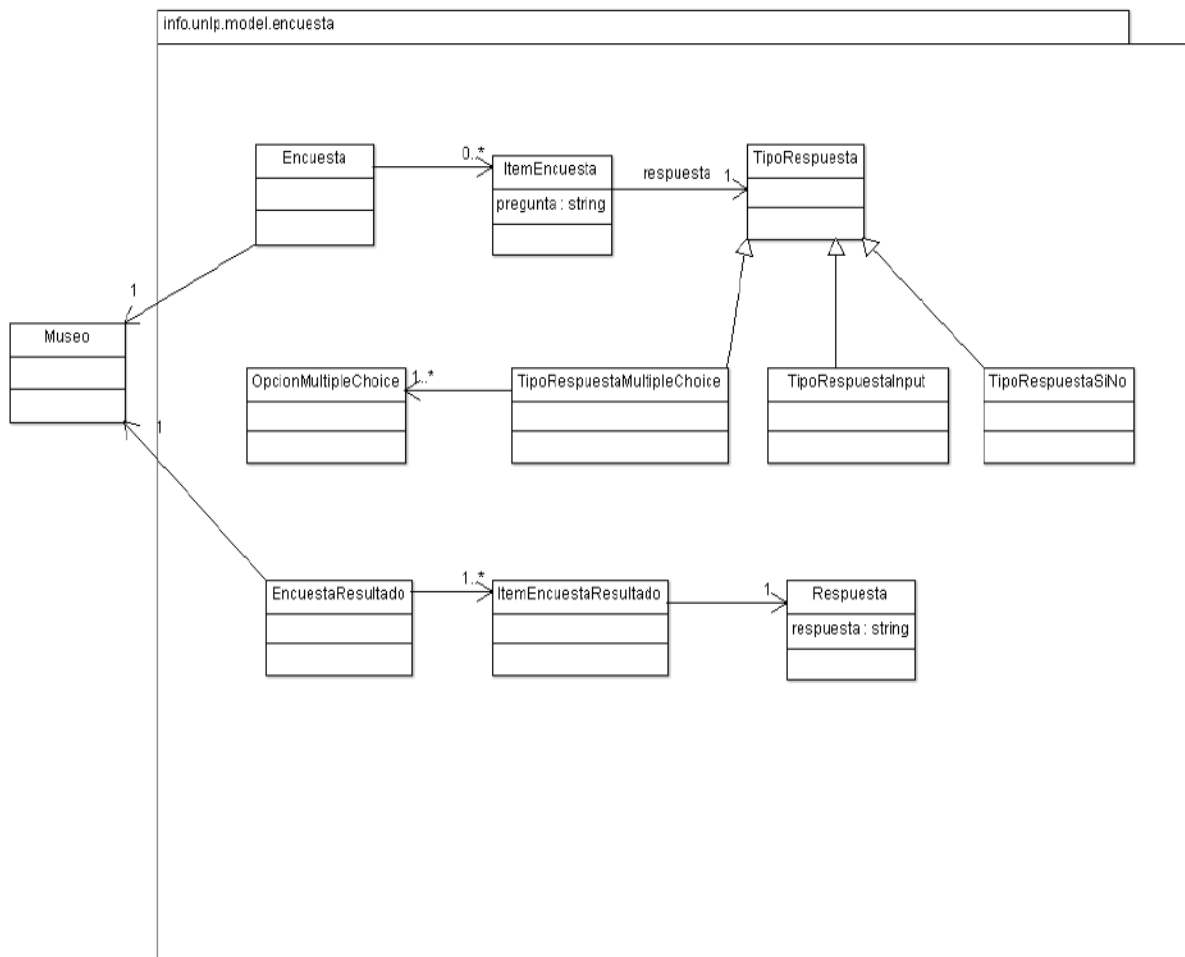


Figura 6.3: Diagrama de clases `info.unlp.model.encuesta`

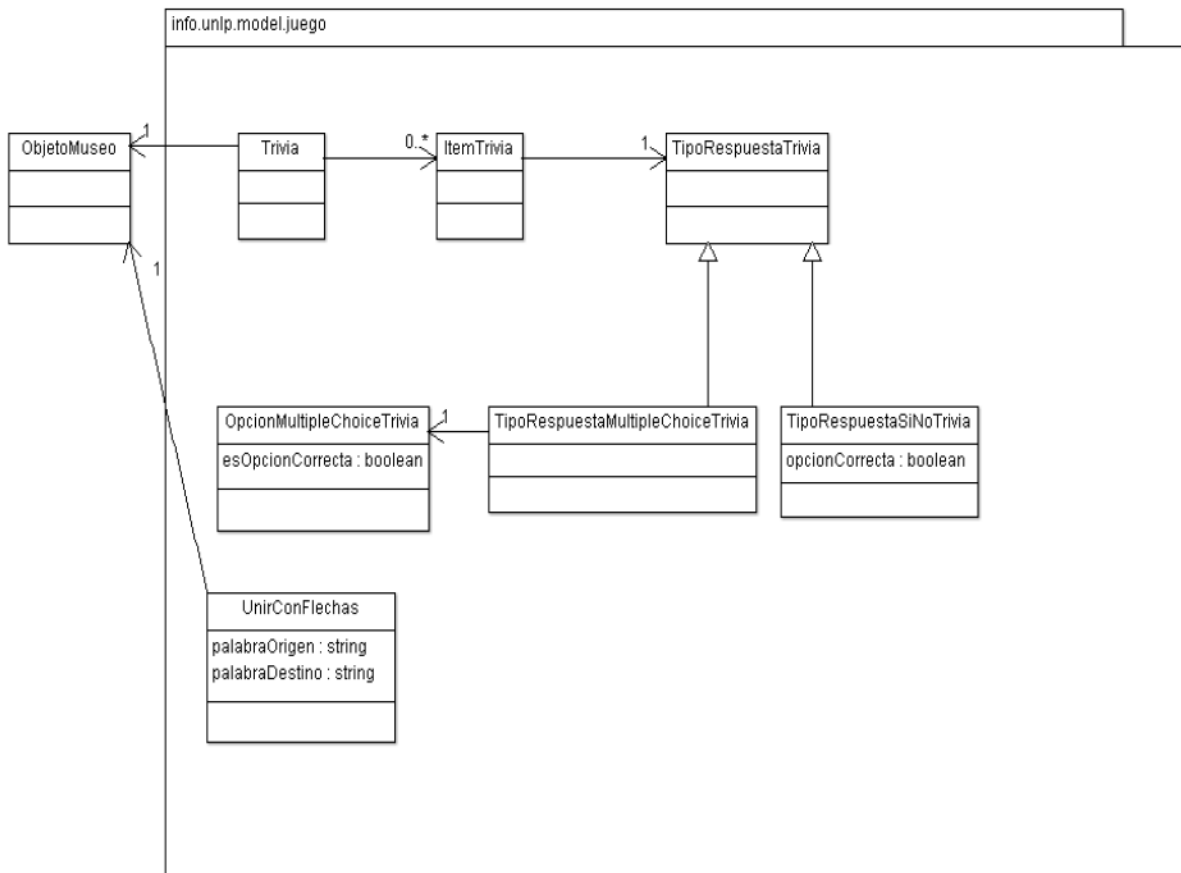


Figura 6.4: Diagrama de clases info.unlp.model.juego

6.6.1 Descripción de las clases

De info.unlp.model:

Museo: Sirve para representar los distintos museos que pueden existir en la aplicación.

ObjetoMuseo: Esta clase representa a los distintos objetos que puede haber en el museo.

MaterialMuseo: Cada objeto del museo posee una colección de elementos. La clase MaterialMuseo sirve para representar cada uno de esos elementos. Cada elemento puede tener un tipo que puede ser una imagen, video, audio o texto.

TipoMaterial: Representar el tipo que puede tener un material. Puede ser audio, imagen, texto o video.

Categoría: Representa la categoría a la cual pertenece un objeto del museo. Los objetos están agrupados físicamente en vitrinas que corresponden a una categoría específica como puede ser óptica o electromagnetismo por ejemplo.

Rol: Representa un rol que puede poseer un usuario.

RolAdministrador: Representa el rol administrador y con el cual se puede dar de alta museos y usuarios.

RolCargaDeDatos: Representa el rol de carga de datos con el cual se pueden hacer la mayoría de las operaciones del sistema como crear objetos, agregar materiales, crear categorías, etc.

Usuario: Representa un usuario del sistema y esta vinculado a un museo específico y posee un rol determinado.

ObjetoSugerido: Es utilizado para cuando desde la aplicación móvil se quieren obtener sugerencias de otros objetos a explorar.

QRData: Representa la información que contiene un código QR. Básicamente posee el id del objeto en la base de datos. Esta clase es transformada a formato Json y el string resultante es el que representa el código QR.

De info.unlp.model.encuesta:

Encuesta: Representa la encuesta del museo. Posee una lista de ItemEncuesta.

ItemEncuesta: Representa cada ítem de la encuesta. Posee una pregunta y un tipo de respuesta.

TipoRespuesta: Es la base de la jerarquía. Los tipos de respuesta pueden ser de tipo input o para ingresar un texto, de tipo multiple choice, o sea, con una lista de opciones posibles, o de tipo Sí/No.

TipoRespuestaInput: Representa el tipo de respuesta en la cual el usuario puede ingresar un valor textual.

TipoRespuestaMultipleChoice: Representa el tipo en el cual el usuario puede elegir entre varias opciones posibles.

TipoRespuestaSiNo: Representa el tipo en el cual el usuario puede elegir por Sí o por No.

OpcionMultipleChoice: Cada tipo de respuesta múltiple choice posee una lista de elementos de esta clase. Como atributo posee el valor de la opción.

EncuestaResultado: Representa la encuesta respondida por el usuario. Posee una lista de ítems de preguntas y respuestas.

ItemEncuestaResultado: Representa un ítem de la encuesta resultado. Posee la pregunta que se respondió y la respuesta.

Respuesta: Representa la respuesta de cada ítem de la encuesta resultado. Como atributo posee un string que es la representación literal de la respuesta del usuario. Por ejemplo, si el usuario respondió Sí, la respuesta dirá "Sí"; si respondió con un valor de tipo input contendrá ese valor y si respondió con varias opciones de multiple choice será un string que concatene los valores de esas opciones.

De info.unlp.model.juego:

Trivia: Representa la trivia del objeto. Posee una lista de ItemTrivia.

ItemTrivia: Representa cada ítem de la trivia. Posee una pregunta y un tipo de respuesta.

TipoRespuestaTrivia: Es la base de la jerarquía. Los tipos de respuesta pueden ser de tipo multiple choice, es decir, con una lista de opciones posibles, o de tipo Sí/No.

TipoRespuestaMultipleChoiceTrivia: Representa el tipo en el cual el usuario puede elegir entre varias opciones posibles.

TipoRespuestaSiNoTrivia: Representa el tipo en el cual el usuario puede elegir por Sí o por No.

OpcionMultipleChoiceTrivia: Cada tipo de respuesta múltiple choice posee una lista de elementos de esta clase. Como atributo posee el valor de la opción.

UnirConFlechas: Representa un ítem del juego de unir con flechas y posee por lo tanto una palabra de origen y otra de destino a la cual está vinculada la primera.

Capítulo 7. Sistema web

En el capítulo anterior hemos hablado sobre requerimientos y casos de uso. En este capítulo explicaremos la arquitectura del prototipo web, los diferentes framework utilizados y ahondaremos en cuestiones técnicas de implementación.

7.1 Arquitectura

La aplicación web constituye la implementación servidor de la aplicación. Es un proyecto Java que sigue el patrón de diseño MVC (modelo - vista- controlador), utilizando como frameworks para su desarrollo:

- Spring [54]
- Spring MVC
- Spring Security
- Hibernate [55]
- jQuery [56]

Para el desarrollo de la parte servidor y web se utilizó Spring MVC. Jsp para la presentación, JQuery como librería para el manejo de funciones JavaScript, Hibernate como Framework para la persistencia de objetos, Spring para la inyección de dependencias y Spring security para la seguridad de la aplicación.

La arquitectura consta de 4 capas:

- la capa de presentación
- la capa de controladores
- la capa de servicios o lógica de negocios
- la capa de acceso a datos o DAO

7.2 Framework Spring

Spring [54] es un framework de código abierto creado para hacer frente a la complejidad del desarrollo de aplicaciones empresariales. Spring hace posible utilizar JavaBeans para lograr cosas que antes sólo eran posible con EJBs. Sin embargo, la utilidad de Spring no se limita al desarrollo del lado del servidor; cualquier aplicación Java puede beneficiarse de Spring en términos de simplicidad, la capacidad de testing y el bajo nivel de acoplamiento. Spring posee dos características importantes que son: la inyección de dependencias y la programación orientada a aspectos.

7.2.1 Inyección de dependencias

También conocido por su nombre más genérico inversión de control o IOC [57], es un patrón de creación de objetos. Es decir que es un patrón que dice algo acerca de cómo un objeto es creado.

Si tenemos dos clases que dependen una de la otra hay varias formas de establecer ese tipo de dependencia. La forma más simple es hacer que la clase A que depende de B instancie B. Pero eso acopla mucho las clases entre sí. Hay otro patrón muy popular usado frecuentemente en J2EE llamado "service locator" que básicamente establece que la clase A busca la clase B en un service locator para su utilización. Esto favorece un acoplamiento menos fuerte que el método anterior, pero este sigue existiendo.

Inyección de dependencia en cambio, hace que la clase que necesita algo no lo cree, ni lo busque en algún lugar, sino que lo obtenga por inyección, mediante un constructor o un setter. Podríamos definir por consiguiente a la inyección de dependencia como un término utilizado para describir el desacoplamiento entre la implementación de un objeto y la construcción de otro objeto del cual depende.

7.2.2 AOP

Aspect-Oriented Programming (AOP) [58] ó programación orientada a aspectos es un paradigma de programación que nos permite separar “aspectos” o “intenciones” de nuestro software en diferentes módulos según su responsabilidad. De esta manera puede verse como otra herramienta para poder diferenciar en nuestro código las diferentes necesidades, encapsulando los diferentes conceptos como son los requerimientos funcionales, seguridad, profiling de tiempos, transaccionabilidad contra una base de datos, logging, entre otros.

Algunos conceptos importantes de AOP son:

- Aspect (Aspecto) es una funcionalidad transversal (cross-cutting) que se va a implementar de forma modular y separada del resto del sistema. El ejemplo más común y simple de un aspecto es el logging (registro de sucesos) dentro del sistema, ya que necesariamente afecta a todas las partes del sistema que generan un suceso.
- Join point (Punto de Cruce o de Unión) es un punto de ejecución dentro del sistema donde un aspecto puede ser conectado, como una llamada a un método, el lanzamiento de una excepción o la modificación de un campo. El código del aspecto será insertado en el flujo de ejecución de la aplicación para añadir su funcionalidad.
- Advice (Consejo) es la implementación del aspecto, es decir, contiene el código que implementa la nueva funcionalidad. Se insertan en la aplicación en los Puntos de Cruce.
- Pointcut (Puntos de Corte) define los Consejos que se aplicarán a cada Punto de Cruce. Se especifica mediante Expresiones Regulares o mediante patrones de nombres (de clases, métodos o campos), e incluso dinámicamente en tiempo de ejecución según el valor de ciertos parámetros.
- Introduction (Introducción) permite añadir métodos o atributos a clases ya existentes. Un ejemplo en el que resultaría útil es la creación de un Consejo de Auditoría que mantenga la fecha de la última modificación de un objeto, mediante una variable y un método `setUltimaModificacion(fecha)`, que podrían ser introducidos en todas las clases (o sólo en algunas) para proporcionarlas esta nueva funcionalidad.
- Target (Destinatario) es la clase aconsejada, la clase que es objeto de un consejo. Sin AOP, esta clase debería contener su lógica, además de la lógica del aspecto.
- Proxy (Resultante) es el objeto creado después de aplicar el Consejo al Objeto Destinatario. El resto de la aplicación únicamente tendrá que soportar al Objeto Destinatario (pre-AOP) y no al Objeto Resultante (post-AOP).

7.3 Spring MVC

Spring MVC [59] ayuda en la construcción de aplicaciones flexibles y débilmente acopladas. El patrón Model-view-controller ayuda en la separación de la lógica de negocios, la lógica de presentación y la lógica de navegación. El modelo es responsable de encapsular los datos de la aplicación. La vista renderiza la respuesta al usuario con la ayuda del objeto del modelo. Los

controladores son responsables de recibir el requerimiento HTTP disparado por el usuario y llamar a los servicios back-end.

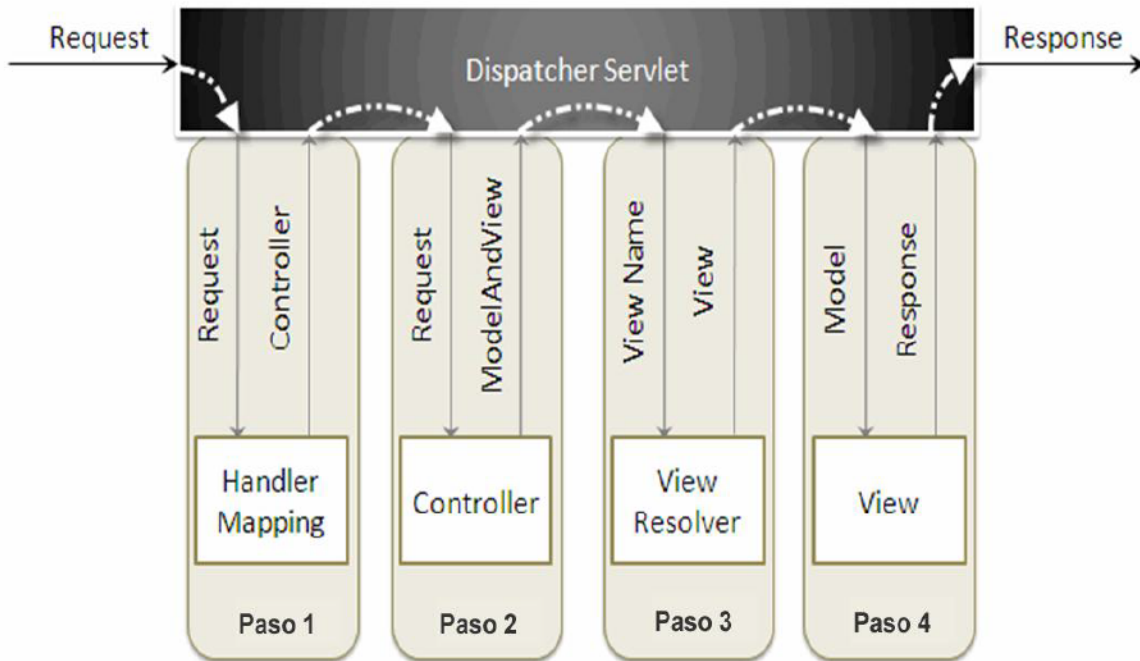


Figura 7.1: Dispatcher Servlet

El framework Spring MVC está diseñado alrededor de un *DispatcherServlet*. Cuando un requerimiento se envía al framework Spring MVC la siguiente secuencia de eventos se suceden:

- 1 El *DispatcherServlet* es el primero en recibir el requerimiento
- 2 El *DispatcherServlet* consulta el *HandlerMapping* e invoca el Controller asociado al requerimiento
- 3 El Controller procesa el requerimiento llamando al método del servicio apropiado y retorna un objeto *ModelAndView* al *DispatcherServlet*. El objeto *ModelAndView* contiene los datos del modelo y el nombre de la vista.
- 4 El *DispatcherServlet* envía el nombre de la vista al *ViewResolver* para encontrar la vista actual a invocar.
- 5 El *DispatcherServlet* ahora pasa el objeto del modelo a la vista para renderizar el resultado.
- 6 La vista, con la ayuda de los datos del modelo renderiza el resultado de vuelta al usuario.

7.4 Spring Security

Spring Security [60] es un poderoso y altamente customizable framework de autenticación y de control de acceso o autorización. Es el estándar de facto para brindar seguridad a las aplicaciones basadas en Spring.

Como es sabido, dos áreas importantes de aplicación de la seguridad son la “autenticación” y la “autorización” (o control de acceso). Estos son los targets principales de Spring security.

Autenticación es el proceso de establecer que una entidad (como puede ser un usuario, dispositivo u otro sistema que intenta realizar una acción en la aplicación) es quien dice ser.

Autorización se refiere al proceso de decidir si a esa entidad se le es permitido realizar cierta acción dentro de la aplicación. Para llegar al punto donde una decisión de autorización es necesaria, la identidad de la entidad ha sido ya establecida por el proceso de autenticación.

7.5 Hibernate

Hibernate [55] es una herramienta de mapeo objeto/relacional (ORM) para ambientes Java. Es un framework de persistencia, basado en objetos, para bases de datos relacionales. Surge como la necesidad de persistir objetos Java en bases de datos relacionales.

7.6 JQuery

JQuery [56] es una librería de JavaScript para acceder a los objetos del DOM de un modo simplificado. La librería jQuery en resumen nos aporta las siguientes ventajas:

- Nos ahorra muchas líneas de código.
- Nos hace transparente el soporte de nuestra aplicación para los navegadores principales.
- Nos provee de un mecanismo para la captura de eventos.
- Provee un conjunto de funciones para animar el contenido de la página en forma muy sencilla.
- Integra funcionalidades para trabajar con AJAX.

7.7 Creación del proyecto

Desde el entorno de desarrollo Eclipse creamos un nuevo proyecto web dinámico. La estructura del mismo es la siguiente:

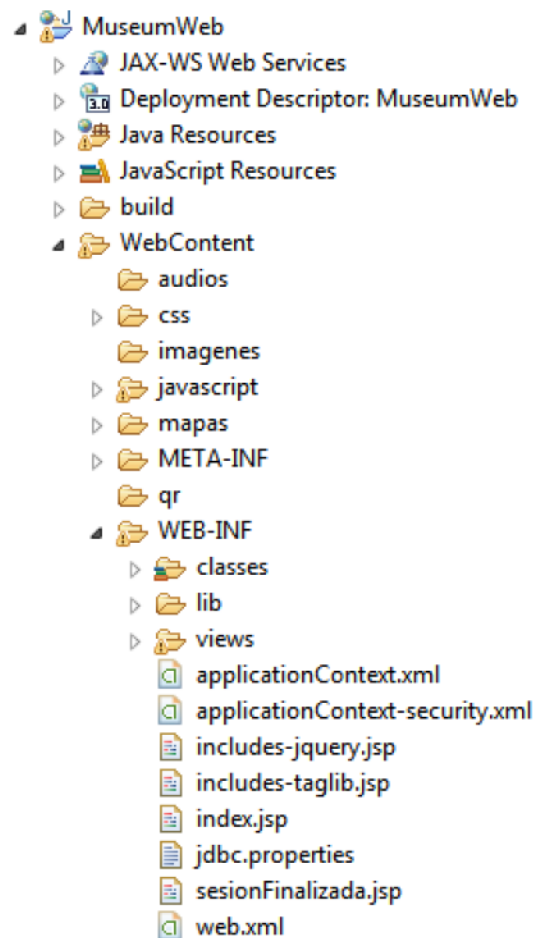


Figura 7.2: Estructura del proyecto web

- **JAX-WS WEB SERVICES:** contiene definiciones de servicios web.
- **Deployment descriptor:** detalla el contenido del descriptor de deployment, que es el archivo web.xml
- **Java resources:** se encuentran los archivos java y archivos de mapeo.
- **JavaScript resources:** se muestran todas las funciones JavaScript definidas en el proyecto.
- **WebContent:** se encuentran todos los recursos web de la aplicación. En la carpeta */audios* se guardan todos los audios que se suben a la aplicación. En la carpeta */imagenes* se guardan todas las imágenes de objetos que son subidas a la aplicación. En */mapas* se guardan los mapas e iconos de los museos. En */qr* se guardan los QR generados por la aplicación.
- Dentro de **WEB-INF** se encuentra el directorio *classes* donde se guardan todos los archivos .class generados por el compilador JAVA; en el directorio *lib* se guardan todas las librerías utilizadas por la aplicación como las propias de Spring e Hibernate, entre otras. En */views* se guardan todos los JSP. En */javascript* se guardan todos los archivos .js propios del lenguaje javascript. En */css* se guardan todos los archivos de estilos .css e imágenes relacionados también con el estilo propio de la aplicación.

7.8 Creación de las clases del modelo

Para ejemplificar se muestra la estructura de una clase del modelo. Al principio del archivo se ve la declaración package que establece el paquete en el que se encuentra esta clase. Luego vienen los imports de otras clases y después la definición propia de la clase.

ObjetoMuseo.java:

```
1 package info.unlp.museum.model;
2
3+ import info.unlp.museum.framework.dao.EntidadBase;
7
8 @SuppressWarnings("serial")
9 public class ObjetoMuseo extends EntidadBase {
10
11     private String nombre;
12
13     private Categoria categoria;
14
15     private List<MaterialMuseo> materiales;
16
17     private Museo museo;
18
19- public ObjetoMuseo() {
20     materiales = new ArrayList<MaterialMuseo>();
21 }
22
23
24- public List<MaterialMuseo> getMateriales() {
25     return materiales;
26 }
27
28- public void setMateriales(List<MaterialMuseo> materiales) {
29     this.materiales = materiales;
30 }
31
```

7.9 Creación del controlador

Hemos dicho ya que Spring MVC nos da soporte para utilizar el conocido patrón MCV (model-view- controller). También hemos visto cómo se crea una clase del modelo. Ahora, para definir una clase como *controller* (*controlador*) se debe añadir en el encabezado de clase y antes del nombre la etiqueta *@controller*. Esta etiqueta claramente es propia del mismo framework, y sirve para que cuando se levante el contexto de Spring (al iniciar la aplicación web) se identifique la clase que posee esta etiqueta como una clase de tipo controller, entonces el framework inyectará las dependencias necesarias y creará esta clase como un bean específico de tipo controller. Existen además muchas otras etiquetas que se pueden utilizar.

Existen dos controladores principales en la aplicación. Uno es LoginController que se encarga de procesar las peticiones que tienen que ver con el manejo de la sesión de usuario. Y el otro es MuseumController que tiene que ver con el manejo del resto de la funcionalidad del sistema como la creación de objetos y el relleno de grillas entre otras.

MuseumController.java:

```

1 package info.unlp.museum.controller;
2
3 import info.unlp.museum.model.Categoria;
4
50
51 /**
52  *
53  * Controlador de la aplicacion web
54  *
55  */
56 @Controller
57 @RequestMapping("/museum")
58 public class MuseumController extends GeneralController {
59
60     public static final long TAMAÑO_MAXIMO_ARCHIVO = 1048576L;
61
62     @Autowired
63     IMuseumService servicio;
64
65     @Autowired
66     IUserarioServicio servicioUsuario;
67

```

En la definición de esta clase vemos que primero se encuentra la etiqueta `@Controller` como ya hemos explicado anteriormente. Después tenemos la etiqueta `@RequestMapping` que define cual es la ruta relativa a la aplicación para este controlador; eso quiere decir que si la ruta de la aplicación es `/MuseumWeb`, el controlador escuchara por peticiones de la forma `/MuseumWeb/museo/*`.

La etiqueta `@Autowired` define que esas variables sean instanciadas al momento que se inicia el contexto de Spring. En esto caso estas variables son los servicios en los cuales el controller delega para que efectúen la lógica de negocios apropiadas. Estas variables serán instancias de una clase que implemente esa interfaz.

Luego tenemos los métodos que devuelven una vista. Nuevamente, con la etiqueta `@RequestMapping` se está diciendo que cuando desde un evento desde el cliente o navegador invoque a la url `/MuseumWeb/museo/altaUsuario.do`, éste requerimiento se mapeará con éste método que lo que hace es devolver un objeto de la clase `ModelAndView`, el cual lleva como nombre identificador en este caso "altaUsuario". Esto hace que el `DispatcherServlet` busque y devuelva una página de nombre "altaUsuario.jsp".

En MuseumController.java:

```

69 @RequestMapping(value = "/altaUsuario.do", method = RequestMethod.GET)
70 public ModelAndView altaUsuario(HttpSession session) throws IOException
71
72     String pagina = "altaUsuario";
73
74     ModelAndView mav = new ModelAndView(pagina);
75
76     List<Museo> lista = servicio.traerMuseos();
77
78     mav.addObject("museos", lista);
79     return mav;
80
81 }

```

Otros métodos importantes son los que sirven para cargar las grillas. De nuevo vemos el url mapping y el método del request que en este caso es por GET. La etiqueta `@ResponseBody` indica que la respuesta del método se transforma en una secuencia de caracteres con el formato Json (JavaScript Simple Object Notation). Nos pareció que era mejor utilizar Json en vez de XML por algunas razones:

- Json es más fácil de leer que XML. Los objetos JSON son prototipos de datos los cuales tiene propiedades que pueden ser accedidas utilizando el punto (objeto.propiedad), por lo cual, extraer datos de un objeto JSON es más fácil que extraer datos de un XML.
- JSON consume menos de ancho de banda, por lo tanto su transmisión es más rápida. Cuando el servidor devuelve un Json devuelve una línea de texto en cambio si devolviera un XML devolverá un tamaño más grande.

MuseumController.java:

```

662 @RequestMapping(value = "/cargarGrillaMuseos.do", method = RequestMethod.GET)
663 public @ResponseBody
664 CustomJsonResponse cargarGrillaMuseos(HttpSession sesion) throws Exception {
665
666     try {
667
668         CustomJsonResponse response = new CustomJsonResponse();
669
670         List<Museo> lista = servicio.traerMuseos();
671
672         // Asigna el resultado desde el servicio a la respuesta
673         response.setRows(lista);
674         // Asigna el numero total de registro encontrados. Es usado por la
675         // pagina
676         response.setRecords(String.valueOf(lista.size()));
677         // Asigna el nro de pagina
678         response.setPage("1");
679         // Asigna el nro total de paginas
680         response.setTotal(String.valueOf(lista.size()));
681
682         return response;
683     } catch (Exception ex) {
684         throw ex;
685
686     }
687 }

```

7.10 Creación de los servicios

Como hemos dicho, el controller delega en servicios la realización de la lógica de negocios. El controller posee un atributo llamado *servicio* que implementa la interfaz *IMuseumService*. Esta variable es instanciada por la clase más acorde que implemente esa interfaz, y de esa acción se

encarga el framework de Spring. En este caso la única clase que la implementa es `MuseumServiceImpl`. Cada clase que cumple la función de servicio debe poseer dos etiquetas: `@Service` y `@Transactional`. La primera le dice a Spring que establezca esta clase como un servicio y que por lo tanto pueda ser utilizada para instanciar un atributo que se definió como `@Autowired` si coincide con el tipo. Y la segunda que agregue transaccionalidad a las operaciones.

`MuseumServiceImpl.java`:

```
54 @Service
55 @Transactional
56 public class MuseumServiceImpl implements IMuseumService {
57
58     @Autowired
59     private IMuseumDAO museoDao;
60
61     @Override
62     public ObjetoMuseo traerPorId(int idObjeto) {
63
64         ObjetoMuseo obj = museoDao.traerPorId(idObjeto);
65         museoDao.inicializarAlPeresozo(obj.getMateriales());
66         return obj;
67     }
68
69     @Override
70     public List<ObjetoMuseo> traerTodos() {
71
72         return museoDao.traerTodos();
73     }
74 }
```

A su vez cada clase de tipo Servicio delega en la capa siguiente, la capa de acceso a datos, que en este caso es cumplida por el atributo `museoDao`. Este atributo también es instanciado de una clase acorde por el contexto de Spring. En este caso, la única clase que la implementa es `MuseumDAOImpl`.

Cada DAO de la aplicación debe extender de la clase genérica y parametrizable `HibernateDAOGenericoImpl`. Los parámetros de esta Clase son la clase a la cual va a parametrizar y el tipo del identificador de cada objeto de esa clase. `HibernateDAOGenericoImpl` extiende de la clase `HibernateDaoSupport`, la cual es propia de Spring y provee acceso a la clase `HibernateTemplate` para realizar las operaciones de consulta sobre objetos.

`HibernateDAOGenericoImpl.java`

```

14 @SuppressWarnings({ "unchecked", "hiding" })
15 public abstract class HibernateDAOGenericoImpl<T, PK extends Serializable>
16     extends HibernateDaoSupport implements DAOGenerico<T, PK> {
17     @Autowired
18     public void init(SessionFactory factory) {
19         setSessionFactory(factory);
20
21     }
22
23     protected abstract Class<T> getEntityClass();
24
25     public void inicializarAlPeresozo(Object proxy) {
26         getHibernateTemplate().initialize(proxy);
27     }
28
29     /*
30     * =====
31     * Métodos a los cuales se le pasa el Type
32     * =====
33     */
34     @Override
35     public <T> T traerPorId(Type type, PK id) {
36         return (T) getHibernateTemplate().get((Class<T>) type, id);
37     }
38
39     @Override
40     public <T> List<T> traerTodos(Type type) {
41         DetachedCriteria criteria = DetachedCriteria.forClass((Class<T>) type);
42         return getHibernateTemplate().findByCriteria(criteria);
43     }
44

```

MuseumDAOImpl.java

```

21 @Repository
22 public class MuseumDAOImpl extends
23     HibernateDAOGenericoImpl<ObjetoMuseo, Integer> implements IMuseumDAO {
24
25     @Override
26     protected Class<ObjetoMuseo> getEntityClass() {
27
28         return ObjetoMuseo.class;
29
30     }
31
32     @Override
33     public void guardarMaterialMuseo(MaterialMuseo mat) {
34
35         this.getHibernateTemplate().save(mat);
36     }
37
38     @Override
39     public void eliminarMaterialMuseo(MaterialMuseo mat) {
40
41         this.getHibernateTemplate().delete(mat);
42
43     }

```

7.11 Comunicación móvil/servidor

La aplicación móvil se comunica con el servidor para recuperar información sobre el modelo de la encuesta, los objetos, el modelo de la trivia, la lista de museos. Para lograr esta comunicación la

aplicación móvil invoca mediante un requerimiento HTTP un método en el controller que devolverá la información requerida en formato Json. Por ejemplo el siguiente método devuelve un String en formato Json con la información de todos los museos.

MuseumController.java

```
2026 @RequestMapping(value = "/recuperarDatosMuseos.do", method = RequestMethod.POST)
2027 public @ResponseBody
2028 MuseoHTTP recuperarDatosMuseos(HttpServletRequest request,
2029                               HttpServletResponse response) throws ServletException, IOException {
2030
2031     MuseoHTTP respuesta = new MuseoHTTP();
2032
2033     try {
2034
2035         List<Museo> lista = servicio.traerMuseos();
2036
2037         if (lista != null && !lista.isEmpty()) {
2038             respuesta.setErrorCode(ErrorCode.RESPUESTA_CORRECTA);
2039             respuesta.setMuseos(lista);
2040         }
2041
2042         else {
2043             respuesta.setErrorCode(ErrorCode.ERROR_SERVIDOR);
2044             respuesta.setMessageError("No se encontraron museos");
2045         }
2046     }
2047
2048
2049     catch (Exception e) {
2050         respuesta.setErrorCode(ErrorCode.ERROR_SERVIDOR);
2051         respuesta.setMessageError(e.getMessage());
2052     }
2053
2054
2055     return respuesta;
2056
```

7.12 Configuración de la aplicación

7.12.1 Web.xml

El archivo web.xml es el archivo descriptor de la aplicación.

7.12.2 applicationContext.xml

En este archivo se definen beans de Spring que realizan la conexión a la base de datos, otros para manejo de la transaccionalidad junto a la integración con hibernate.

```

26 <bean id="propertyConfigurer"
27     class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
28     <property name="locations">
29         <list>
30             <value>/WEB-INF/jdbc.properties</value>
31         </list>
32     </property>
33 </bean>
34
35 <bean id="dataSource"
36     class="org.springframework.jdbc.datasource.DriverManagerDataSource">
37     <property name="driverClassName" value="${jdbc.driverClassName}" />
38
39     <!-- DESARROLLO -->
40     <property name="url"
41         value="${jdbc.url}" />
42
43     <property name="username" value="${jdbc.username}" />
44     <property name="password" value="${jdbc.password}" />
45 </bean>
46
47 <bean id="sessionFactory"
48     class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
49     <property name="dataSource" ref="dataSource"></property>
50     <property name="hibernateProperties">
51         <props>
52             <prop key="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect
53             </prop>
54             <prop key="hibernate.show_sql">true</prop>
55             <prop key="hibernate.cache.provider_class">org.hibernate.cache.HashtableCacheProvider
56
57             </prop>
58             <prop key="hibernate.cache.use_second_level_cache">true</prop>
59             <prop key="hibernate.cache.use_query_cache">true</prop>

```

7.12.3 applicationContext-security.xml

En este archivo de configuración se establecen los parámetros de seguridad de la aplicación. Al inicio se indican las url que son invocadas por la aplicación móvil y que no tiene restricciones en el acceso. Eso se indica cuando se pone *access="permitAll"*.

Para el resto de las url que se quiere acceder se debe estar autenticado, eso se declara cuando se declara *access="isAuthenticated()"*.

Se declaró en el proyecto una clase llamada CustomAuthenticationManager que es la que se encarga de guardar en el contexto de la aplicación el usuario y los roles del usuario.


```

21 <security:http use-expressions="true" entry-point-ref="authenticationEntryPoint">
22
23     <security:intercept-url pattern="/museum/recuperarSugerenciasObjetos.do" access="permitAll"/>
24     <security:intercept-url pattern="/museum/sobreElMuseo.do" access="permitAll"/>
25     <security:intercept-url pattern="/museum/recuperarInirConFlechas.do" access="permitAll"/>
26     <security:intercept-url pattern="/museum/recuperarModeloTrivia.do" access="permitAll"/>
27     <security:intercept-url pattern="/museum/crearEncuesta.do" access="permitAll"/>
28     <security:intercept-url pattern="/museum/recuperarDatosMuseos.do" access="permitAll"/>
29     <security:intercept-url pattern="/museum/recuperarDatosObjeto.do" access="permitAll"/>
30     <security:intercept-url pattern="/museum/recuperarModeloEncuesta.do" access="permitAll"/>
31     <security:intercept-url pattern="/museum/**" access="isAuthenticated()" />
32     <security:intercept-url pattern="/index.jsp" access="permitAll" />
33
34
35
36
37     <security:session-management
38         invalid-session-url="/Login/sesionFinalizada.do" >
39         <security:concurrency-control max-sessions="1" error-if-maximum-exceeded="true"
40             expired-url="/login.htm" />
41     </security:session-management>
42
43 </security:http>
44
45
46 <bean id="authenticationEntryPoint"
47     class="org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint"
48     p:loginFormUrl="/login/getLoginPage.do" />
49

```

7.12.4 jdbc.properties

Este archivo define los parámetros de conexión a la base de datos.

```

1 jdbc.driverClassName=org.postgresql.Driver
2 jdbc.url=jdbc:postgresql://localhost:5433/museo
3 jdbc.username=postgres
4 jdbc.password=postgres

```

Diagrama de base de datos

A continuación se muestra a modo de pantallazo general el diagrama de base de datos.

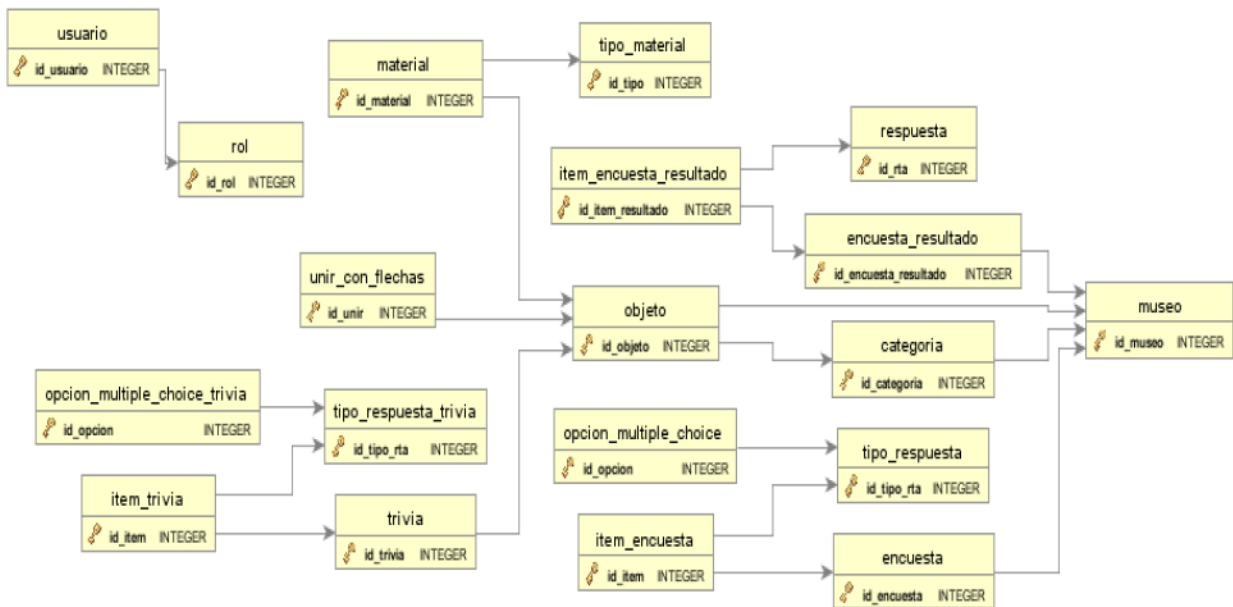


Figura 7.3: Diagrama de base de datos

Capítulo 8. Implementación de la Aplicación Móvil

En el capítulo anterior hemos detallado como fue la implementación del prototipo web. En este capítulo explicaremos similarmente detalles de implementación y cuestiones técnicas relacionadas al prototipo móvil.

8.1 Prototipo de la interfaz

Para desarrollar una aplicación resulta importante conocer además del SDK de Android, un conjunto de reglas básicas de diseño de interfaces de usuario. Además de las cuestiones de diseño explicadas en los apartados anteriores, se encuentran los patrones de diseño que se definieron como una solución global para un problema recurrente. Para definir la interfaz gráfica nos hemos basado en **Patrones Android** [61], un recurso orientado a diseñadores de interacción que necesitan resolver dudas sobre diseño y patrones de comportamiento típicos que se dan en las aplicaciones. Un patrón de interacción es un resumen de una solución de diseño que se ha demostrado que funciona más de una vez. Esta comunidad de diseñadores aconseja utilizar los patrones que describen como una guía y no como una ley. Clasifican los patrones en las siguientes categorías:

- **Manejo de Datos:** Se compone de patrones relacionados con mostrar, ver, ordenar, filtrar, navegar y buscar en un conjunto de datos. Por ejemplo, las aplicaciones de búsqueda, barras de estado, etc.
- **Entrada de Datos:** Cubre las formas comunes en que los usuarios pueden insertar datos en un contexto determinado. Por ejemplo, cuando el usuario interactúa con los diferentes widgets de un formulario (cajas de texto, check boxes, etc).
- **Navegación:** La navegación es el proceso de ir de un lugar a otro. Tiene dos objetivos; en primer lugar, la navegación le permite al usuario saber qué información está disponible en la aplicación; y en segundo lugar, ayuda a los usuarios a encontrar la información deseada lo más rápido posible. Por ejemplo: menús contextuales, barra de herramientas, etc.
- **Notificaciones:** Las notificaciones le dicen al usuario acerca de un evento que ocurre en la aplicación. Puede que en algunas de ellas se requiera de la respuesta del usuario, y en otras ocasiones la notificación simplemente suministre alguna información. En este apartado describen cómo utilizar los mensajes de tipo toast, las notificaciones y las barras de progreso.
- **Personalizar:** Por medio de la personalización, los usuarios son capaces de realizar cambios, guardar información personal y administrar ajustes de seguridad en el dispositivo Android. Aquí describen patrones para hacer pantallas de login, crear cuentas, etc.
- **Interacciones de pantalla:** Las interacciones de pantalla incluye todas las interacciones que puede realizar el usuario con su dedo sobre la pantalla a excepción del “pulsación simple” ya que esta acción siempre corresponde al gesto de seleccionar.
- **Social:** Los patrones de diseño contenidos en esta categoría, le permiten al usuario comunicarse e interactuar con otros usuarios. Por ejemplo, responder a cierto contenido a través de un comentario, hacer una valoración de algún contenido, etc.

A continuación describiremos el prototipo de la interfaz gráfica de la aplicación describiendo cada uno de los patrones de diseño utilizados.

Las pantallas de la aplicación va a mantener un formato general compuesto por tres partes principales: en la parte superior un encabezado, donde se muestra el icono de la aplicación; en el centro el contenido principal y debajo un pie de página, donde se detallan quienes desarrollaron la aplicación.



Figura 8.1: Formato general de la aplicación

1. Menú Principal

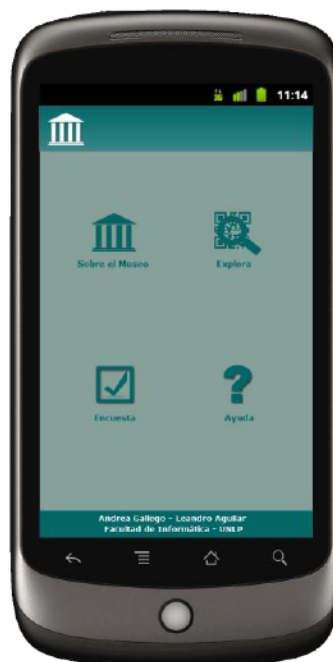


Figura 8.2: menú principal

En el menú principal, el contenido está compuesto por un tablero de opciones principales, el cual fue diseñado a partir del siguiente patrón de diseño:

Tablero de opciones



Figura 8.3: Patrón Tablero de opciones

El tablero de opciones es un patrón dentro de la categoría “Navegación” mencionada anteriormente. Un tablero de opciones es la pantalla de bienvenida de la aplicación, proporciona un punto de partida para el usuario. Además, muestra lo que el usuario puede hacer con la aplicación. El tablero de opciones puede ser estático o dinámico.

Un tablero de opciones proporciona fácil acceso a las tareas y funciones importantes. Las opciones se muestran con un ícono más un título en un diseño con forma de cuadrícula.

Un tablero de opciones se puede utilizar cuando la aplicación está orientada a las tareas y es compatible con múltiples tareas o funciones. Es ideal si se quiere proporcionar una visión general de tareas interesantes, nuevas o utilizadas con frecuencia. Un tablero de opciones se puede combinar con una barra de acción, por ejemplo, para proporcionar una opción de búsqueda para los usuarios. [62]

Aspectos positivos:

- Un usuario tiene un fácil acceso a las tareas más importantes
- Ofrece una visión general de la funcionalidad de la aplicación
- Ayuda al usuario a acceder de forma rápida al elemento más utilizado.

Aspectos negativos:

- Un tablero de opciones ocupa una gran cantidad de espacio en pantalla
- Las opciones que se muestran en el tablero puede ser percibida como las únicas características de la aplicación

Ejemplo: **Twitter**

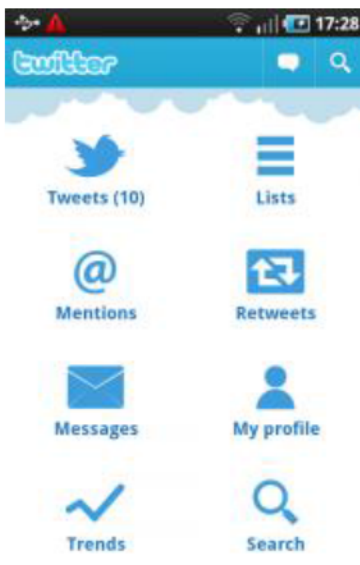


Figura 8.4: Pantalla de inicio Twitter

La pantalla de inicio de Twitter es un tablero de opciones, que muestra las funcionalidades más importantes. En esta captura de pantalla también muestra la cantidad de nuevos tweets que hay. La barra de acción proporciona enlaces rápidos al inicio de la aplicación, búsqueda y Twitter.

Listados



Figura 8.5: Carga de listado de museos

El diseño de los listados se realizó en base al siguiente patrón ubicado dentro de la categoría “Manejo de datos”:

Lista estática

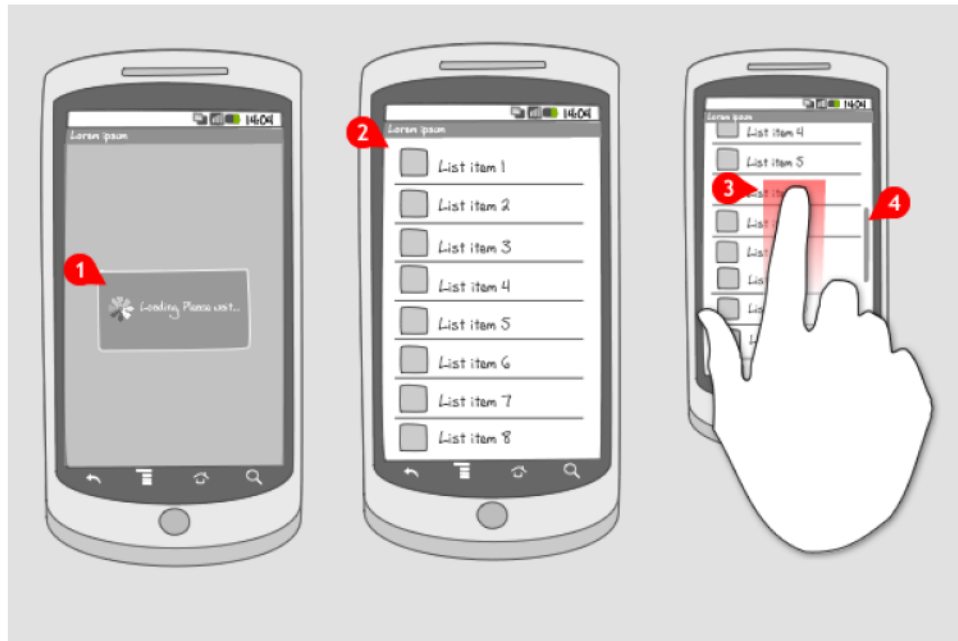


Figura 8.6: Patrón de diseño Lista Estática

- 1 Al cargar una lista estática, es posible mostrar un diálogo de progreso para que los usuarios sepan que el contenido se está cargando. Esto se hace generalmente cuando se necesita más de x segundos para cargar el contenido.
- 2 El contenido se muestra en una lista. Los elementos contienen un título y suelen ir acompañados de un icono o imagen. Es posible mostrar información adicional, siempre y cuando sea notorio que se trata de una lista.
- 3 Cuando el usuario se mueve con el dedo en la dirección que él quiere para desplazarse (en este caso hacia arriba), la lista se mueve hacia arriba a través de la pantalla, y los nuevos elementos son visibles.
- 4 Cuando la pantalla está aún en movimiento, un indicador muestra la posición de los elementos que se ven en la lista completa. Cuando el usuario deja de moverse y la lista deja de moverse, el indicador desaparece.

Una lista puede ser útil muchas veces y por lo tanto se puede aplicar con mucha frecuencia. La lista es una manera simple y directa para mostrar elementos de navegación, especialmente cuando el número de elementos no es muy extenso y se necesita que el contenido sea visto en una pantalla. Los elementos de la lista se colocan en un solo nivel, la lista es desplazable verticalmente. [63]

Aspectos Positivos

- Todo el contenido está en la pantalla al mismo tiempo, no hay que esperar o presionar para ver más contenido

Aspectos Negativos

- Puede tomar más tiempo para cargar
- Requiere más desplazamiento que el contenido paginado

Ejemplo: Engadget

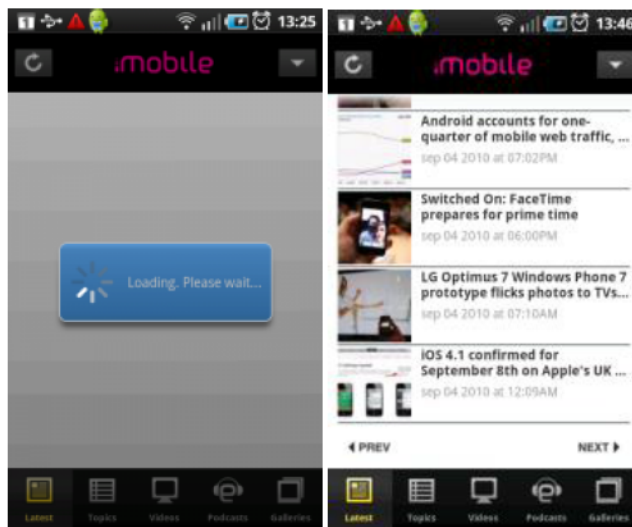


Figura 8.7: Lista estática en la aplicación Engadget

Cuando el usuario carga una nueva página de artículos en Engadget, se muestra un cuadro de diálogo de progreso. En la parte inferior de la página siguiente que contiene 20 elementos, las dos flechas se puede utilizar para el paso hacia la página siguiente o hacia atrás.

Elemento del Museo



Figura 8.8: Pestaña para el texto, audio y listado de imágenes

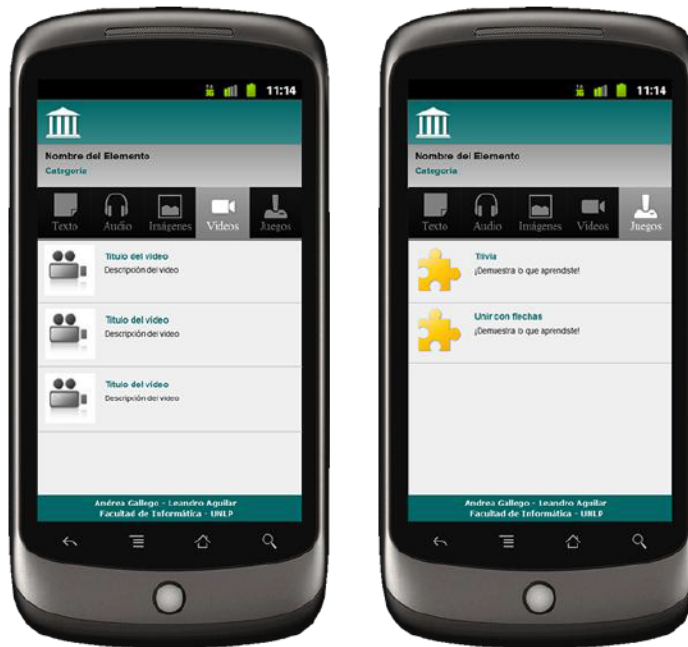


Figura 8.9: Pestañas para el listado de videos y juegos

Para la visualización de los diferentes contenidos del elemento se utilizó el siguiente patrón, ubicado en la categoría “Navegación”:

Pestañas



Figura 8.10: Patrón de diseño Pestañas

El contenido se divide en tres pestañas. Cada ficha representa un aspecto o sección del tema.

- 1 La primera pestaña está aquí seleccionada. Se muestra información sobre el tema.
- 2 Cuando la segunda pestaña se selecciona se resalta y el contenido correspondiente a ese aspecto se muestra.

- 3 Cuando la última pestaña se selecciona se resalta y el contenido correspondiente se muestra.

Las pestañas se utilizan cuando se desea mostrar una cantidad de información sobre un tema a distribuir en diferentes secciones. Estas secciones se muestran en una fila horizontal de 2 a 5 pestañas. Las etiquetas se muestran en la sección de la pestaña. [64]

Aspectos Positivos

- Manera directa para distribuir datos a través de diferentes secciones
- Originalmente proviene de interfaces web, por lo que los usuarios probablemente está familiarizado

Aspectos Negativos

- Puede ocupar mucho espacio en la pantalla
- Cuando hay más de 3 pestañas no queda mucho espacio para la etiqueta.

Ejemplo: Facebook

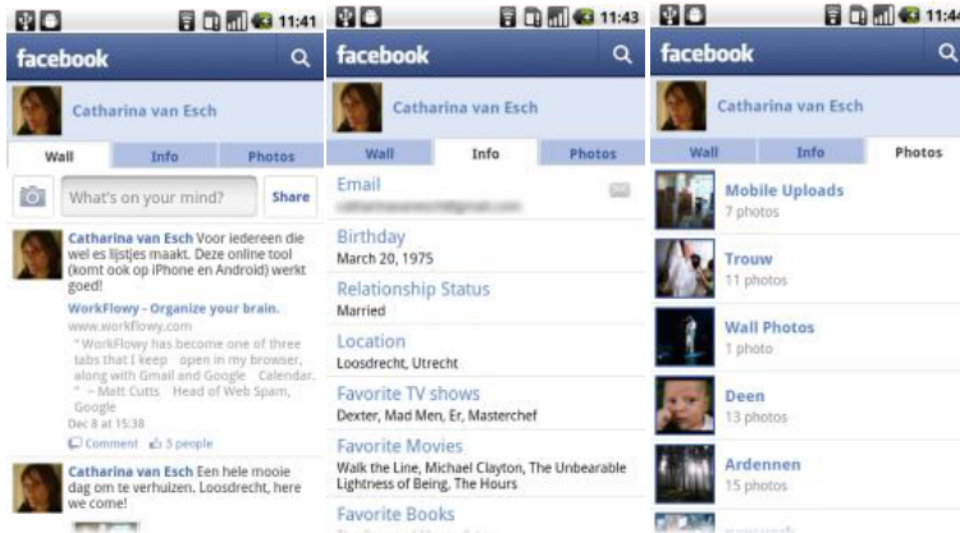


Figura 8.11: Pestañas en Facebook

En facebook se selecciona un usuario y la información sobre ese usuario se divide en tres pestañas. En la primera imagen se selecciona la primer pestañas: muro.

Tocando en la segunda pestaña se visualiza la información sobre el usuario.

Y en la tercera pestaña se muestran todas las fotos del usuario.

Formularios



Figura 8.12: Vista de la Encuesta

Para la entrada de datos de los usuarios en los formularios se utilizó el siguiente patrón, ubicado en la categoría “Entrada de datos”:

Expandir y ocultar teclado



Figura 8.13: Patrón de diseño Expandir y ocultar teclado

- 1 Cuando se selecciona un campo de entrada
- 2 La pantalla se desplaza hacia arriba, permitiendo que el teclado se incluya en pantalla debajo del campo de entrada.
- 3 El campo de entrada se destaca y el texto se puede introducir en el campo de entrada usando el teclado.

- 4 Cuando el teclado está oculto, la pantalla vuelve a su posición original y el texto introducido aparece en el campo de entrada.

Se utilizan para organizar el espacio disponible entre la aplicación y el teclado en pantalla y cuando en la aplicación los campos de entrada se encuentran por debajo de la mitad de la pantalla. [65]

Aspectos Positivos

- Tanto la aplicación y el teclado son visibles: el contexto de la entrada queda claro

Aspectos Negativos

- Requiere que la pantalla se mueva, lo que podría resultar confuso para el usuario
- El contenido no puede desplazarse adecuadamente.

Ejemplo: Calendario

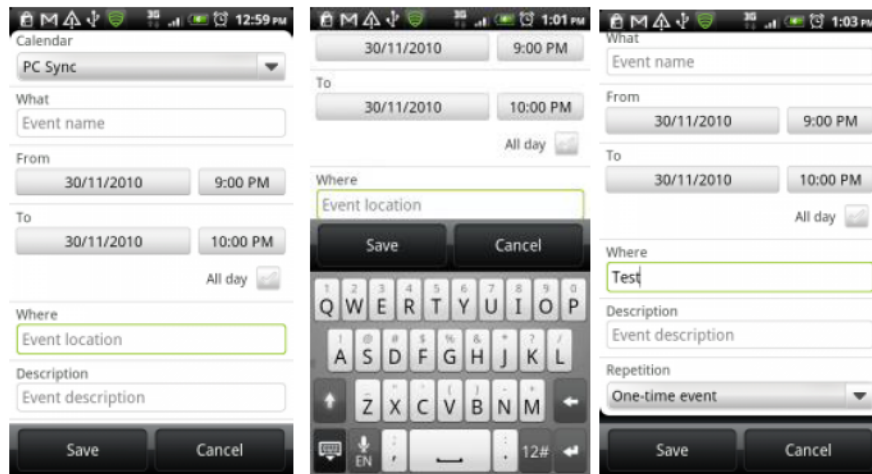


Figura 8.14: Ejemplo del patrón de diseño “Expandir y ocultar teclado”

La configuración de un nuevo evento de calendario incluye un campo de entrada para la ubicación del evento. Cuando se selecciona, toda la pantalla se desplaza, permitiendo que el teclado se ubique debajo del campo de entrada. Se puede introducir texto y el botón de abajo a la izquierda se puede utilizar para ocultar el teclado y en cuyo caso la pantalla vuelve a su posición original y el texto introducido se muestra en el campo de entrada.

Avuda



Figura 8.15: Vista de la Ayuda

Para el diseño de la ayuda se utilizó el siguiente patrón, ubicado en la categoría “Manejo de datos”:

Lista Expandible



Figura 8.16: Patrón de diseño Lista Expandible

Los elementos se organizan en grupos (a menudo por categoría) en una lista de dos niveles. Los grupos pueden expandirse para mostrar a sus hijos. Las listas expandibles muestran un indicador junto a cada elemento, indicando el estado actual del mismo:

- 1 Contraído
- 2 Expandido.

Las listas expandibles son útiles cuando el contenido tiene que ser visible en la misma pantalla. El contenido se puede agrupar en categorías que abarcan dos niveles.

Aspectos Positivos

- La información se puede comparar con mayor facilidad
- Ahorra espacio y se centra en el primer nivel de la Información.

Aspectos Negativos

- Para una gran cantidad de información en una pantalla el desplazamiento es inevitable
- La información no es inmediatamente visible a menos que las listas se expanda. Por lo tanto, es difícil comparar la información en los niveles secundario.

Ejemplo: RSS Feeds (HTC Desire)

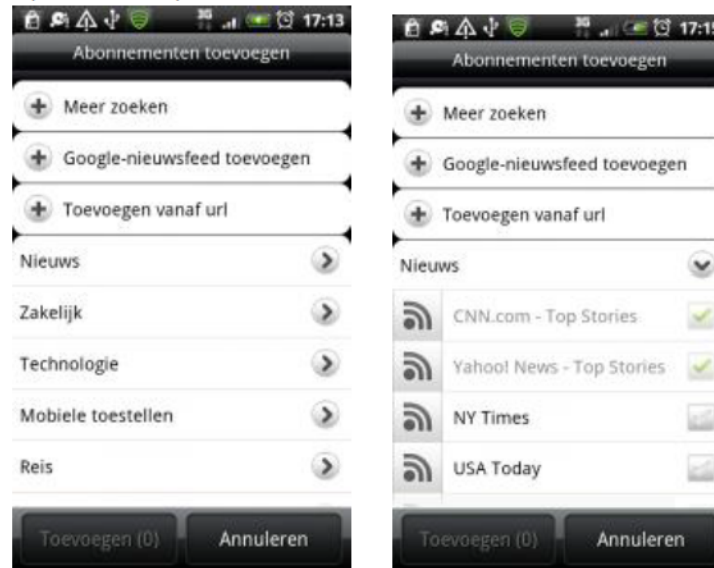


Figura 8.17: Ejemplo patrón de diseño Lista Expandible

Muestra una lista en estado contraído con un indicador hacia la derecha. Al tocar en un elemento de la lista, se muestra el estado expandido del elemento, con el indicador apunta hacia abajo.

Carga de datos remotos

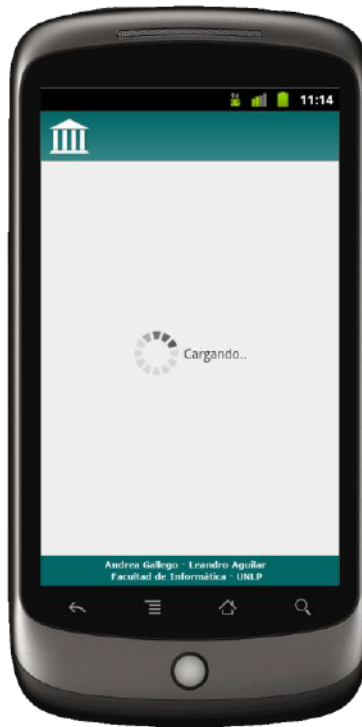


Figura 8.18: Carga de datos remotos

En las pantallas en las que es necesaria la comunicación con el servidor para poder visualizar los datos se utilizó el siguiente patrón ubicado en la categoría “Notificaciones”:

Rueda de Progreso



Figura 8.19: Patrón de diseño Rueda de progreso

- 1 El usuario pulsa sobre un elemento para abrirlo.
- 2 Una página en blanco con una rueda de progreso indicando al usuario que el contenido se está cargando.
- 3 Cuando se carga, se muestra el contenido.

Una rueda de progreso se utiliza para indicar que una aplicación se encuentra ocupada cargando la información a mostrar. Se puede aplicar a una pantalla completa, cuando toda la página tiene que ser cargada, o sólo a una parte de la página. [66]

Aspectos Positivos

- El usuario sabe que se está cargando el contenido
- Debido a que la carga del contenido se realiza en el interior de la pantalla, en lugar de en la parte superior de la misma, el usuario puede irse fácilmente a otra parte de la aplicación

Aspectos Negativos

- El usuario no sabe cuánto tiempo tardará el contenido en ser cargado

8.2 Implementación del prototipo

8.2.1 Creación del proyecto

Como dijimos anteriormente Android ofrece un plugin para Eclipse, el ADT plugin, que extiende la funcionalidad de este y facilita el desarrollo de aplicaciones para Android. Una vez instalado debemos descargar las SDK que elegimos y las herramientas necesarias para el desarrollo desde el Android SDK and AVD Manager.

Para crear la aplicación desde eclipse debemos ir a la opción File -> New -> Android Project

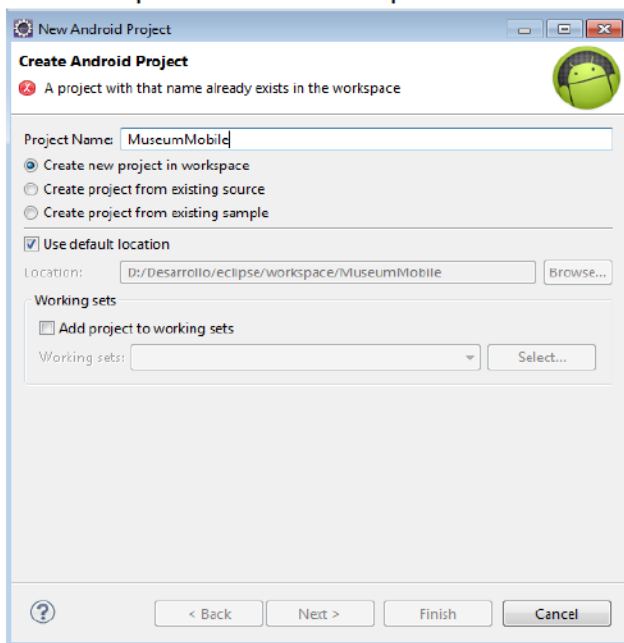


Figura 8.20: Crear proyecto Android

Luego nos pedirá que configuremos las siguientes opciones:

Project Name: es la denominación de Eclipse para nuestro proyecto. A través de este nombre se creará una carpeta en el workspace de Eclipse. En nuestro caso será "MuseumMobile".

Application Name: es el nombre de la aplicación, cómo aparecerá en el dispositivo.

Package Name: Este campo designa un paquete de Java. En nuestro caso será info.unlp.museum.

Minimun SDK: representa la versión mínima de Android SDK que se debe instalar en la dispositivo para que se ejecute esta aplicación en particular. Normalmente, este número se corresponderá al nivel de la API que se eligió para la aplicación. En nuestra caso elegimos la 8 que corresponde a la versión 2.2 de Android.

8.2.2 Estructura del proyecto

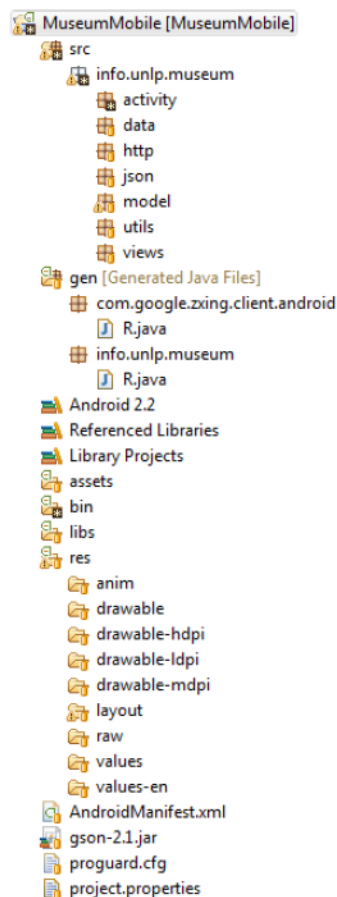


Figura 8.21: Estructura del proyecto

La carpeta /src/ contiene todo el código fuente de la aplicación. Dividimos las clases en los siguientes paquetes:

- **activity:** contiene todas las actividades necesarias para nuestra aplicación:
- **data:** contiene las clases necesarias para el almacenamiento de datos necesarios de la aplicación.
- **http:** contiene las clases necesarias para la comunicación con el servidor.
- **json:** contiene las clases necesarias para el paseo de datos de tipo json.
- **model:** contiene un conjunto de clases utilizadas para enviar y recibir información desde el servidor y para estructurar otros datos necesarios para la aplicación.
- **utils:** aquí se encuentran las clases que sirven de diferentes utilidades en la aplicación.
- **views:** todas las clases utilizadas para la vista de la aplicación.

AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="info.unlp.museum"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.CAMERA" />

    <application

        android:icon="@drawable/museo"
        android:label="@string/app_name"
        android:theme="@style/MyTheme" >

        <activity
            android:name=".activity.SplashAppActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".activity.MenuPrincipalActivity" />
        <activity android:name=".activity.ElementoMuseoActivity" />
        <activity android:name=".activity.AudioActivity" />
        <activity android:name=".activity.ImagenActivity" />
        <activity android:name=".activity.JuegoActivity" />
        <activity android:name=".activity.TextoActivity" />
        <activity android:name=".activity.VideoActivity" />
    </application>

```

Aquí se muestra una parte del archivo de manifiesto de la aplicación en la que se definen varias características de la aplicación:

- <uses-sdk> donde se especifica la versión mínima requerida por la aplicación. En nuestro caso en la versión 8.
- <uses-permission> Se utiliza para declarar todos los permisos que el usuario debe otorgar para el funcionamiento de la aplicación. En nuestro caso necesitaremos el permiso de acceso a internet, conocer el estado de la conexión a internet y la utilización de la cámara.
- <application> se describen las características básicas de la aplicación y sus actividades.
- <activity> define cada una de las actividades de la aplicación y sus características.
- <intent-filter> especifica el propósito de la actividad y dentro de este se indica la acción y la categoría del mismo. Por ejemplo la acción android.intent.action.MAIN y la categoría android.intent.category.LAUNCHER indica que es la actividad principal de la aplicación que se ejecutará.

8.2.3 Componentes de la aplicación

Como dijimos anteriormente las actividades representan el componente principal de la interfaz gráfica de una aplicación Android. Una actividad es implementada como una subclase de Activity. A

continuación se muestra a través de diagramas de clases las distintas actividades que contiene la aplicación.

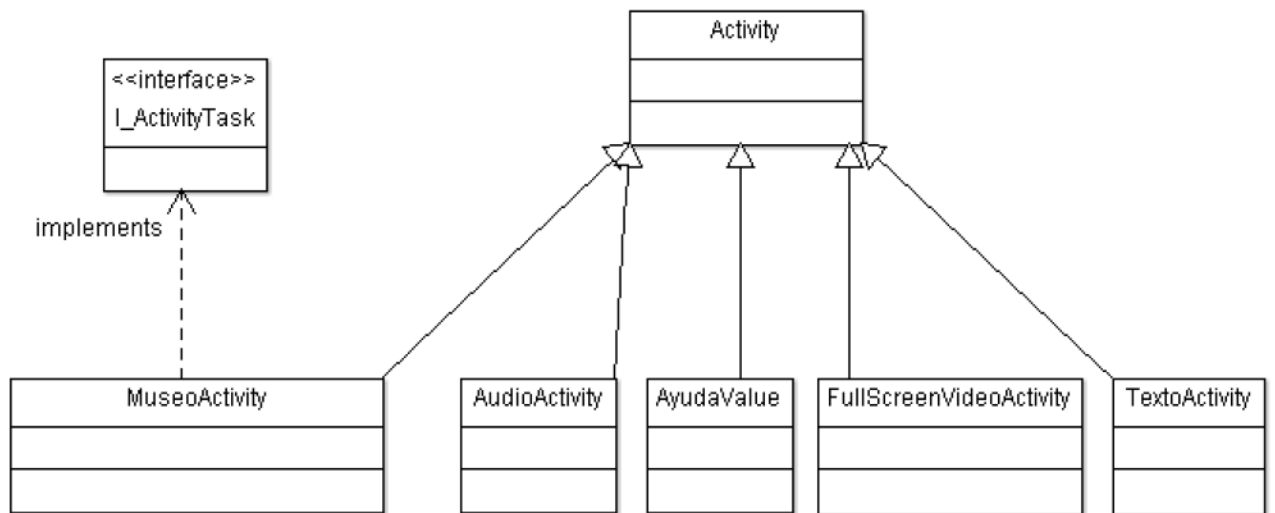


Figura 8.22: Diagrama de clases (Activity)

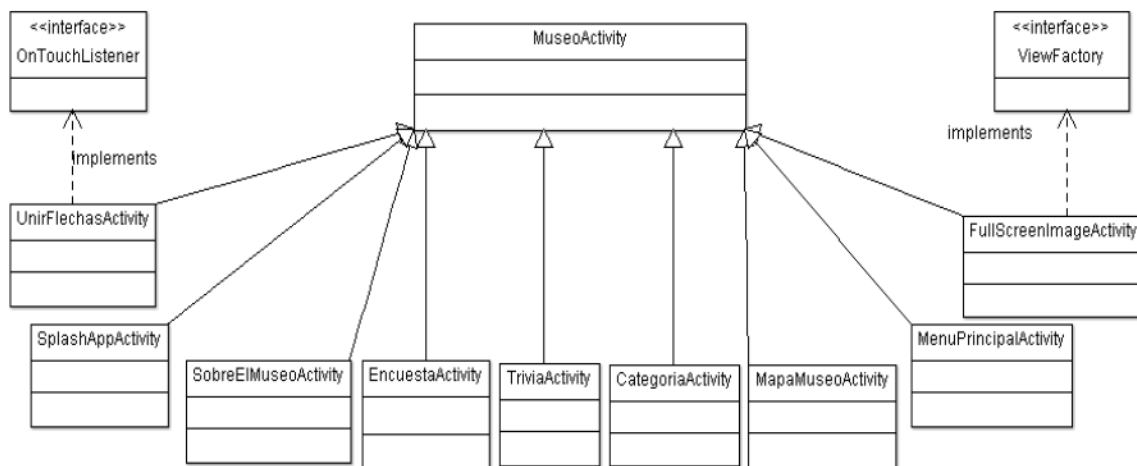


Figura 8.23: Diagrama de clases (MuseoActivity)

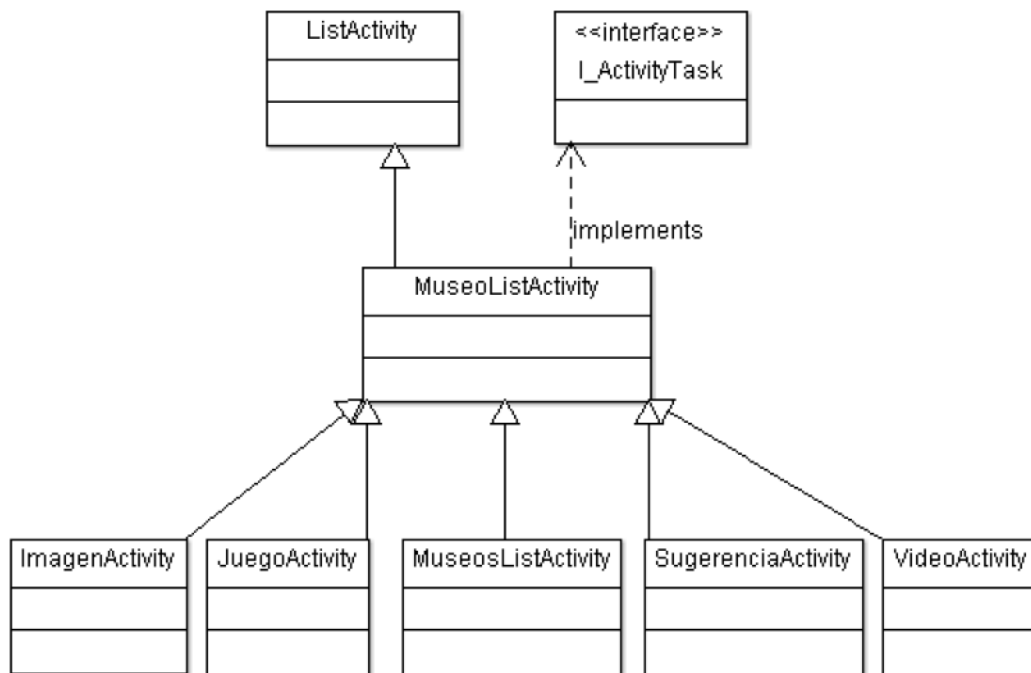


Figura 8.24: Diagrama de clases (ListActivity)

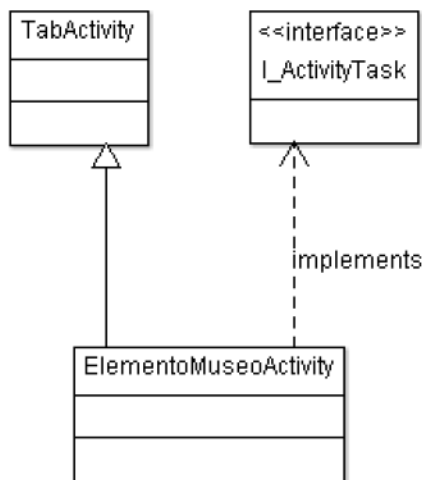


Figura 8.25: Diagrama de clases (TabActivity)

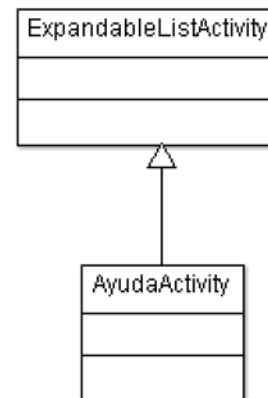


Figura 8.26: Diagrama de clases (ExpandableListActivity)

La mayoría de la actividades de la aplicación tienen el mismo comportamiento. Es por este motivo que decidimos implementar clases abstractas para que realicen el comportamiento común a las demás actividades. Estas actividades son MuseoActivity y MuseoListActivity. Ambas tienen un comportamiento similar, la diferencia es que MuseoActivity extiende de la clase android.app.Activity y MuseoListActivity extiende de la clase android.app.ListActivity. Esta última se utiliza para aquellos casos en los que es necesario mostrar un listado de elementos. A continuación se detalla el comportamiento de cada una de ellas.

MuseoActivity

MuseoActivity.java

```
1 package info.unlp.museum.activity;
2
3
4 import info.unlp.museum.R;
22
23 public abstract class MuseoActivity extends Activity implements I_ActivityTask{
24
25     private EstadoActividad estado;
26
27     private SharedPreferences sharedPreferences;
28
29     private Dialog dialog;
30
31     private Object objeto;
```

Esta clase además de extender de la clase `android.app.Activity`, implementa la interfaz `info.unlp.museum.http.I_ActivityTask` de la cual hereda el comportamiento necesario para la comunicación con el servidor que más adelante explicaremos.

Esta clase contiene cuatro propiedades o variables de instancia principales:

Estado: está representada por la clase `EstadoActivity.java`, se utiliza para guardar el estado en el que se encuentra la actividad en un momento dado. Es muy útil para los casos en los que se producen cambios en tiempo de ejecución como por ej: un cambio de orientación de pantalla. Cuando un cambio de orientación se produce la actividad se destruye, es decir, se llama al método `onDestroy()` y luego se llama nuevamente al método `onCreate()`. Por lo tanto, es necesario guardar el estado en el que se encontraba la actividad para poder restaurarlo posteriormente. A continuación se muestra los datos que necesitamos guardar de la actividad:

EstadoActivity.java

```
1 package info.unlp.museum.model;
2
3 import info.unlp.museum.http.MuseoAsyncTask;
4
5 public class EstadoActividad {
6
7     public enum Estado{
8         INICIANDO,
9         VALIDANDO,
10        EJECUTANDO,
11        FINALIZANDO,
12        INACTIVO
13    }
14
15    private Estado estado;
16
17    private MuseoAsyncTask task;
18
19    private CacheImagen cacheImagen;
20
21    private int operacion;
22
23    private Boolean noHayIntenet = false;
24
```

sharedPreference: este objeto se utiliza para tener acceso a los datos que necesitamos guardar en la aplicación para compartírselos entre las diferentes actividades. En android se puede almacenar información de diversas maneras. Para grandes cantidades de datos se dispone de bases de datos

mediante SQLite [67]. Para guardar pequeñas configuraciones o datos únicos, la plataforma pone a nuestra disposición la clase `android.content.SharedPreferences`. Esta clase almacena información accesible desde cualquier actividad de la aplicación.

SharedPreferences.java

```
6 public class SharedPreferences {
7
8     private Context context;
9
10    private SharedPreferences preferences;
11
12    private static final String COLECTION = "info.unlp.museum";
13    private static final String ID_MUSEO = "id_museo";
14    private static final String OBJETO_MUSEO = "objeto_museo";
15
16    public SharedPreferences(Context contex){
17        setContext(contex);
18        preferences = context.getSharedPreferences(COLECTION,Context.MODE_PRIVATE);
19    }
20
21    public void guardarIdMuseo(Integer idMuseo){
22        SharedPreferences.Editor editor = preferences.edit();
23        editor.putInt(ID_MUSEO, idMuseo);
24        editor.commit();
25    }
26
27    public void guardarObjeto(String objeto){
28        SharedPreferences.Editor editor = preferences.edit();
29        editor.putString(OBJETO_MUSEO, objeto);
30        editor.commit();
31    }
32
33    public Integer getIdMuseo(){
34        return preferences.getInt(ID_MUSEO, 0);
35    }
36
37    public String getObjetoMuseo(){
38        return preferences.getString(OBJETO_MUSEO, null);
39    }
}
```

La clase `SharedPreferences` [68] permite guardar y recuperar pares persistentes clave-valor de tipos de datos primitivos: booleanos, reales, enteros, long y string. Es decir, cada uno de los pares estará compuesta por un identificador único (p.e. "email") y un valor asociado a dicho identificador (p.e. "prueba@email.com"). Estos datos se mantienen durante la sesión del usuario y permanecen a pesar de que salgamos de la aplicación.

La API para el manejo de estas preferencias es muy sencilla. Toda la gestión se centraliza en la clase `SharedPreferences`, que representará a una colección de preferencias. Una aplicación Android puede gestionar varias colecciones de preferencias, que se diferencian mediante un identificador único. Para obtener una referencia a una colección determinada utilizaremos el método `getSharedPreferences()` al que pasaremos el identificador de la colección y un *modo de acceso*. El modo de acceso indicará qué aplicaciones tendrán acceso a la colección de preferencias y qué operaciones tendrán permitido realizar sobre ellas. Así, tendremos tres posibilidades principales:

- `MODE_PRIVATE`. Sólo nuestra aplicación tiene acceso a estas preferencias.

- `MODE_WORLD_READABLE`. Todas las aplicaciones pueden leer estas preferencias, pero sólo la nuestra modificarlas.
- `MODE_WORLD_WRITABLE`. Todas las aplicaciones pueden leer y modificar estas preferencias.

Una vez que hemos obtenido una referencia a nuestra colección de preferencias, ya podemos obtener, insertar o modificar preferencias utilizando los métodos `get` o `put` correspondientes al tipo de dato de cada preferencia

Para actualizar o insertar nuevas preferencias el proceso será igual de sencillo, con la única diferencia de que la actualización o inserción no la haremos directamente sobre el objeto `SharedPreferences`, sino sobre su objeto de edición `SharedPreferences.Editor`. A este último objeto accedemos mediante el método `edit()` de la clase `SharedPreferences`. Una vez obtenida la referencia al editor, utilizaremos los métodos `put` correspondientes al tipo de datos de cada preferencia para actualizar/insertar su valor, por ejemplo `putString(clave, valor)`, para actualizar una preferencia de tipo `String`. Finalmente, una vez actualizados/insertados todos los datos necesarios llamaremos al método `commit()` para confirmar los cambios.

En nuestro caso utilizamos esta clase para guardar el museo que selecciona el usuario cuando ingresa a la aplicación y el objeto que se encuentra visualizando actualmente, que explicaremos con más detalle más adelante.

dialog: esta variable es utilizada para mantener el diálogo que se encuentra visualizando el usuario en un momento dado. De esta manera, podemos modificar su estado en cualquier momento del ciclo de vida de la actividad. Para notificar al usuario utilizamos diálogos de alerta. Este tipo de diálogo se limita a mostrar un mensaje sencillo al usuario y botones de confirmación. A estos diálogos los construiremos mediante la clase `AlertDialog` y más concretamente su subclase `AlertDialog.Builder`. Su utilización es muy sencilla, bastará con crear un objeto de tipo `AlertDialog.Builder` y establecer las propiedades del diálogo mediante sus métodos correspondientes: título (`setTitle()`), mensaje (`setMessage()`) y el texto y comportamiento de los botones (`setPositiveButton()` y `setNegativeButton()`). A continuación mostramos la implementación de uno de los diálogos utilizados en la aplicación, el mismo se utiliza para notificar al usuario que no hay conexión a Internet:

MuseoActivity.java

```

85     AlertDialog.Builder builder = new AlertDialog.Builder(this);
86     builder.setIcon(R.drawable.waning);
87     builder.setTitle("Error al conectarse");
88     builder.setMessage(mensaje);
89     builder.setCancelable(false);
90     builder.setPositiveButton("Configuración de red", new DialogInterface.OnClickListener() {
91         public void onClick(DialogInterface dialog, int id) {
92             dialog.cancel();
93             startActivity(new Intent(Settings.ACTION_WIRELESS_SETTINGS));
94         }
95     });
96     builder.setNegativeButton("Intentar de nuevo", new DialogInterface.OnClickListener() {
97         public void onClick(DialogInterface dialog, int id) {
98             dialog.cancel();
99             getEstado().setNoHayIntenet(false);
100            realizarOperacion(operacion);
101        }
102    });
103     AlertDialog alert = builder.create();
104     alert.setCancelable(false);
105     setDialog(alert);
106     alert.show();

```

En este caso tiene dos botones: "Configuración de red", que se utiliza para mostrar al usuario la configuración de red del dispositivo para que de esta manera pueda solucionar el problema de la conexión, y el botón "Intentar de nuevo" que permite ejecutar la tarea actual nuevamente.

objeto: esta propiedad se utiliza para guardar el objeto que fue retornado en el método `onRetainNonConfigurationInstance()`. Como mencionamos anteriormente se pueden producir cambios de configuración durante la ejecución de la actividad. Si al reiniciar la actividad se requiere recuperar grandes conjuntos de datos se puede aliviar la carga de reinicializar la actividad mediante la retención de un objeto stateful cuando la actividad se reinicia debido a un cambio de configuración.

Para conservar un objeto durante un cambio de configuración en tiempo de ejecución:

- 1 Hay que sobrescribir el método **`onRetainNonConfigurationInstance()`** para devolver el objeto que se desea conservar.
- 2 Cuando la actividad se crea de nuevo, se llama al método **`getLastNonConfigurationInstance()`** para recuperar el objeto.[69]

MuseoActivity.java

```
73 @Override
74 public Object onRetainNonConfigurationInstance() {
75     if (getEstado().getTask() != null)
76         getEstado().getTask().setActivity(null);
77     return getEstado();
78 }

41     objeto = getLastNonConfigurationInstance();
```

Cuando el sistema Android apaga la actividad debido a un cambio de configuración, llama al método `onRetainNonConfigurationInstance()` entre el método `OnStop()` y `OnDestroy()` del ciclo de vida de la actividad. En la implementación de `onRetainNonConfigurationInstance()`, se puede devolver cualquier objeto que se necesita para restaurar eficientemente su estado después del cambio de configuración.

Un escenario en el que esto es valioso es si la aplicación carga una gran cantidad de datos desde la web. Si el usuario cambia la orientación del dispositivo, se reinicia la actividad y la aplicación debe volver a recuperar los datos esto podría ser lento. Por lo tanto en nuestro caso nos guardamos el estado de la actividad, mientras que el recupero de los datos desde el servidor se realiza en segundo plano.

Ciclo de vida de la actividad

A continuación se detalla la implementación del ciclo de vida de la actividad para que se comporte correctamente de acuerdo a los distintos cambios de estado que esta puede sufrir.

OnCreate()

MuseoActivity.java


```

33 @Override
34 public void onCreate(Bundle savedInstanceState) {
35     super.onCreate(savedInstanceState);
36
37     setContentView(getLayoutXML());
38
39     setSharedPreferences(new SharedPreferences(getApplicationContext()));
40
41     objeto = getLastNonConfigurationInstance();
42
43     inicializar();
44
45 }

```

El método onCreate() se llama al crear la actividad. Es donde se prepara la interfaz gráfica de la pantalla. En este caso la vista de la actividad dependerá de la actividad que extiende de la misma. Para ello se definió un método abstracto getLayoutXML(), cuya función es retornar id correspondiente a la vista de la actividad dentro de clase R. Mediante la llamada al método setContentView de la clase Activity le indicamos a Android que vista queremos mostrar en la pantalla. El id referencia a un archivo XML donde describimos la interfaz de usuario para una actividad determinada. Por ejemplo: R.layout.splash_layout referencia a la vista de la actividad SplashAppActivity.

```

57     protected abstract void inicializar();
58
59     protected abstract int getLayoutXML();

```

Además se inicializa la variable sharedPreferences y se llama al método abstracto inicializar() para que cada actividad realice lo que necesite al momento que se crea la actividad.

OnResume()

MuseoActivity.java

```

47 @Override
48 protected void onResume() {
49     super.onResume();
50     if (getEstado() != null && getEstado().getNoHayIntenet()){
51         getEstado().setNoHayIntenet(false);
52         realizarOperacion(getEstado().getOperacion());
53     }
54 }
55

```

Este método se ejecuta justo antes de que el usuario pueda interactuar con la actividad, por lo que se utiliza para aquellos momentos en los que el usuario perdió la conexión de internet y decide ir a la configuración de red para solucionar el problema. Luego que el usuario realizó los cambios correspondientes en la configuración de red para conectarse a la red nuevamente, en el método onResume() se valida si se encuentra en esa situación y si es así, se ejecuta nuevamente la tarea que estaba realizando en el momento que se quedó sin conexión.

onPause()

MuseoActivity.java

```

140 @Override
141 protected void onPause() {
142     super.onPause();
143     if (getDialog() != null && getDialog().isShowing()){
144         getDialog().dismiss();
145     }
146 }

```

Este método se llama cuando la actividad va a ser ocultada por otra actividad, es por este motivo que se valida si en ese momento se encuentra visualizando un diálogo y si es así se elimina el diálogo de la pantalla. Se elimina el diálogo, porque cuando se vuelve a activar la actividad dependiendo del estado en el que se encontraba la actividad se puede volver a mostrar el diálogo o no.

onRetainNonConfigurationInstance()

MuseoActivity.java

```

73 @Override
74 public Object onRetainNonConfigurationInstance() {
75     if (getEstado().getTask() != null)
76         getEstado().getTask().setActivity(null);
77     return getEstado();
78 }

```

Como explicamos anteriormente éste método se llama debido a un cambio de configuración. Es a través de este método que retornamos el estado en el que se encuentra la actividad en ese momento para poder restaurarlo correctamente cuando se vuelva a crear la actividad (onCreate()). Unas de las variables del estado de la actividad es una tarea, de la cual hablaremos con más detalle posteriormente, en este momento es importante aclarar que como la actividad se destruye es necesario desvincular la tarea con esa actividad y vincularla nuevamente cuando se restaura el estado de la misma, y de esta manera la tarea puede seguir ejecutándose mientras que la actividad se vuelve a crear.

onDestroy()

MuseoActivity.java

```

168 @Override
169 protected void onDestroy() {
170     super.onDestroy();
171
172     if (isFinishing()){
173         if (getEstado() != null && getEstado().getEstado() == Estado.EJECUTANDO){
174             if (getTask() != null && !getTask().isTerminado()){
175                 getTask().cancel(true);
176             }
177         }
178     }
179 }

```

Este método se llama antes de destruir la actividad. El método isFinishing() que nos provee la clase Activity nos permite saber si la actividad va a ser terminada o si, por ejemplo, simplemente se esta realizando un cambio de orientación de la pantalla. En el caso que la actividad está finalizando, nos interesa saber si en ese momento se está ejecutando una tarea. Si es así, debemos cancelarla para que se deje de ejecutar ya que no la necesitaremos y de esta manera liberamos recursos de la actividad que ya no utilizará.

MuseoListActivity

```
34 public abstract class MuseoListActivity extends ListActivity implements I_ActivityTask{
35
36     private EstadoActividad estado;
37
38     private SharedPreferences sharedPreferences;
39
40     private Dialog dialog;
41
42     private Object objeto;
43
44     private Vector<RowData> dataVector = new Vector<RowData>();
45
46     private int imagenDefault;
47
48     private LayoutInflater mInflater;
```

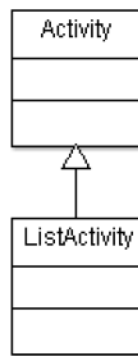


Figura 8.27 Diagrama de la clase ListActivity

La clase MuseoListActivity además de implementar la interfaz `info.unlp.museum.http.I_ActivityTask` de la cual hereda el comportamiento necesario para la comunicación con el servidor extiende de la clase `android.app.ListActivity`. Esta clase, que extiende de la clase `android.app.Activity` se diseñó para simplificar la visualización de elementos en una lista.

La clase MuseoListActivity contiene tres propiedades más además de las propiedades que contiene la clase MuseoActivity:

dataVector: esta propiedad es de tipo `java.util.Vector` que contiene elementos de tipo `RowData`. `RowData` es una clase interna de MuseoListActivity que representa los datos de cada elemento que se va a visualizar en la lista. Por lo tanto, la propiedad `dataVector` contiene todos los elementos que se visualizan en el listado.

MuseoListActivity.class

```

217 public class RowData {
218
219     private Integer mId;
220
221     private String titulo;
222
223     private String url;
224
225     private String detalle;

```

imagenDefault: cada elemento de la lista contiene una imagen y en la mayoría de los casos la carga de estas imágenes es dinámica y se obtiene desde el servidor. Pero hay otros casos en los que no es necesario obtener la imagen desde la red y es por ello, que utilizamos este atributo, el cual contiene una imagen a mostrar por defecto.

mInflater: este atributo es de tipo `android.view.LayoutInflater` que básicamente instancia archivos XML correspondientes a una vista en objetos java para que podamos manipularlos. En este caso se utilizó para la carga de datos de los elementos de la lista.

Implementación de una lista

Android proporciona la clase `ListView` que es capaz de mostrar una lista desplegable de elementos. Estos elementos pueden ser de cualquier tipo. A continuación podemos ver cómo podemos añadir un control `ListView` a nuestra interfaz de usuario:

```

10 <ListView
11     android:id="@android:id/list"
12     android:layout_width="fill_parent"
13     android:layout_height="wrap_content" />

```

Un `ListView` recibe sus datos a través de un adaptador. El adaptador también define la forma en la se muestra cada fila. Un adaptador extiende de la clase `BaseAdapter`. Android proporciona varios adaptadores estándar, los más importantes son `ArrayAdapter` y `CursorAdapter`.

Para controlar la asignación de los datos se puede implementar un adaptador propio como es en este caso `CustomAdapter` que extiende de la clase `ArrayAdapter` que puede manejar datos basados en matrices o listas.

MuseoListActivity.java

```

273 public class CustomAdapter extends ArrayAdapter<RowData> {
274
275     private HashMap<String, Object> cache;
276
277     public CustomAdapter(Context context, int resource,
278         int textViewResourceId, List<RowData> objects) {
279
280         super(context, resource, textViewResourceId, objects);
281     }
282
283
284     @Override
285     public View getView(int position, View convertView, ViewGroup parent) {
286
287         ViewHolder holder = null;
288         TextView title = null;
289         TextView detail = null;
290         ImageView imageView = null;
291         RowData rowData = getItem(position);
292         if (null == convertView) {
293             convertView = mInflater.inflate(R.layout.lista_elementos, null);
294             holder = new ViewHolder(convertView);
295             convertView.setTag(holder);
296         }
297         holder = (ViewHolder) convertView.getTag();
298         title = holder.getTitle();
299         title.setText(rowData.titulo);
300         detail = holder.getDetail();
301         detail.setText(rowData.detalle);
302
303         imageView = holder.getImage();
304
305         if (getCache() != null){
306             Bitmap drawable = (Bitmap) getCache().get(rowData.url);
307             imageView.setImageBitmap(drawable);
308         }else{
309             imageView.setImageResource(getImagenDefault());
310         }
311
312         return convertView;
313     }

```

Posteriormente, redefinimos el método encargado de generar y rellenar con nuestros datos todos los controles necesarios de la interfaz gráfica de cada elemento de la lista. Este método es `getView()`. El método `getView()` se llamará cada vez que haya que mostrar un elemento de la lista. Lo primero que se debe hacer es “inflar” el diseño XML que hemos creado. Esto consiste en consultar el XML de nuestro diseño y crear e inicializar la estructura de objetos java equivalente. Para ello, se utiliza el servicio del sistema `LayoutInflater`. Para obtener acceso a este servicio se puede realizar a través del método `getSystemService(Activity.LAYOUT_INFLATER_SERVICE)` de la actividad y una vez que tenemos acceso a este objeto generaremos la estructura de objetos mediante su método `inflate(id_layout)`.

La creación de objetos Java, cuando se infla un archivo XML, es costosa en cuanto a tiempo y consumo de memoria. Para aliviar este problema, Android nos propone un método que permite reutilizar algún diseño que hayamos inflado con anterioridad y que ya no nos haga falta por algún motivo, por ejemplo porque el elemento correspondiente de la lista ha desaparecido de la pantalla al hacer scroll. De esta forma evitamos todo el trabajo de crear y estructurar todos los objetos asociados al diseño. Entonces, si Android determina que una vista que representa a una fila no es visible, permite volver a utilizarlo a través del parámetro `convertView` del método `getView()`. En caso de que la vista no está disponible para su reutilización, Android pasará un valor nulo en el parámetro `convertView`. Por lo tanto, en la implementación del adaptador es necesario realizar esta comprobación.

MuseoListActivity.java

```
327     private class ViewHolder {
328         private View mRow;
329         private TextView title = null;
330         private TextView detail = null;
331         private ImageView i11 = null;
332
333     public ViewHolder(View row) {
334         mRow = row;
335     }
336
337     public TextView gettitle() {
338         if (null == title) {
339             title = (TextView) mRow.findViewById(R.id.title);
340         }
341         return title;
342     }
343
344     public TextView getdetail() {
345         if (null == detail) {
346             detail = (TextView) mRow.findViewById(R.id.detail);
347         }
348         return detail;
349     }
350
351     public ImageView getImage() {
352         if (null == i11) {
353             i11 = (ImageView) mRow.findViewById(R.id.img);
354         }
355         return i11;
356     }
357 }
```

El modo típico de crear una fila para un ListView es ir cargando todos los elementos gráficos que hay en una fila asignándole el dato que le corresponde a cada uno. Para ello, se parte del diseño de una fila como explicamos anteriormente, y se va cargando cada elemento de esta fila accediendo a través del método `findViewById()` y editando el valor correspondiente. Pero ésta búsqueda por ID de un control determinado dentro del árbol de objetos de un diseño puede ser una tarea costosa dependiendo de la complejidad del propio diseño, es por ello que se diseñó el patrón `ViewHolder`, que evita utilizar el método `findViewById()` cuando reutilizamos el `convertView`. Una clase `ViewHolder` es una clase interna en el adaptador que contienen referencias a las correspondientes vistas del diseño. Por lo tanto, se crea e inicializa el objeto `ViewHolder` la primera vez que inflamos un elemento de la lista y asociamos a dicho elemento de forma que posteriormente podamos recuperarlo fácilmente. Esta referencia a la vista se asigna como una etiqueta a través del método `setTag()` de la clase `View`.

Si recibimos un objeto `convertView`, podemos obtener la instancia del `ViewHolder` a través del método `getTag()` y asignar los nuevos atributos a las vistas a través de la referencia del `ViewHolder`.

Si bien esto suena complejo es más rápido que utilizando el método `findViewById()` cada vez.

Ahora bien, si quisiéramos realizar cualquier acción al pulsarse sobre un elemento de la lista creada tendremos que implementar el evento `onItemClickListener`. En este caso el método será abstracto para que cada clase lo implemente de acuerdo a la tarea que necesite cada una.

```
271 | public abstract void onListItemClick(ListView parent, View v, int position, long id);
```

No explicaremos el ciclo de vida de esta actividad ya que es análogo al de la actividad MuseoActivity.

8.2.4 Comunicación con el servidor

En el capítulo anterior explicamos cómo el servidor despliega los datos que serán consumidos por los clientes. Entonces, dijimos que esa información viajaba a través de la red en formato JSON. Previamente al diseño, tenemos que decir que la aplicación tenga acceso a Internet, esto se realiza a través del archivo de manifiesto.

```
9 | <uses-permission android:name="android.permission.INTERNET" />
```

De esta manera, nuestra aplicación podrá acceder a Internet, ya sea por WIFI o 3G. Además utilizamos los siguientes permisos para comprobar el estado de la red:

```
10 | <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />  
11 | <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Android cuenta con muchas clases dedicadas exclusivamente a las diferentes cuestiones de comunicaciones. Por ejemplo, de Java hereda, entre otros de javax.net. Dentro de sus paquetes exclusivos como plataforma, incluye una amplísima familia de paquetes bajo el nombre raíz android.net y sobre todo org.apache.http, la cual usaremos en esta aplicación.

Utilizamos un cliente HTTP estándar, con la clase DefaultHttpClient. Ésta cuenta con el método execute(), que precisa como parámetros la petición a enviar y se encarga de ejecutar la conexión. Para recibir la correspondiente respuesta, se define un objeto HttpResponse que recibe el resultado del método execute().

Para finalizar, la respuesta desde el servidor se procesa mediante clases JSON y para poder ser finalmente convertirlas al modelo de datos. La implementación que utilizamos para JSON es Gson [70], una librería de Google para serializar y deserializar objetos JSON.

En la aplicación implementamos la clase ConexionHTTP que será la encargada de realizar la comunicación con el servidor.

ConexionHTTP.java

```

24 public class ConexionHTTP {
25
26     private String url;
27
28     private ArrayList<NameValuePair> params;
29
30     private ArrayList <NameValuePair> headers;
31
32     private int responseCode;
33
34     private String message;
35
36     private String response;
37
38
39     public ConexionHTTP(String url){
40         setUrl(url);
41         setParams(new ArrayList<NameValuePair>());
42         setHeaders(new ArrayList<NameValuePair>());
43     }

```

Esta clase contiene las siguientes propiedades:

- **url:** es la dirección a la que se debe conectar.
- **params:** son los parámetros que se necesitan realizar en la petición.
- **headers:** se utiliza para enviar encabezados http en caso que sea necesario.
- **responseCode:** guarda el código de estado de la respuesta a la petición.
- **message:** guarda el mensaje correspondiente al código de estado de la respuesta.
- **response:** guarda la respuesta recibida.

El método principal de esta clase es el método execute, el cual atiende tanto requerimientos GET como POST.

ConexionHTTP.java:


```

46 public void execute(RequestMethod method) throws Exception
47 {
48     switch(method) {
49         case GET:
50             {
51                 String combinedParams = "";
52
53                 if(!params.isEmpty()){
54                     combinedParams += "?";
55
56                     for(NameValuePair p : params){
57                         String paramString = p.getName() + "=" + URLEncoder.encode(p.getValue(),"UTF-8");
58                         if(combinedParams.length() > 1){
59                             combinedParams += "&" + paramString;
60                         }else{
61                             combinedParams += paramString;
62                         }
63                     }
64                 }
65
66                 HttpGet request = new HttpGet(url + combinedParams);
67
68                 for(NameValuePair h : headers){
69                     request.addHeader(h.getName(), h.getValue());
70                 }
71
72                 executeRequest(request);
73                 break;
74             }
75         case POST:
76             {
77                 HttpPost request = new HttpPost(url);
78
79                 for(NameValuePair h : headers){
80                     request.addHeader(h.getName(), h.getValue());
81                 }
82
83                 if(!params.isEmpty()){
84                     request.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));
85                 }
86
87                 executeRequest(request);
88                 break;
89             }
90     }
91 }

```

Para el caso que se realice un requerimiento de tipo GET, se crea un objeto HttpGet con la url junto con los parámetros en el formato correspondiente al método GET. Para el caso de un requerimiento de tipo POST se creará un objeto HttpPost con la url y los parámetros correspondientes. En ambos caso se llama al método executeRequest que recibe como parámetro el requerimiento correspondiente al método elegido y realiza la conexión al servidor.

ConexionHTTP.java

```

93 private void executeRequest(HttpUriRequest request)
94 {
95     HttpClient client = new DefaultHttpClient();
96
97     HttpResponse httpResponse;
98
99     try {
100        httpResponse = client.execute(request);
101        responseCode = httpResponse.getStatusLine().getStatusCode();
102        message = httpResponse.getStatusLine().getReasonPhrase();
103
104        HttpEntity entity = httpResponse.getEntity();
105
106        if (entity != null) {
107
108            InputStream instream = entity.getContent();
109            response = convertStreamToString(instream);
110            instream.close();
111        }
112
113        } catch (ClientProtocolException e) {
114            client.getConnectionManager().shutdown();
115            e.printStackTrace();
116        } catch (IOException e) {
117            client.getConnectionManager().shutdown();
118            e.printStackTrace();
119        }
120    }

```

Creamos un cliente HTTP de la interfaz HttpClient implementado por DefaultHttpClient. Luego, creamos un objeto HttpResponse sobre el cual recibimos el resultado de ejecutar la petición creada a través del cliente HTTP. Si el resultado de la petición no es nulo, convertimos el contenido de la respuesta a string y lo guardamos en la variable response la cual será luego consultada para obtener la correspondiente respuesta.

Tareas en segundo plano

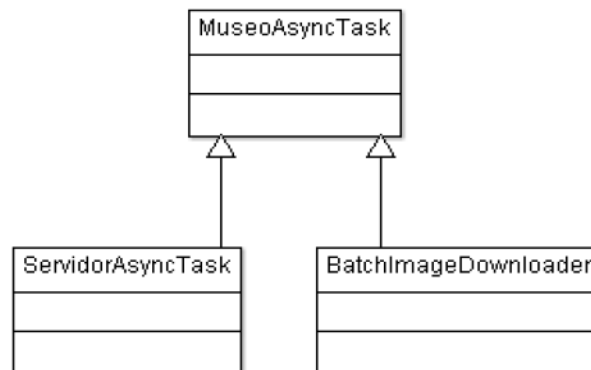


Figura 8.28: Diagrama de clases AsyncTask

Todos los componentes de una aplicación Android, tanto las actividades, los servicios o los receptores de eventos se ejecutan en el mismo hilo de ejecución, el llamado hilo principal, main thread o GUI thread, que como éste último nombre indica también es el hilo donde se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación. Es por ello, que

cualquier operación larga o costosa que realicemos en este hilo va a bloquear la ejecución del resto de componentes de la aplicación y por supuesto también la interfaz, produciendo al usuario un efecto evidente de lentitud, bloqueo o mal funcionamiento en general. Para evitar esto debemos ejecutar tareas en segundo plano. Android proporciona la clase `AsyncTask` para ello.

La forma básica de utilizar la clase `AsyncTask` consiste en crear una nueva clase que extienda de ella y sobrescribir varios de sus métodos. Estos métodos son los siguientes:

- `onPreExecute()`. Se ejecutará antes del código principal de nuestra tarea. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz, etc.
- `doInBackground()`. Contendrá el código principal de nuestra tarea.
- `onProgressUpdate()`. Se ejecutará cada vez que llamemos al método `publishProgress()` desde el método `doInBackground()`.
- `onPostExecute()`. Se ejecutará cuando finalice nuestra tarea, o dicho de otra forma, tras la finalización del método `doInBackground()`.
- `onCancelled()`. Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

En nuestro caso implementamos una clase abstracta `MuseoAsyncTask` con el comportamiento básico que necesitamos para las clases que extienda de ella:

Esta clase contiene una referencia a la actividad que lo está ejecutando y de esta manera podemos utilizarla en los métodos `onPreExecute()` para mostrarle la espera al usuario y `onPostExecute()` para notificarle a la actividad que la tarea se ha completado. El método `doInBackground()` deberá implementarse las clases concretas, tales como `ServidorAsyncTask` y `BatchImageDownloader`.

Como podemos ver definimos un tipo enumerativo `Task` el cual sirve para indicarle al `AsyncTask` la tarea que deseamos ejecutar las cuales explicaremos con más detalle posteriormente.

`MuseoAsyncTask.java`

```

7 public abstract class MuseoAsyncTask extends AsyncTask<Void, Void, Void> {
8
9
10 private I_ActivityTask activity;
11 private HashMap<String, Object> respuestas;
12 private HashMap<String, Object> parametros;
13 private Task tarea;
14 private boolean terminado;
15
16
17 public enum Task {
18     BUSCAR_ELEMENTO,
19     GET_MUSEOS,
20     GET_MODELO_ENCUESTA,
21     GET_MUSEO,
22     GET_IMAGENES,
23     GET_UNIR_CON_FLECHAS,
24     ENVIAR_ENCUESTA,
25     GET_MODELO_TRIVIA,
26     GET_OBJETOS_SUGERIDOS
27 }
28
29 public MuseoAsyncTask(I_ActivityTask activity) {
30     setActivity(activity);
31     setRespuestas(new HashMap<String, Object>());
32     setParametros(new HashMap<String, Object>());
33     setTerminado(false);
34 }
35
36 public void ejecutarTarea(Task task, HashMap<String, Object> parametros) {
37     setParametros(parametros);
38     setTarea(task);
39     execute();
40 }
41
42 @Override
43 protected void onPreExecute() {
44     getActivity().mostrarEspera();
45 }
46
47 @Override
48 protected void onPostExecute(Void result) {
49     setTerminado(true);
50     notifyActivityTaskCompleted();
51 }
52
53 private void notifyActivityTaskCompleted() {
54     if (null != getActivity()) {
55         getActivity().recibirRespuesta();
56     }
57 }

```

ServidorAsyncTask

Esta clase se utiliza para realizar la comunicación con el servidor, es decir, para realizar las peticiones correspondientes a la información del Museo.

ServidorAsyncTask.java

```

7 public class ServidorAsyncTask extends MuseoAsyncTask {
8
9     private ClienteHTTP cliente;
10
11
12 public ServidorAsyncTask(I_ActivityTask activity) {
13     super(activity);
14     setCliente(new ClienteHTTP((Activity) activity));
15 }
16
17
18 @Override
19 protected Void doInBackground(Void... params) {
20     switch (getTarea()) {
21         case BUSCAR_ELEMENTO:
22             getRespuestas().put("respuestaHTTP", cliente.getObjetoMuseo((QRData) getParametros().get("qrdata")));
23             break;
24         case GET_MUSEOS:
25             getRespuestas().put("museoHTTP", cliente.getMuseos());
26             break;
27         case GET_MODELO_ENCUESTA:
28             getRespuestas().put("encuestaHTTP", cliente.getModeloEncuesta((Integer) getParametros().get("idMuseo")));
29             break;
30         case GET_MUSEO:
31             getRespuestas().put("sobreElMuseoHTTP", cliente.getMuseo((Integer) getParametros().get("idMuseo")));
32             break;
33         case GET_UNIR_CON_FLECHAS:
34             getRespuestas().put("unirConFlechasHTTP", cliente.getUnirConFlechas((Integer) getParametros().get("idObjeto")));
35             break;
36         case ENVIAR_ENCUESTA:
37             EncuestaResultado resul = (EncuestaResultado) getParametros().get("encuesta");
38             int idMuseo = (Integer) getParametros().get("idMuseo");
39             getRespuestas().put("enviarEncuestaHTTP", cliente.enviarEncuesta(resul,idMuseo));
40             break;
41         case GET_MODELO_TRIVIA:
42             getRespuestas().put("triviaHTTP", cliente.getModeloTrivia((Integer) getParametros().get("idobjeto")));
43             break;
44         case GET_OBJETOS_SUGERIDOS:
45             getRespuestas().put("sugerenciaHTTP", cliente.getObjetosSugeridos((Integer) getParametros().get("idobjeto")));
46             break;
47         default:
48             break;
49     }
50     return null;
51 }

```

Tiene una variable de tipo ClienteHTTP la cual se encarga de validar la conexión a internet y recibir la respuesta del objeto ConexionHTTP que explicamos anteriormente.

ClienteHTTP.java

```

25 public class ClienteHTTP {
26
27     public static final String URL = "http://192.168.1.100:8080/MuseumWeb";
28
29     private Gson gson;
30
31     private Context context;
32
33     public ClienteHTTP(Context context) {
34         setGson(new Gson());
35         setContext(context);
36     }
37
38     private boolean hayConexion() {
39         ConnectivityManager cm = (ConnectivityManager) getContext()
40             .getSystemService(Context.CONNECTIVITY_SERVICE);
41         NetworkInfo internet = cm.getActiveNetworkInfo();
42         return internet != null && internet.isConnectedOrConnecting();
43     }

```

La comprobación del estado de la red se realiza a través del método `hayConexion()` el cual utiliza el servicio `CONNECTIVITY_SERVICE` para obtener el objeto `ConnectivityManager`. A través de este objeto podemos consultar si esta conectado o se esta conectando utilizando el método `isConnectedOrConnecting()`.

En el siguiente apartado, a medida que expliquemos cada una de las pantallas de la aplicación, explicaremos los métodos que ejecuta esta clase.

BatchImageDownloader

Esta clase se utiliza para la descarga de imágenes desde el servidor.

Teniendo en cuenta que se está trabajando con memoria limitada, debemos cargar en memoria una versión de la imagen de baja resolución. La versión de baja resolución debe coincidir con el tamaño del componente de interfaz de usuario que se visualiza. Una imagen con una resolución alta no proporciona ningún beneficio visible y ocupa mucha memoria. Es por este motivo que debemos decodificar imágenes de gran tamaño sin exceder el límite de memoria de la aplicación mediante la carga de una versión más pequeña en memoria.

BatchImageDownloader.java

```

37     protected Void doInBackground(Void... params) {
38         switch (getTarea()) {
39             case GET_IMAGENES:
40
41                 ImagenHTTP imagenHTTP = new ImagenHTTP();
42
43                 if (hayConexion()) {
44
45                     List<String> urls = (List<String>) getParametros().get("urls");
46
47                     for (String url : urls) {
48                         if (!imagenHTTP.getImagenes().containsKey(url)) {
49                             Bitmap bm = downloadImage(ClienteHTTP.URL + url);
50                             if (null != bm) {
51                                 imagenHTTP.getImagenes().put(url, bm);
52                             }
53                         }
54                     }
55                 } else {
56                     imagenHTTP.setErrorCode(ErrorCode.NO_HAY_INTERNET);
57                 }
58
59                 getRespuestas().put("imagenHTTP", imagenHTTP);
60                 break;
61             }
62         return null;
63     }
64
65     public Bitmap downloadImage(String url) {
66         Bitmap bitmap = decodeSampledBitmapFromResource(url,
67             (Integer) getParametros().get("width"),
68             (Integer) getParametros().get("height"));
69         return bitmap;
70     }

```

BatchImageDownloader.java

```

96     public static Bitmap decodeSampledBitmapFromResource(String url,
97         int reqWidth, int reqHeight) {
98
99         final BitmapFactory.Options options = new BitmapFactory.Options();
100        options.inJustDecodeBounds = true;
101        try {
102            DefaultHttpClient httpClient = new DefaultHttpClient();
103            HttpGet request = new HttpGet(url);
104            HttpResponse response = httpClient.execute(request);
105            InputStream stream = response.getEntity().getContent();
106            BitmapFactory.decodeStream(stream, null, options);
107        } catch (MalformedURLException e) {
108            e.printStackTrace();
109        } catch (IOException e) {
110            e.printStackTrace();
111        }
112
113        options.inSampleSize = calculateInSampleSize(options, reqWidth,
114            reqHeight);
115
116        options.inJustDecodeBounds = false;
117
118        try {
119            return BitmapFactory.decodeStream(new URL(url).openConnection()
120                .getInputStream(), null, options);
121        } catch (MalformedURLException e) {
122            e.printStackTrace();
123        } catch (IOException e) {
124            e.printStackTrace();
125        } catch (OutOfMemoryError e) {
126            e.printStackTrace();
127        }
128        return null;
129    }

```

La clase BitmapFactory proporciona varios métodos de decodificación (decodeByteArray(), decodeFile(), decodeStream(), etc.) para crear un objeto Bitmap a partir de diversas fuentes. En nuestro caso utilizamos el método decodeStream() para poder pasarle como parámetro el

InputStream que obtenemos de realizar un requerimiento a la url de la imagen que deseamos descargar.

Cada método de decodificación mencionado anteriormente intenta asignar memoria para el objeto Bitmap creado y por lo tanto puede derivar fácilmente en una excepción `java.lang.OutOfMemory`. Para evitar esto los métodos de decodificación permiten especificar opciones de decodificación a través de la clase `BitmapFactory.Options`. Ajustando la propiedad `inJustDecodeBounds` como verdadera evita la asignación de memoria, devolviendo un valor nulo para el objeto Bitmap, pero estableciendo las propiedades `outWidth`, `outHeight` y `outMimeType`. Esta técnica permite leer las dimensiones y el tipo de los datos de la imagen antes de la construcción y la asignación de memoria del objeto Bitmap. Entonces, para evitar la excepción `java.lang.OutOfMemory` debemos comprobar las dimensiones del Bitmap antes de la decodificación y decidir si la imagen completa puede ser cargada en la memoria o si debemos modificar el tamaño de la misma.

Para decirle al decodificador que modifique el tamaño de la imagen y cargue una versión más pequeña en la memoria, se debe ajustar el tamaño de la muestra a través de la propiedad `inSampleSize` del objeto `BitmapFactory.Options`. Si se establece un valor > 1 , pide al decodificador modificar la imagen original, devolviendo una imagen más pequeña para ahorrar memoria. El tamaño de la muestra es el número de píxeles en ambas dimensiones que corresponden a un único píxel en el Bitmap decodificado. Por ejemplo, `inSampleSize == 4` devuelve una imagen que es $1/4$ del ancho/altura de la original y $1/16$ el número de píxeles. Aquí se muestra el método para calcular el valor del tamaño de la muestra en base al ancho y altura de la imagen:

BatchImageDownloader.java

```
72 public static int calculateInSampleSize(BitmapFactory.Options options,
73     int reqWidth, int reqHeight) {
74
75     final int height = options.outHeight;
76     final int width = options.outWidth;
77     int inSampleSize = 1;
78
79     if (height > reqHeight || width > reqWidth) {
80         if (width > height) {
81             inSampleSize = Math.round((float) height / (float) reqHeight);
82         } else {
83             inSampleSize = Math.round((float) width / (float) reqWidth);
84         }
85
86         final float totalPixels = width * height;
87         final float totalReqPixelsCap = reqWidth * reqHeight * 2;
88
89         while (totalPixels / (inSampleSize * inSampleSize) > totalReqPixelsCap) {
90             inSampleSize++;
91         }
92     }
93     return inSampleSize;
94 }
```

Para utilizar este método, primero la decodificación `inJustDecodeBounds` se establece en verdadero, se obtienen las propiedades de la imagen y luego se decodifica nuevamente con el nuevo valor `inSampleSize` estableciendo la propiedad `inJustDecodeBounds` en falso, obteniendo de esta manera la imagen que se mostrará al usuario.

8.2.5 Implementación de los componentes de la aplicación

Anteriormente hemos explicado el comportamiento básico que tendrán las actividades que extenderán de las clases MuseoActivity y MuseoListActivity. A continuación explicaremos de qué manera se implementan cada una de las pantallas de la aplicación utilizando estas clases o las clases de base que proporciona Android.

1. Inicio de la aplicación



Figura 8.29: SplashAppActivity

SplashAppActivity es la actividad de bienvenida de la aplicación, tiene como objetivo mostrar una imagen y presentar la aplicación mientras que por detrás se recuperan los museos registrados en el sistema. Por lo tanto, la imagen de presentación se mostrará la cantidad de tiempo que se demore en recuperar esta información.

Vista de la actividad

splash_layout.xml

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background_header"
    android:gravity="center"
    android:orientation="vertical" >

    <ImageView
        android:layout_width="159dp"
        android:layout_height="0dip"
        android:layout_weight="0.16"
        android:clickable="false"
        android:scaleType="center"
        android:contentDescription="@string/app_name"
        android:src="@drawable/app_icon" />

</LinearLayout>

```

La vista tiene básicamente un LinearLayout conteniendo un ImageView. Los layouts son elementos no visuales destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior. Estos componentes extienden a la clase base ViewGroup, como muchos otros componentes contenedores, es decir, capaces de contener a otros controles. Hay distintos tipos de layout, entre los que encontramos, el FrameLayout, LinearLayout, RelativeLayout, etc. El LinearLayout apila uno de tras de otro todos sus elementos hijos de forma horizontal o vertical según se establezca su propiedad android:orientation.

Los elementos contenidos en un LinearLayout pueden establecer sus propiedades android:layout_width y android:layout_height para determinar sus dimensiones dentro del layout. En el caso de un LinearLayout, tendremos otro parámetro con el que jugar, la propiedad android:layout_weight. Esta propiedad nos va a permitir dar a los elementos contenidos en el layout dimensiones proporcionales entre ellas.

El control ImageView permite mostrar imágenes en la aplicación. La propiedad más interesante es android:src, que permite indicar la imagen a mostrar. Lo normal será indicar como origen de la imagen el identificador de un recurso de nuestra carpeta /res/drawable, por ejemplo android:src="@drawable/app_icon".

Implementación

La implementación de la clase SplashAppActivity consistió en extender la clase MuseoActivity y sobrescribir los métodos necesarios para cumplir el objetivo de la misma.

SplashAppActivity.java

```

19 public class SplashAppActivity extends MuseoActivity{
20
21     @Override
22     protected void inicializar() {
23         if ( getObjeto() instanceof EstadoActividad ) {
24             setEstado((EstadoActividad) getObjeto());
25             switch (getEstado().getEstado()) {
26                 case EJECUTANDO:
27                     getTask().setActivity(this);
28                     break;
29                 case INACTIVO:
30                     break;
31             }
32         } else {
33             getMuseos();
34         }
35     }
36
37     private void getMuseos() {
38         cambiarEstado(Estado.EJECUTANDO);
39         setTask(new ServidorAsyncTask(this));
40         HashMap<String, Object> p = new HashMap<String, Object>();
41         ejecutarTarea(Task.GET_MUSEOS, p);
42     }
43
44     @Override
45     public void recibirRespuesta() {
46         MuseoHTTP respuesta = (MuseoHTTP) getTask().getRespuestas().get("museoHTTP");
47
48         switch (respuesta.getErrorCode()) {
49             case ErrorCode.NO_HAY_INTERNET:
50                 noHayConexion(1, getResources().getString(R.string.no_hay_internet));
51                 break;
52             case ErrorCode.RESPUESTA_CORRECTA:
53                 respuestaCorrecta(respuesta);
54                 break;
55             case ErrorCode.ERROR_SERVIDOR:
56                 crearDialogo(respuesta.getMessageError(), ViewUtil.ERROR);
57                 break;
58         }
59     }
60
61
62     private void respuestaCorrecta(MuseoHTTP respuesta) {
63         String objetoMuseos = new Gson().toJson(respuesta.getMuseos());
64         Intent intentAct = new Intent(getApplicationContext(),MuseosListActivity.class);
65         intentAct.putExtra("museos", objetoMuseos);
66         startActivity(intentAct);
67         finish();
68     }
69
70     @Override
71     protected int getLayoutXML() {
72         return R.layout.splash_layout;
73     }

```

El objetivo de esta clase es obtener un listado de museos registrados en el museo para que posteriormente el usuario pueda elegir con cual interactuar. Para ello, utilizamos una tarea asincrónica `ServidorAsyncTask` para comunicarnos con el servidor, al que le indicamos que ejecute el requerimiento `GET_MUSEOS`.

ServidorAsyncTask.java

```
24         case GET_MUSEOS:
25             getRespuestas().put("museoHTTP", cliente.getMuseos());
26             break;
```

La tarea simplemente realiza la petición al ClienteHTTP y guarda la respuesta en un HashMap para que pueda ser consumida posteriormente por la actividad.

ClienteHTTP.java

```
157 public MuseoHTTP getMuseos() {
158
159     MuseoHTTP museoHTTP = new MuseoHTTP();
160
161     if (hayConexion()) {
162         ConexionHTTP con = new ConexionHTTP(URL + "/museum/recuperarDatosMuseos.do");
163
164         try {
165             con.execute(RequestMethod.POST);
166         } catch (Exception e) {
167             e.printStackTrace();
168         }
169
170         String json_response = con.getResponse();
171         String jsonAux = null;
172         try {
173             jsonAux = new String(json_response.getBytes("iso-8859-1"), "UTF-8");
174         } catch (UnsupportedEncodingException e) {
175             e.printStackTrace();
176         }
177
178         museoHTTP = getGson().fromJson(jsonAux, MuseoHTTP.class);
179
180     } else {
181         museoHTTP.setErrorCode(ErrorCode.NO_HAY_INTERNET);
182     }
183
184     return museoHTTP;
185 }
```

En el método `getMuseos()` de la clase `ClienteHTTP` creamos un objeto `ConexionHTTP` con la url `MuseumWeb/museo/recuperarDatosMuseos.do`, correspondiente al requerimiento que devuelve un objeto `MuseoHTTP`. Este objeto contiene un código de error y un mensaje en caso de que ocurra algún error en la comunicación con el servidor y un listado de museos, que serán los que se le mostrarán al usuario en la siguiente actividad. Como la respuesta que recibe es de tipo JSON es necesario realizar una conversión a su representación en objetos JAVA. Para ello utilizamos la librería `Gson`.

Una vez que obtenemos la respuesta del requerimiento, la tarea `ServidorAsyncTask` le notifica a la actividad que la tarea se ha completado.

MuseoAsyncTask.java

```

53 private void notifyActivityTaskCompleted() {
54     if (null != getActivity()) {
55         getActivity().recibirRespuesta();
56     }
57 }

```

La actividad SplashAppActivity cuando recibe la respuesta, a través del método recibirRespuesta(), valida el código de error que recibió y en base al mismo ejecuta el código correspondiente. Los códigos de error que puede recibir en este caso son:

- **ErrorCode.NO_HAY_INTERNET:** este código nos indica que el dispositivo no tiene acceso a internet y debemos comunicárselo al usuario para que arregle la situación. Para ello, le mostramos el siguiente diálogo:



Figura 8.30: Error de conexión

- **ErrorCode.ERROR_SERVIDOR:** este código nos indica que se produjo un error en la comunicación o de procesamiento en el servidor. Esta situación también se le informa al usuario a través de un diálogo al cual le seteamos el mensaje que viene en la respuesta.
- **ErrorCode.RESPUESTA_CORRECTA:** por último tenemos el código de error que realmente nos interesa, el que nos indica que la respuesta recibida es correcta. En este caso, iniciamos la actividad MuseosListActivity y le enviamos la lista de museos que recibimos a través de un objeto Intent y finalizamos la actividad.

2. Listado de museos

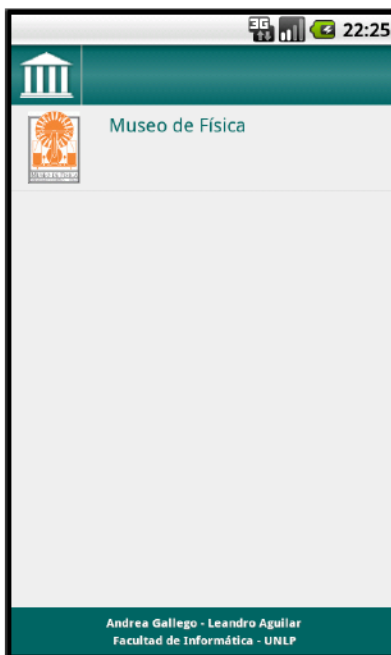


Figura 8.31: MuseosListActivity

Como dijimos anteriormente la actividad MuseosListActivity muestra el listado de museos que se encuentran registrados en el sistema. El usuario debe elegir entre los museos cual es el que va a empezar a recorrer. En este caso, sólo se encuentra registrado el Museo de Física, que se muestra junto a su ícono.

Vista de la actividad

museos.xml

```

2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   android:id="@+id/body"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent"
6   android:background="@drawable/gray_list"
7   android:orientation="vertical" >
8
9   <include layout="@Layout/header"/>
10
11 <FrameLayout
12   android:layout_width="fill_parent"
13   android:layout_height="0dip"
14   android:layout_weight="1"
15   >
16   <ListView
17     android:id="@android:id/list"
18     android:layout_width="fill_parent"
19     android:layout_height="fill_parent"
20   />
21
22   <TextView
23     android:id="@android:id/empty"
24     android:layout_width="fill_parent"
25     android:layout_height="fill_parent"
26     android:gravity="center_vertical|center_horizontal"
27     android:text="@string/no_hay_resultados"
28     style="@style/ListEmpty"/>
29
30   <include layout="@Layout/cargando"/>
31 </FrameLayout>
32
33 <include layout="@Layout/footer"/>
34
35 </LinearLayout>

```

La vista está compuesta por un contenedor LinearLayout, que explicamos en la actividad anterior, al que le incorporamos tres componentes: el encabezado, un FrameLayout y el pie de página.

El encabezado y pie de página fueron definidos en un archivo xml separado de manera tal que nos permita incluirlo en todas las vistas que necesitemos.

header.xml

```
2 <RelativeLayout xmlns:android='http://schemas.android.com/apk/res/android'
3     android:layout_width='fill_parent'
4     android:layout_height='50dp'
5     android:gravity="center_vertical"
6     android:background="@drawable/background_header"
7     android:padding="0dp"
8     >
9
10    <ImageView android:id="@+id/icon"
11        android:layout_width='wrap_content'
12        android:layout_height='fill_parent'
13        android:layout_alignParentTop='true'
14        android:layout_alignParentBottom='true'
15        android:layout_margin='6dip'
16        android:background="@drawable/m_header"
17        android:contentDescription="@string/app_name"
18    />
19
20    <View
21        android:layout_toRightOf="@id/icon"
22        android:layout_width="1dip"
23        android:layout_height="fill_parent"
24        android:background="@drawable/background_green" />
25
26 </RelativeLayout>
```

Para el encabezado utilizamos un contenedor RelativeLayout que permite especificar la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout. De esta forma, al incluir un nuevo elemento X podremos indicar por ejemplo que debe colocarse debajo del elemento Y y alineado a la derecha del layout padre. En este caso se incorporó una imagen a través de un ImageView y una línea vertical a su derecha a través de un componente básico como es el View.

footer.xml

```
2 <LinearLayout
3     xmlns:android='http://schemas.android.com/apk/res/android'
4     android:layout_width="fill_parent"
5     android:layout_height="40dp"
6     android:background="@drawable/start_green"
7     android:gravity="center"
8     android:orientation="vertical"
9     >
10
11    <TextView
12        style="@style/FontFooter"
13        android:layout_width="wrap_content"
14        android:layout_height="wrap_content"
15        android:text="@string/app_footer" />
16
17    <TextView
18        style="@style/FontFooter"
19        android:layout_width="wrap_content"
20        android:layout_height="wrap_content"
21        android:text="@string/app_footer2" />
22
23 </LinearLayout>
```

El pie de página es muy simple, consta de un LinearLayout que contiene dos TextView, el cual se utiliza para mostrar un texto determinado al usuario. El texto del control se establece mediante la propiedad android:text.

En la vista de la actividad mencionamos también que incluimos un FrameLayout. Éste es el más simple de todos los layouts de Android. Un FrameLayout coloca todos sus controles hijos alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente (a menos que éste último tenga transparencia). Los componentes incluidos en un FrameLayout podrán establecer sus propiedades android:layout_width y android:layout_height, que

podrán tomar los valores "fill_parent" (para que el control hijo tome la dimensión de su layout contenedor) o "wrap_content" (para que el control hijo tome la dimensión de su contenido). En este caso, utilizamos un FrameLayout para mostrar un ListView o un ProgressBar (barra de progreso). Es decir, mientras se está transfiriendo la información se visualiza la barra de progreso y una vez que se completó dicha transferencia se muestra el listado con los datos recibidos. El ProgressBar lo incorporamos a través del layout cargando.xml

cargando.xml

```
2 <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:id="@+id/cargando"
5     android:orientation="horizontal"
6     android:layout_width="fill_parent"
7     android:layout_height="fill_parent"
8     android:background="@drawable/gray_list"
9     android:gravity="center"
10    android:layout_gravity="center_vertical|center_horizontal"
11    >
12    <ProgressBar
13        android:layout_width="wrap_content"
14        android:layout_height="wrap_content"
15        style="Widget.ProgressBar.Small.Inverse"
16
17        />
18    <TextView android:text="@string/cargando"
19        android:layout_width="wrap_content"
20        android:layout_height="wrap_content"
21        android:layout_marginLeft="5dip"/>
22 </LinearLayout>
```

Implementación

MuseosListActivity.java


```

31 public class MuseosListActivity extends MuseoListActivity {
32
33     private List<Museo> museos;
34
35     @Override
36     protected void inicializar() {
37
38         Bundle extras = getIntent().getExtras();
39         String json = extras.getString("museos");
40
41         Type collectionType = new TypeToken<List<Museo>>().getType();
42         museos = new Gson().fromJson(json, collectionType);
43
44         if ( getObjeto() instanceof EstadoActividad ) {
45             setEstado((EstadoActividad) getObjeto());
46
47             switch (getEstado().getEstado()) {
48                 case EJECUTANDO:
49                     getTask().setActivity(this);
50                     break;
51                 case FINALIZANDO:
52                     getTask().setActivity(this);
53                     salirDeLaAplicacion();
54                     break;
55                 case INACTIVO:
56                     ocultarEspera();
57                     if (!hayElementos())
58                         cargarListado(getEstado().getCacheImagen().getImagenes());
59                     break;
60             }
61         } else {
62             if (!hayElementos())
63                 getImagenes();
64         }
65     }
66
67     private void getImagenes() {
68         cambiarEstado(Estado.EJECUTANDO);
69         setDataVector(new Vector<MuseoListActivity.RowData>());
70         setTask(new BatchImageDownloader(this));
71
72         HashMap<String, Object> p = new HashMap<String, Object>();
73         ArrayList<String> urls = new ArrayList<String>();
74         for (Museo elemento : museos) {
75             urls.add(elemento.getUrlIcono());
76         }
77         p.put("urls", urls);
78         p.put("width", 60);
79         p.put("height", 60);
80
81         ejecutarTarea(Task.GET_IMAGENES, p);
82     }

```

Esta actividad tiene como objetivo mostrar el listado de museos que se obtuvieron en la actividad anterior, es por ello que extiende de la actividad MuseoListActivity. En este listado se muestra el icono del museo junto a su nombre y utilizamos la tarea asincrónica BatchImageDownloader para descargar las imágenes correspondientes a cada uno de estos iconos. Una vez que se han descargado todas las imágenes, se procede a mostrar el listado al usuario.

MuseosListActivity.java

```
122 private void mostrarDatos(ImagenHTTP respuesta) {
123     cargarListado(respuesta.getImagenes());
124     CacheImagen cache = new CacheImagen();
125     cache.setImagenes(respuesta.getImagenes());
126     getEstado().setCacheImagen(cache);
127     cambiarEstado(Estado.INACTIVO);
128 }
129
130 private void cargarListado(HashMap<String, Object> imagenes) {
131     for (Museo elemento : museos) {
132         RowData rd = new RowData(elemento.getId(), elemento.getNombre(),
133             elemento.getSubTitulo(), elemento.getUrlIcono());
134         getDataVector().add(rd);
135     }
136
137     CustomAdapter adapter = new CustomAdapter(this,
138         R.layout.lista_elementos, R.id.title, getDataVector());
139     adapter.setCache(imagenes);
140
141     setListAdapter(adapter);
142 }
143
144 ...
```

Para manejar la selección de un elemento de la lista se debe implementar el método `onListItemClick()`. Este método recibe como parámetro la posición del elemento seleccionado la cual nos permite acceder al elemento dentro del adapter. En este caso, cuando se selecciona uno de los museos de la lista guardamos el id del museo en las preferencias de la aplicación para saber en todo momento sobre qué museo está interactuando el usuario. Luego, llamamos a la actividad del menú principal.

MuseosListActivity.java

```
84 public void onListItemClick(ListView parent, View v, int position, long id) {
85
86     RowData row = (RowData) parent.getAdapter().getItem(position);
87
88     getSharedPreferences().guardarIdMuseo(row.getmId());
89
90     Intent intent = new Intent(v.getContext(), MenuPrincipalActivity.class);
91
92     startActivity(intent);
93
94 }
```

3. Menú principal



Figura 8.32: MenuPrincipalActivity

A través de la actividad MenuPrincipalActivity implementamos el menú principal de la aplicación. Con las opciones que se presentan, podemos acceder a las actividades principales que ofrece la aplicación, de las cuales hablaremos más adelante. Ahora nos centraremos en explicar cómo implementamos esta actividad.

Vista de la actividad

dashboard_layout.xml

```

2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/home_root"
4     android:orientation="vertical"
5     android:layout_width="fill_parent"
6     android:layout_height="fill_parent">
7
8     <include layout="@layout/header"/>
9     <include layout="@layout/fragment_layout"/>
10    <include layout="@layout/footer"/>
11
12 </LinearLayout>

```

fragment_layout.xml

```

20 <info.unlp.museum.views.DashboardLayout
3   xmlns:android="http://schemas.android.com/apk/res/android"
4   android:layout_width="fill_parent"
5   android:layout_height="fill_parent"
6   android:layout_weight="1"
7   android:background="@drawable/background_green" >
8
9   <Button
10      android:id="@+id/boton_museo"
11      style="@style/DashboardButton"
12      android:drawableTop="@drawable/button_museo_states"
13      android:onClick="sobreELMuseo"
14      android:text="@string/boton_historia_museo" />
15
16
17   <Button
18      android:id="@+id/boton_lector"
19      style="@style/DashboardButton"
20      android:onClick="verElemento"
21      android:drawableTop="@drawable/button_leerqr_states"
22      android:text="@string/boton_lectorQR" />
23
24
25   <Button
26      android:id="@+id/btn_encuesta"
27      style="@style/DashboardButton"
28      android:onClick="verEncuesta"
29      android:drawableTop="@drawable/button_encuesta_states"
30      android:text="@string/boton_encuesta" />
31
32   <Button
33      android:id="@+id/btn_ayuda"
34      style="@style/DashboardButton"
35      android:drawableTop="@drawable/button_states"
36      android:text="@string/boton_ayuda"
37      android:onClick="verAyuda" />
38
39 </info.unlp.museum.views.DashboardLayout>

```

Para la vista de esta actividad debemos implementar el patrón “Tablero de opciones” que explicamos en el cap 8.1.

En el caso que necesitemos cierta funcionalidad que no está presente en los componentes estándar de Android, nos vemos en la necesidad de crear nuestros propios controles personalizados, diseñados a medida de nuestros requisitos.

Android admite crear controles personalizados y permite hacerlo de diferentes formas:

- 1 Extendiendo la funcionalidad de un control ya existente.
- 2 Combinando varios controles para formar otro más complejo.
- 3 Diseñando desde cero un nuevo control.

Para este caso, nosotros diseñamos un componente llamado DashboardLayout que extiende de la clase ViewGroup como el resto de los Layout que provee Android. Este nuevo componente permite disponer en forma de cuadrícula las diferentes opciones del menú, adaptándose al tamaño y orientación de cada pantalla. Cada una de estas opciones está formada por un componente Button el cual representa a un botón en la pantalla. A través de las propiedades drawableTop y text se le asigna una imagen de fondo un texto respectivamente. Con la propiedad onClick definimos el nombre del método al que se llama cada vez que el botón es seleccionado. Estos métodos deben

definirse en la actividad correspondiente a la vista, en este caso la actividad MenuPrincipalActivity.java

Implementación

Como podemos ver en esta clase se implementan cada uno de los métodos que fueron asignados a los botones de la vista.

MenuPrincipalActivity.java

```
14 public class MenuPrincipalActivity extends MuseoActivity {
15
16
17     private Long idMuseo;
18
19     @Override
20     protected void inicializar() {
21         setEstado(new EstadoActividad(Estado.INICIANDO));
22     }
23
24
25
26     public void sobreElMuseo(View button) {
27         Intent intentAct = new Intent(getApplicationContext(), SobreElMuseoActivity.class);
28         intentAct.putExtra("idMuseo", getIdMuseo());
29         startActivity(intentAct);
30     }
31
32     public void verEncuesta(View button) {
33         Intent intentAct = new Intent(getApplicationContext(),
34             EncuestaActivity.class);
35         startActivity(intentAct);
36     }
37
38
39     public void verElemento(View button) {
40         Intent intent = new Intent("info.unlp.museum.SCAN");
41         intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
42         startActivityForResult(intent, 0);
43     }
44
45     public void verAyuda(View button) {
46         Intent intentAct = new Intent(getApplicationContext(), AyudaActivity.class);
47         startActivity(intentAct);
48     }
```

Como dijimos anteriormente, cuando un usuario hace clic en el botón, el sistema Android llama al método de la actividad nombreMetodo(View). Para que esto funcione, el método debe aceptar una vista como único parámetro. Esta vista corresponde al widget que fue seleccionado.

La creación y activación de una actividad es un proceso asíncrono que se realiza usando las funciones startActivity o startActivityForResult, heredadas de la superclase y que se diferencian en que la última permite obtener un resultado.

El método onClick() de los botones será el encargado de mandar un objeto de tipo Intent hacia la actividad que llamamos que se define como segundo parámetro en el constructor al momento de crearlo. Después, con ayuda de putExtra() pasamos cada uno de los valores que queremos que reciba la actividad. Este método recibe como primer parámetro una clave que se utiliza en la segunda actividad para recuperar la información, que corresponde al segundo parámetro.

En el caso del método `verElemento(View)` como necesitamos recibir la información de la actividad a la que llamamos y realizar algo con los datos que recibimos llamamos al método `startActivityForResult()`. En caso de que no esperemos alguna respuesta de la actividad secundaria, únicamente utilizaríamos el método `startActivity()`, como lo hicimos con el resto de los métodos.

Por último, para que podamos recibir satisfactoriamente la respuesta de la actividad secundaria, sobrescribimos el método `onActivityResult()` que se utiliza para recuperar un código de respuesta y realizar lo que corresponda de acuerdo a la respuesta recibida. El método `onActivityResult()` recibe como primer parámetro un `requestCode` que es un identificador que se definió líneas arriba en la actividad principal y que nos sirve para diferenciar qué `Intent` y qué actividad es la que se llamó. El segundo parámetro es un `resultCode` que se define en la actividad secundaria. Puede tomar el valor de `RESULT_OK` en caso de que todo haya ido bien al momento de procesar la información o `RESULT_CANCELED` si la actividad no ha obtenido algún resultado o si algo falló.

Integrar la librería ZXing

Vamos a explicar como configurar nuestro entorno de desarrollo para que nos sea posible trabajar con códigos QR. Hicimos uso de una librería llamada ZXing [71], un proyecto open source implementado en Java, que centra su capacidad en el uso de la cámara de los dispositivos para escanear y decodificar códigos de barras directamente en el dispositivo sin necesidad de comunicarse con el servidor.

Esta librería es la más popular y nos facilita mucho integrar la lectura de códigos a través de un `Intent`.

1 Descargar la librería ZXing: Nos descargamos el archivo `ZXing-2.0.zip` desde <http://code.google.com/p/zxing/downloads/list>. Lo descomprimos en algún directorio dentro de nuestro disco rígido. De todos los directorios que encontramos en la librería, los que nos ayudan a trabajar con aplicaciones Android son el directorio `/android` y `/core`.

2 Crear la librería ZXing: Como la versión 2.0 nos provee el archivo `core.jar` no es necesario generarlo como sí hacía falta en versiones anteriores. Entonces procedemos a crear el proyecto para poder crear la librería que posteriormente incorporaremos a nuestro proyecto. Seleccionamos la opción *File > New > Android Project*, seleccionamos la opción *Create project from existing source* y con el botón `Browse...` navegamos hasta el directorio donde descomprimos el archivo de ZXing y seleccionamos el directorio `android` que es un proyecto Android. El nombre que le pusimos al proyecto es `ZxingLib`. Damos clic en `Finish`.

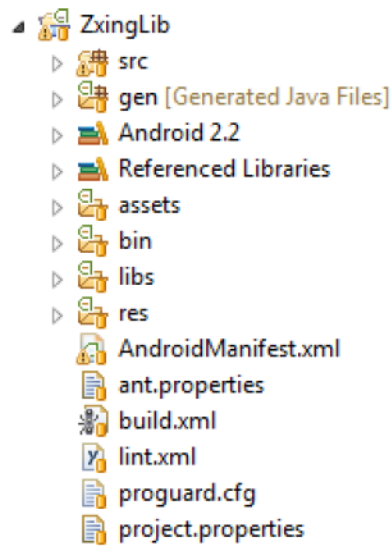


Figura 8.33: Estructura del proyecto ZxingLib

Hasta este momento, nuestro proyecto aún necesita importar la librería que es el archivo `core.jar` que se encuentra en el directorio `/core`. Damos clic derecho sobre el proyecto `ZxingLib` > `Properties` > `Java Build Path` > `Add External JARs...` y seleccionamos el archivo `core.jar`.

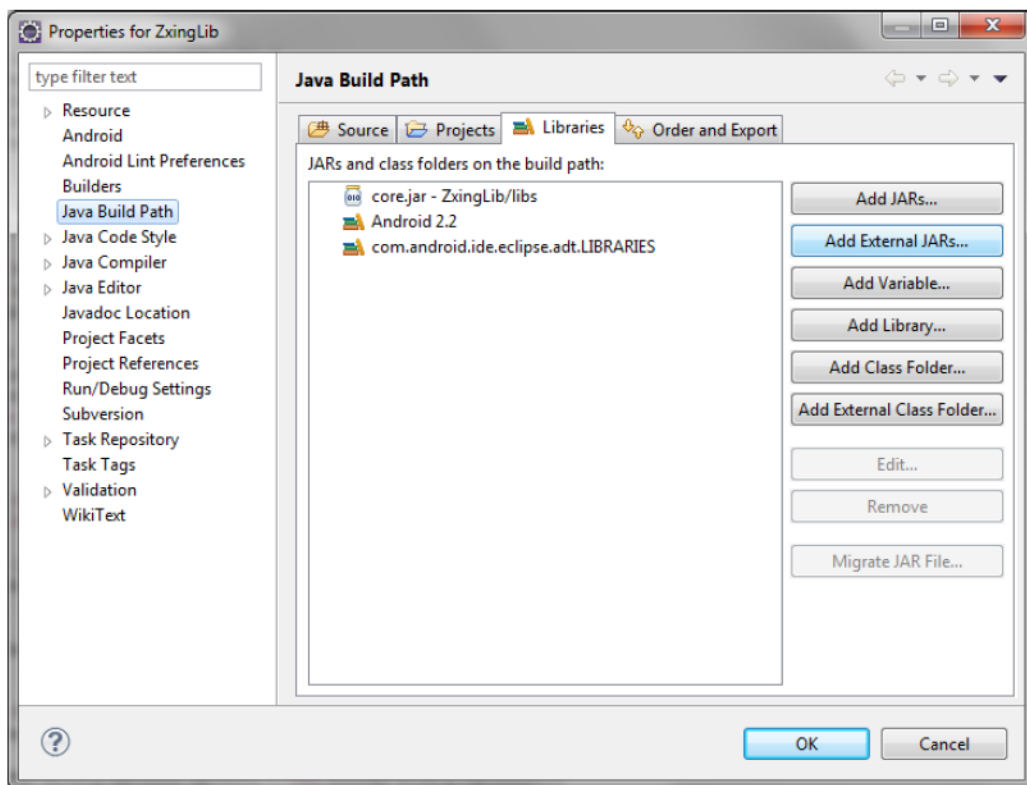


Figura 8.34: Importar librería

Luego debemos indicar que este proyecto es una librería. Para ello hacemos clic derecho sobre el proyecto `ZxingLib` > `Properties` > `Android` y tildamos la opción `Is Library`.

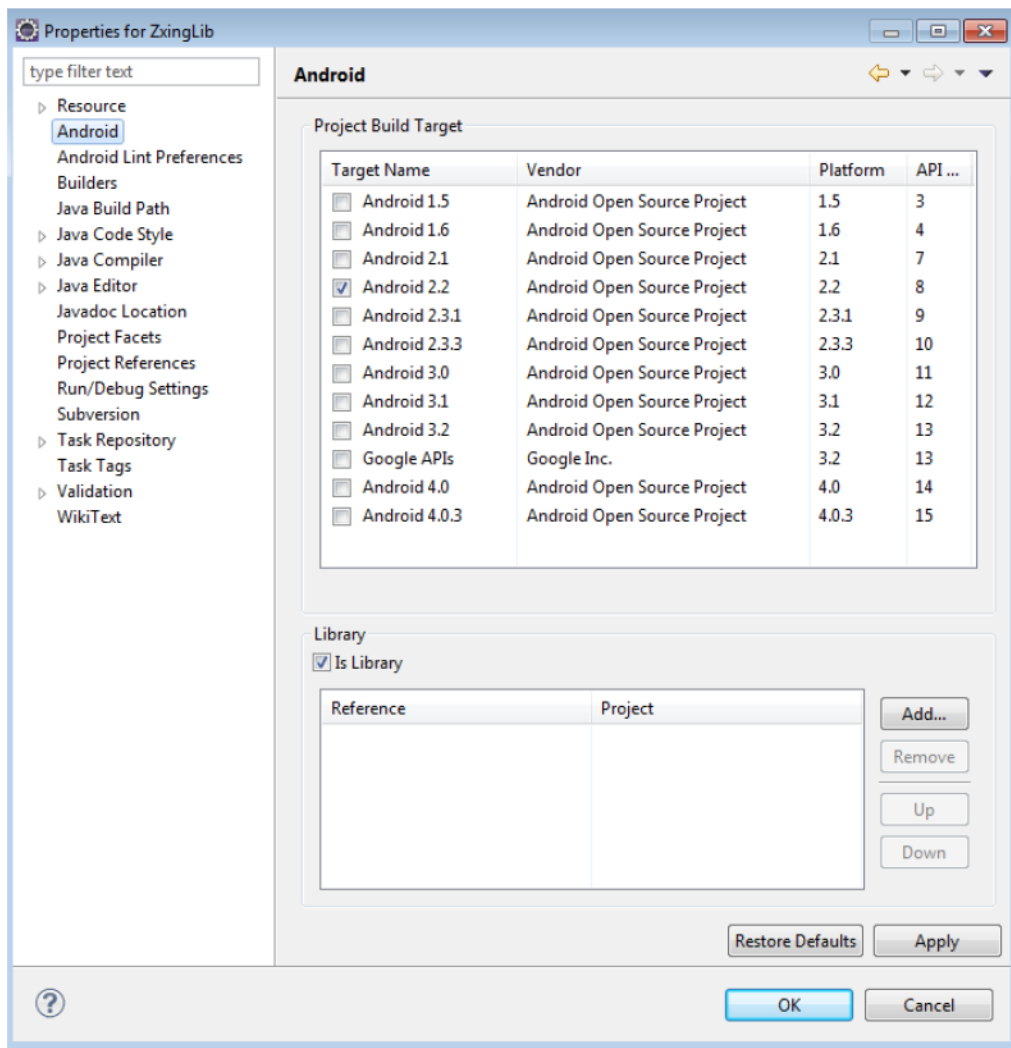


Figura 8.35: Indicar que es una librería

- Incorporar la librería a nuestro proyecto:** Una vez que tenemos creada la librería ya la podemos incorporar a nuestro proyecto. Para ello damos clic derecho sobre el *MuseumMobile* > *Properties* > *Android* y en la sección de librerías damos clic sobre el botón *Add...* dónde deberemos ver la opción de *ZxingLib* y la seleccionamos.

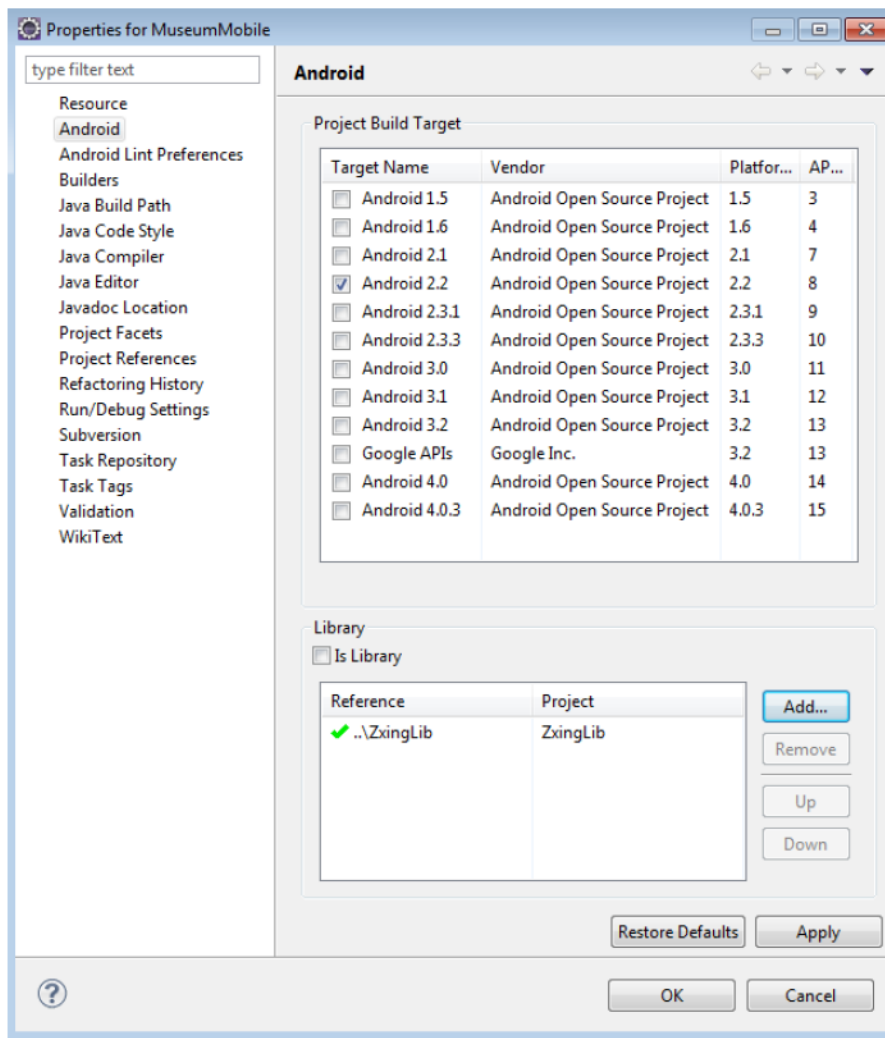


Figura 8.36: Incorporar la librería

También tenemos que añadir el core.jar al build-path, como hicimos con el proyecto de la librería ZXing.

Luego en el evento del botón que queremos que ejecute esta librería deberemos realizar lo siguiente:

```
Intent intent = new Intent("com.google.zxing.client.android.SCAN");
intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
startActivityForResult(intent, 0);
```

En Android cuando hay más de una aplicación que resuelve la acción que fue lanzada, en este caso sería com.google.zxing.client.android.SCAN, le muestra al usuario un listado de opciones para que elija una de ellas. Como nosotros queremos evitar esta situación decidimos cambiarle el nombre a la acción y la llamamos info.unlp.museum.SCAN. Por lo que el evento se ejecutará de la siguiente manera:

MenuPrincipalActivity.java

```

39 public void verElemento(View button) {
40     Intent intent = new Intent("info.unlp.museum.SCAN");
41     intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
42     startActivityForResult(intent, 0);
43 }
..

```

Luego en la misma actividad, como explicamos anteriormente cuando utilizamos el método `startActivityForResult`, necesitamos hacer lo siguiente para recuperar los resultados:

```

51 @Override
52 protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
53
54     if (requestCode == 0) {
55         if (resultCode == RESULT_OK) {
56             String contents = intent.getStringExtra("SCAN_RESULT");
57
58             QRParser qrParser = new QRParser(contents);
59             QRData qrData = null;
60             try {
61                 qrData = qrParser.parseQR();
62             } catch (JsonSyntaxException e) {
63                 crearDialogo(getResources().getString(R.string.msg_error_qr), ViewUtil.ERROR);
64             }
65
66             if (qrData != null){
67                 Intent intentAct = new Intent(getApplicationContext(),
68                     ElementoMuseoActivity.class);
69                 intentAct.putExtra("objetoLeido", contents);
70                 startActivity(intentAct);
71             }
72         } else if (resultCode == RESULT_CANCELED) {
73
74         }
75     }
76 }
77

```

Además, para que esto funcione correctamente tenemos que añadir en el archivo de manifiesto de la aplicación la información sobre el SCAN de ZXing y el permiso para utilizar la cámara:

```

12 <uses-permission android:name="android.permission.CAMERA" />

```

```

49 <activity
50     android:clearTaskOnLaunch="true"
51     android:stateNotNeeded="true"
52     android:configChanges="orientation|keyboardHidden"
53     android:name="com.google.zxing.client.android.CaptureActivity"
54     android:screenOrientation="Landscape"
55     android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
56     android:windowSoftInputMode="stateAlwaysHidden" >
57     <intent-filter >
58         <action android:name="android.intent.action.MAIN" />
59         <category android:name="android.intent.category.DEFAULT" />
60     </intent-filter>
61     <intent-filter >
62         <action android:name="info.unlp.museum.SCAN" />
63         <category android:name="android.intent.category.DEFAULT" />
64     </intent-filter>
65 </activity>
66 </application>
--

```

Como nosotros modificamos el nombre de la acción necesitamos modificar también el archivo de manifiesto de la librería ZxingLib para que pueda atender la acción info.unlp.museum.SCAN. Con todos estos pasos realizados ya disponemos dentro de nuestro proyecto de un lector de códigos QR.

4. Sobre el Museo



Figura 8.37: SobreEIMuseoActivity y MapaMuseoActivity

El objetivo de estas actividades es contar algo acerca del museo y visualizar un mapa interno del mismo. Para ello implementamos dos clases SobreEIMuseoActivity y MapaMuseoActivity. A continuación explicamos cada una de estas clases.

SobreEIMuseoActivity

Esta clase la utilizamos para mostrar un texto en el cual se describe la historia del museo y un botón a través del cual accedemos al mapa interno.

Vista de la actividad

sobre_el_museo.xml

```
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@+id/home_root"
4      android:orientation="vertical"
5      android:layout_width="fill_parent"
6      android:layout_height="fill_parent">
7
8      <include layout="@Layout/header"/>
9      <include layout="@Layout/encabezado"/>
10
11  <ScrollView
12      android:layout_width="fill_parent"
13      android:layout_height="0dip"
14      android:background="@drawable/gray_list"
15      android:fillViewport="true"
16      android:layout_weight="1"
17      android:padding="10dp" >
18
19  <FrameLayout
20      android:layout_width="fill_parent"
21      android:layout_height="wrap_content" >
22
23  <LinearLayout
24      xmlns:android="http://schemas.android.com/apk/res/android"
25      android:layout_width="fill_parent"
26      android:layout_height="fill_parent"
27      android:orientation="vertical" >
28      <TextView
29          android:id="@+id/historia"
30          android:layout_width="fill_parent"
31          android:layout_height="wrap_content"
32          android:layout_weight="1.0"
33          android:inputType="textMultiline"
34          style="@style/TextStyle"
35      />
36
37      <Button
38          android:id="@+id/verMapa"
39          android:layout_marginTop="10dp"
40          android:layout_width="wrap_content"
41          android:layout_height="wrap_content"
42          android:layout_gravity="center_horizontal"
43          android:text="@string/ver_mapa"
44          android:onClick="verMapa"/>
45
46  </LinearLayout>
47  <include layout="@Layout/cargando"/>
48
49  </FrameLayout>
50
51  </ScrollView>
52  <include layout="@Layout/footer"/>
53
54  </LinearLayout>
```

Cuando trabajamos en espacios pequeños como lo son las pantallas de los teléfonos, debemos presentar la información al usuario de una forma cómoda, atractiva y práctica sin importar lo limitado que sea el espacio con el que contemos. Una forma de conseguir esto es el uso del scroll, que nos permite mostrar sólo una parte de la información total que nos interesa mostrarle al usuario y lo que resta se mostrará conforme el usuario se desplace hacia arriba o hacia abajo, según sea el caso.

Aquí utilizamos el ScrollView, que es el contenedor que ofrece una barra de desplazamiento para el contenido que pongamos dentro de él. Es muy útil en los casos en los que el diseño pueda ser

demasiado grande para algunas pantallas. La solución será envolver este contenido en un ScrollView. Lo que pasará después es que el usuario sólo verá parte del diseño a la vez, y el resto estará disponible a través del desplazamiento.

El resto de los componentes que utilizamos ya los explicamos con anterioridad por lo que omitimos su explicación en este caso.

Implementación

Esta clase extiende de la clase MuseoActivity. En el método inicializar podemos ver que seteamos un flag sobre la ventana de la aplicación. El flag que seteamos es FLAG_KEEP_SCREEN_ON que nos permite mantener la pantalla del dispositivo encendida y brillante.

SobreELMuseoActivity.java

```
21 public class SobreELMuseoActivity extends MuseoActivity{
22
23     private Museo museo;
24
25     @Override
26     protected void inicializar() {
27
28         getWindow().setFlags(android.view.WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON,
29             android.view.WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
30
31         TextView titulo = (TextView) findViewById(R.id.txtTitulo);
32         titulo.setText(getResources().getString(R.string.boton_historia_museo));
33
34         if ( getObjeto() instanceof EstadoActividad ) {
35             setEstado((EstadoActividad) getObjeto());
36
37             switch (getEstado().getEstado()) {
38                 case EJECUTANDO:
39                     getTask().setActivity(this);
40                     break;
41                 case INACTIVO:
42                     ocultarEspera();
43                     ((Button)findViewById(R.id.verMapa)).setEnabled(Boolean.FALSE);
44                     break;
45             }
46         } else {
47             getMuseo();
48         }
49     }
50
51
52     private void getMuseo() {
53         cambiarEstado(Estado.EJECUTANDO);
54         setTask(new ServidorAsyncTask(this));
55         HashMap<String, Object> p = new HashMap<String, Object>();
56         p.put("idMuseo", getSharedPreferences().getIdMuseo());
57         ejecutarTarea(Task.GET_MUSEO, p);
58     }
}
```

Luego en el método OnDestroy debemos eliminarla de la siguiente manera:

SobreELMuseoActivity.java

```
117     @Override
118     protected void onDestroy() {
119         super.onDestroy();
120
121         getWindow().clearFlags(android.view.WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
122     }
}
```

Para la comunicación con el servidor, utilizamos la tarea asincrónica ServidorAsyncTask al la cual le solicitamos la información del museo con la opción GET_MUSEO a la que le debemos pasar como parámetro el id del museo que el usuario seleccionó al iniciar la aplicación. Como este id lo

tenemos almacenado en las preferencias de la aplicación se lo solicitamos a la clase SharedPreferences

ServidorAsyncTask.java

```
30         case GET_MUSEO:
31             getRespuestas().put("sobreElMuseoHTTP", cliente.getMuseo((Integer) getParametros().get("idMuseo")));
32             break;
```

La tarea realiza la petición al ClienteHTTP junto con el idMuseo que recibió de la actividad.

ClienteHTTP.java

```
187 public SobreElMuseoHTTP getMuseo(Integer id) {
188
189     SobreElMuseoHTTP sobreElMuseoHTTP = new SobreElMuseoHTTP();
190
191     if (hayConexion()) {
192
193         ConexionHTTP con = new ConexionHTTP(URL + "/museum/sobreElMuseo.do");
194         con.addParam("id", id.toString());
195
196         try {
197             con.execute(RequestMethod.POST);
198         } catch (Exception e) {
199             e.printStackTrace();
200         }
201
202         String json_response = con.getResponse();
203         String jsonAux = null;
204         try {
205
206             jsonAux = new String(json_response.getBytes("iso-8859-1"), "UTF-8");
207
208         } catch (Exception e) {
209
210             e.printStackTrace();
211         }
212         sobreElMuseoHTTP = getGson().fromJson(jsonAux,
213             SobreElMuseoHTTP.class);
214     } else {
215         sobreElMuseoHTTP.setErrorCode(ErrorCode.NO_HAY_INTERNET);
216     }
217
218     return sobreElMuseoHTTP;
219 }
```

En el método getMuseo() de la clase ClienteHTTP creamos un objeto ConexionHTTP con la url MuseumWeb/museo/sobreElMuseo.do, correspondiente al requerimiento que devuelve un objeto SobreElMuseoHTTP. Este objeto contiene un código de error y un mensaje en caso de que ocurra algún error en la comunicación con el servidor y un objeto de tipo Museo, el cual contiene la historia y la url del mapa. Nuevamente como la respuesta que recibe es de tipo JSON es necesario realizar un conversión a su representación en objetos JAVA.

Una vez que realizó la petición a la tarea asincrónica verifica el resultado de la respuesta. En caso que haya algún error se comporta de la misma manera que explicamos en la implementación de clases anteriores. En el caso que la respuesta sea correcta seteamos el texto de la vista con el contenido de la historia del museo y se la mostramos al usuario.

SobreElMuseoActivity.java

```

61 @Override
62 public void recibirRespuesta() {
63
64     SobreElMuseoHTTP respuesta = (SobreElMuseoHTTP) getTask()
65         .getRespuestas().get("sobreElMuseoHTTP");
66
67     switch (respuesta.getErrorCode()) {
68     case ErrorCode.NO_HAY_INTERNET:
69         noHayConexion(1, getResources().getString(R.string.no_hay_internet));
70         break;
71     case ErrorCode.RESPUESTA_CORRECTA:
72         mostrarDatos(respuesta);
73         break;
74     case ErrorCode.ERROR_SERVIDOR:
75         crearDialogo(respuesta.getMessageError(), ViewUtil.ERROR);
76         ((Button) findViewById(R.id.verMapa)).setEnabled(Boolean.FALSE);
77         break;
78     }
79
80     ocultarEspera();
81
82 }
83
84 private void mostrarDatos(SobreElMuseoHTTP respuesta) {
85     setSobreElMuseo(respuesta.getMuseo());
86     TextView historia = (TextView) findViewById(R.id.historia);
87     historia.setText(respuesta.getMuseo().getHistoria());
88 }

```

Como dijimos anteriormente la vista también contiene un botón. El evento que se ejecuta cuando lo seleccionamos es el siguiente:

SobreELMuseoActivity.java

```

90 public void verMapa(View button) {
91     Intent intentAct = new Intent(getApplicationContext(),
92         MapaMuseoActivity.class);
93     intentAct.putExtra("urlMapa", getSobreElMuseo().getUrlMapa());
94     startActivity(intentAct);
95 }

```

A través de este evento llamamos a la actividad MapaMuseoActivity y le pasamos la url del mapa del museo.

MapaMuseoActivity

Esta clase se limita a visualizar el mapa del museo por lo que la vista consiste principalmente en un ImageView utilizado para mostrar la imagen.

Vista de la actividad

mapa_museo.xml

```

2② <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:background="@drawable/gray_list"
6     android:gravity="center">
7     <ImageView
8         android:id="@+id/mapa"
9         android:scaleType="fitCenter"
10        android:layout_width="wrap_content"
11        android:layout_height="wrap_content"
12        android:layout_gravity="center"
13        android:contentDescription="@string/ver_mapa"
14    />
15    <include layout="@layout/cargando"/>
16 </FrameLayout>

```

Implementación

Esta clase también extiende de la clase MuseoActivity, pero aquí necesitamos utilizar la tarea asincrónica BatchImageDownloader para obtener la imagen desde el servidor que corresponde a la url que recibió de la actividad anterior. Como hemos mencionado, para solicitar una imagen a la tarea se utiliza la opción GET_IMAGENES a la que le debemos indicar la url y el tamaño con el que queremos que se visualice. En este caso será el tamaño de la pantalla, que los obtenemos a través del objeto Display que nos provee el manejador de ventanas de Android.

Una vez que recibimos la respuesta de la tarea seteamos el Bitmap que recibimos en el componente ImageView de la vista para mostrársela al usuario.

MapaMuseoActivity.java


```

23 public class MapaMuseoActivity extends MuseoActivity {
24
25     private String url;
26
27     @Override
28     protected void inicializar() {
29
30         Intent intent = getIntent();
31         url = intent.getExtras().getString("urlMapa");
32
33         if (getObjeto() instanceof EstadoActividad) {
34             setEstado((EstadoActividad) getObjeto());
35
36             switch (getEstado().getEstado()) {
37                 case EJECUTANDO:
38                     getTask().setActivity(this);
39                     break;
40                 case INACTIVO:
41                     ocultarEspera();
42                     ImageView imageView = (ImageView) findViewById(R.id.mapa);
43                     imageView.setImageBitmap((Bitmap) getEstado().getCacheImagen()
44                                             .getImagenes().get(url));
45                     break;
46             }
47         } else {
48             getMapa();
49         }
50     }
51
52     private void getMapa() {
53         cambiarEstado(Estado.EJECUTANDO);
54         setTask(new BatchImageDownloader(this));
55         HashMap<String, Object> p = new HashMap<String, Object>();
56
57         ArrayList<String> urls = new ArrayList<String>();
58         urls.add(url);
59         p.put("urls", urls);
60
61         Display display = getWindowManager().getDefaultDisplay();
62         p.put("width", display.getWidth());
63         p.put("height", display.getHeight());
64
65         ejecutarTarea(Task.GET_IMAGENES, p);
66     }
67 }

```

5. Explora



Figura 8.38: ElementoMuseoActivity

Podemos decir que ésta es la pantalla principal de la aplicación que junto con el Menú Principal nos permiten llevar a cabo el proceso de Realidad Aumentada. Como explicamos en el Menú Principal, cuando el usuario seleccione la opción Explora del menú se ejecutará la tarea dedicada a la lectura del código QR. Una vez que leyó el código procesamos el resultado que recibimos de la lectura, la cual representa el id del objeto sobre el cual está interactuando el usuario. Una vez que obtenemos esa información desde la clase MenuPrincipalActivity llamamos a la actividad ElementoMuseoActivity a la cual le pasamos estos datos.

A continuación explicamos el comportamiento de la clase ElementoMuseoActivity y todos los componentes que incluye los cuales nos permiten explorar el objeto y conocer más detalles acerca de este, tales como su nombre, categoría a la que pertenece, su descripción, imágenes, videos y juegos.

Vista de la actividad

La vista de esta actividad además de contener el encabezado y pie de página como el resto de las actividades, está compuesta por dos partes principales:

- Un encabezado para el objeto, donde se indica el nombre y la categoría del mismo.
- Pestañas, que como dijimos anteriormente nos permiten dividir en diferentes secciones la información.

elemento_museo.xml

```

15<LinearLayout android:id="@+id/elemento_leido"
16    android:orientation="vertical"
17    android:layout_width="fill_parent"
18    android:layout_height="wrap_content"
19    android:background="@drawable/white"
20    >
21<LinearLayout
22    android:layout_width="fill_parent"
23    android:layout_height="wrap_content"
24    android:background="@drawable/background_encabezado"
25    android:orientation="vertical">
26    <TextView android:id="@+id/txt_nombre_elemento"
27        android:layout_width='fill_parent'
28        android:layout_height='fill_parent'
29        android:gravity='left'
30        style="@style/FontElemento"
31    />
32    <TextView android:id="@+id/txt_categoria_elemento"
33        android:layout_width='fill_parent'
34        android:layout_height='fill_parent'
35        android:gravity='left'
36        style="@style/FontCategoria"
37        android:clickable="true"
38        android:onClick="verCategoria"
39        android:textColor="@drawable/texto_categoria_states"
40    />
41
42</LinearLayout>
43<TabHost
44    xmlns:android="http://schemas.android.com/apk/res/android"
45    android:id="@android:id/tabhost"
46    android:layout_width="fill_parent"
47    android:layout_height="@0dip"
48    android:layout_weight="1">
49<LinearLayout
50    android:orientation="vertical"
51    android:layout_width="fill_parent"
52    android:layout_height="fill_parent">
53    <TabWidget
54        android:id="@android:id/tabs"
55        android:layout_width="fill_parent"
56        android:layout_height="wrap_content"
57    />
58    <FrameLayout
59        android:id="@android:id/tabcontent"
60        android:layout_width="fill_parent"
61        android:layout_height="fill_parent" />
62</LinearLayout>
63</TabHost>

```

Para mostrar el nombre y la categoría simplemente utilizamos un `LinearLayout` que contiene dos `TextView`. Para la implementación de las pestañas o secciones utilizamos los componentes `TabHost`, `TabWidget` y `FrameLayout`.

La definición de un `TabHost` es contenedor para una vista de ventanas con pestañas. Este objeto tiene dos hijos: un conjunto de etiquetas en las que el usuario hace clic para seleccionar una pestaña específica y un objeto `FrameLayout` que muestra el contenido de esta pestaña.

El `TabHost` es el elemento raíz o principal del layout de la actividad principal, el cual esta dividió en 2 partes principales y personalizables, la primera es donde estarán las pestañas que vamos a

utilizar en nuestra aplicación, la cual se llama TabWidget. La segunda parte principal es donde estará el contenido que realiza cada pestaña y el cual es un FrameLayout.

Un TabWidget es un atributo que muestra una lista de las etiquetas de la pestaña que representa cada página. El objeto contenedor de este Widget es TabHost. Cuando el usuario selecciona una pestaña, este objeto envía un mensaje al contenedor principal, TabHost, para indicarle que debe cambiar la página mostrada. Por lo general no se utilizan muchos métodos directamente en este objeto. El contenedor TabHost se utiliza para agregar etiquetas y gestión de devoluciones de llamada.

Así que básicamente un TabWidget es un elemento que contiene las pestañas con las cuales se interactúa cuando la aplicación está corriendo.

Por último, ya comentamos que un FrameLayout está diseñado para bloquear un área en la pantalla para mostrar un solo elemento a la vez.

Implementación

Para poder implementar las pestañas la clase ElementoMuseoActivity debe extender de la clase TabActivity. Como también necesitamos realizar comunicación con el servidor deberá implementar la interfaz I_ActivityTask.

ElementoMuseoActivity.java

```
35 public class ElementoMuseoActivity extends TabActivity implements I_ActivityTask{
36
37     private EstadoActividad estado;
38
39     private ObjetoMuseo objetoMuseo;
40
41     private SharedPreferences sharedPreferences;
42
43     private Dialog dialog;
44
45     private Object objeto;
46
```

Lo primero que debemos hacer en esta actividad es obtener los datos del objeto que el usuario se encuentra explorando. Para ello le realizamos la petición BUSCAR_ELEMENTO a la tarea asincrónica ServidorAsyncTask y le pasamos como parámetro los datos que obtuvimos del código QR.

ElementoMuseoActivity.java

```

85 private void buscarElemento() {
86     String qrData = getIntent().getStringExtra("objetoLeido");
87     if (qrData != null){
88         cambiarEstado(Estado.EJECUTANDO);
89         setTask(new ServidorAsyncTask(this));
90         HashMap<String, Object> p = new HashMap<String, Object>();
91         p.put("qrdata", new QRParser(qrData).parseQR());
92         ejecutarTarea(Task.BUSCAR_ELEMENTO, p);
93     }
94 }

```

El ServidorAsyncTask realiza la petición al ClienteHTTP de la siguiente manera:

ServidorAsyncTask.java

```

21     case BUSCAR_ELEMENTO:
22         getRespuestas().put("respuestaHTTP", cliente.getObjetoMuseo((QRData) getParametros().get("qrdata")));
23         break;

```

Luego, en el método getObjetoMuseo() de la clase ClienteHTTP creamos un objeto ConexionHTTP con la url MuseumWeb/museo/recuperarDatosObjeto.do, correspondiente al requerimiento que devuelve un objeto RespuestaHTTP. Este objeto contiene un código de error y un mensaje en caso de que ocurra algún error en la comunicación con el servidor y un objeto de tipo ObjetoMuseo, el cual contiene el nombre del objeto, su categoría y el listado de los materiales que dispone, tales como un texto que corresponde a la descripción del objeto, audio, imágenes, videos y juegos.

ClienteHTTP.java

```

45 public RespuestaHTTP getObjetoMuseo(QRData qrData) {
46     RespuestaHTTP respuesta = new RespuestaHTTP();
47
48     if (hayConexion()) {
49         ConexionHTTP con = new ConexionHTTP(URL
50             + "/museum/recuperarDatosObjeto.do");
51         con.addParam("idObjeto", qrData.getIdObjeto().toString());
52
53         try {
54             con.execute(RequestMethod.POST);
55         } catch (Exception e) {
56             e.printStackTrace();
57         }
58
59         String json_response = con.getResponse();
60         respuesta = getGson().fromJson(json_response, RespuestaHTTP.class);
61
62     } else {
63         respuesta.setErrorCode(ErrorCode.NO_HAY_INTERNET);
64     }
65     return respuesta;
66 }

```

Una vez que recibe la respuesta del servidor procedemos a crear las distintas pestañas que corresponden a cada una de las secciones y seteamos los valores correspondientes al nombre y la categoría en los TextView respectivos.

Además, guardamos los datos del objeto a través de SharedPreferences para tener disponible la información del objeto en el resto de las actividades.

```

114 TabHost tabHost = getTabHost();
115 tabHost.getCurrentTab();
116 String objetoJson = new Gson().toJson(respuesta.getObjetoMuseo());
117 getSharedPreferences().guardarObjeto(objetoJson);
118 objetoMuseo = respuesta.getObjetoMuseo();
119
120 TextView txtNombre = (TextView) findViewById(R.id.txt_nombre_elemento);
121 txtNombre.setText(objetoMuseo.getNombre());
122
123 TextView txtCategoria = (TextView) findViewById(R.id.txt_categoria_elemento);
124 txtCategoria.setText(objetoMuseo.getCategoria().getNombre());
125
126 tabHost.clearAllTabs();
127 TabHost.TabSpec spec;
128 Intent intent;
129 Resources resources = getResources();
130
131 intent = new Intent().setClass(this, TextoActivity.class);
132 spec = tabHost
133     .newTabSpec(resources.getString(R.string.texto))
134     .setIndicator(resources.getString(R.string.texto),
135                 resources.getDrawable(R.drawable.texto))
136     .setContent(intent);
137 tabHost.addTab(spec);
138
139 intent = new Intent().setClass(this, AudioActivity.class);
140 spec = tabHost
141     .newTabSpec(resources.getString(R.string.audio))
142     .setIndicator(resources.getString(R.string.audio),
143                 resources.getDrawable(R.drawable.audio))
144     .setContent(intent);
145 tabHost.addTab(spec);
146
147 intent = new Intent().setClass(this, ImagenActivity.class);
148 spec = tabHost
149     .newTabSpec(resources.getString(R.string.imagenes))
150     .setIndicator(resources.getString(R.string.imagenes),
151                 resources.getDrawable(R.drawable.imagen))
152     .setContent(intent);
153 tabHost.addTab(spec);
154
155 intent = new Intent().setClass(this, VideoActivity.class);
156 spec = tabHost
157     .newTabSpec(resources.getString(R.string.videos))
158     .setIndicator(resources.getString(R.string.videos),
159                 resources.getDrawable(R.drawable.video))
160     .setContent(intent);
161 tabHost.addTab(spec);
162
163 intent = new Intent().setClass(this, JuegoActivity.class);
164 spec = tabHost
165     .newTabSpec(resources.getString(R.string.juegos))
166     .setIndicator(resources.getString(R.string.juegos),
167                 resources.getDrawable(R.drawable.juego))
168     .setContent(intent);
169 tabHost.addTab(spec);

```

Para cada una de las pestañas debemos indicar el Intent que utilizaremos en cada caso, es decir, a qué actividad debemos llamar cada vez que el usuario seleccione una pestaña. A continuación explicaremos la función de cada una de estas pestañas:

5.1 Texto

Esta sección se implementó con la clase `TextoActivity`, que tiene como objetivo mostrar la descripción del objeto en cuestión.

Vista de la actividad

La vista de la actividad consiste simplemente en un TextView contenido por un ScrollView para los casos en los que el texto no entra completamente en la pantalla del dispositivo.

```
20 <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:fillViewport="true"
6     android:background="@drawable/gray_list"
7     android:gravity="center"
8     android:padding="10dp"
9     >
10
11     <TextView
12         android:id="@+id/textObjeto"
13         android:layout_width="fill_parent"
14         android:layout_height="wrap_content"
15         android:inputType="textMultiLine"
16         style="@style/TextStyle"
17     />
18
19
20 </ScrollView>
```

Implementación

Como en esta clase no debemos realizar una comunicación con el servidor ya que disponemos de la descripción del objeto en las preferencias de la aplicación, simplemente extiende de la clase Activity.

TextoActivity.java

```

14 public class TextoActivity extends Activity {
15
16     private SharedPreferences sharedPreferences;
17
18     @Override
19     public void onCreate(Bundle savedInstanceState) {
20
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.texto);
23
24         getWindow().setFlags(android.view.WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON,
25                             android.view.WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
26
27         sharedPreferences = new SharedPreferences(getApplicationContext());
28
29         String descripcion = null;
30
31         String objetoMuseo = sharedPreferences.getObjetoMuseo();
32
33         ObjetoMuseo obj = (new Gson()).fromJson(objetoMuseo, ObjetoMuseo.class);
34
35         for (MaterialMuseo elemento : obj.getMateriales()) {
36             if (elemento.getTipo().getNombre().equals(ConstantesMuseo.TIPO_TEXTO)){
37                 descripcion = elemento.getDescripcion();
38             }
39         }
40
41         TextView edit = (TextView) findViewById(R.id.textObjeto);
42
43         if (descripcion == null){
44             edit.setText(getResources().getString(R.string.no_hay_descripcion));
45         } else{
46             edit.setText(descripcion);
47         }
48     }

```

En esta actividad también hacemos uso del flag `FLAG_KEEP_SCREEN_ON` el cual le permitirá al usuario leer cómodamente el texto sin que la pantalla se le apague.

Finalmente lo que realizamos en esta clase es obtener la información del objeto guardado en las preferencias de la aplicación, buscamos el material correspondiente al texto del mismo y seteamos su valor en el componente `TextView` para que sea visible al usuario. En caso de que el texto no existe se muestra un valor por defecto indicándole al usuario que no se dispone de la descripción del objeto.

5.2 Audio



Figura 8.39: AudioActivity

En esta sección el usuario puede escuchar la descripción del objeto a través de la reproducción de un audio.

Vista de la actividad

La vista la dividimos en tres secciones principales: una barra de progreso, para indicar el progreso del audio, una sección para los botones de reproducción y una barra para aumentar o disminuir el volumen.

Para implementar la barra de progreso, tanto para el progreso del audio como para el volumen, utilizamos el componente SeekBar.

audio.xml

```

21         <SeekBar
22             android:id="@+id/seekbar"
23             android:layout_width="fill_parent"
24             android:layout_height="wrap_content"
25             android:max="100"
26             android:paddingBottom="10dip"
27             android:progressDrawable="@drawable/progress"
28             android:thumb="@drawable/thumb_states"
29             android:thumbOffset="0dip"
30             android:layout_marginLeft="10dp"
31             android:layout_marginRight="10dp"
32         />

```

Un SeekBar deriva de la clase ProgressBar y le agrega un indicador (thumb) de posición. El usuario puede seleccionar el indicador y arrastrar hacia la izquierda o hacia la derecha para ajustar el nivel de progreso actual. Lo que nos permitía en este caso adelantar o retrasar la pista de audio o aumentar o bajar el volumen.

Implementación

Para implementar esta actividad utilizamos la clase `AudioActivity` que al igual que la actividad anterior extiende de la clase `Activity`.

```
28 public class AudioActivity extends Activity {
29
30     private SharedPreferences sharedPreferences;
31     private SeekBar seekbar = null;
32     private MediaPlayer player = null;
33     private AudioManager audioManager;
```

Las dos API's que Android provee para trabajar con sonidos son `SoundPool` y `MediaPlayer`. Podemos utilizar `SoundPool` para reproducir pequeñas pistas de audio. Con esta clase podemos repetir la reproducción de sonidos y hasta reproducir múltiples sonidos de manera simultánea. Un dato importante que hay que tomar en cuenta cuando se trabaja con esta clase, es que los archivos de sonido que se quiera reproducir no deben sobrepasar 1 MB de tamaño. Pero cuando queramos trabajar con pistas de audio más largas debemos utilizar la clase `MediaPlayer`. En el método `onCreate()` de la actividad inicializamos cada uno de los componentes.

AudioActivity.java

```
95     seekbar = (SeekBar) findViewById(R.id.seekbar);
96     seekbar.setProgress(0);
97     seekbar.setOnSeekBarChangeListener(seekBarChanged);
98
99     playButton = (ImageButton) findViewById(R.id.play);
100    prevButton = (ImageButton) findViewById(R.id.prev);
101    nextButton = (ImageButton) findViewById(R.id.next);
102
103    playButton.setOnClickListener(onButtonClick);
104    nextButton.setOnClickListener(onButtonClick);
105    prevButton.setOnClickListener(onButtonClick);
106
107    player = new MediaPlayer();
108    player.setOnCompletionListener(onCompletion);
109    player.setOnErrorListener(onError);
110
111    setVolumeControlStream(AudioManager.STREAM_MUSIC);
112    audioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
113    int maxVolume = audioManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
114    int curVolume = audioManager.getStreamVolume(AudioManager.STREAM_MUSIC);
115    SeekBar volControl = (SeekBar) findViewById(R.id.seekBar1);
116    volControl.setMax(maxVolume);
117    volControl.setProgress(curVolume);
118    volControl.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
119        public void onStopTrackingTouch(SeekBar arg0) {
120        }
121
122        public void onStartTrackingTouch(SeekBar arg0) {
123        }
124
125        public void onProgressChanged(SeekBar arg0, int arg1,
126            boolean arg2) {
127            audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, arg1, 0);
128        }
129    });
```

Para el caso del `SeekBar` inicializamos la barra de progreso en cero y le asignamos el escuchador `seekBarChange` para que atienda el evento cada vez que el usuario mueve el indicador de la barra hacia la izquierda o derecha y actualice el reproductor a la posición correcta.

AudioActivity.java

```

326 private SeekBar.OnSeekBarChangeListener seekBarChanged = new SeekBar.OnSeekBarChangeListener()
327
328     public void onStopTrackingTouch(SeekBar seekBar) {
329         isMoveingSeekBar = false;
330     }
331
332     public void onStartTrackingTouch(SeekBar seekBar) {
333         isMoveingSeekBar = true;
334     }
335
336     public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
337         if(isMoveingSeekBar){
338             player.seekTo(progress);
339         }
340     }
341 };

```

Le indicamos a los botones del reproductor para que atiendan lo eventos iniciar, adelantar o retrasar la pista de audio. Dependiendo del botón que es seleccionado atenderá el evento adecuado.

AudioActivity.java

```

251 private View.OnClickListener onClick = new View.OnClickListener() {
252     public void onClick(View v) {
253         switch(v.getId()){
254             case R.id.play:{
255                 if(player.isPlaying()){
256                     handler.removeCallbacks(updatePositionRunnable);
257                     player.pause();
258                     playButton.setImageResource(R.drawable.media_play_states);
259                 }
260                 else{
261                     if(isStarted){
262                         player.start();
263                         playButton.setImageResource(R.drawable.media_pause_states);
264                         updatePosition();
265                     }else{
266                         startPlay(url);
267                     }
268                 }
269                 break;
270             }
271             case R.id.next:{
272                 if (isStarted){
273                     int seekto = player.getCurrentPosition() + STEP_VALUE;
274                     if(seekto > player.getDuration())
275                         seekto = player.getDuration();
276                     if (player.isPlaying()){
277                         player.pause();
278                         player.seekTo(seekto);
279                         player.start();
280                     }else{
281                         player.seekTo(seekto);
282                         seekbar.setProgress(player.getCurrentPosition());
283                         ((TextView) findViewById(R.id.textTiempoActual)).setText(
284                             utils.millisSecondsToTimer(player.getCurrentPosition()));
285                     }
286                 }
287                 break;
288             }
289             case R.id.prev:{
290                 if (isStarted){
291                     int seekto = player.getCurrentPosition() - STEP_VALUE;
292                     if(seekto < 0)
293                         seekto = 0;
294                     if (player.isPlaying()){
295                         player.pause();
296                         player.seekTo(seekto);
297                         player.start();
298                     }else{
299                         player.seekTo(seekto);
300                         seekbar.setProgress(player.getCurrentPosition());
301                         ((TextView) findViewById(R.id.textTiempoActual)).setText(
302                             utils.millisSecondsToTimer(player.getCurrentPosition()));
303                     }
304                 }
305                 break;
306             }
307         }
308     }
309 }

```

Luego inicializamos el reproductor creando una instancia de la clase MediaPlayer, al que le asignamos un escuchador en caso de que ocurra algún error y otro para que cuando el audio se haya completado podamos reiniciar el reproductor nuevamente.

AudioActivity.java

```

309 private MediaPlayer.OnCompletionListener onCompletion = new MediaPlayer.OnCompletionListener() {
310     public void onCompletion(MediaPlayer mp) {
311         stopPlay();
312     }
313 };
314
315 private MediaPlayer.OnErrorListener onError = new MediaPlayer.OnErrorListener() {
316     public boolean onError(MediaPlayer mp, int what, int extra) {
317         return false;
318     }
319 };

```

Por último, para controlar el volumen del audio lo hacemos a través de la instancia del AudioManager a la que le solicitamos los valores del volumen actual y volumen máximo para inicializar el SeekBar encargado de manejar el volumen. En este caso, cada vez que el usuario

mueva el indicador de la barra se actualizará el volumen a través del método `setStreamVolume()` del `AudioManager`.

Una vez que tenemos los componentes inicializados y el usuario inicia el audio debemos preparar el reproductor con la url que se corresponde al audio del objeto. Para ello, utilizamos el método `setDataSource()` al que le pasamos como parámetro la url, preparamos el reproductor de forma síncrona a través del método `prepare()` el cual puede durar unos segundos y por último le decimos al reproductor que comience con la reproducción con el método `start()`. Una vez iniciado, se actualiza el texto con la duración total y le indicamos al `seekBar` la duración máxima.

AudioActivity.java

```
207 private void startPlay(String url) {
208     seekbar.setProgress(0);
209     player.stop();
210     player.reset();
211     try {
212         player.setDataSource(url);
213         player.prepare();
214         player.start();
215         ((TextView) findViewById(R.id.textDuracionTotal)).setText(utils
216             .millisecondsToTimer(player.getDuration()));
217     } catch (IllegalArgumentException e) {
218         e.printStackTrace();
219     } catch (IllegalStateException e) {
220         e.printStackTrace();
221     } catch (IOException e) {
222         e.printStackTrace();
223     }
224     seekbar.setMax(player.getDuration());
225     playButton.setImageResource(R.drawable.media_pause_states);
226     updatePosition();
227     isStarted = true;
228 }
```

El método `updatePosition()` se utiliza para actualizar la barra de progreso y el texto para indicar la duración actual a medida que el audio avanza. El objeto `Handler` nos permite ejecutar tareas en segundo plano de una manera muy sencilla. Para que lo entendamos mejor un `handler` es el puente que hay entre un hilo secundario (`thread`) y el hilo principal (aplicación). Como nosotros deseamos que transcurridos cierto tiempo se actualice automáticamente el progreso del audio en la interfaz del usuario, le indicamos al objeto `Handler` a través del método `postDelayed` que ejecute un objeto `Runnable` con cierta frecuencia. El objeto `Runnable` simplemente vuelve a llamar al método `updatePosition()`, logrando de esta manera que la interfaz se actualice constantemente hasta su finalización.

AudioActivity.java

```
247 private void updatePosition() {
248     handler.removeCallbacks(updatePositionRunnable);
249     seekbar.setProgress(player.getCurrentPosition());
250     ((TextView) findViewById(R.id.textTiempoActual)).setText(utils
251         .millisecondsToTimer(player.getCurrentPosition()));
252     handler.postDelayed(updatePositionRunnable, UPDATE_FREQUENCY);
253 }
```

En el método `stopPlay()` principalmente lo que hacemos es detener la reproducción del audio con todo lo que ello implica y lo reiniciamos.

AudioActivity.java

```
230 private void stopPlay() {
231     if (player != null) {
232         estadoAudio = new EstadoAudio();
233         estadoAudio.setIsPlaying(player.isPlaying());
234         estadoAudio.setIsStarted(isStarted);
235         estadoAudio.setProgress(player.getCurrentPosition());
236         estadoAudio.setUrl(url);
237
238         player.stop();
239         player.reset();
240         playButton.setImageResource(R.drawable.ic_media_play);
241         handler.removeCallbacks(updatePositionRunnable);
242         seekbar.setProgress(0);
243         isStarted = false;
244     }
245 }
```

Por último, es importante que en el método `onDestroy()` de la actividad liberemos los recursos ocupados por el reproductor. Esto lo hacemos a través del método `release()`.

AudioActivity.java

```
195 @Override
196 protected void onDestroy() {
197     super.onDestroy();
198     if (player != null) {
199         player.release();
200         player = null;
201     }
202
203     getWindow().clearFlags(
204         android.view.WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
205 }
```

5.3 Imágenes

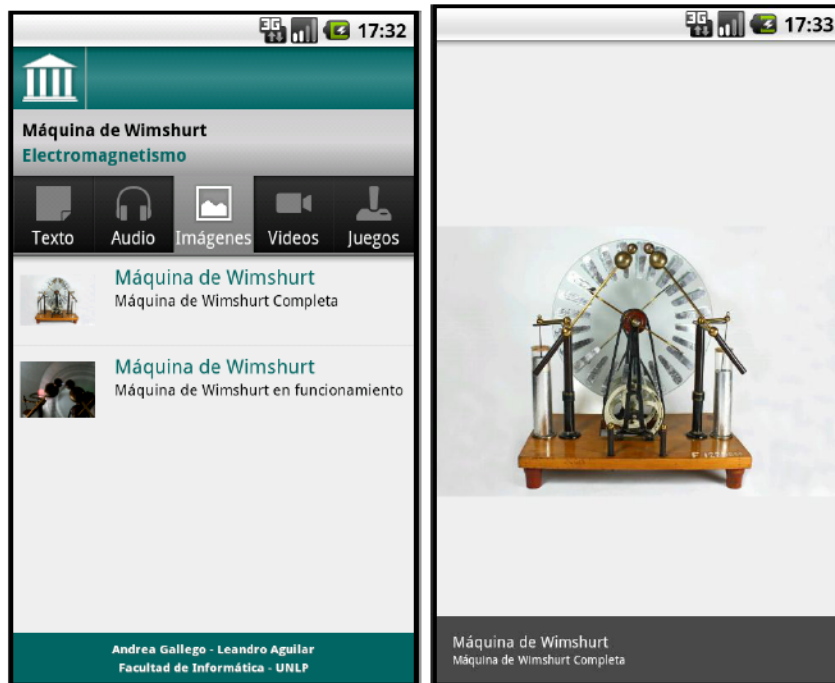


Figura 8.40: ImagenActivity y FullScreenImageActivity

A través de esta sección el usuario puede visualizar un conjunto de imágenes asociadas al objeto. En la primera pantalla puede ver el listado de imágenes en la que se muestra una imagen pequeña junto al nombre y una breve descripción de la misma. Cuando el usuario selecciona una de ellas, la imagen se visualiza en tamaño completo y podrá navegar hacia la izquierda o derecha entre las imágenes disponibles sin necesidad de volver al listado.

Como la actividad ImagenActivity, que es un listado de imágenes, tiene el mismo comportamiento que la actividad en la que se muestra el listado de museos explicada anteriormente, nos enfocaremos principalmente en explicar la implementación de la segunda actividad, FullScreenImageActivity la cual es llamada desde la actividad ImagenActivity cuando se selecciona un elemento de la lista.

Vista de la actividad

full_image.xml

```

20 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
21     android:layout_width="fill_parent"
22     android:layout_height="fill_parent"
23     android:background="@drawable/gray_list"
24     android:gravity="center"
25 >
26     <FrameLayout
27         android:layout_width="fill_parent"
28         android:layout_height="fill_parent">
29         <ImageSwitcher
30             android:id="@+id/imageSwitcher"
31             android:layout_width="fill_parent"
32             android:layout_height="fill_parent"
33             android:onClick="imageClick"
34             android:keepScreenOn="true"
35             android:layout_gravity="center">
36         </ImageSwitcher>
37         <LinearLayout
38             android:id="@+id/textoImagen"
39             android:layout_width="fill_parent"
40             android:layout_height="wrap_content"
41             android:layout_gravity="center_horizontal|bottom"
42             android:padding="12dip"
43             android:orientation="vertical"
44             android:background="@drawable/gray_text">
45             <TextView
46                 android:layout_width="fill_parent"
47                 android:layout_height="wrap_content"
48                 android:id="@+id/tituloImagen"
49                 android:textSize="12dp"
50                 android:textColor="@drawable/white"
51             />
52             <TextView
53                 android:layout_width="fill_parent"
54                 android:layout_height="wrap_content"
55                 android:id="@+id/descripcionImagen"
56                 android:textSize="9dp"
57                 android:textColor="@drawable/white"
58             />
59         </LinearLayout>
60     </FrameLayout>
61     <include layout="@layout/cargando"/>
62 </FrameLayout>

```

Como hemos hecho en otras actividades, en esta vista también incluimos el layout para indicarle al usuario que estamos cargando la imagen.

Pero lo más importante que debemos explicar en esta vista es la utilización del componente ImageSwitcher. Este componente nos permite mostrar un imagen en tamaño completo y es muy útil para cambiar entre dos imágenes y proporciona formas de transición de uno a otro a través de animaciones apropiadas. A continuación explicamos cómo se implementa dicho componente.

Implementación

Lo que hicimos entonces es crear una clase que no sólo extienda de MuseoActivity sino también que implemente la interfaz ViewFactory. ViewFactory es una interfaz que crea la vista que necesita ser mostrada en el componente ImageSwitcher. Tiene un método makeView() que debemos implementar.

FullScreenImageActivity.java


```

35 public class FullScreenImageActivity extends MuseoActivity implements ViewFactory{
36
37     private ImageSwitcher imageSwitcher ;
38     private ArrayList<MaterialMuseo> elementos;
39     int curIndex=0;
40     int downX,upX;
41
42     public View makeView() {
43         ImageView i = new ImageView(this);
44         i.setScaleType(ImageView.ScaleType.FIT_CENTER);
45         i.setLayoutParams(new ImageSwitcher.LayoutParams(
46             LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
47
48         return i;
49     }

```

En el método makeView() simplemente creamos el componente ImageView donde queremos que se visualice la imagen correspondiente.

Lo primero que debemos hacer en el método inicializar de la clase FullScreenImageActivity es obtener el listado de materiales del objeto que se corresponden a las imágenes que fueron enviados desde la actividad ImagenActivity a través de un Intent. Dichos materiales contienen la url de las imágenes para poder descargarlas desde el servidor. Desde la actividad anterior también se envía la posición de la imagen seleccionada para saber cuál es la imagen del listado recibido que debemos visualizar por primera vez.

FullScreenImageActivity.java

```

54     Type collectionType = new TypeToken<List<MaterialMuseo>>() {
55     }.getType();
56     elementos = new Gson().fromJson(
57         getIntent().getStringExtra("elementos"), collectionType);
58
59     curIndex = getIntent().getIntExtra("position", 0);
--

```

Una vez hecho esto, debemos inicializar el componente ImageSwitcher.

FullScreenImageActivity.java

```

61     imageSwitcher = (ImageSwitcher) findViewById(R.id.imageSwitcher);
62     imageSwitcher.setFactory(this);
63     imageSwitcher.setInAnimation(AnimationUtils.LoadAnimation(this,
64         android.R.anim.fade_in));
65     imageSwitcher.setOutAnimation(AnimationUtils.LoadAnimation(this,
66         android.R.anim.fade_out));
67     imageSwitcher.setOnTouchListener(new OnTouchListener() {
68     public boolean onTouch(View v, MotionEvent event) {
69         if (event.getAction() == MotionEvent.ACTION_DOWN) {
70             downX = (int) event.getX();
71             return true;
72         } else if (event.getAction() == MotionEvent.ACTION_UP) {
73             upX = (int) event.getX();
74             if (upX - downX == 0) {
75                 ocultarMostrarTexto();
76             } else if (upX - downX > 100) {
77                 curIndex--;
78                 if (curIndex < 0) {
79                     curIndex = elementos.size() - 1;
80                 }
81                 imageSwitcher.destroyDrawingCache();
82                 imageSwitcher.setInAnimation(AnimationUtils
83                     .LoadAnimation(FullScreenImageActivity.this,
84                         android.R.anim.slide_in_left));
85                 imageSwitcher.setOutAnimation(AnimationUtils
86                     .LoadAnimation(FullScreenImageActivity.this,
87                         android.R.anim.slide_out_right));
88                 getImagen();
89             } else if (downX - upX > -100) {
90                 curIndex++;
91                 if (curIndex > elementos.size() - 1) {
92                     curIndex = 0;
93                 }
94                 imageSwitcher.destroyDrawingCache();
95                 imageSwitcher.setInAnimation(AnimationUtils
96                     .LoadAnimation(FullScreenImageActivity.this,
97                         R.anim.slide_in_right));
98                 imageSwitcher.setOutAnimation(AnimationUtils
99                     .LoadAnimation(FullScreenImageActivity.this,
100                         R.anim.slide_out_left));
101                 getImagen();
102             }
103             return true;
104         }
105         return false;
106     }
107 });

```

A este componente además de indicarle que animación deseamos que realice cada vez que la imagen entre o salga de la pantalla, a través de los métodos `setInAnimation` y `setOutAnimation` respectivamente, debemos asignarle un escuchador para eventos táctil que se produzcan en la pantalla. De esta manera podemos interpretar cuando el usuario hace un gesto con el dedo de izquierda a derecha, de derecha a izquierda o simplemente toca la pantalla. Para los dos primeros casos, significa que el usuario quiere navegar sobre la galería y recorrer las imágenes hacia adelante o hacia atrás respectivamente. En cambio cuando toca la pantalla, lo que hacemos es mostrar u ocultar la descripción de la imagen.

Para obtener la imagen a visualizar utilizamos la posición actual, obtenemos la url dentro del listado de materiales y ejecutamos la tarea asincrónica `BatchImageDownloader` para descargar dicha imagen desde el servidor. Una vez que tenemos la imagen debemos asignársela al `ImageSwitcher` a través del método `setImageDrawable` y actualizar la descripción de la misma.

`FullScreenImageActivity.java`

```

166     public void recibirRespuesta() {
167
168         ImagenHTTP respuesta = (ImagenHTTP) getTask().getRespuestas().get(
169             "imagenHTTP");
170
171         switch (respuesta.getErrorCode()) {
172             case ErrorCode.NO_HAY_INTERNET:
173                 noHayConexion(1, getResources().getString(R.string.no_hay_internet));
174                 break;
175             case ErrorCode.RESPUESTA_CORRECTA:
176                 mostrarImagen(respuesta);
177                 break;
178         }
179
180         ocultarEspera();
181     }
182
183     private void mostrarImagen(ImagenHTTP respuesta) {
184         curIndex = getEstado().getCacheImagen().getPosition();
185         imageSwitcher.setImageDrawable(new BitmapDrawable((Bitmap) respuesta
186             .getImagenes().get(getElementos().get(curIndex).getUrl())));
187
188         getEstado()
189             .getCacheImagen()
190             .getImagenes()
191             .put(getElementos().get(curIndex).getUrl(),
192                 (Bitmap) respuesta.getImagenes().get(getElementos().get(curIndex).getUrl()));
193
194         actualizarTexto(getElementos().get(curIndex).getTitulo(),
195             getElementos().get(curIndex).getDescripcion());
196
197         cambiarEstado(Estado.INACTIVO);
198     }

```

Guardamos la imagen en una caché para tenerla disponible en caso que el usuario quiera girar la pantalla para ver la imagen en otra posición y de esta forma nos evitamos comunicarnos nuevamente con el servidor.

5.4 Videos

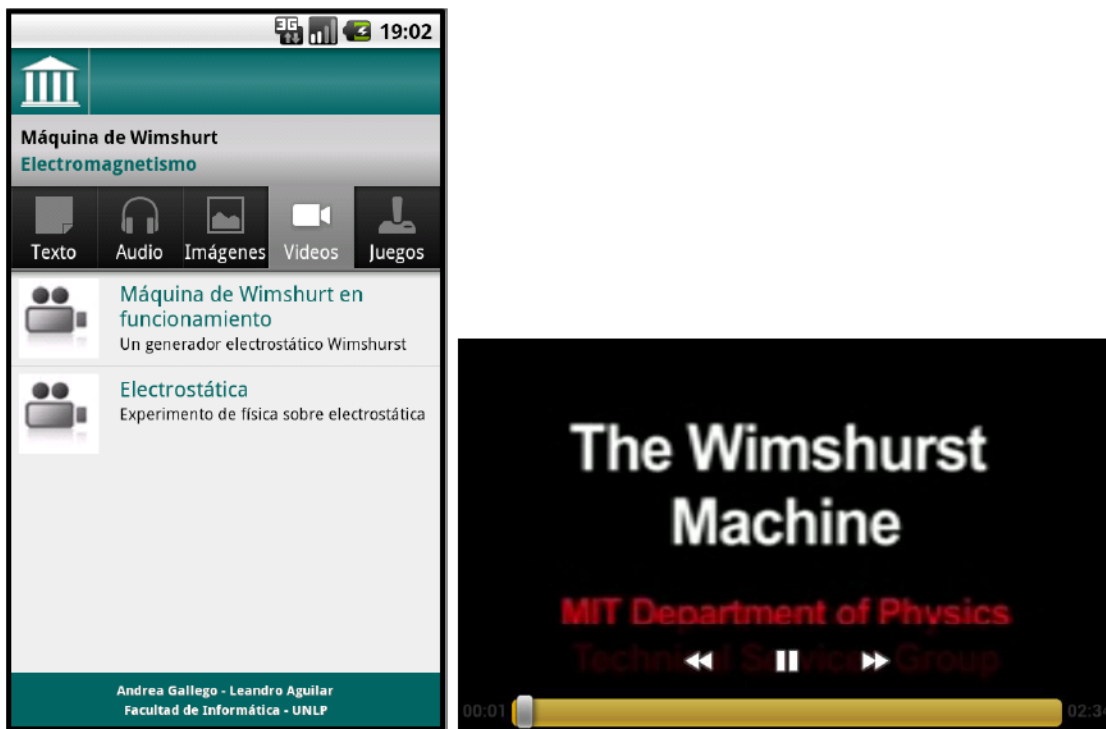


Figura 8.41: VideoActivity y FullScreenVideoActivity

A través de esta sección el usuario podrá ver distintos videos relacionados al objeto. Para ello, deberá seleccionar uno de los videos del listado y podrá visualizar el mismo en pantalla completa. Como en el caso anterior nos centraremos en explicar la segunda actividad FullScreenVideoActivity.

Vista de la actividad

Existen dos formas de crear interfaces de usuario en android. Una forma es declarativa y la otra es mediante programación.

La creación de interfaces de usuario declarativa se realiza con código XML, donde se definen como se verán los componentes en la pantallas y es como lo veníamos haciendo hasta el momento. Pero para este caso, nos pareció mejor la segunda opción, crear la interfaz programáticamente.

Implementación

Esta clase extiende directamente de Activity.

FullScreenVideoActivity.java

```

42 public class FullScreenVideoActivity extends Activity {
43
44     protected ProgressBar mProgressBar;
45     protected TextView mProgressMessage;
46     protected VideoView mVideoView;
47     protected String mVideoId = null;
48     private Dialog dialog;
49
50     @Override
51     protected void onCreate(Bundle savedInstanceState) {
52         super.onCreate(savedInstanceState);
53
54         requestWindowFeature(Window.FEATURE_NO_TITLE);
55         getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
56                             WindowManager.LayoutParams.FLAG_FULLSCREEN);
57
58         getWindow().setFlags(android.view.WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON,
59                             android.view.WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
60
61         setupView();
62
63         mVideoId = this.getIntent().getStringExtra("url");
64     }
65
66     --

```

En el método onCreate() indicamos que no queremos que se muestre la barra de título de la aplicación utilizando el método requestWindowFeature y seteamos el FLAG_FULLSCREEN, para utilizar el tamaño completo de la pantalla y FLAG_KEEP_SCREEN_ON, para mantener la ventana visible al usuario.

La forma más simple que disponemos en Android para visualizar videos es a través del componente VideoView.

FullScreenVideoActivity.java

```

167     Display display = getWindowManager().getDefaultDisplay();
168
169     mVideoView = new Video(this, display.getWidth(), display.getHeight());
170     mVideoView.setId(3);
171     android.widget.RelativeLayout.LayoutParams lVidViewLayoutParams = new android.widget.RelativeLayout.LayoutParams(
172         ViewGroup.LayoutParams.FILL_PARENT,
173         ViewGroup.LayoutParams.FILL_PARENT);
174     lVidViewLayoutParams.addRule(RelativeLayout.CENTER_IN_PARENT);
175     mVideoView.setLayoutParams(lVidViewLayoutParams);
176     mVideoView.setKeepScreenOn(true);
177
178     mVideoView.setOnCompletionListener(new OnCompletionListener() {
179     public void onCompletion(MediaPlayer pMp) {
180         FullScreenVideoActivity.this.finish();
181     }
182     });
183
184     mVideoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
185     public void onPrepared(MediaPlayer pMp) {
186         FullScreenVideoActivity.this.mProgressBar
187             .setVisibility(View.GONE);
188         mVideoView.requestFocus();
189         mVideoView.start();
190     }
191     });
192
193     MediaController lMediaController = new MediaController(
194         FullScreenVideoActivity.this);
195     lMediaController.show(0);
196
197     mVideoView.setMediaController(lMediaController);
198

```

Al VideoView le asignamos un componente MediaController. Este componente es una vista que contiene los controles de un reproductor de multimedia. Normalmente contiene los botones como

iniciar, pausar, adelantar, etc. y una barra de progreso. Se ocupa de la sincronización de los controles con el estado de la reproducción. El MediaController creará un conjunto predeterminado de los controles y los pondrá en una ventana flotante encima de la aplicación. Estos desaparecerán si se deja inactiva durante tres segundos y vuelven a aparecer cuando el usuario toca la pantalla.

La información que disponemos del video que el usuario seleccionó es el id del video de youtube. Por ejemplo, en la url <http://www.youtube.com/watch?v=ZilvI9tS0Og> el id es ZilvI9tS0Og. Entonces, para poder reproducir este video, debemos acceder al archivo xml con a la información que se corresponde a ese id. Para ello, youtube nos provee de la url <https://gdata.youtube.com/feeds/api/videos/>. Dentro de este archivo xml encontramos la url del video con formato 3gp el cual soporta el componente VideoView. Para poder parsear el xml y recorrerlo utilizamos las clases `javax.xml.parsers.DocumentBuilderFactory`, `javax.xml.parsers.DocumentBuilder` y `org.w3c.dom.Document`.

Una vez que encontramos la url se la seteamos al VideoView a través del método `setVideoURI` y una vez listo para reproducirse iniciará la reproducción del video a través del escuchador `OnPreparedListener` asignado durante la inicialización.

FullScreenVideoActivity.java

```
105 private void verVideo() {
106     mProgressBar.bringToFront();
107     mProgressBar.setVisibility(View.VISIBLE);
108
109     DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
110     DocumentBuilder builder = null;
111     try {
112         builder = factory.newDocumentBuilder();
113     } catch (ParserConfigurationException e) {
114         e.printStackTrace();
115     }
116
117     String url = null;
118     try {
119         Document doc = builder
120             .parse("https://gdata.youtube.com/feeds/api/videos/"
121                 + mVideoId);
122
123         for (int i = 0; i < doc.getElementsByTagName("media:content")
124             .getLength(); i++) {
125             if (doc.getElementsByTagName("media:content").item(i)
126                 .getAttributes().getNamedItem("type").getNodeValue()
127                 .equals("video/3gpp")) {
128                 url = doc.getElementsByTagName("media:content").item(i)
129                     .getAttributes().getNamedItem("url").getNodeValue();
130                 break;
131             }
132         }
133
134         mVideoView.setVideoURI(Uri.parse(url));
135
136     } catch (SAXException e) {
137         e.printStackTrace();
138     } catch (IOException e) {
139         e.printStackTrace();
140     }
141 }
```

Una vez que se completó la reproducción, el escuchador `OnCompletionListener` finalizará la actividad.

5.5 Juegos



Figura 8.42: JuegoActivity

En esta sección, el usuario podrá acceder a dos juegos relacionados al objeto para que pueda demostrar lo que aprendió utilizando el resto de los recursos disponibles. Estos juegos consisten en en una trivía y un unir con flechas.

5.5.1 Trivia

El desarrollo de este juego consiste en responder un conjunto de preguntas seleccionando las opciones que el usuario considere correctas. En algunas preguntas debe seleccionar una o más opciones y en otras debe optar entre la respuestas sí o no.

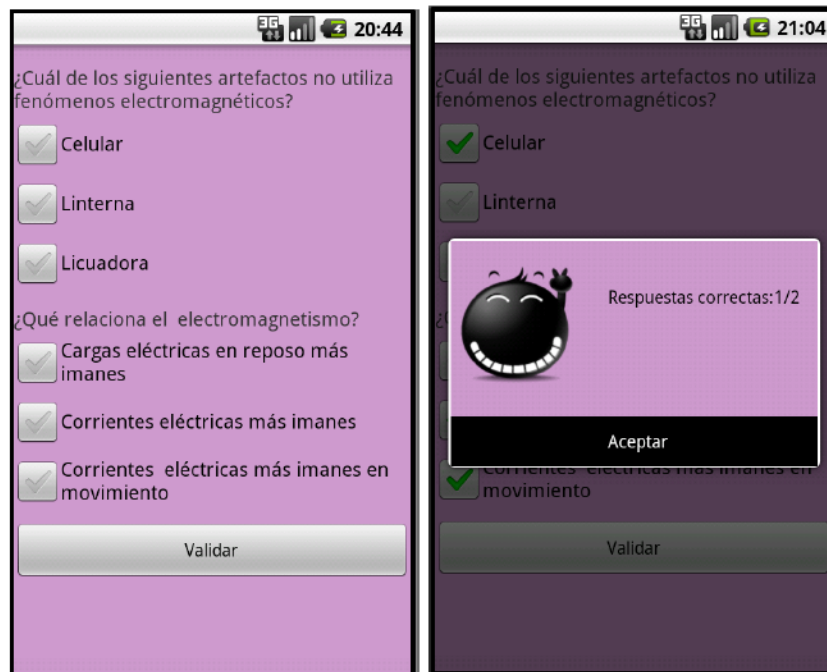


Figura 8.43: TriviaActivity

Vista de la actividad

trivia.xml

```
2 <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/scroll_trivia"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     android:background="@drawable/gray_list"
7     android:fillViewport="true"
8     >
9     <FrameLayout
10        android:layout_width="fill_parent"
11        android:layout_height="wrap_content" >
12
13        <LinearLayout
14            android:id="@+id/linear_trivia"
15            android:layout_width="fill_parent"
16            android:layout_height="fill_parent"
17            android:orientation="vertical"
18            android:background="@drawable/fondo"
19            android:paddingTop="15dp"
20            >
21
22            </LinearLayout>
23
24            <include layout="@layout/cargando"/>
25        </FrameLayout>
26 </ScrollView>
27
```

La trivia es dinámica, es decir, varía de acuerdo al objeto que el usuario se encuentra visualizando y además las preguntas que se pueden utilizar para crear la trivia pueden ser de distinto tipo, como multiple choice o del estilo Sí/No. Es por este motivo, que los distintos componentes utilizados para representar las preguntas y respuestas de la trivia, se crean dinámicamente al iniciar la actividad.

Implementación

La clase que utilizamos para implementar este juego es TriviaActivity, la cual extiende de MuseoActivity, ya que debemos obtener los datos de la trivia que se corresponde al objeto.

TriviaActivity.java

```
34 public class TriviaActivity extends MuseoActivity {
35
36     private List<IControlEncuestaUI> listaItemsViews;
37     private LinearLayout ll;
38     private ObjetoMuseo objeto;
39
```

Para buscar la trivia que se corresponde al objeto utilizamos la tarea asincrónica ServidorAsyncTask, a la que le realizamos el requerimiento GET_MODELO_TRIVIA y le pasamos como parámetro el id del objeto correspondiente.

TriviaActivity.java


```

41 private void buscarTrivia() {
42     cambiarEstado(Estado.EJECUTANDO);
43     setTask(new ServidorAsyncTask(this));
44     HashMap<String, Object> p = new HashMap<String, Object>();
45     p.put("idobjeto", objeto.getId());
46     getEstado().getTask().ejecutarTarea(Task.GET_MODELO_TRIVIA, p);
47
48 }

```

Luego la tarea se la solicita al ClienteHTTP.

ServidorAsyncTask.java

```

41     case GET_MODELO_TRIVIA:
42         getRespuestas().put("triviaHTTP", cliente.getModeloTrivia((Integer) getParametros().get("idobjeto")));
43         break;

```

En el método getModeloTrivia() de la clase ClienteHTTP creamos un objeto ConexionHTTP con la url MuseumWeb/museo/recuperarModeloTrivia.do, correspondiente al requerimiento que devuelve un objeto TriviaHTTP. Este objeto contiene un código de error y un mensaje en caso de que ocurra algún error en la comunicación con el servidor y un objeto de tipo Trivia, el cual contiene el listado de preguntas y respuestas de la trivia.

ClienteHTTP.java

```

270 public TriviaHTTP getModeloTrivia(Integer idobjeto) {
271
272     TriviaHTTP respuesta = new TriviaHTTP();
273
274     if (hayConexion()) {
275
276         ConexionHTTP con = new ConexionHTTP(URL
277             + "/museum/recuperarModeloTrivia.do");
278         con.addParam("id", idobjeto.toString());
279
280         try {
281             con.execute(RequestMethod.POST);
282         } catch (Exception e) {
283             e.printStackTrace();
284         }
285
286         String json_response = con.getResponse();
287         String jsonAux = null;
288         try {
289             jsonAux = new String(json_response.getBytes("iso-8859-1"),
290                 "UTF-8");
291         } catch (Exception e) {
292             e.printStackTrace();
293         }
294
295         Gson gsb = new GsonBuilder().registerTypeAdapter(TriviaHTTP.class,
296             new TriviaHTTPAdapter()).create();
297         respuesta = gsb.fromJson(jsonAux, TriviaHTTP.class);
298
299     } else {
300         respuesta.setErrorCode(ErrorCode.NO_HAY_INTERNET);
301     }
302
303     return respuesta;
304 }

```

Una vez que recibimos la respuesta desde el servidor y verificamos que no haya ocurrido ningún error procedemos a generar la encuesta en base a los datos recibidos.

TriviaActivity.java

```

68 private void mostrarModeloTrivia(TriviaHTTP respuesta) {
69
70     if (respuesta.getTriviaModelo() != null) {
71         for (ItemTrivia item : respuesta.getTriviaModelo().getItems()) {
72             String label = item.getPregunta();
73             if (item.getRespuesta().getTipo().equals("Si-No")) {
74                 TipoRespuestaSiNoTrivia tipo = (TipoRespuestaSiNoTrivia) item
75                     .getRespuesta();
76                 ViewItemSiNo v = new ViewItemSiNo(this, label,
77                     tipo.getOpcionCorrecta());
78                 listaItemsViews.add(v);
79                 ll.addView(v);
80             }
81
82             if (item.getRespuesta().getTipo().equals("MultipleChoice")) {
83                 TipoRespuestaMultipleChoiceTrivia tipoMC = (TipoRespuestaMultipleChoiceTrivia) item
84                     .getRespuesta();
85                 ViewItemMultipleChoice v = new ViewItemMultipleChoice(this,
86                     label);
87                 for (OpcionMultipleChoiceTrivia opcion : tipoMC
88                     .getOpciones()) {
89                     v.agregarCheckBox(opcion.getNombre(),
90                         opcion.getEsOpcionCorrecta());
91                 }
92                 listaItemsViews.add(v);
93                 ll.addView(v);
94             }
95         }
96         this.agregarBotonEnviar();
97     }
98 }

```

En el método mostrarModeloTrivia(), podemos ver que en base al tipo de respuesta se genera la vista correspondiente. Cuando es de tipo “Sí-No” generamos componentes RadioButton y cuando es de tipo “MultipleChoice” utilizamos componentes CheckBox.

Una vez que el usuario responde la trivia, se valida la cantidad de respuestas correctas que realizó y se las mostramos a través de un diálogo.

TriviaActivity.java

```

123 protected void validarTrivia() {
124     int totalItems = listaItemsViews.size();
125     int totalCorrectas = 0;
126     for (IControlEncuestaUI elemento : listaItemsViews) {
127         if (elemento.estaBienRespondida()) {
128             totalCorrectas++;
129         }
130     }
131
132     TextView texto = (TextView) getDialog().findViewById(
133         R.id.text_dialog_trivia);
134     texto.setText("Respuestas correctas:" + totalCorrectas + "/"
135         + totalItems);
136     this.getDialog().show();
137 }
138 }

```

5.5.2 Unir con flechas

El desarrollo de este juego consiste en relacionar un concepto de la columna izquierda con uno de la columna derecha. Para ello, el usuario debe seleccionar un concepto de alguna de las columnas para activar la flecha y arrastrar la misma hacia el concepto de la otra columna que el usuario considera que se corresponde. El juego termina cuando logre unir correctamente todos los conceptos.

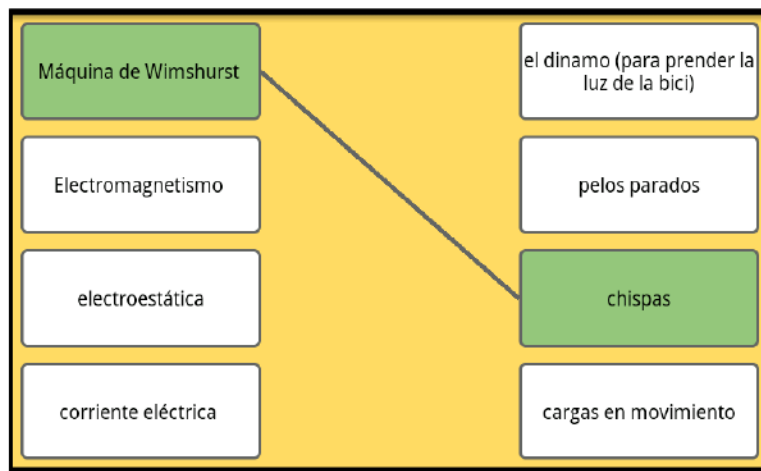


Figura 8.44: UnirFlechasActivity

Vista de la actividad

Como solo se permiten cuatro opciones en total por juego, la vista de la actividad consiste simplemente en dos LinearLayout que contienen un conjunto de cuatro botones cada uno.

Implementación

La clase UnirFlechasActivity extiende de la clase MuseoActivity y además implementa la interfaz onTouchListener.

UnirFlechasActivity.java

```

39 public class UnirFlechasActivity extends MuseoActivity implements onTouchListener {
40
41     private int corx, cory;
42     private FlechaView flechaView;
43     private Boolean flechaActiva = false;
44     private Boolean inicializado = false;
45     private ResueltasView resueltasView;

```

Como en el caso anterior debemos utilizar la tarea ServidorAsyncTask para solicitarle el conjunto de preguntas y respuestas que se corresponden al objeto que está visualizando el usuario. En este caso, utilizamos la tarea GET_UNIR_CON_FLECHAS.

UnirFlechasActivity.java

```
109 private void getUnirConFlechas() {
110     cambiarEstado(Estado.EJECUTANDO);
111     setTask(new ServidorAsyncTask(this));
112     HashMap<String, Object> p = new HashMap<String, Object>();
113     p.put("idObjeto", getObjeto().getId());
114     ejecutarTarea(Task.GET_UNIR_CON_FLECHAS, p);
115
116 }
```

ServidorAsyncTask.java

```
33     case GET_UNIR_CON_FLECHAS:
34         getRespuestas().put("unirConFlechasHTTP", cliente.getUnirConFlechas((Integer) getParametros().get("idObjeto")))
35         break;
```

CienteHTTP.java

```
132 public UnirConFlechasHTTP getUnirConFlechas(Integer idObjeto) {
133
134     UnirConFlechasHTTP unirHTTP = new UnirConFlechasHTTP();
135     if (hayConexion()) {
136         ConexionHTTP con = new ConexionHTTP(URL
137             + "/museum/recuperarUnirConFlechas.do");
138         con.addParam("id", idObjeto.toString());
139         try {
140             con.execute(RequestMethod.POST);
141         } catch (Exception e) {
142             e.printStackTrace();
143         }
144         String json_response = con.getResponse();
145         String jsonAux = null;
146         try {
147             jsonAux = new String(json_response.getBytes("iso-8859-1"),
148                 "UTF-8");
149         } catch (Exception e) {
150         }
151         unirHTTP = getGson().fromJson(jsonAux, UnirConFlechasHTTP.class);
152     } else {
153         unirHTTP.setErrorCode(ErrorCode.NO_HAY_INTERNET);
154     }
155     return unirHTTP;
156 }
```

Una vez que recibimos los datos, cargamos cada una de las preguntas en la columna de botones izquierda, cargamos las respuestas en la columna derecha y mezclamos las respuestas aleatoriamente. Esto lo hacemos a través del método shuffle de la clase Collections.

UnirFlechasActivity.java

```

141 private void mostrarJuego(UnirConFlechasHTTP respuesta) {
142     mostrarBotones();
143     for (UnirConFlechas pregutanRespuesta : respuesta.getItems()) {
144         ItemJuego preg = new ItemJuego(pregutanRespuesta.getPalabraOrigen());
145         ItemJuego resp = new ItemJuego(pregutanRespuesta.getPalabraDestino());
146         preg.setRelacionado(resp);
147         resp.setRelacionado(preg);
148         getPreguntas().add(preg);
149         getRespuestas().add(resp);
150     }
151
152     for (int i = 0; i < getPreguntas().size(); i++) {
153         getPreguntas().get(i).setButton((Button) findViewById(getButtonsPreguntas()[i]));
154         getPreguntas().get(i).getButton().setText(getPreguntas().get(i).getTexto());
155     }
156
157     Collections.shuffle(getRespuestas());
158
159     for (int i = 0; i < getRespuestas().size(); i++) {
160         getRespuestas().get(i).setButton((Button) findViewById(getButtonsRespuestas()[i]));
161         getRespuestas().get(i).getButton().setText(getRespuestas().get(i).getTexto());
162     }
163 }
...

```

Para detectar los diferentes eventos que produce el usuario cuando toca la pantalla y arrastra, lo hacemos a través de la interfaz `OnTouchListener`, en la cual debemos implementar el método `onTouch(View v, MotionEvent event)`. En este método mediante el objeto `event` obtenemos la coordenada `x` e `y` donde el usuario seleccionó con el dedo, entonces podemos identificar si se encuentra o no posicionado sobre uno de los botones. En el caso que esté posicionado sobre un botón se activará la flecha y se mantendrá en ese estado hasta que el usuario suelte el dedo. Cuando suelta el dedo, debemos verificar si lo hizo sobre otro botón y si, es así, debemos validar si este botón corresponde a la respuesta correcta.

UnirFlechasActivity.java

```

330 public boolean onTouch(View v, MotionEvent event) {
331     if (!isJuegoResuelto()) {
332         int x = (int) event.getX();
333         int y = (int) event.getY();
334         inicializarRectangulos();
335         switch (event.getAction()) {
336             case MotionEvent.ACTION_DOWN:
337                 itemInicio = seleccionoItem(x, y);
338                 if (itemInicio != null) {
339                     flechaView.x = x;
340                     flechaView.y = y;
341                     flechaActiva = true;
342                 }
343                 break;
344             case MotionEvent.ACTION_MOVE:
345                 if (flechaActiva) {
346                     corx = x;
347                     cory = y;
348                     flechaView.invalidate();
349                 }
350                 break;
351             case MotionEvent.ACTION_UP:
352                 if (flechaActiva) {
353                     itemFin = seleccionoItem(x, y);
354                     flechaActiva = false;
355                     if (itemFin != null
356                         && itemFin.equals(itemInicio.getRelacionado())) {
357                         itemFin.setEstado(EstadoItem.RESUELTO);
358                         itemInicio.setEstado(EstadoItem.RESUELTO);
359                         resueltasView.invalidate();
360                         playSound(OK_SOUND);
361                     } else {
362                         playSound(ERROR_SOUND);
363                     }
364                     itemInicio = null;
365                     itemFin = null;
366                     flechaView.invalidate();
367                 }
368                 break;
369             }
370         }
371         return true;
372     }

```

Cuando el usuario une todos los conceptos correctamente, se le muestra un diálogo indicando esta situación.

5.6 Categoría

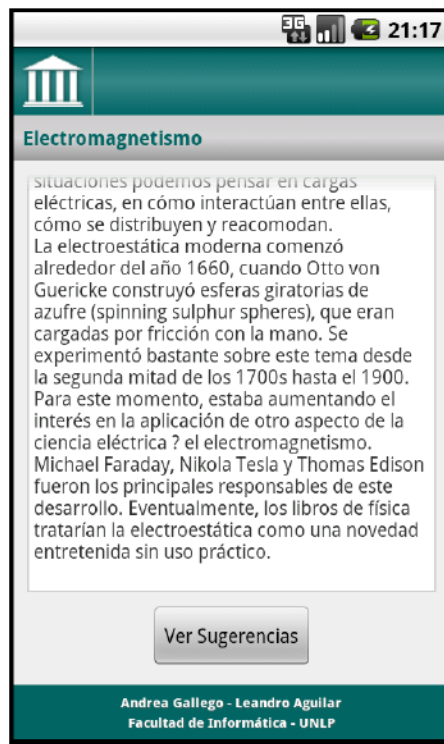


Figura 8.45: CategoríaActivity

Cuando seleccionamos el nombre de la categoría que se muestra en el encabezado del objeto, le permite acceder al usuario a la descripción de la misma. Además, podemos ver sugerencias de objetos a los que podemos seguir recorriendo. Dichos objetos pueden ser de la misma categoría o de otras que se encuentre en exposición en el museo.

Como el comportamiento es similar a la actividad `SobreElMuseoActivity` omitiremos la explicación de su implementación.

6. Encuesta

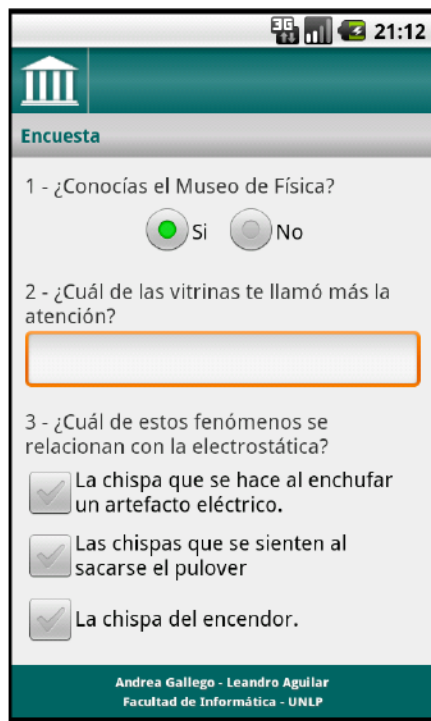


Figura 8.46: EncuestaActivity

El objetivo de esta actividad es que el usuario complete una encuesta. Luego, el personal del museo tendrá acceso a los resultados a través del sitio Web, lo que les permitirá sacar distintas conclusiones con respecto a la visita que realizó el usuario. Como la encuesta es propia del museo, es decir, cada museo genera su propia encuesta, la carga de la encuesta en la actividad se realiza dinámicamente al igual que hacíamos en el caso de la trivía. Para este caso, a diferencia de la trivía, se permiten respuesta de tipo texto para que el usuario pueda ingresar su propia respuesta. Otra diferencia que tenemos con respecto a la trivía es que aquí no hay respuestas correctas, sino que simplemente, se envían los resultados al servidor para que puedan ser almacenados y consultados posteriormente.

7. Ayuda

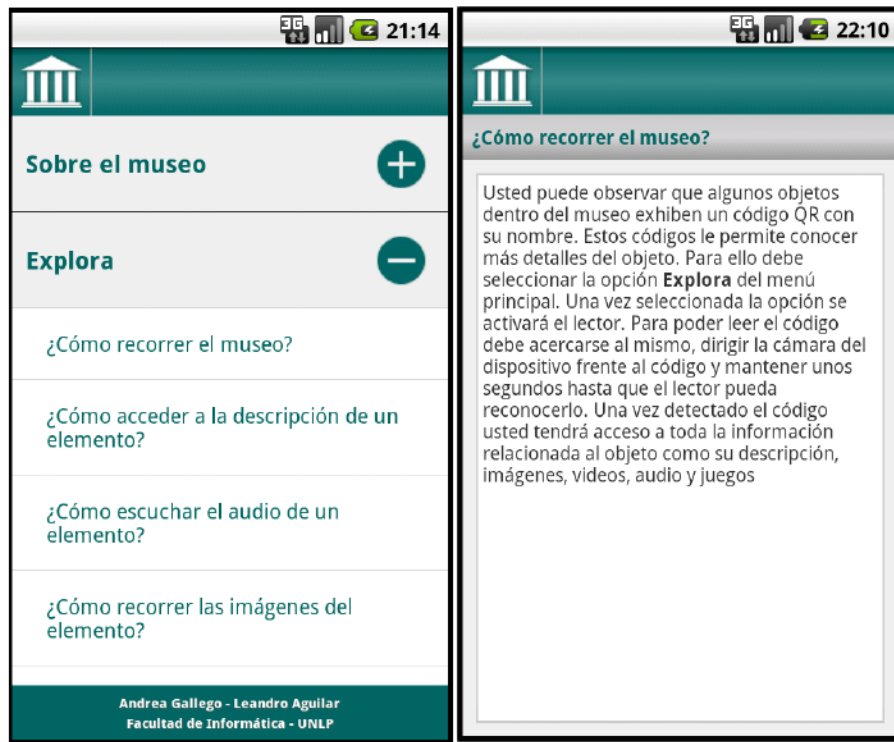


Figura 8.47: AyudaActivity y AyudaValueActivity

Esta actividad sirve de asistencia al usuario, es decir, le permite al usuario aprender a utilizar cada una de las opciones que se ofrecen en la aplicación.

Vista de la aplicación

ayuda.xml

```

20 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:orientation="vertical"
6     android:gravity="center"
7     android:background="@drawable/gray_list">
8
9     <include layout="@Layout/header"/>
10
11     <ExpandableListView
12         android:id="@+id/android:list"
13         android:layout_width="fill_parent"
14         android:layout_height="0dip"
15         android:layout_weight="1"
16         android:background="@drawable/gray_list"
17         android:groupIndicator="@drawable/group_indicator"
18         android:layout_gravity="center_vertical"
19     />
20
21     <include layout="@Layout/footer"/>
22 </LinearLayout>

```

Para la vista de la ayuda utilizamos el componente ExpandableListView, el cual nos permite implementar el patrón de diseño elegido para este caso. A través de este componente podemos organizar la información en grupos y crear una lista de dos niveles. El grupo se corresponde a cada opción del menú principal de la aplicación y en el segundo nivel se visualizan las acciones que se pueden realizar en cada una de ellas.

Implementación

Para implementar esta actividad debemos extender de la clase `ExpandableListActivity` y crear nuestro propio adaptador. Al igual que con el `ListView` debemos crear un adaptador que gestione la colección de datos que mostraremos. En este caso creamos uno llamado `MyExpandableListAdapter` el cual extiende de la clase `BaseExpandableListAdapter` e implementamos los métodos necesario para que la vista se muestre correctamente.

AyudaActivity.java

```
18 public class AyudaActivity extends ExpandableListActivity {
19
20     private Help help;
21
22     private ExpandableListAdapter mAdapter;
23
24     @Override
25     public void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.ayuda);
28
29         setHelp(new Help(getApplicationContext()));
30
31         mAdapter = new MyExpandableListAdapter(this);
32
33         setListAdapter(mAdapter);
34
35         DisplayMetrics metrics = new DisplayMetrics();
36         getWindowManager().getDefaultDisplay().getMetrics(metrics);
37         int width = metrics.widthPixels;
38
39         getExpandableListView().setIndicatorBounds(
40             width - GetDipsFromPixel(50), width - GetDipsFromPixel(10));
41
42         getExpandableListView().setOnChildClickListener(this);
43     }
```

Cuando el usuario selecciona un elemento del segundo nivel, se llama a la actividad `AyudaValueActivity` en la que se muestra la descripción correspondiente a la opción elegida.

AyudaActivity.java

```
45     @Override
46     public boolean onChildClick(ExpandableListView parent, View v,
47         int groupPosition, int childPosition, long id) {
48
49         Intent i = new Intent(getApplicationContext(),
50             AyudaValueActivity.class);
51         i.putExtra("groupPosition", groupPosition);
52         i.putExtra("childPosition", childPosition);
53         startActivity(i);
54
55         return false;
56     }
```

Con esta última actividad hemos completado la descripción de los componentes que forman parte de la aplicación.

8.2.6 Internacionalización

En Android es algo muy sencillo implementar una aplicación que soporte varios idiomas. Para ello debemos utilizar el archivo strings.xml que se encuentra en el directorio res/values de nuestro proyecto Android. El archivo strings.xml almacena todas las cadenas de texto utilizando el formato XML para su definición y para que nos sea sencillo acceder a cada recurso y utilizarlos en cualquier archivo xml o código android. Este archivo nos facilita la traducción de una aplicación a distintos idiomas

res/values/string.xml

```
<resources>
  <string name="app_name">Museo</string>
  <string name="app_footer">Andrea Gallego - Leandro Aguilar</string>
  <string name="app_footer2">Facultad de Informática - UNLP</string>
  <string name="boton_historia_museo">Sobre el Museo</string>
  <string name="boton_lectorQR">Explora</string>
  <string name="boton_encuesta">Encuesta</string>
  <string name="boton_ayuda">Ayuda</string>
  ....
</resources>
```

El elemento raíz de este recurso es <resources></resources>. Cada vez que necesitemos definir una nueva cadena de texto deberemos agregar un elemento <string> con un atributo name cuyo valor debe ser un identificador de la cadena.

Siempre que se quiera asignar cadenas de texto a un archivo de layout, nos posicionaremos en el atributo android:text del widget de nuestro interés y en lugar de definir el texto estático vamos a utilizar el formato @string/[texto del atributo name que definimos para el string].

La internacionalización de una aplicación en Android se resuelve implementando varios archivos strings.xml, uno por cada idioma que necesitemos implementar.

Como vamos a publicar la aplicación en dos idiomas: inglés y castellano tenemos que crear una carpeta res/value-en, copiar el archivo strings.xml que teníamos en res/values desde la creación del proyecto y traducir cada valor al idioma inglés. Dejaremos las cadenas en español (que será el idioma por defecto) en el archivo localizado en la carpeta res/values/strings.xml, y las cadenas en inglés en res/values-en/strings.xml. Aquí estamos usando el sufijo 'en' de Inglés para las cadenas de texto en este idioma. También hay sufijos por idioma y por región o país, como son es-AR, es-ES, etc. Así si requerimos traducir la aplicación a otros idiomas, tendremos que incluir tantos directorios values-[códigoDelIdioma] como sean necesarios.

Por último, cabe mencionar que si el idioma del dispositivo en el que se instale nuestra aplicación coincide con el código de alguno de los directorios de idioma que anexamos, ese será tomado por default para desplegar la aplicación.

res/values-en/string.xml

```
<resources>
  <string name="app_name">Museo</string>
  <string name="app_footer">Andrea Gallego - Leandro Aguilar</string>
  <string name="app_footer2">Facultad de Informática - UNLP</string>
  <string name="boton_historia_museo">About the museum</string>
  <string name="boton_lectorQR">Explore</string>
  <string name="boton_encuesta">Poll</string>
  <string name="boton_ayuda">Help</string>
  ....
</resources>
```

Capítulo 9. Análisis de usabilidad

En el capítulo anterior hemos visto detalles de la implementación del prototipo móvil. En este capítulo veremos cuestiones respecto al análisis de la usabilidad que se hizo sobre la experiencia del visitante del museo usando la aplicación móvil. Se detallarán los perfiles de usuarios y se expondrán las conclusiones.

9.1 Armado de Ambiente

El análisis de usabilidad de la aplicación móvil fue llevada a cabo en el contexto del Museo de Física. Para ello, se creó el ambiente de producción del sistema. Como servidor para la aplicación Web se utilizó la PC que disponía allí, la cual tenía las siguientes características:

Procesador: Intel core 2 duo CPU 2.33GHz

Memoria: 1 GB de RAM

Sistema Operativo: Windows XP

Luego, se conectó un router inalámbrico para que los smartphones utilizados para la prueba se puedan conectar a Internet y de esa manera tengan acceso a la información del servidor del Museo.

Una vez que se encontraba todo lo anterior funcionando correctamente se procedió a la carga de datos de un objeto en particular en la aplicación Web, se imprimió el QR y se expuso frente al objeto en cuestión.

Por último, se instaló la aplicación móvil en los siguientes smartphones disponibles:

- Samsung Galaxy SII, Versión Android 4.0.3
- Motorola Milestone, Versión Android 2.3
- HTC Nexus One, Versión Android 2.3

9.2 Objetivo

Debido a que la Realidad Aumentada es una tecnología emergente, todavía no forma parte de la vida cotidiana de los usuarios. Por lo tanto, el museo es una interesante oportunidad para introducir a un público variado a la Realidad Aumentada. El estudio de la interacción entre los visitantes del museo y el dispositivo en un entorno natural, pero controlado, permite identificar las situaciones donde la RA es útil y relevante.

En los últimos años, más y más estudios han combinado la RA y los museos. Sin embargo, estos estudios se han centrado en la evaluación de los dispositivos de RA o en examinar los beneficios de la inversión en RA en el campo cultural, pero nunca describen cómo la RA puede transformar la actividad de visitar un museo.

Para estudiar el papel de la RA durante una visita a un museo, se debe establecer y definir un modelo de actividades y utilizar un marco teórico para integrar una herramienta como una guía de museo aumentada.

El test tiene dos objetivos principales, por un lado, observar de qué manera impacta la RA en una visita a un museo y por otro, comprobar que la aplicación sea fácil de usar para los usuarios y si este es capaz de obtener los resultados deseados de una manera sencilla.

9.3 Perfiles de usuarios

La evaluación se llevó a cabo con usuarios clasificados en los siguientes grupos:

Grupo A: Con experiencia en el uso de celulares y con experiencia en el uso de aplicaciones de RA

Grupo B: Con experiencia en el uso de celulares y sin experiencia en el uso de aplicaciones de RA

Grupo C: Sin experiencia en el uso de celulares y con experiencia en el uso de aplicaciones de RA

Grupo D: Sin experiencia en el uso de celulares y sin experiencia en el uso de aplicaciones de RA

9.4 Encuesta

Como parte del test de usabilidad de la aplicación móvil se invitó a un grupo de personas a la que se le explicó el objetivo de la aplicación y se les entregó un cuestionario, el cual estaba dividido en la siguientes partes:

Parte A: Perfil del Usuario

Cada persona tenía que completar sus datos personales e indicar cuál era su experiencia en el uso de teléfonos celulares y su experiencia en el uso de aplicaciones de realidad aumentada. Estas preguntas nos permitía identificar a qué grupo de usuarios de los mencionados anteriormente pertenecía.

El criterio que utilizamos para determinar quienes pertenecían a cada uno de los grupos fue:

- Aquellos que respondían que no tenían celular o si lo poseían pero su experiencia en su uso era básica o no tenían; o nunca accedieron a Internet o redes sociales desde su celular los consideramos como “*sin experiencia en el uso de celulares*”.
- Los que respondían que utilizaban el celular con mucha frecuencia y accedían a Internet y redes sociales los consideramos como “*con experiencia en el uso de celulares*”.
- Con respecto a la experiencia en aplicaciones de RA, consideramos como “*con experiencia en el uso de aplicaciones de RA*” a aquellos que indicaron que habían utilizado una aplicación de RA alguna vez y “*sin experiencia en el uso de aplicaciones de RA*” en el caso contrario.

Parte B: Sobre las tareas realizadas

Las tareas que se llevaron a cabo fueron de menor complejidad a mayor complejidad y se basaron en el reconocimiento de la opción que debían seleccionar para completar la tarea en forma rápida. Las tareas a realizar son las siguientes:

Tarea 1: Cuéntanos algo acerca del museo.

Consiste en acceder a la información que hay sobre el museo, accediendo desde el menú principal.

Tarea 2: Explora un objeto del museo. ¿Qué características tiene?

Consiste en acceder a la opción “Explora” y leer el código QR para acceder a la información del objeto.

Tarea 3: Buscar imágenes del mismo objeto. ¿Te ayudaron a comprender más el objeto?

Consiste en ver las imágenes que tiene el objeto. Para ello, puede continuar desde el estado en el que se encuentra de la tarea anterior, accediendo a la pestaña “Imágenes” del objeto.

Tarea 4: Escuchar la explicación del objeto. ¿Qué pudiste aprender?

Consiste en escuchar el audio que hay sobre el objeto explorado accediendo a la pestaña “Audio” y utilizar el reproductor de audio.

Tarea 5: Elije un video del objeto y observa su funcionamiento. ¿Qué te pareció?

Consiste en seleccionar un video, dentro del listado de videos en la sección “Videos” del objeto explorado.

Tarea 6: Elije el juego que más te guste. ¿Qué puntaje sacaste?

Consiste en seleccionar alguna de las actividades que se encuentran en la sección “Juegos”, como la trivía o el unir con flechas y demostrar lo que pudo aprender con la información que estuvo observando en las tareas anteriores.

Parte C: Sobre opiniones y sugerencias

Esta parte nos permite conocer las opiniones que tiene el encuestado con respecto a la aplicación y así poder obtener tanto los aspectos positivos como negativos de la misma.

9.5 Análisis de los resultados

Cantidad de participantes por perfil:

	Con experiencia en el uso de aplicaciones de RA	Sin experiencia en el uso de aplicaciones de RA
Con experiencia en el uso de celulares	Grupo A 3	Grupo B 4
Sin experiencia en el uso de celulares	Grupo C 0	Grupo D 6

Porcentaje de realización por perfil de usuarios:

	Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5	Tarea 6
Grupo A	3 (100%)	3 (100%)	3 (100%)	2 (66%)	3 (100%)	3 (100%)
Grupo B	4 (100%)	4 (100%)	4 (100%)	4 (100%)	2 (50%)	4 (100%)
Grupo D	6 (100%)	6 (100%)	6 (100%)	5 (83%)	4 (66%)	6 (100%)

Observaciones: La tarea de menor porcentaje de realización fue la tarea 5 y se debió a problemas en la conexión a Wi-Fi, pero todos los participantes encontraron la sección referida a la tarea.

9.6 Conclusiones

En cuanto al primer objetivo del test, se puede decir que la guía RA puede constituir un obstáculo en cuanto a que puede generar un distanciamiento entre el visitante y el objeto que se expone en el museo. Por otra parte, algunas características de la aplicación enriquecían la visita, en particular a través del contenido multimedia que se ofrecía y donde se podía apreciar información que no era visible en la realidad.

El test pone de manifiesto la idoneidad de la RA para dar una descripción precisa del objeto del museo y ayudar a los visitantes a analizar el mismo de una manera lúdica, gracias a la exploración de la información multimedia y los juegos proporcionados.

Un criterio importante para los visitantes, fue apreciar para poder confrontar y comparar la información proporcionada por la guía RA y la realidad. Era muy importante para los participantes ser capaces de encontrar la información pertinente sobre el objeto a simple vista.

En cuanto a la facilidad de uso, el resultado del test fue muy favorable, la mayoría de los participantes lograron realizar todas las tareas en forma simple y rápida. Para el caso de las personas que no poseía experiencia en el uso de celular, necesitaron asistencia para el uso básico del celular, pero una vez que aprendieron su uso, la aplicación les resultó intuitiva y fácil de utilizar.

Los aspectos negativos de la aplicación que fueron encontrados por los participantes se basaron principalmente en el contenido de la información del objeto que visitaron.

En cambio, los aspectos positivos de la aplicación se basaron principalmente en la facilidad de uso, practicidad y en la interactividad de la aplicación, además de la posibilidad de acceder a más información sobre los objetos del museo.

Capítulo 10. Conclusiones

En el capítulo anterior se hizo un análisis sobre la usabilidad del prototipo móvil. En este apartado daremos un repaso general del proyecto, presentando las conclusiones finales en función de los resultados obtenidos e indicaremos posibles trabajos futuros que puedan presentarse como consecuencia de éste.

10.1 Cumplimiento de los objetivos iniciales

Para comprobar que el proyecto ha obtenido los resultados esperados, basta con ver que se han alcanzado los objetivos que se pautaron al inicio del proyecto.

La oferta de smartphones es cada vez más grande. Los teléfonos móviles modernos tienen capacidades que los hacen ideales para desarrollos de aplicaciones interactivas. La llegada de sensores como GPS, brújulas electrónicas, banda ancha y la alta capacidad computacional, han convertido a los celulares en el perfecto campo de juego para las aplicaciones de RA.

La tecnología de RA es un campo de exploración con mucho futuro. Sus posibilidades de ofrecer contenido y una experiencia de alta calidad en el ámbito cultural y educativo son variadas y numerosas.

A lo largo de todo el presente proyecto se ha conseguido obtener un conocimiento amplio del sistema operativo Android. Su arquitectura, sus componentes y características, así como el funcionamiento y posibilidades ofrecidas por un sistema como Android se han ido conociendo gracias principalmente a la extensa y, en general, completa documentación que Google ha puesto a disposición de los desarrolladores. Especialmente en las primeras fases, esta documentación es útil y fácil de asimilar, lo que permite acercarse poco a poco a las formas y la tecnología de esta nueva plataforma.

Android incluye todo el software para que funcione un teléfono móvil, pero sin los obstáculos propietarios que han impedido la innovación en el teléfono móvil. El resultado en última instancia será un ritmo de innovación más rápido y mejor que proporcionará a los clientes móviles aplicaciones y capacidades hasta ahora difíciles de conseguir. Android es una estrategia importante para acercarnos al objetivo de proporcionar acceso a la información a los usuarios dondequiera que estén.

10.2 Resultados obtenidos

A lo largo de este trabajo de investigación hemos expuesto distintos conceptos y temas relacionados a aplicaciones móviles y realidad aumentada. También hemos realizado una introducción a las principales características de la plataforma Android, su API y entorno de desarrollo principal, para luego concluir con el desarrollo de un prototipo de aplicación para dispositivos móviles.

El prototipo cuenta con un administrador Web que forma parte del backend de la aplicación, actuando como parte servidor de la arquitectura. Está desarrollado en Java sobre el Framework Spring. El administrador Web permite ingresar la información respectiva al museo, a los objetos del museo, así como también ver las encuestas respondidas por los visitantes del museo.

En la aplicación móvil se utilizan diferentes componentes básicos de la plataforma Android, como son las actividades. Además, se trabajó con interfaces de usuarios, conexiones HTTP, declaración de manifiesto de aplicación y un manejo de recursos tanto internos como remotos.

Desde el punto de vista del usuario del museo, la aplicación ha sido de fácil uso, intuitiva y altamente responsiva. El usuario ha podido realizar todas las actividades que le ofrece la aplicación sin mayores inconvenientes y ha podido mejorar su experiencia aprendiendo de una manera interactiva y moderna. Mediante el feedback devuelto en las encuestas, los visitantes mismos han cambiado la manera en que el museo los ve a ellos mismos.

Desde el punto de vista del museo mismo, la implementación del sistema de realidad aumentada inició un proceso de modernización de sus recursos tecnológicos muy necesario desde hace tiempo. El nuevo sistema de información permite registrar información de vital importancia sobre el museo como son los objetos y la multimedia asociada a éste.

Se ha logrado desarrollar una aplicación de RA en el ámbito del museo de física. La aplicación es parametrizable y extendible a varios museos. Se ha analizado la usabilidad de la aplicación mediante tests para ver cómo era la interacción con los usuarios. A partir de este análisis se obtuvieron resultados satisfactorios sobre el desempeño del sistema.

En conclusión, hemos desarrollado un prototipo estable y funcional.

10.3 Evaluación del proyecto

Como resultado de las pruebas de usabilidad vimos que los visitantes del museo se mostraron muy cómodos utilizando la aplicación. Como ya hemos explicado en el capítulo anterior, se acercaron de una manera sencilla e intuitiva al concepto de realidad aumentada.

Por parte del museo la evaluación fue muy satisfactoria. Al preguntarles que opinaban sobre la implementación de este nuevo sistema de información ellos se mostraron conformes y agradecidos ya que también entendieron el valor agregado que se suma a la experiencia de visita al museo.

10.4 Trabajo futuro

Quedan muchas mejoras que se pueden hacer sobre este prototipo, a conocer desde la aceptación de los usuarios y necesidades de negocio específicas.

La implementación de esta primera versión del prototipo ha dejado ver puntos débiles que deberían ser mejorados o añadidos que generarían nuevos aspectos funcionales del mismo:

- Refactorización general del código: Hemos puesto empeño en la desarrollar funcionalidades en el prototipo dejando de lado algunos detalles estéticos de código fuente. Por lo tanto creemos que se podría mejorar la implementación.
- Nuevas funcionalidades en la aplicación móvil: se podría agregar la opción para marcar como favorito al estilo “me gusta” de Facebook.
- Mejorar el administrador Web: Se podría mejorar la eficiencia general de la aplicación utilizando una caché para almacenar los objetos. Así se evitarían demasiados accesos a la base de datos.

El crecimiento que están teniendo las tecnologías de realidad aumentada en el mercado de las aplicaciones móviles ha dado lugar al nacimiento de startups y empresas de desarrollo. Es por esto que este tipo de aplicaciones se perfilan como promotores del marketing móvil. Su aplicación cubre varios sectores, tanto públicos como empresariales.

Capítulo 11. Bibliografía

- [1] Frank H. P. Fitzek, Frank Reichert. Mobile Phone Programming and Its Application to Wireless Networking. Springer, 2007
- [2] Fling Brian. Mobile Design and Development. 1° Edición, O'Reilly Media, Inc. 2009
- [3] Ajay R. Mishra. Cellular Technologies for Emerging Markets: 2G, 3G and Beyond. John Wiley & Sons, Ltd. 20120
- [4] Zheng, Lionel Ni. Smart Phone and Next Generation Mobile Computing. Elsevier Inc. 2006
- [5] Datos estadísticos. [en línea] <<http://googleblog.blogspot.com.ar/2012/05/new-research-shows-smartphone-growth-is.html>>
- [6] JAVA ME [en línea]. <http://www.java.com/es/download/faq/whatis_j2me.xml>
- [7] BREWMP [en línea]. <<https://www.brewmp.com/>>
- [8] Windows Phone. [en línea]. <<http://www.windowsphone.com>>
- [9] Limo [en línea]. <www.limofoundation.org>
- [10] WebOS [en línea]. <<http://www.openwebosproject.org/>>
- [11] Blackberry [en línea]. <<http://ar.blackberry.com/>>
- [12] iPhone [en línea]. <<http://www.apple.com/es/iphone/>>
- [13] Michael J. Jipping. Symbian OS Communications Programming. Editorial Wiley, 2002
- [14] Frank McPherson. How to Do Everything with Windows Mobile. Editorial McGraw-Hill, 2006
- [15] Saad Z. Asif. Next Generation Mobile Communications Ecosystem: Technology Management for Mobile.
- [16] iOS. [en línea]. <<https://developer.apple.com/>>
- [17] Licencia Apache. [en línea] <<http://www.apache.org/licenses/LICENSE-2.0.html>>
- [18] Android. [en línea] <<http://developer.android.com>>
- [19] George H. Forman y John Zahorjan. The Challenges of Mobile Computing. IEEE Computer Society, Vol 27
- [20] Andrei Cristian Spataru. Agile Development Methods for Mobile Applications.
- [21] Adam Warterski y otros. Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone [en línea]. <http://www.adamwesterski.com/wp-content/files/docsCursos/Agile_doc_TemasAnv.pdf>
- [22] Azuma, R. (1997) "A Survey of Augmented Reality". Presence: Teleoperators and Virtual Environments, 6(4), pp 355-385.
- [23] Azuma, R.; Baillot, Y.; Behringer, R.; Feiner, S.; Julier, S. and MacIntyre, B. (2001) "Recent Advances in Augmented Reality". IEEE Computer Graphics and Applications, 21(6), 34-47.
- [24] María José Abasolo. Capítulo 3. Realidad aumentada. Curso Postgrado Facultad de Informática (UNLP). Septiembre 2012.
- [25] ARToolKit. [en línea] <<http://www.hitl.washington.edu/artoolkit/>>
- [26] María José Abasolo. Capítulo 1. Realidad aumentada. Curso Postgrado Facultad de Informática (UNLP). Septiembre 2012.

- [27] Zhou, F.; Dhu, H. and Billinghurst, M. (2008) "Trends in Augmented Reality Tracking, Interaction and Display: A Review of Ten Years of ISMAR" Proceedings of 7th IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR 2008), pp 193-202.
- [28] Kato, H. and Billinghurst, M. (1999) "Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System." Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99), pp 85-94.
- [29] ARTag [en línea] <<http://www.artag.net>>
- [30] ARToolkitPlus [en línea] <<http://handheldar.icg.tugraz.at/artoolkitplus.php>>
- [31] Thrun, S.; Liu, Y.; Koller, D.; Ng A. and Durrant-Whyte, H. (2004) "Simultaneous Localisation and Mapping with Sparse Extended Information Filters", International Journals on Robotics Research, 23(7-8), pp 693-716
- [32] QR code [en línea]. <http://www.itsc.org.sg/pdf/synthesis08/Three_QR_Code.pdf>
- [33] Denso Wave Corp. Documentation of the QRcodes <<http://www.denso-wave.com/qrcode/index-e.html>>
- [34] Open Handset Alliance, [en línea] <<http://www.openhandsetalliance.com/>>
- [35] Gargenta Marko. Learning Android. 1º Edición, O'Reilly Media, Inc. 2011
- [36] Distribución de Versiones Android. [en línea] <<http://developer.android.com/about/dashboards/index.html>>
- [37] Fundamentos de una aplicación Android. [en línea] <<http://developer.android.com/guide/components/fundamentals.html>>
- [38] Intents y Intents Filters. [en línea] <<http://developer.android.com/guide/components/intents-filters.html>>
- [39] Procesos en Android. [en línea] <<http://developer.android.com/guide/components/processes-and-threads.html>>
- [40] Instalacion SDK Android. [en línea]. <<http://developer.android.com/sdk/installing.html>>
- [41] Emulador Android. [en línea] <<http://developer.android.com/tools/devices/emulator.html>>
- [42] DDMS Android. [en línea] <<http://developer.android.com/tools/debugging/ddms.html>>
- [43] ADB Android. [en línea]. <<http://developer.android.com/tools/help/adb.html>>
- [44] ISO 9241-11. [en línea] <http://www.usabilitynet.org/tools/r_international.htm#9241-11>
- [45] Vos, Tanja. Modelos de Madurez de Usabilidad.
- [46] Jonathan Gladden. Research Paper Art 894x12. History of Computer Graphics
- [47] Interfaces hápticas. [en línea]. <<http://www.usabilityfirst.com/glossary/haptic-interface/>>
- [48] Interfaces modales. [en línea]. <<http://www.usabilityfirst.com/glossary/modal/>>
- [49] Sergio Ortega Santamaría. Introducción a la usabilidad y su evaluación. Universitat Oberta de Catalunya. 2011
- [50] Soporte de múltiples pantallas. [en línea] <http://developer.android.com/guide/practices/screens_support.html>
- [51] Fernández Zumaquero, S. Métodos de evaluación de la usabilidad para entornos de Realidad Virtual, Realidad Aumentada y Sistemas Ubicuos.
- [52] JSON. [en línea]. <<http://www.json.org/>>

- [53] PostgreSQL. [en línea]. <<http://www.postgresql.org/>>
- [54] Framework Spring. [en línea] <<http://www.springsource.org/>>
- [55] Hibernate. [en línea] <<http://www.hibernate.org/>>
- [56] JQuery. [en línea]. <<http://jquery.com>>
- [57] Inversión de Control. <<http://static.springsource.org/spring/docs/2.5.3/reference/beans.html>>
- [58] AOP. [en línea]. <<http://static.springsource.org/spring/docs/2.5.5/reference/aop.html>>
- [59] Spring MVC. [en línea]
<<http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html>>
- [60] Spring Security. [en línea] <<http://www.springsource.org/spring-security>>
- [61] Patrones Android. [en línea] <<http://www.androidpatterns.com/>>
- [62] Patrón de diseño: Tablero de opciones. [en línea]
<http://www.androidpatterns.com/uap_pattern/dashboard-%E2%80%93-features>
- [63] Patrón de diseño: Lista estática. [en línea] <http://www.androidpatterns.com/uap_pattern/static-list>
- [64] Patrón de diseño: Pestañas. [en línea] <http://www.androidpatterns.com/uap_pattern/module-tabs>
- [65] Patrón de diseño: Expandir y ocultar teclado. [en línea]
<http://www.androidpatterns.com/uap_pattern/soft-keyboard-pan-scan>
- [66] Patrón de diseño: Rueda de Progreso. [en línea]
http://www.androidpatterns.com/uap_pattern/progress-wheel
- [67] SQLite. [en línea] <<http://www.sqlite.org/docs.html>>
- [68] SharedPreferences. [en línea]
<<http://developer.android.com/reference/android/content/SharedPreferences.html>>
- [69] Cambios en tiempo de ejecución Android. [en línea]
<<http://developer.android.com/guide/topics/resources/runtime-changes.html>>
- [70] GSON. [en línea] <<http://code.google.com/p/google-gson/>>
- [71] Librería ZXing. [en línea] <<http://code.google.com/p/zxing/>>