Título: Aplicación para realizar cuestionarios desde dispositivos Android

Autor: Sarah Bouayad

Fecha: 22 de Enero de 2013

Directora: María José Casany Guerrero

Departamento de la directora: ESSI

Co-Director: Marc Alier Forment

Departamento del co-director: ESSI

Titulación: Ingeniería informática

Centro: Facultat d'Informàtica de Barcelona (FIB) Universidad: Universitat Politècnica de Catalunya (UPC)

Datos del proyecto

Título del proyecto: Aplicación para realizar cuestionarios desde dispositivos Android

Nombre del estudiante: Sarah Bouayad

Titulación: Ingeniería informática superior

Créditos: 37.5

Directora: Maria José Casany Guerrero

Co-director: Marc Alier Forment

Departamento: ESSI

Miembros del tribunal

Directora: Maria José Casany Guerrero

Co-director: Marc Alier Forment

Presidente: Enrique Mayol Sarroca

Vocal: José Antonio Lubary Martínez

Calificación

Calificación numérica:

Calificación descriptiva:

Fecha:

Agradecimientos

En primer lugar, me gustaría agradecer a María José Casany el haber aceptado dirigir mi proyecto final de carrera y el haberme dado la oportunidad de participar en el proyecto Moodbile estos meses. Su implicación y su comprensión han sido esenciales para que este proyecto se llevara a cabo con éxito.

En segundo lugar, agradecer a todas las personas que cada una a su manera me han aportado algo durante todos mis años de estudio, profesores, amigos, compañeros de clase, compañeros de trabajo, etc.

Por último, agradecer a mis padres el apoyo incondicional que siempre me han dado. Me gustaría dedicarles este proyecto final de carrera ya que sin su ayuda, no estaría en el lugar en el que estoy a día de hoy. Gracias por todo.

Índice

Índice de figuras

1. Definición:

1.1. Introducción

En los últimos años, se ha ido observando una subida exponencial del uso de los llamados *smartphones* (teléfonos inteligentes), así cómo de los hábitos de sus consumidores. Lo que antes eran llamadas y SMS, se ha convertido en *WhatsApps*, notificaciones y actualizaciones de estado en toda y cada una de las redes sociales en las que estamos dados de alta.

Es un hecho ya, que desde que aparecieron las potentes conexiones de datos, prácticamente todas las tareas que antes requerían del uso de un ordenador, se pueden ahora llevar a cabo en los *smartphones*, sumándole además, el factor movilidad que nos brindan estos últimos.

Figura 1 – Logo de Android

Prueba de ello es el aumento de las ventas de móviles smartphones en España, y en el resto del mundo frente al desplome de las ventas de móviles *tradicionales.* En la siguiente figura, vemos como han evolucionado las ventas de teléfonos móviles en España en los últimos años.

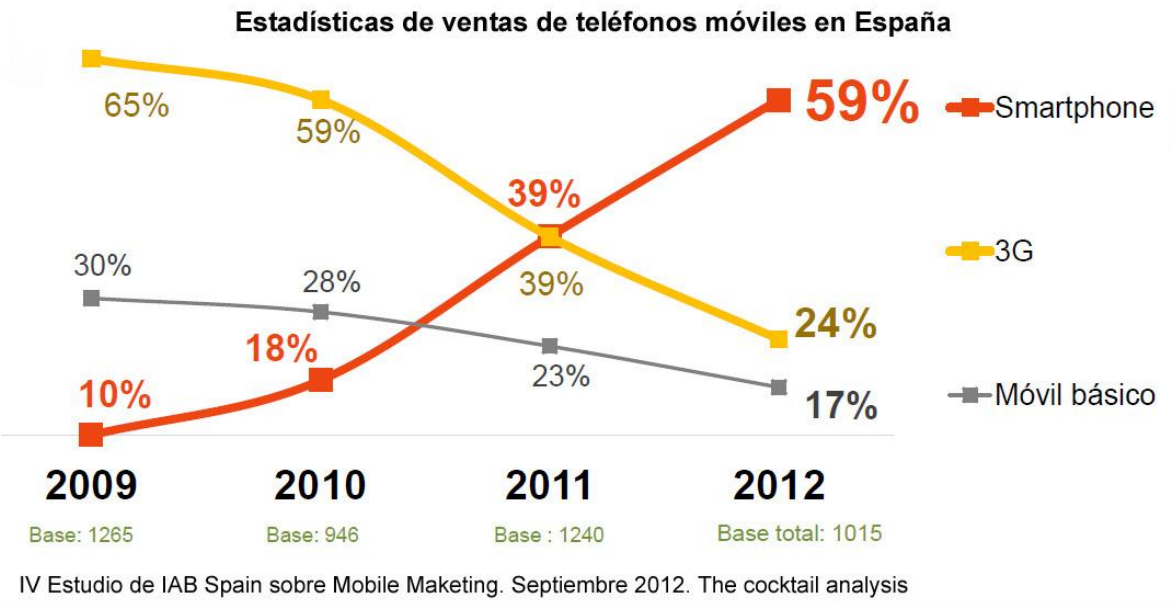


Figura 2 - Estadísticas de ventas de teléfonos móviles en España

Actualmente, existen cuatro sistemas operativos sobre los cuales se basa el desarrollo de las principales aplicaciones móviles:

- Android de Google
- iOS de Apple
- Windows Phone de Microsoft
- BlackBerryOS de RIM

Android es un sistema operativo desarrollado en sus inicios por Android Inc., firma que fue comprada posteriormente por Google en el año 2005. A día de hoy, tiene más de 700.000 aplicaciones disponibles en su tienda Google Play (el anteriormente

llamado Android Market), cifra confirmada por Google a finales de Octubre del 2012; lo que le permite estar a punto de alcanzar su competidor más directo, Apple, que cuenta con unas 750.000 aplicaciones en la AppStore.

A pesar de tener menos aplicaciones, Google puede presumir de ser el sistema operativo más usado en dispositivos móviles, ya que tres de cada cuatro smartphones vendidos tienen Android como sistema operativo, como se puede apreciar en la siguiente figura.

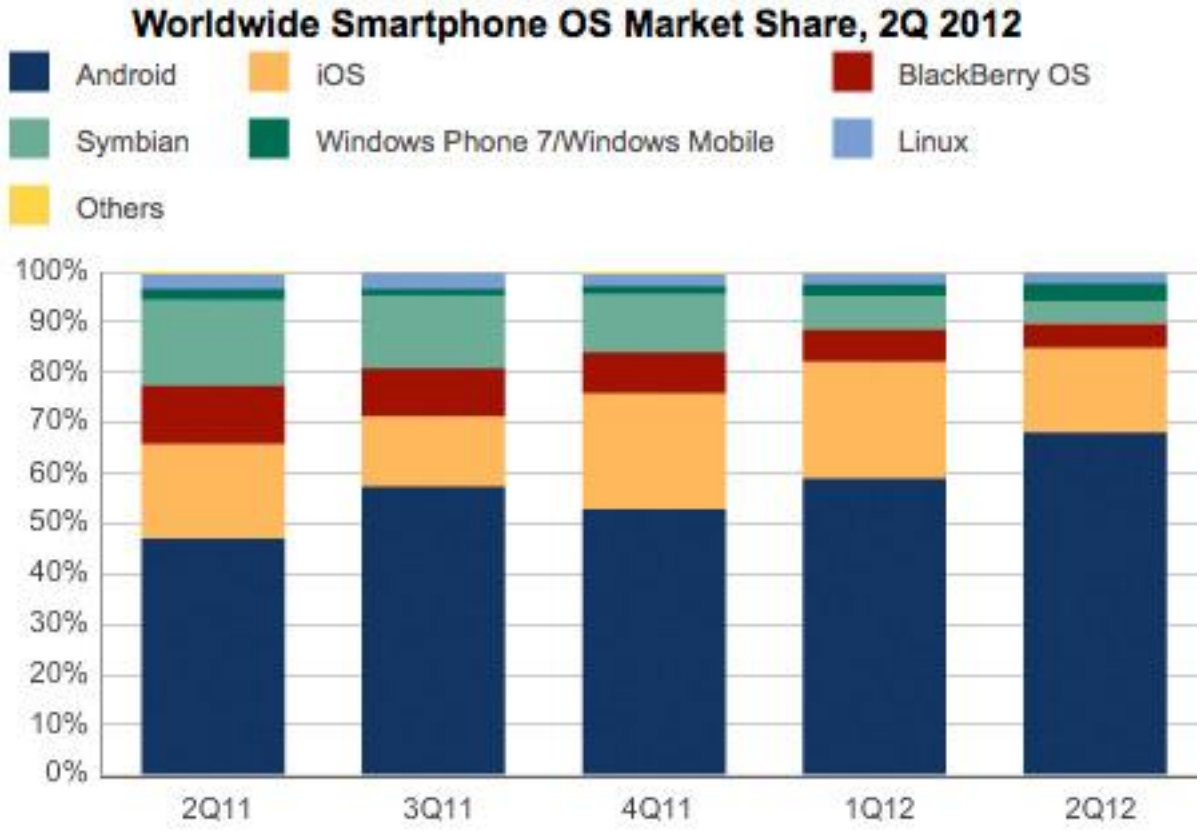


Figura 3 - Cuota de mercado de los smartphones, 2Q 2012

Debido a la rápida expansión de los *smartphones*, hoy en día nos encontramos con todo tipo de aplicaciones, pensadas para proporcionar multitud de servicios a los usuarios de móviles. Las aplicaciones más famosas son las orientadas a redes sociales (Facebook, Twitter,…) o a servicios de mensajería (Whatsapp), pero tenemos también

aplicaciones de banca online, aplicaciones de localización, aplicaciones orientadas a uso empresarial, etc…, además del tipo que nos interesa en este proyecto, que son las aplicaciones educativas.

Este proyecto se centra en el desarrollo de una aplicación nativa para dispositivos móviles con sistema operativo Android que permita a los alumnos de *Moodle 2.0* acceder e interactuar con los módulos de cuestionarios y tareas.

A grandes rasgos, *Moodle* es una plataforma educativa virtual orientada a la gestión de cursos que permite a los profesores crear comunidades de aprendizaje en línea. Consta de varios módulos personalizables (módulo de tareas, módulo de consulta, módulo foro, módulo diario, módulo cuestionario, módulo recurso, módulo encuesta, módulo wiki...).

Una de las principales ventajas de este proyecto es la optimización del factor visual, ya que al ser una aplicación nativa, el usuario no necesita un navegador, y permite una visualización mucho más nítida y más adaptada a dispositivos android, para un mejor manejo de los contenidos. Otra de las ventajas, es que los usuarios podrán interactuar con los contenidos de este módulo sin necesidad de tener conexión permanente a Internet, sino sólo para acciones puntuales de acceso al servidor.

En definitiva, mi proyecto consiste en implementar un cliente capaz de conectarse con el servidor de Moodle, y de ofrecer los contenidos de los módulos *Quiz* y *Assignment* a dispositivos móviles con sistema operativo Android.

Para que una aplicación con estas propiedades fuera posible, era necesario contar con una API de *webservices* que permiten a Moodle ofrecer un conjunto de sus funcionalidades y datos a clientes externos. Esta API está desarrollada y mantenida por los miembros del equipo de trabajo del *proyecto Moodbile*. *Moodbile* es un proyecto en desarrollo, cuyo objetivo es la creación de aplicaciones móviles que accedan a un servidor de *Moodle*. Una de estas aplicaciones cliente está basada en Android, y uno de

los objetivos esenciales de este proyecto es la integración de los dos nuevos módulos mencionados anteriormente dentro de la misma.

Por último, cabe destacar que dentro del proyecto *Moodbile* existen dos aplicaciones más, una desarrollada para IPhone (con iOS), y otra en HTML5, pensada para funcionar sobre navegadores web de los dispositivos móviles indistintamente de su sistema operativo.

1.2. Objetivos

Este proyecto tiene como objetivo principal diseñar y desarrollar una aplicación que funcione sobre dispositivos Android, que permita a usuarios móviles acceder a los módulos *Quiz* (Cuestionario) y *Assignment* (Tarea) de *Moodle* 2.0.

En un primer tiempo, el proyecto consistía en la adaptación completa del módulo Quiz a dispositivos Android, pero dada la carga de trabajo que esto implicaba tanto para mí como para los desarrolladores de Webservices y la necesidad de coordinación con los miembros del equipo de trabajo Moodbile, se redujeron las funcionalidades de este módulo y se agregó la implementación del módulo Assignment y la integración de ambos en el cliente que contiene los demás módulos. Más adelante, explicaremos con más detalle las causas de este cambio de requisitos.

A continuación, detallamos los principales objetivos de este proyecto.

1.2.1. Funcionalidades de los módulos Quiz y Assignment de Moodle:

Nuestra aplicación cliente debe ser capaz de interactuar con el sistema de Moodle, y ofrecer al usuario la posibilidad de utilizar las funcionalidades más importantes de estos dos módulos.

- Módulo Quiz:
  - *Seleccionar un curso y listar los quizzes que tenga asociados*
  - *Descargar quiz*
  - *Contestar quiz*
  - *Grabar respuestas en el dispositivo móvil*

- Módulo Assignment:
  - *Listar todas las tareas de un usuario (agrupadas por cursos)*
  - *Listar las tareas de un curso (después de seleccionarlo)*
  - *Seleccionar una tarea y ver sus detalles*
  - *Upload de un fichero al servidor con las respuestas del assignment*
  - *Recibir nota y feedback*

Comentar que estas funcionalidades se han implementado para *Assignments* de tipo "Single upload", dejando los demás tipos para un futuro proyecto.

1.2.2. Adaptar los contenidos de Moodle a un dispositivo móvil:

Ha sido necesario adaptar y filtrar los contenidos de Moodle a dispositivos móviles, para obtener una visualización acorde con este tipo de pantallas.

Se ha intentado implementar un diseño coherente con el diseño de los demás módulos de Moodbile, que a su vez aplicaban patrones en las vistas para cumplir con las exigencias de los usuarios y permitir que la apariencia sea compatible a distintos tipos de pantallas, ya que Android (a diferencia de iOS por ejemplo) está implantado en dispositivos móviles de distintos fabricantes.

1.2.3. Interactuar con el sistema de manera diferente según la conexión disponible:

Según si se dispone de una conexión WiFi o una conexión de datos, la aplicación se comportará de manera distinta para minimizar gastos innecesarios. Por otro lado, nos basamos en el concepto de persistencia de datos, para permitir que la aplicación pueda seguir funcionando incluso en ausencia de conexión. De este modo, sólo hace falta estar conectado para la descarga de quiz y para el upload de un assignment.

Cualquier otra acción, como contestar al cuestionario, se puede desarrollar sin necesidad de estar conectado basándonos en todo momento en la base de datos del móvil para evitar la pérdida de datos.

1.2.4. Integración de los módulos a Moodbile:

Uno de los principales objetivos del proyecto es conseguir una implementación de los módulos que se pueda adaptar e integrar con facilidad a la aplicación Moodbile.

Para ello, ha sido necesario estudiar detenidamente la arquitectura y el diseño de la aplicación ya existente, y aplicar la misma metodología y los mismos patrones en los nuevos módulos. Una vez integrados los módulos, las funcionalidades logradas son las siguientes:

- *Autenticación a la aplicación vía user/password o vía OAuth.*
- *Acceso a todas las funcionalidades y todos los módulos ya implementados de la aplicación.*
- *Acceso a los nuevos módulos implementados.*

1.3. Alcance del proyecto

Existe una gran variedad de sistemas operativos para dispositivos móviles, que a su vez soportan unos determinados lenguajes de programación para el desarrollo de aplicaciones. Este proyecto se ha basado en el ya desarrollado cliente para *Android*, pero, como se mencionó anteriormente, existe un cliente HTML5 para usarlo desde los navegadores web de distintos sistemas operativos, y otro para iOS (iPhone) con una gran cuota de mercado también.

De esta forma, conseguimos aprovechar todas las propiedades y el potencial de dos de los sistemas operativos para dispositivos móviles más usados, además de contar con un cliente más versátil que se pueda usar a través del navegador de distintos sistemas operativos.

La complejidad del proyecto hace inviable ponerle fin en un proyecto final de carrera, de hecho ya son varios los proyectos que han precedido y seguramente varios los que sucederán.

1.4. Visión general del sistema

A continuación, explicaremos de modo genérico las funcionalidades más importantes del sistema desarrollado. Algunas de estas funcionalidades, ya estaban implementadas en la aplicación existente, y no ha sido necesario modificarlas, pero hemos juzgado necesario volver a mencionarlas, o bien porque están fuertemente ligadas a las funcionalidades nuevas, o bien por su importante peso en la aplicación.

- *Entrar al sistema de moodle*

Si se dispone de una cuenta de usuario en el servidor de *Moodle,* se podrá acceder como alumno a la ventana principal de la aplicación previa validación de las credenciales.

- *Acceder a los módulos de la aplicación Moodbile*

El usuario podrá acceder a todos los módulos de Moodle ya implementados, como los foros, participantes, cursos, etc…, e interactuar correctamente con todos sus contenidos y funcionalidades.

- *Obtener la lista de quizzes asociados a un curso*

Una vez autenticado el usuario y obtenidos sus datos, puede acceder al listado de cursos que tiene asociados, y a continuación, seleccionar uno y ver la lista de cuestionarios (entre otros módulos) que tiene asociados, junto a una breve descripción de cada uno.

- *Descargar quiz*

Desde el listado de módulos de un curso, el usuario puede seleccionar un Quiz y descargarlo. Esta acción guarda el quiz en la base de datos del dispositivo móvil, para que esté disponible para consultas futuras.

Comentar que el Quiz se descarga *una única vez* desde el servidor de Moodle, por lo que cualquier consulta a partir de aquel momento, se haría sobre los datos obtenidos de la base de datos y no del Servidor. Se ha tomado esta decisión al considerar que una vez puesto un Quiz a disposición de los alumnos, es muy poco coherente que se cambien las preguntas a posteriori, ya que eso podría crear conflictos a la hora de enviar las respuestas por parte de los alumnos.

- *Contestar quiz*

El usuario puede contestar a cualquier quiz descargado anteriormente incluso estando en modo offline. Puede ir guardando las respuestas una a una sin necesidad de contestar al cuestionario entero de una única pasada. Estas respuestas se graban en

la base de datos del dispositivo móvil, y mientras no se haya hecho la entrega del Quiz, se pueden rectificar en cualquier momento.

- *Listar assignments asociados a un usuario*

Un usuario autenticado puede acceder a la lista de todos sus assignments, agrupados por curso. Puede acceder desde la pantalla principal seleccionando el "Listado de assignments", o bien seleccionando el icono de Assignments en el menú deslizante.

- *Listar assignments asociados a un curso*

Un usuario autenticado puede acceder al listado de cursos que tiene asociados, y a continuación, seleccionar uno y ver la lista de tareas (entre otros módulos) que tiene asociadas.

- *Seleccionar un assignment y acceder a sus detalles*

Desde ambos listados, el usuario puede seleccionar un assignment, y consultar sus detalles: texto de la tarea, fecha límite de entrega, fichero entregado si lo hay.

- *Enviar un fichero como entrega de una tarea*

El usuario, desde la pantalla de la tarea, puede escoger un fichero y entregarlo como respuesta de esta tarea. En el caso de que este assignment tenga ya un fichero entregado, se sobrescribirá.

- *Recibir la nota y el feedback del profesor*

Desde la pantalla principal y el menú deslizante, el usuario puede acceder a una lista de elementos, llamados *graduables.* Los elementos graduables son aquellos elementos que el alumno puede enviar al servidor de Moodle, y seguidamente, recibir una nota y/o un feedback por parte de su profesor. En esta lista, aparecerán entre otros

los assignments, y el usuario puede seleccionar uno para acceder a consultar la nota que le haya concedido el profesor.

- *Uso sin conexión*

La aplicación permite que el usuario, *una vez autenticado*, pueda consultar toda la información visitada previamente aunque se pierda la conexión. Por lo tanto, tendrá acceso a todos los contenidos almacenados en la caché y en la base de datos del dispositivo móvil. Esto es, no tendrá acceso a la información actualizada, sino a versiones correspondientes a la última conexión.

- *Desconectarse y cambiar de usuario*

El usuario puede desconectarse en todo momento, dejando de estar validado y volviendo a la pantalla de conexión inicial.

- *Configurar estilo visual*

El usuario tiene la posibilidad de escoger entre distintos temas visuales desde las opciones.

- *Vaciar caché*

Cada vez que se autentique un nuevo usuario, se vaciará la caché de manera automática. Se podrá vaciar manualmente también para borrar la información persistente cuando lo desee el usuario.

2. Entorno de trabajo

En este capítulo, describiremos un poco el entorno de trabajo sobre el cual nos basamos para desarrollar la aplicación. Por una parte, *Moodle* que es la plataforma desde dónde obtendremos los datos y contenidos vía servicios web, y por otra, *Android* y todas las herramientas, librerías, etc… que hicieron falta en el desarrollo técnico del proyecto.

2.1. Moodle

De un tiempo para hoy, se están implantando cada vez más plataformas de aprendizaje en línea ó *e-learning* como una herramienta más dentro de la enseñanza para apoyar la formación presencial.

Estas plataformas permiten entre otras cosas, el acceso a contenidos y recursos, la realización de actividades, y la comunicación entre el profesorado y los alumnos.

Una de las más usadas es *Moodle*, ya que es una plataforma altamente personalizable, que además es de software libre.

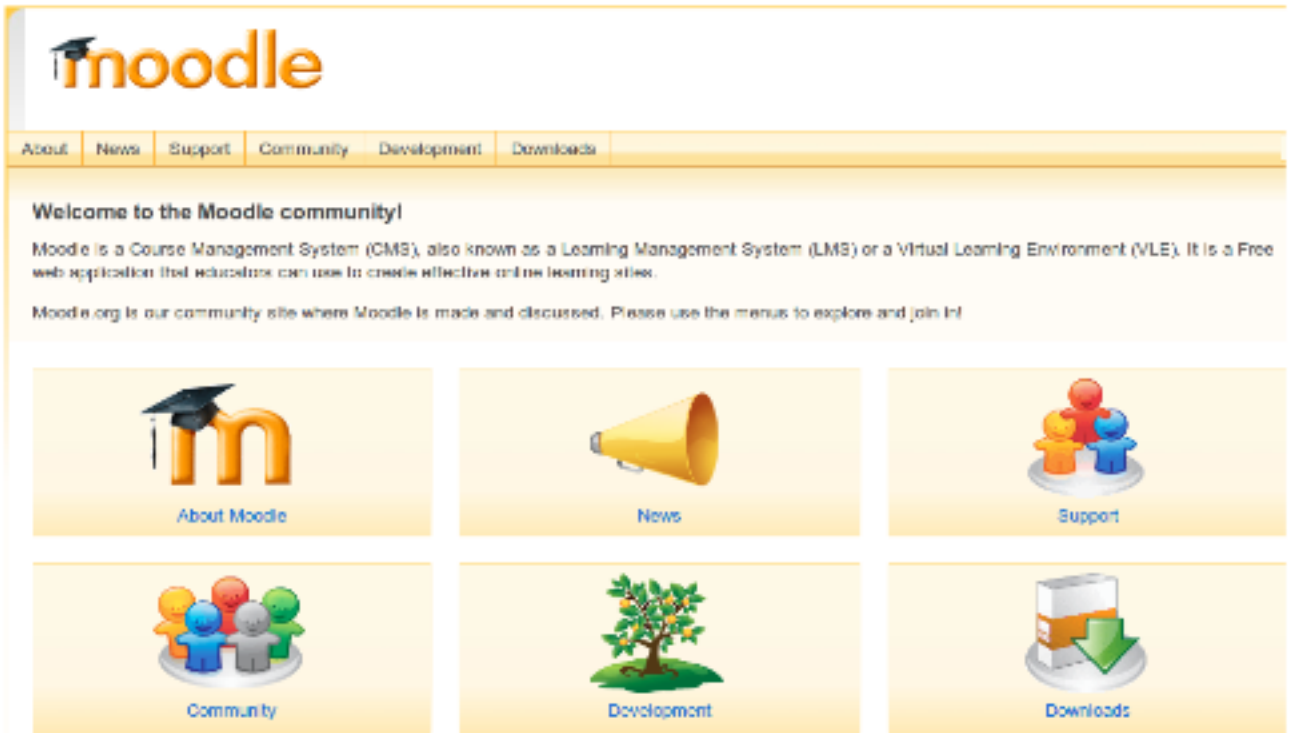


Figura 4 - Portal de Moodle

Hace unos años, las estadísticas de Moodle apuntaban a que tenía dos millones de usuarios en el mundo. Hoy son 25 millones y es una cifra a la baja, ya que el registro en la *web* de Moodle es voluntario y minoritario. En España, las universidades han adoptado Moodle masivamente y múltiples proyectos institucionales ofrecen apoyo para implementarlo en escuelas e institutos.

Para instalar la plataforma virtual es preciso contar con un servidor web (local o en la red) que cuente con el servidor Apache y un sistema de bases de datos como MySQL. Sobre el servidor, se instala la plataforma Moodle descargada previamente desde la página www.moodle.org. En nuestro caso, para poder hacer las pruebas, nos hemos basado en un servidor en internet.

Según se indica en la propia web de Moodle en español, Moodle es un paquete de software para la creación de cursos y sitios Web basados en Internet. Es un proyecto

en desarrollo, diseñado para dar soporte a un marco de educación social constructivista. Se distribuye como software libre bajo una licencia GNU GPL.

En otras palabras, Moodle es un campus virtual donde el profesor puede distribuir materiales, encuestas y cuestionarios a los alumnos, crear foros de debate, glosarios, estadísticas y calendarios de asignaturas, comunicarse con los estudiantes por correo o mensajería instantánea, hacer tutorías electrónicas en privado o en grupo, recoger trabajos, repartir notas, responder a las dudas de los alumnos, evaluar su participación... Todo de forma fácil y automatizada.

A continuación, explicaré brevemente el funcionamiento de los módulos de Quiz y Assignment que son los que nos interesan en este proyecto:

- Módulo *quiz* de Moodle

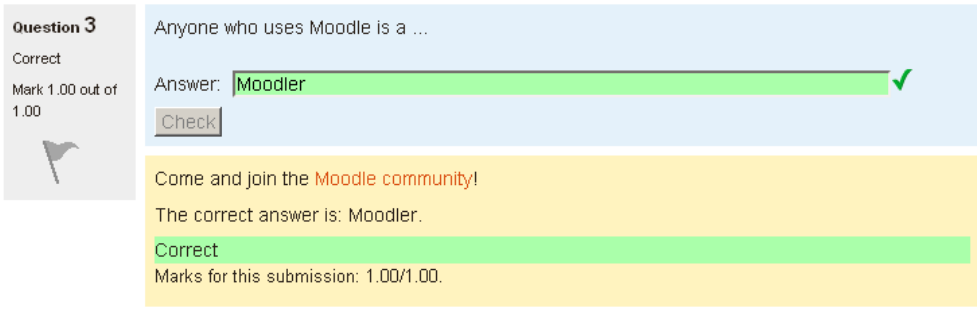


Figura 5 - Módulo quiz de Moodle

Este módulo permite al profesor diseñar y publicar cuestionarios. Existe una amplia variedad de tipos de preguntas (opción múltiple, verdadero/falso, respuestas cortas, etc...). Podemos permitir a los estudiantes repetir intentos en una pregunta o bien que respondan el cuestionario varias veces, y obtener una puntuación final calculada automáticamente. El profesor puede decidir si muestra algún mensaje o las respuestas correctas al finalizar el examen.

- Módulo *assignment* de Moodle

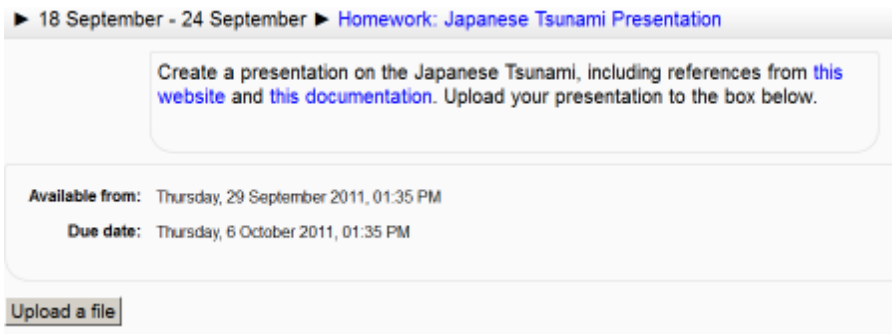


Figura 6 - Módulo assignment de Moodle

El módulo de tareas permite a los profesores recoger el trabajo de los estudiantes, revisarlo y calificarlo.

Los alumnos pueden entregar cualquier contenido digital, por ejemplo, documentos de texto, hojas de cálculo, imágenes, audio y video. Los assignments no tienen que consistir necesariamente en la carga de archivos. Los profesores pueden pedir a los alumnos escribir directamente en Moodle utilizando un assignment de tipo texto online. También hay tareas offline que se pueden utilizar para registrar las calificaciones en Moodle para actividades que no tienen un componente en línea.

Los cuatro tipos de tareas de Moodle son:

- Single upload
- Advances upload
- Online text
- Offline activity

Evidentemente, a la hora de adaptar estos dos módulos a móvil, hemos intentado dar soporte a las funcionalidades más importantes, ya que no pretendemos tener una aplicación que sea una copia conforme de Moodle, sino un cliente móvil, fácil

de usar, con las funcionalidades justas e imprescindibles y lo más adaptado posible a este tipo de dispositivos.

2.2. Android

Figura 7 - Evolución del número de terminales con Android

Android es un sistema operativo para dispositivos móviles, *smartphones* y actualmente, también para *tablets*. Android es el principal producto de la *Open Handset Alliance* (OHA), una alianza de 78 compañías que tratan de desarrollar estándares para dispositivos móviles, como Google, HTC, Dell, Intel, Motorola, Qualcomm, Samsung, LG, T-Mobile, Nvidia, etc...

Una de las cosas que hacen que Android sea un sistema operativo distinto a otros como iOS y Windows Phone, es que se desarrolla de forma abierta, lo que significa que no sólo pueden mejorarlo los desarrolladores de Google, sino que también se nutre de las aportaciones de desarrolladores externos. Así, cualquier desarrollador puede crear aplicaciones para Android sin la necesidad de pagar nada para obtener el kit de desarrollo (SDK) a diferencia de los desarrolladores de Apple.

Android está basado en el *kernel* de Linux y utiliza una variación del lenguaje de programación de Java. Asimismo, tiene una adaptación a la JVM (*Java Virtual Machine*) que se llama Dalvik.

A continuación, vemos una ilustración típica de la arquitectura de un sistema Android:

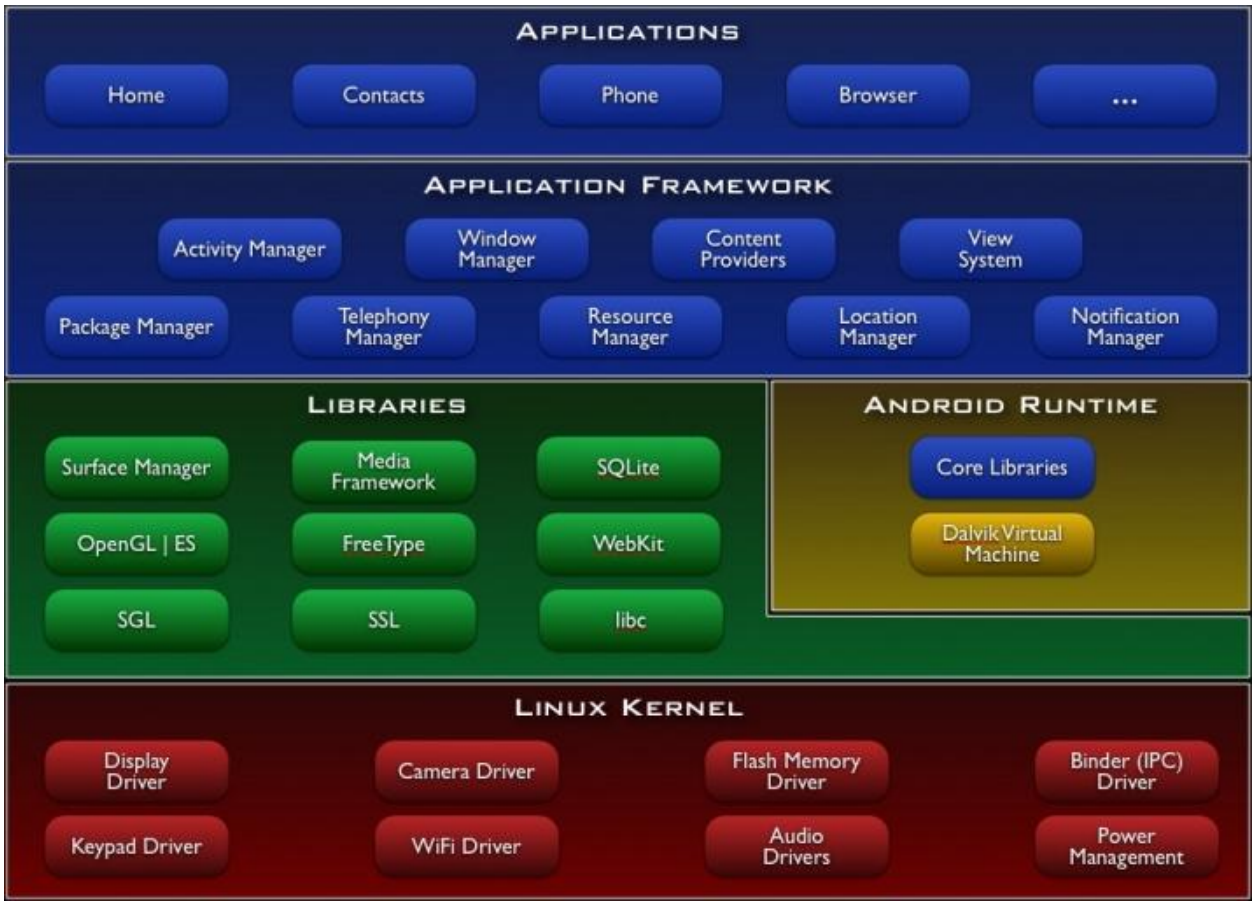


Figura 8 - Arquitectura de un sistema Android

Cada una de las capas utiliza elementos de la capa inferior para realizar sus funciones, por eso a este tipo de arquitectura se le conoce también como *pila*.

Android es una pila de software que incluye un sistema operativo, middleware y una capa de aplicaciones para que el teléfono pueda realizar funciones más allá de los dispositivos que se usaban antaño.

Vamos a hablar a continuación de la distribución de las distintas versiones de Android, y de la elección que se hizo para este proyecto.

Google en una nota del día 2 de Julio de 2012, actualizó los datos de su sistema operativo para móviles. De esta noticia, cabe destacar que Gingerbread (2.3) sigue a la cabeza con un 64% del total de móviles Android. Le sigue de lejos Froyo (2.2) con un 17%, Eclair (2.0, 2.1) con un 5% y HoneyComb (3.0, 3.1, 3.1) con un pobre 2,5%.

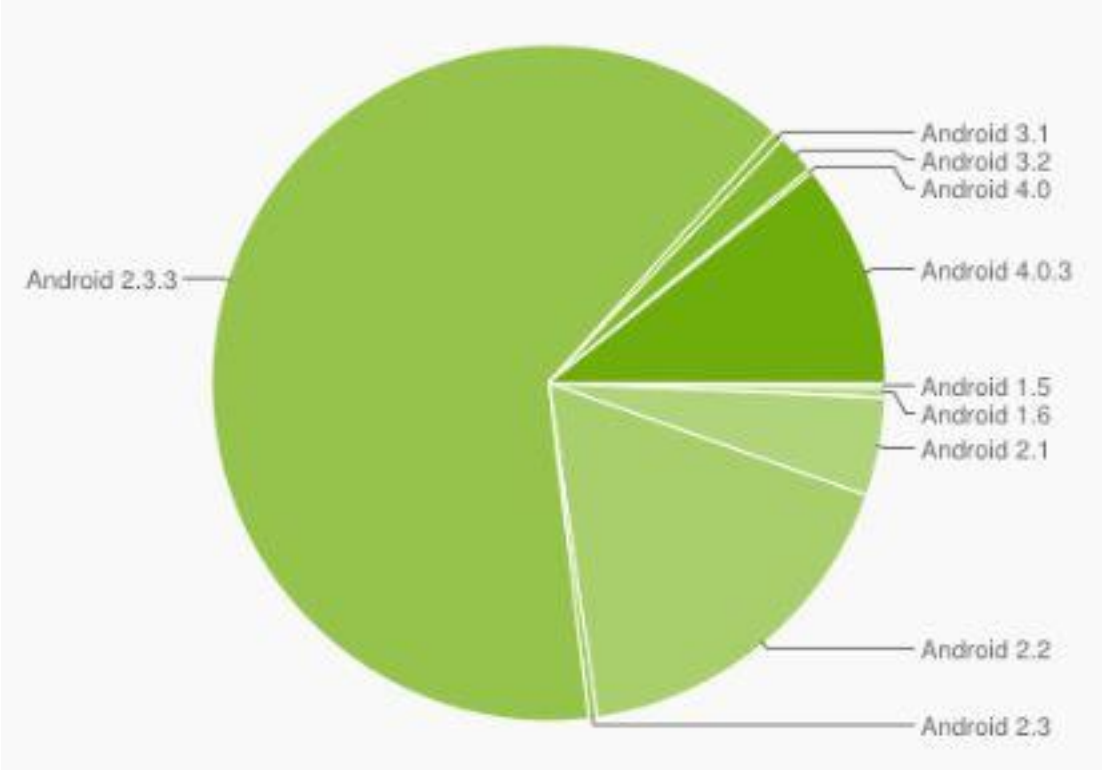


Figura 9 - Distribución de las versiones de Android (Julio 2012)

Estas estadísticas son las que han servido de base para elegir la versión sobre la cual se desarrolló el cliente ya existente, para que tenga el mayor alcance posible. Por lo tanto, sólo nos queda seguir con la misma versión, esto es la versión 2.1 mínima.

2.3. Herramientas de trabajo y otro software

Para el desarrollo de la aplicación, he decidido usar *Eclipse*. Primero, porqué al haber trabajado muchos años con este entorno, estoy muy familiarizada con todas sus funcionalidades, por lo que no me ha hecho falta ningún tutorial; esto, y que considero que es el programa que mejor se adapta a mis necesidades de desarrolladora Java. Dispone de una variedad de herramientas para depurar errores y la posibilidad de instalar un plugin que provee Android para el entorno de programación. Incluye un emulador de dispositivos móviles, que aunque consume muchos recursos, y funciona de manera excesivamente lenta, nos ha servido en alguna que otra fase del desarrollo.

Para la gestión de la base de datos SQLite, he usado el programa Navicat, que es un gestor de bases de datos ligeras, ideal para importar bases de datos y generar el diagrama de tablas correspondiente.

3. Tecnologías

En este apartado, describiremos las tecnologías que se han usado para llevar a cabo este proyecto, tecnologías para la comunicación con el servidor, para la autenticación, etc...

La API de Moodbile proporciona servicios web que permiten la comunicación de aplicaciones externas con Moodle, vía los siguientes protocolos:

- JSON-AJAX
- JSON-P con autenticación OAuth
- JSON-RPC
- *JSON-RPC con autenticación OAuth*
- REST con autenticación OAuth

Para que esta aplicación sea fácilmente integrable con la aplicación que contiene los demás módulos, hemos trabajado con el mismo protocolo que había escogido previamente el compañero, es decir JSON-RPC con autenticación Oauth.

Pero según la documentación de la que disponíamos así cómo el código que teníamos entre las manos, antes de esta elección, se hizo otra implementación con JSON-RPC sin autenticación Oauth. Después de estudiarlo detenidamente, me pareció que podía ser interesante implementar algo parecido para los nuevos WebServices, por lo que al final tenemos también dos formas de acceder al servidor:

- JSON-RPC
- JSON-RPC con autenticación Oauth 1.0a

La ventaja de la autenticación con Oauth es que nuestro cliente no trata de ninguna manera con los datos de acceso del usuario directamente. El procedimiento es el siguiente: el cliente se conecta a Moodle directamente a través de un navegador, el usuario introduce sus credenciales, y es el propio servidor de Moodle el encargado de

validarlos o no. De esta manera, se consigue más seguridad, ya que a partir de ahí, los servicios json-rpc con oauth se limitan a firmar las peticiones para que Moodle las acepte.

3.1. Webservices con JSON-RPC 2.0

Para poder acceder a las funcionalidades y contenidos de Moodle, hace falta establecer una comunicación con el servidor y hacer llamadas a los servicios web de la API de Moodbile. Para ello, nos hemos basado en el protocolo JSON-RPC como hemos mencionado en el anterior apartado.

JSON-RPC es un protocolo ligero de llamada a procedimiento remoto codificado en JSON. Es un protocolo muy simple, que define sólo unos pocos tipos de datos y comandos. Funciona enviando una solicitud al servidor. El cliente (en este caso la aplicación android) suele ser software que quiere llamar a un método único de un

sistema remoto. El método remoto es invocado por el envío de una petición a través de HTTP (se puede enviar por TCP/IP también, pero no es el caso en nuestra aplicación).

La petición debe contener tres propiedades determinadas:

- *método* - Una cadena con el nombre del método que se invoca.
- *params* - Una lista de objetos que se pasan como parámetros al método.
- *id* - Un valor de cualquier tipo, que se utiliza para que la respuesta coincida con la solicitud.

La respuesta debe, a su vez, contener tres propiedades determinadas:

- *resultado* - Los datos devueltos por el método invocado. Si se produce un error mientras se invoca el método, este valor debe ser nulo.
- *error* - Un código de error especificado si hubo un error de invocar el método, si no nulo.
- *id* - El identificador de la solicitud que está respondiendo.

De este modo, con JSON-RPC ya hemos establecido la comunicación necesaria entre el cliente y el servidor Moodle.

3.2. Autenticación por oauth 1.0.a

Oauth es un estándar abierto, lanzado hacia finales de 2007, que define un mecanismo para que una aplicación cliente pueda acceder a la información de un usuario en otra (el servidor de Moodle en nuestro caso) sin tener que informar a la primera del usuario y contraseña.

A continuación, voy a explicar, de manera muy breve, el proceso de autenticación por Oauth. No entraré en detalles, ya que este proceso está completamente heredado de la implementación del compañero, por lo que sólo nos interesa conocerlo a grandes rasgos.

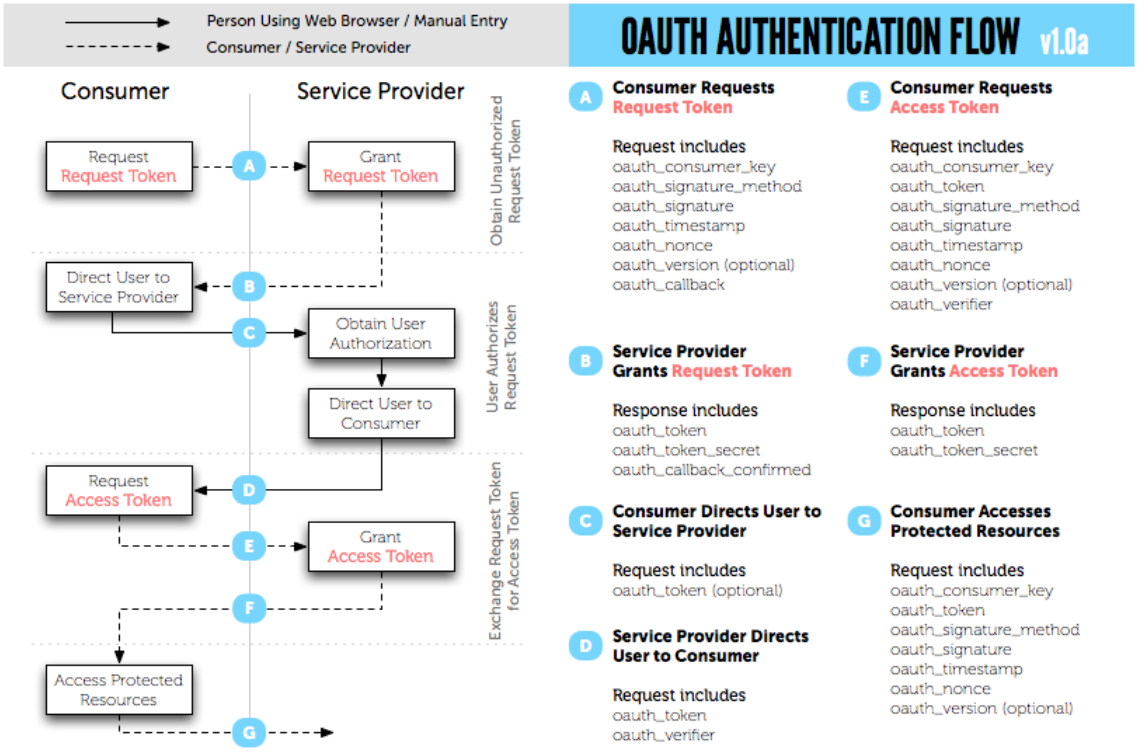


Figura 10 - Flujo de autenticación por OAuth v1.0a

Para este proceso, se necesitan tres partes:

- Proveedor de servicios (Servidor): Es un sitio o servicio web que contiene información de usuarios cuyo acceso es restringido. Estos *providers* publican una API que soporta el protocolo de autenticación Oauth.
- Usuario: Cualquier persona que tiene una cuenta en el *Service Provider*.
- Consumidor (Cliente): Es un sitio o una aplicación web o móvil que solicita permiso a un usuario para acceder a sus datos de acceso a un *provider.* El usuario puede autorizar o denegar el acceso a sus datos.

El *flujo* de autenticación se puede resumir como sigue:

- Obtenemos un *Request Token*
- Solicitamos la autorización del *Usuario* (para acceder a los datos de su cuenta) enviándole a una página especial de login del *Service Provider.*

- Cambiamos el *Request Token* por un *Access Token*
- Guardamos el *Access Token*

3.3. Otras librerías destacables

Para facilitar el desarrollo, hemos necesitado instalar algunas librerías útiles; a continuación las más destacables:

- *JDOM*: Hemos utilizado esta librería para el parse de documentos en formato .xml. Hemos necesitado este tipo de *parse* en la lectura del resultado del WS que devuelve un quiz del servidor, ya que nos devuelve el quiz en forma de un fichero xml .
- *Explorador*: Nos proporciona un explorador para dispositivos android, ya que la propia api no lo proporciona. Esta librería nos ha sido útil en el módulo de Assignment, ya que necesitamos recorrer los ficheros del dispositivo móvil para elegir el que se quiere elegir para la entrega de la tarea.

4. Análisis de requisitos

A continuación, describiremos los requisitos que debe cumplir nuestra aplicación. Se desglosan en requisitos funcionales (características, funcionamiento del sistema, etc...) y requisitos no funcionales (rendimiento, interfaz, etc...).

4.1. Requisitos funcionales

Además de lo ya desarrollado en la aplicación original, se deben poder ejecutar correctamente las siguientes funcionalidades:

- Listar *tareas* de un usuario
- Listar *cuestionarios* y *tareas* asociados a un curso
- Seleccionar un quiz y descargarlo
- Contestar quiz
- Grabar respuestas en el dispositivo móvil
- Seleccionar una tarea y consultar sus detalles
- Entregar un fichero en una tarea
- Recibir nota y feedback

4.2. Requisitos no funcionales

Estos requisitos no especifican funciones que debe cumplir nuestra aplicación, sino las cualidades genéricas que debe tener el sistema:

4.2.1. La interfaz

La interfaz de nuestra aplicación debe ser intuitiva y fácil de utilizar. Nuestros dos nuevos módulos deben mantener el mismo estilo Moodbile del cliente Android en el que nos basamos, tanto a nivel de fuentes, como a nivel de ilustraciones, organización visual, etc...

4.2.2. Estándares de diseño

El diseño de nuestra aplicación debe estar adaptado a dispositivos móviles, con imágenes de tamaño reducido, así como fuentes y botones grandes, para que sea utilizable.

Por otra parte, la aplicación debe seguir los estándares de diseño y usabilidad establecidos por Android. En nuestro caso, debemos asegurarnos de que el diseño es el mismo que se aplica a los antiguos módulos de Moodbile ya implementados, para que haya coherencia, ya que el objetivo es que el usuario no note diferencia ninguna entre los distintos módulos de la aplicación.

4.2.3. Estándares de uso

La interacción del usuario sobre la aplicación deberá responder a los estándares de uso en aplicaciones móviles, respondiendo a los toques, deslices y pinzamientos del usuario tal y como se espera en una aplicación de estas características.

4.2.4. Persistencia de datos

Se espera que los datos de la aplicación sean persistentes en la memoria del dispositivo móvil, de tal manera que el usuario pueda seguir usando la aplicación y

consultar sus datos, incluso cuando se encuentre sin conexión. Esta propiedad dota la aplicación de la capacidad de seguir funcionando durante los períodos de no conectividad.

Comentar que todo el mecanismo de la persistencia de datos se ha heredado del cliente Android ya existente, ya que éste cubría todas las características necesarias.

4.2.5. Coherencia entre los nuevos módulos y los ya desarrollados

Como ya se mencionó en los puntos anteriores, tiene que haber total coherencia entre todos los módulos de tal manera que el usuario no note en ningún momento que se hayan desarrollado en fases distintas del proyecto. Esto implica seguir exactamente los mismo patrones en el diseño, ya que el usuario sólo tiene acceso directo a la parte visual del proyecto y nos interesa que ésta fuera lo más compacta posible.

5. Especificación

Nuestro cliente se ha implementado hasta la hora para ser accedido exclusivamente por estudiantes. Por lo tanto, tenemos un único actor: usuario de tipo *Student* (alumno).

5.1. Diagrama de casos de uso

A continuación, definimos un diagrama de casos de uso que describe los múltiples escenarios que puede encontrar un actor dentro de la aplicación. No mostraremos *todos* los casos de uso de la aplicación, sino que nos limitaremos a describir los nuevos estados de uso (marcados en rojo), y aquellos que ya existían pero que tienen algún tipo de relevancia dentro de la nueva implementación.

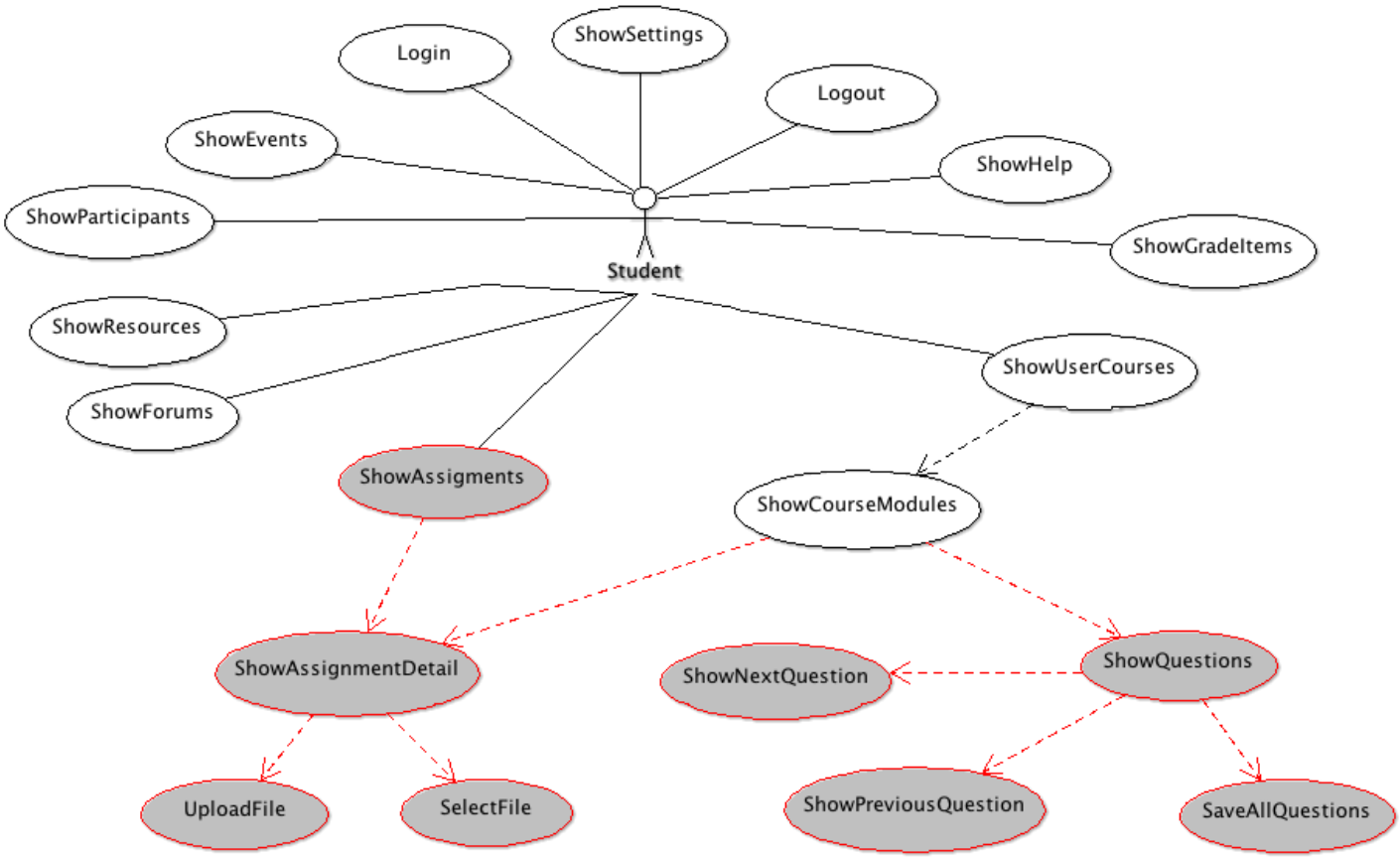


Figura 11 - Diagrama de casos de uso

5.2. Casos de uso

A continuación tenemos una descripción de algunos casos de usos de la aplicación. Algunos de estos casos de uso ya estaban implementados y no he necesitado cambiar nada, otros han necesitado alguna modificación y por último, algunos son totalmente nuevos.

Para cada *caso de uso*, daremos una breve descripción, el actor que en nuestro caso es siempre el usuario, las precondiciones y postcondiciones de la ejecución del caso de uso, los flujos normales, y los flujos alternativos.

Quiero resaltar que podría resultar interesante tener un WebService que permita obtener *todos los quizzes de un usuario* en concreto, y así poder incluir este módulo también dentro del listado del inicio y en el menú deslizante.

5.2.1. Case de uso : ShowCourseModules

Descripción: Se muestra el listado de módulos de un curso.

Actor: Alumno

Precondiciones: El usuario está dentro de la lista de cursos del usuario.

Postcondiciones: El usuario obtiene los módulos del curso seleccionado.

Flujo normal: 1. El usuario selecciona un curso del listado.

2. El usuario obtiene la lista de módulos del curso seleccionado.

Flujo alternativo: -

5.2.2. Caso de uso : ShowQuizQuestions

Descripción: Acceder al conjunto de preguntas de un quiz

Actor: Alumno

Precondiciones: El usuario está dentro de la lista de módulos de un curso.

Postcondiciones: El usuario obtiene las preguntas del quiz seleccionado.

Flujo normal: 1. El usuario selecciona un quiz del listado de módulos.

2. El usuario accede a la primera pregunta del conjunto de preguntas del quiz.

Flujo alternativo: -

5.2.3. Caso de uso : ShowNextQuestion

Descripción: Acceder a la siguiente pregunta del quiz

Actor: Alumno

Precondiciones: El usuario está visualizando una pregunta del quiz.

Postcondiciones: El usuario obtiene la siguiente pregunta.

Flujo normal: 1. El usuario hace clic sobre el botón “Next”

2. El sistema muestra la siguiente pregunta.

Flujo alternativo: 2B. El quiz no tiene más preguntas, se queda visualizando la pregunta actual.

5.2.4. Caso de uso : ShowPreviousQuestion

Descripción: Acceder a la pregunta anterior del quiz

Actor: Alumno

Precondiciones: El usuario está visualizando una pregunta del quiz.

Postcondiciones: El usuario obtiene la pregunta anterior.

Flujo normal: 1. El usuario hace clic sobre el botón “Previous”

2. El sistema muestra la pregunta anterior.

Flujo alternativo: 2B. El usuario está visualizando la primera pregunta del quiz, se queda visualizando la pregunta actual.

5.2.5. Caso de uso : SaveAllQuestions

Descripción: Guardar todas las respuestas introducidas del quiz

Actor: Alumno

Precondiciones: El usuario está visualizando una pregunta del quiz

Postcondiciones: Las respuestas previamente introducidas se salvan en la base de datos del dispositivo.

Flujo normal: 1. El usuario hace clic sobre el botón “Save”

2. El sistema salva todas las respuestas previamente introducidas.

Flujo alternativo: -

5.2.6. Caso de uso : ShowAssignments

Descripción: Se muestra el listado de Assignments de un usuario.

Actor: Alumno

Precondiciones: El usuario está autenticado.

Postcondiciones: El usuario obtiene los assignments de todos sus cursos.

Flujo normal: 1. El usuario selecciona consultar la lista de sus assignments.

2. El usuario obtiene la lista los assignments de todos sus cursos.

Flujo alternativo: -

*Sólo se han contemplado los assignment de tipo uploadsingle, queda pendiente implementar los demás tipos.

5.2.7. Caso de uso : ShowAssignmentDetail

Descripción: Consultar el detalle de un assignment

Actor: Alumno

Precondiciones: El usuario está dentro de la lista de assignments/ El usuario está dentro de la lista de módulos.

Postcondiciones: El usuario accede al detalle del assignment seleccionado

Flujo normal: 1. El usuario selecciona un assignment del listado.

2. El usuario accede al detalle del assignment seleccionado.

Flujo alternativo: -

5.2.8. Caso de uso : SelectFile

Descripción: Seleccionar un archivo para la tarea

Actor: Alumno

Precondiciones: El usuario está consultando un assignment single upload.

Postcondiciones: El usuario asigna un archivo del dispositivo a la tarea.

Flujo normal: 1. El usuario hace clic sobre el botón “Select file”

2. El usuario selecciona un fichero

3. El sistema asocia este fichero a la tarea

4. El sistema muestra este fichero en el detalle de la tarea.

Flujo alternativo: 2B. El usuario no selecciona ningún fichero.

3B. El sistema vuelve al detalle del assignment.

5.2.9. Caso de uso : UploadFile

Descripción: Entregar un archivo.

Actor: Alumno

Precondiciones: El usuario está en el detalle de una tarea single upload.

Postcondiciones: Se crea una nueva entrega (*submission*) de la tarea en el servidor o se modifica si ya existía.

Flujo normal: 1. El usuario hace clic cobre el botón “Upload file”

2. El sistema notifica al usuario que la entrega se ha hecho con éxito.

Flujo alternativo: 2B. El sistema indica al usuario que se encuentra sin conexión

2C. El sistema indica al usuario que este archivo está actualmente subido.

2D. El sistema indica al usuario que el tamaño del archivo supera el tamaño permitido.

6. Integración de los nuevos módulos dentro de la aplicación

En este capítulo, definiremos la arquitectura general del sistema, la comunicación con el servidor y la arquitectura propia del cliente móvil. Explicaremos cómo se ha integrado la arquitectura específica de los nuevos módulos dentro de la arquitectura global, y los pasos que hemos seguido para conseguirlo.

6.1. Arquitectura genérica del sistema

Para acceder y obtener a los contenidos de Moodle, nos hemos apoyado en el proyecto desarrollado y mantenido por el equipo de trabajo de Moodbile que consiste en *una capa de Interoperabilidad de Moodle.*

La capa de interoperabilidad hace de puente entre Moodle y las aplicaciones externas que quieran comunicar con él. Esta capa está dividida en tres capas a su vez:

- Capa del núcleo: Esta capa consiste en una API de *core functions* para el uso de las aplicaciones externas.
- Capa externa: Esta capa es la que contiene los servicios que Moodle ofrece al exterior.
- Capa de conectores: Contiene conectores que, bajo distintos protocolos, transforman las *external functions* en WebServices.

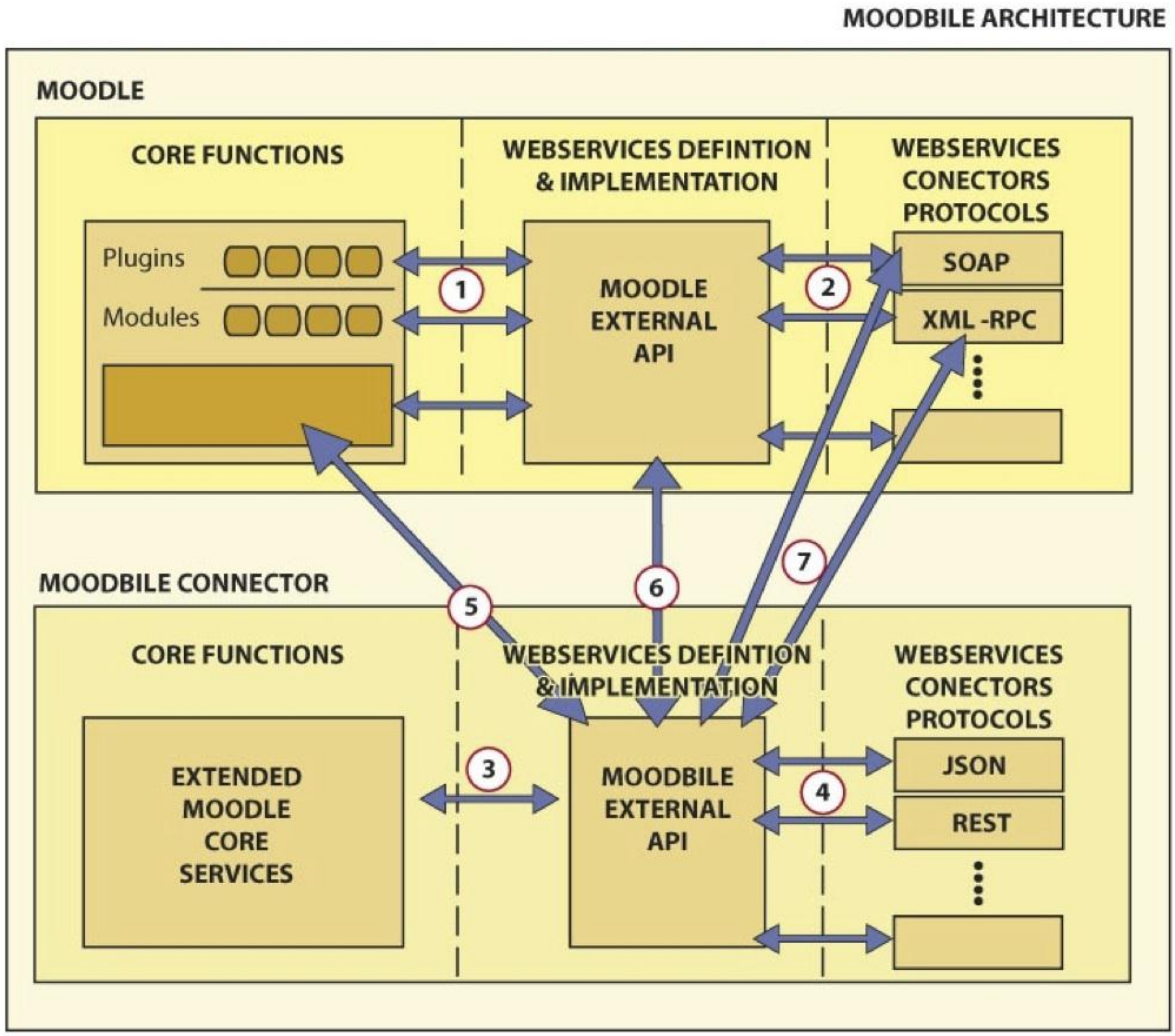


Figura 12 - Arquitectura del sistema de Moodbile

6.1.1. Diagrama de clases:

Dado que lo que pretendemos conseguir es una aplicación que contenga los mismos contenidos en Moodle, nos hemos basado en las estructuras de datos que nos devuelven la API de Moodbile, para construir nuestra estructura de clases.

A continuación, tenemos una parte del diagrama de clases del cliente. Las nuevas clases, correspondientes a las necesidades de los nuevos módulos, están coloreadas en gris.

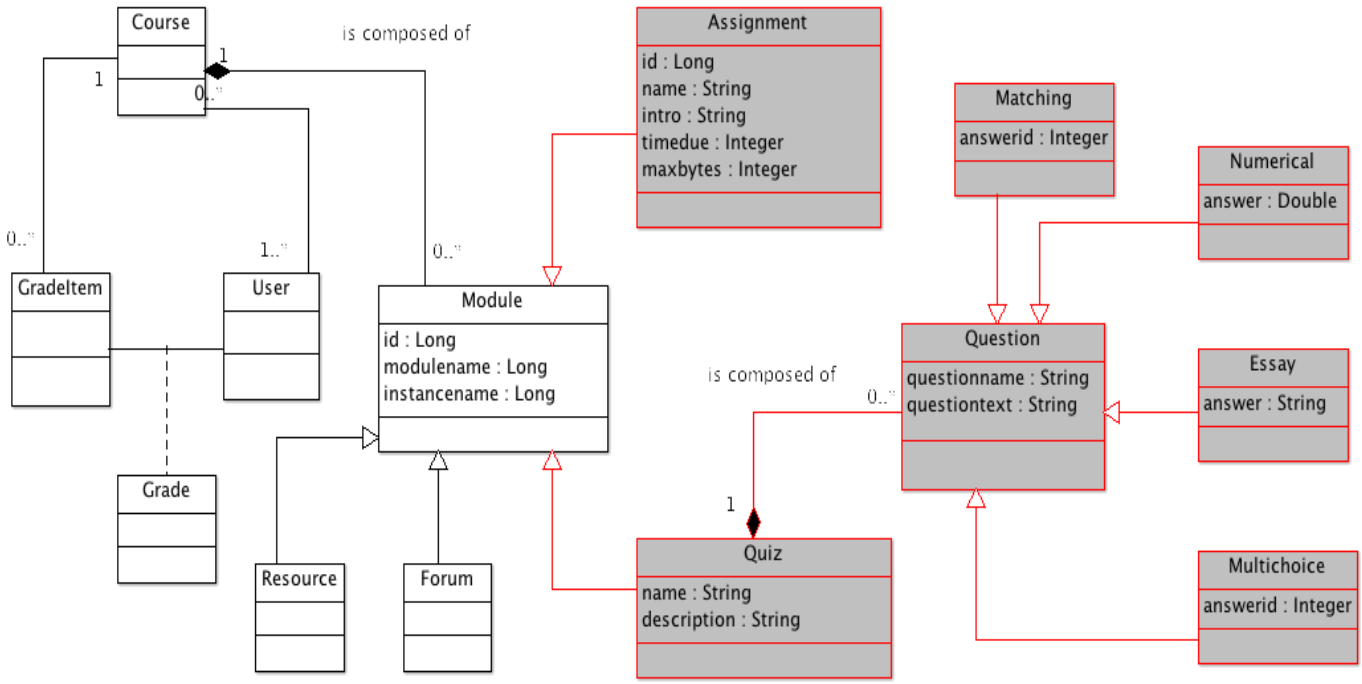


Figura 13 - Diagrama de clases

Nuestro objetivo al construir la estructura de datos de los nuevos módulos, ha sido crear coherencia entre los atributos de las clases y los campos de los objetos devueltos por los servicios web. La principal razón es facilitar la obtención y el tratamiento de datos por parte del cliente.

A continuación, tenemos el ejemplo del objeto que se devuelve al llamar el WebService *mbl_assign_get_assignment_by_id*, que consiste en devolver un *Assignment*, dado un id concreto. Podemos ver que los campos de este objeto coinciden con los atributos de la clase Assignment.

```
object {
    id int Required
    assignment record id

    name text Required
    multilang compatible name

    intro raw Required
    assignment description text

    assignmenttype alpha Required
    assignment type: upload, online, uploadsingle, offline

    maxbytes int Required
    maximium bytes per submission

    timedue int Required
    assignment due time

    grade int Required
    grade scale for assignment
}
```

En el módulo de los cuestionarios, la situación es relativamente distinta. El caso es que no hay un Webservice que devuelva un objeto Quiz; sino que disponemos de un Webservice que dado el id de un quiz, nos devuelve un conjunto de las preguntas que lo componen, en forma de fichero XML, que nos encargamos de parsear a posteriori. En consecuencia, hemos tenido que construir el objeto Quiz y sus atributos de manera más intuitiva, y basándonos sobretodo en los atributos del objeto módulo. Esto, de momento no importa al no existir una pantalla que muestre explícitamente el detalle de un quiz (nombre, descripción,…) como es el caso del assignment.

6.1.2. Llamadas a WebServices:

Para construir todas las llamadas al servidor de Moodle necesarias, nos hemos basado esencialmente en el código del cliente base, así cómo de la documentación de la Wiki de Moodbile, que describe con todo detalle, un procedimiento estandarizado para este propósito.

Sea cual sea el protocolo usado, hay que hacer una petición, indicando el método que se pretende llamar y los parámetros necesarios para este método, obteniéndose la respuesta como objeto o lista de objetos.

A continuación, mostramos un ejemplo sencillo de cómo se hace esta llamada en el caso del WebService ***mbl_quiz_export_quiz_to_xml:***

```
String responseBody = "";
JSONObject jsonParams = new JSONObject();
                        jsonParams.put("quizid",quizid);
                        responseBody = getServerResponse("mbl_quiz_export_quiz_to_xml", jsonParams);
```

Para la implementación de los módulos *Assignment* y *Quiz*, destacamos los siguientes Webservices (cada uno de ellos acompañado de una breve descripción).

Aunque algunos de ellos se estaban usando ya en la versión anterior, consideramos que es interesante volver a destacarlos ya que tienen una amplia importancia en nuestros nuevos módulos.

- <u>Assignment:</u>
  - mbl_assign_get_assignment_by_id : Obtiene un objeto Assignment a partir de su id.
  - mbl_assign_get_assignments_by_courseid : Obtiene todos los assignment de un curso.
  - mbl_assign_get_submission_files : Obtiene todos los fichero entregados en un assignment.
  - mbl_files_upload : Envía un fichero al servidor de moodle, y devuelve su id.
  - mbl_assign_submit_singleupload : Entrega un fichero (a través del id devuelto por el método anterior) en una tarea de tipo SingleUpload.

- mbl_grade_get_user_grade_by_itemid : Obtiene la nota y el feedback de un elemento graduable a partir de su id. Se puede aplicar tanto a Quiz como a Assignment.

- Quiz:

  - mbl_quiz_export_quiz_to_xml : Obtiene el id (dentro del servidor) de un fichero XML que contiene el conjunto de preguntas que componen el quiz indicado.
  - mbl_files_get_file_url : Usamos este *Webservice* para recuperar el fichero del servidor a partir del id que nos ha devuelto la llamada anterior.
  - mbl_course_get_course_modules : Este Webservice se usa mucho en el cliente, pero queríamos destacarlo, porque es el webservice que usamos para obtener los datos de un quiz como el nombre, la descripción,… ya que no hay uno específico como ya se mencionó anteriormente.

6.2. Arquitectura propia del cliente

En este capítulo, explicaremos la arquitectura específica de nuestro cliente. La elección de los elementos, patrones y relaciones que se van a explicar a continuación, ha sido decisión del compañero que desarrolló la versión inicial de la aplicación. Por lo tanto, nuestro trabajo se ha concentrado principalmente en estudiar estas decisiones, entender las motivaciones y aplicar lo mismo sobre los nuevos módulos con el fin de conseguir que la aplicación sea lo más homogénea posible. Destacar que la implementación es bastante flexible y permite añadir módulos y funcionalidades nuevas con muchísima facilidad.

Funcionalmente hablando, cuando se agrega un nuevo módulo a la aplicación Moodbile, lo que se pretende es, por una parte, conseguir visualizarlo en la lista

principal y en el menú deslizante, poder seleccionarlo y que se muestra un listado con los elementos de este módulo, y por otra, poder visualizarlo en la lista de módulos de un curso previamente seleccionado. El comportamiento concreto de un módulo es otro tema, ya que depende de la funcionalidad que tiene cada uno y no afecta tanto en la integración.

Con la implementación de la que disponíamos al iniciar nuestro proyecto, ya estaban integrados tanto el módulo de Quiz como el de Assignment a la lista de módulos de un curso, por lo que nos quedaba agregar el módulo *Assignment* a la lista principal y desarrollar el comportamiento concreto al seleccionar un elemento. En el caso de *Quiz*, no hemos añadido este elemento en la lista principal, al no haber un WebService específico para este propósito, como en el caso de Forums o Assignments.

A continuación, explicaremos cómo está construida la arquitectura de nuestro proyecto, capa a capa, centrándonos en los cambios que ha sido necesario introducir para poder integrar los nuevos módulos. Salvo pequeñas diferencias que se irán señalando, la integración de los dos módulos se ha hecho de manera similar.

De aquí en adelante, nos basaremos en el módulo de Assignment cómo ejemplo en los pasos de la integración.

6.2.1. Lógica:

El diseño de la arquitectura de nuestro cliente se basa principalmente en el patrón *Capas* para conseguir el menor acoplamiento posible entre distintas partes de la aplicación. Se ha establecido una estructura lógica en cuatro capas separadas según sus responsabilidades, como se puede observar en la siguiente figura:

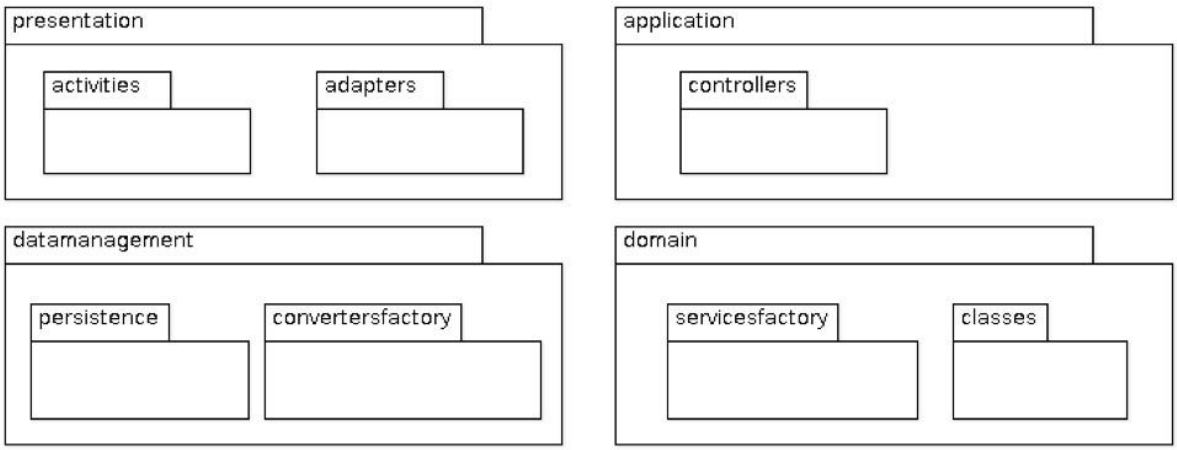


Figura 14 - Esquema de las cuatro capas de la aplicación

- *Presentación:* Incluye las distintas ventanas de la aplicación (*Activities*) y *adaptadores* que son los encargados de transformar la información de los objetos en información visual.

- *Aplicación:* La capa de aplicación gestiona las peticiones de la capa de presentación para que esta no se las haga directamente a la capa de dominio. Contiene un controlador por cada clase de dominio y es el encargado de gestionar las peticiones que se tienen que ejecutar sobre su respectiva clase. Cada vez que la interfaz reciba un evento que implique al dominio, una función de estas clases será la que se encargue de transferirlo.

- *Dominio:* Dentro de esta capa encontramos los servicios que se detallaran más adelantes, así como las clases de dominio que ya se han explicado en el diagrama de clases anterior.

- *Persistencia:* Esta capa incluye todo el paquete de persistencia (creación de la base de datos, etc..) y los conversores (DBConverters) que se encargan de

convertir los datos de los objetos del modelo a datos de base de datos y vice versa.

La arquitectura lógica no ha cambiado con respecto al cliente original. Nuestro esfuerzo se ha centrado más bien en introducir cambios y nuevos elementos en estas capas para conseguir el correcto funcionamiento de nuestros nuevos módulos sin estropear la estructura ni el bajo acoplamiento. Hablaremos de estos cambios con más detalle, cuando pasemos a detallar las capas una a una.

6.2.2. Presentación

En Android, la presentación se basa esencialmente en las *Activities* y los *layouts*. En este cliente usamos también *adapters*, que serán los encargados de transformar los objetos del modelo en datos visualmente entendibles y manejables.

Integración

Comentar que los elementos *assignment* ya aparecerían en la lista de módulos de un curso sin ningún tipo de desarrollo adicional, por lo que sólo nos quedaría asignarles el comportamiento correcto al seleccionar un elemento.

- En la clase *HomeAdapter.java,* agregamos el elemento Assignment a los demás módulos, para que sea visible en el listado principal y en el menú deslizante.

```
protected AssignmentsTabHelper assignmentsHelper;
…
/*
** Setup View Tab by position

private void setupTabViewByPos(final int position) {
final Resources resources = getApplicationContext().getResources();
        switch(position) {
                …
                case BaseTabHelper.ASSIGNMENTS_POS:
                        setupTabView(BaseTabHelper.ASSIGNMENTS_TAG, resources.getString(R.string.Assignments),
                                assignmentsHelper.setupAssignmentsTabView(BaseTabHelper.getLastUid()));
                        break;
                …
        }
}
```

- En la clase *BaseTabActivity.java*, agregamos el nuevo módulo en la lista principal, y le añadimos el comportamiento adecuado para que al seleccionarlo, se visualice la lista de Assignments.

```
protected AssignmentsTabHelper assignmentsHelper;
…
/*
** Setup View Tab by position
*/
private void setupTabViewByPos(final int position) {
final Resources resources = getApplicationContext().getResources();
        switch(position) {
                …
                case BaseTabHelper.ASSIGNMENTS_POS:
                        setupTabView(BaseTabHelper.ASSIGNMENTS_TAG, resources.getString(R.string.Assignments),
                                assignmentsHelper.setupAssignmentsTabView(BaseTabHelper.getLastUid()));
                        break;
                …
        }
```

- En la clase *BaseTabHelper.java*, especificamos comportamientos básicos como, cuando se hace clic sobre el botón *Back* del terminal, etc…

```
// Assignments attributes
public static final int ASSIGNMENTS_POS = 7;
public static final String ASSIGNMENTS_TAG = "assignments";
…
public static int getPreviousTabViewPos(int lastTabViewPos) {
        switch(lastTabViewPos) {
                …
                case ASSIGNMENTS_POS:
                        return HOME_POS;
                case ASSIGNMENT_UPLOADSINGLE_POS:
                        return isLastAssignments? ASSIGNMENTS_POS : MODULES_POS;
                …
        }
}
```

- Agregar la clase *AssignmentsTabHelper.java* que hereda de *BaseTabHelper.java*

En esta clase, llamaremos a nuestro controlador para obtener los datos que nos interesan; esto es, el listado de tareas del usuario conectado en este momento agrupados por curso.

```
public class AssignmentsTask extends AsyncTask<Pair<Long, Integer>, Void, Void> {
                @Override
                protected void onPreExecute() {…}
                @Override
                protected Void doInBackground(Pair<Long, Integer>... params) {
                        assignmentsAdap.addItems(assignmentsCtrl.getAssignmentsBy(
                                        params[0].first, params[0].second.intValue() != 0));
                        return null;
                }
                @Override
                protected void onPostExecute(Void unused) {…}
        }
```

- En esta misma clase, debemos especificar el comportamiento que queremos conseguir al seleccionar un elemento del listado de tareas. En nuestro caso, queremos abrir un nuevo *Tab* que muestre los detalles de este assignment.

```
private ExpandableListView.OnChildClickListener onAssignmentsChildClick = new ExpandableListView.OnChildClickListener(){
        public boolean onChildClick(ExpandableListView parent, View v,int groupPosition, int childPosition, long id) {
                BaseTabHelper.setLastAid(Long.valueOf(assignmentsAdap.getChildId(groupPosition, childPosition)));
                BaseTabHelper.setLastAssignments(true); //TODO use history
                ((BaseTabActivity) context).showTabView(ASSIGNMENT_UPLOADSINGLE_POS);
                return true;
        }
};
```

- Agregar la clase *AssignmentsAdapter.java* que en nuestro caso hereda de *BaseExpandableListAdapter.java*. Esta clase es la encargada de adaptar las listas de objetos Assignment de manera que se puedan manejar por los layouts correspondientes.

- Construir los layouts correspondientes al detalle, y a la lista de assignments, y agregar la nueva vista de la lista de assignment (*listview*) a los demás tabs de módulos en el fichero *tabwidget.xml.*

- Añadir los valores string que hagan falta y las ilustraciones correspondientes.

6.2.3. Controllers

Como ya se ha comentado anteriormente, estamos ante una arquitectura basada en el patrón Modelo-Vista-Controlador.

El modelo son los objetos que representan cierta información sobre el dominio, la vista representa la visión del modelo en la interfaz de usuario y el controlador es el encargado de cambiar esta información manipulando el modelo y provocando que la vista se actualice.

Integración:

Agregamos una nueva clase *AssignmentsController.java,* para hacer de mediador entre los modelos de datos y las clases de presentación.

Esta clase debe contener, entre otros elementos, un método que nos permita obtener la lista de todos los assignments de un usuario a partir de su *idUser.* Esta lista es la que se visualiza en la pantalla principal del cliente.

```
public HashMap<String,ArrayList<IConvertible>> getAssignmentsBy(Long uid, boolean isUpdate) {
        HashMap<String,ArrayList<IConvertible>> assignments = new HashMap<String,ArrayList<IConvertible>>();
        HashMap<Long,String> coursesShortnames = getCoursesShortnamesBy(uid);
        if (coursesShortnames != null) {
                for (Map.Entry<Long,String> entry : coursesShortnames.entrySet()) {
                        ArrayList<IConvertible> courseAssignments = new ArrayList<IConvertible>();
                        Collection<IConvertible> c
                                = assignmentsService.getAssignmentsBy(entry.getKey(), isUpdate).values();
                        Iterator<IConvertible> itr = c.iterator();
                        while (itr.hasNext()) {
                                courseAssignments.add(itr.next());
                        }
                                assignments.put(entry.getValue(), courseAssignments);
                }
        }
        return assignments;
}
```

6.2.4. Servicios

En la capa de servicio, nos apoyamos en una factoría de servicios (*Services Factory*) para la generación de los objetos *Service.* Cada servicio tiene una implementación *Offline* y otra *Online,* para obtener los datos, de la caché, de la base de datos del dispositivo móvil ó bien del servidor, según la conexión, y la validez de los datos de los que disponemos en el momento de la llamada.

Mediante el uso de la factoría de servicios, patrón *Factoría*, y el uso de los adaptadores, patrón *Adaptador*, conseguimos que se puedan cambiar en el futuro los servicios sin que afecte al resto, lo que se conoce como patrón *Variaciones Protegidas*.

Integración:

- Agregamos el nuevo servicio en la clase *ServicesFactory.java*. Como norma general, cuando una clase cualquiera necesita acceder a algún servicio, primero se accede al servicio *Offline,* y es éste el encargado de decidir de qué manera se obtendrán los datos.

```
public class ServicesFactory {
        private IAssignmentsService assignmentsService;
        …
        public IAssignmentsService getAssignmentsService() {
                if (assignmentsService == null) {
                        AssignmentsOfflineService localService
                                = new AssignmentsOfflineService(new AssignmentsOnlineService());
                        assignmentsService = (IAssignmentsService) localService;
                }
                return assignmentsService;
        }
        …
}
```

Creamos una *interface* *IAssignmentsService.java*, implementada por dos clases service *AssignmentsServiceOnline.java* y *AssignmentsServiceOffline.java*. Como se ha comentado anteriormente, si tenemos conexión y no estamos en modo *Roaming* siempre se intenta obtener los datos a través del servicio Online, para tenerlos lo más actualizados posible. En este caso, accedemos al servidor mediante llamadas Webservice, recuperamos los contenidos que nos interesan, y acto seguido los grabamos en la base de datos del móvil para tenerlos disponibles.

En caso contrario, si no tenemos conexión de datos, obtendremos los datos de la caché (si hace poco que se ha perdido la conexión), o bien de la base de datos.

A continuación vemos como está implementado este proceso en las clases *Service*:

```
public interface IAssignmentsService {
        public IConvertible getAssignment(Long aid);
        public HashMap<Long,IConvertible> getAssignmentsBy(Long cid, boolean isUpdate);
        public Boolean uploadAssignmentFile(Long cid, Long aid, String filePath);
}
```

```
public class AssignmentsOnlineService extends JsonRpcOnlineService implements IAssignmentsService{

	public IConvertible getAssignment(Long aid) {
		Assignment assignment = null;
		String responseBody = "";
		…
		JSONObject jsonParams = new JSONObject();
		jsonParams.put("assigid", aid.longValue());
		responseBody = getServerResponse("mbl_assign_get_assignment_by_id", jsonParams);
		…
		JSONObject post = new JSONObject(responseBody);
		…
		JSONObject entry = post.getJSONObject("result");
		assignment = new Assignment();
		assignment.setTimecached(AppUtils.getInstance().getDateTime());
		assignment.setOid(new Long(entry.getLong("id"))); //TODO
		assignment.setId(new Long(entry.getLong("id")));
		assignment.setName(entry.getString("name"));
		…
		return assignment;
	}
}
```

```
public class AssignmentsOfflineService extends AbstractOfflineService implements IAssignmentsService, Runnable{
…
	public IConvertible getAssignment(Long aid) {
		IConvertible assignment;
		assignment = PersistenceFacade.getInstance().get(aid, Assignment.class);
		if (assignment == null|| isOnline() && !isCacheHit(assignment)) {
			IConvertible onlineAssignment = assignmentsOnlineService.getAssignment(aid);
			if (onlineAssignment != null) {
				PersistenceFacade.getInstance().put(onlineAssignment);
				assignment = PersistenceFacade.getInstance().get(aid, Assignment.class);
			}
		}
		return assignment;
	}
…
}
```

El caso del módulo de Quiz tiene unas pocas diferencias con el resto de los módulos. Sólo requerimos una única conexión al servidor para obtener los datos de un Quiz. Una vez recuperados los contenidos, se graban en la base de datos. Y a partir de este momento, sólo se obtendrán de ahí independientemente de la conexión de la que disponemos. Funcionalmente, se traduce en qué un quiz si no existe en el móvil, se obtiene del servidor, pero de existir, no se vuelve a actualizar. En el código se refleja de la siguiente forma:

```
public HashMap<Long, IConvertible> getQuestionsBy(Long qid, boolean isUpdate) {
        HashMap<Long, IConvertible> questions = new HashMap<Long, IConvertible>();
        questions = PersistenceFacade.getInstance().getBy(qid, Question.class);
        if (questions.isEmpty()) {
                HashMap<Long, IConvertible> onlineQuestions =
questionsOnlineService.getQuestionsBy(qid, isUpdate);                    if
(!onlineQuestions.isEmpty()) {
                        questions.putAll(onlineQuestions);
                        PersistenceFacade.getInstance().putAll(onlineQuestions, Question.class);
                        …
                        questions = PersistenceFacade.getInstance().getBy(qid, Question.class);
                }
                return questions;
        }
}
```

6.2.5. Persistencia

Para conseguir la persistencia de datos en nuestro cliente, nos hemos basado en una base de datos SQLite. SQLite es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y por supuesto ser de código libre. Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas una completa API para llevar a cabo de manera sencilla todas las tareas necesarias.

Al ser el paquete de persistencia ya construido, hemos concentrado nuestros esfuerzos en estudiarla al detalle y definir los elementos donde se tienen que introducir cambios para agregar nuestros nuevos módulos. Cabe destacar en este sentido, que nuestra base de datos se apoya en una caché en memoria por cuestiones de eficiencia. Así se evita materializar el objeto de la base de datos lo que es importante al ser una operación lenta. Entendemos por materialización convertir registros de la base de datos a objetos y por desmaterialización el proceso contrario.

Para establecer la correspondencia entre las clases del dominio y los registros se utilizan unas clases *Converter*, una por cada tipo de clase del dominio, que heredan el

comportamiento común de una clase abstracta *DBConverter* del paquete de persistencia. Los objetos persistentes implementan la interfaz *IConvertible* a su vez. El paquete de persistencia cuenta además con una *Facade* que hace de interfaz uniforme escogiendo el *Converter* apropiado en función de la clase.

Para las operaciones sobre la base de datos, se presentaban las funciones get() y put() respectivamente a las que se han agregado getBy(), getAll(), putBy() y putAll() para cubrir el caso de las colecciones. Las variantes de put() reciben un objeto o una colección de ellos como parámetro. Se han creado también funciones insert(), update(), delete() y reload() para insertar, actualizar, borrar o volver a cargar un objeto respectivamente.

Integración:

A continuación, detallamos los pasos necesarios para introducir un nuevo módulo a la capa de persistencia:

- Agregar las tablas correspondientes al nuevo módulo así cómo una tabla que represente la relación con el elemento curso. Esto se refleja en la clase *DBH.java* que hereda de *SQLiteOpenHelper.java*:

```
protected static final String ASSIGNMENT_TABLE = "assignment";
protected static final String ASSIGNMENT_TIMECACHED = "timecached";
protected static final String ASSIGNMENT_OID = "oid";
protected static final String ASSIGNMENT_ID = "id";
protected static final String ASSIGNMENT_NAME = "name";
protected static final String ASSIGNMENT_INTRO = "intro";
protected static final String ASSIGNMENT_FILE = "file";
protected static final String ASSIGNMENT_TYPE = "type";
protected static final String ASSIGNMENT_MAXBYTES = "maxbytes";
protected static final String ASSIGNMENT_TIMEDUE = "timedue";
protected static final String ASSIGNMENT_GRADE = "grade";
protected static final String ASSIGNMENT_COURSEID = "courseid";
protected static final String ASSIGNMENT_CREATE = "create table "
                + ASSIGNMENT_TABLE                      + " ("
                + ASSIGNMENT_TIMECACHED         + " integer not null, "
                + ASSIGNMENT_OID                        + " integer primary key, "
                + ASSIGNMENT_ID                         + " integer unique, "
                + ASSIGNMENT_NAME                       + " text not null, "
                + ASSIGNMENT_INTRO                      + " text, "
                + ASSIGNMENT_FILE                       + " text, "
                + ASSIGNMENT_TYPE                       + " text, "
                + ASSIGNMENT_MAXBYTES           + " integer, "
                + ASSIGNMENT_TIMEDUE            + " integer, "
                + ASSIGNMENT_GRADE                      + " integer, "
                + ASSIGNMENT_COURSEID           + " integer not null"
                + ");";
protected static final String COURSEASSIGNMENT_TABLE = "courseassignment";
protected static final String COURSEASSIGNMENT_CID = "cid";
protected static final String COURSEASSIGNMENT_AID = "aid";
protected static final String COURSEASSIGNMENT_CREATE = "create table "
                + COURSEASSIGNMENT_TABLE         + " ("
                + COURSEASSIGNMENT_CID           + " integer, "
                + COURSEASSIGNMENT_AID           + " integer, "
                + "primary key(" + COURSEASSIGNMENT_CID + ", " + COURSEASSIGNMENT_AID + ")"
                + ");";
…
database.execSQL(ASSIGNMENT_CREATE);
database.execSQL(COURSEASSIGNMENT_CREATE);
```

- Añadir el nuevo módulo, junto con la relación con el elemento curso, a la clase *CacheMemory.java* para que se tenga en cuenta a la hora de guardar objetos Assignment en la memoria caché.

```
private HashMap<Long,IConvertible> assignments = new HashMap<Long,IConvertible>();
private HashMap<Long,ArrayList<Long>> coursesAssignments = new HashMap<Long,ArrayList<Long>>();
…

public IConvertible getAssignment(Long aid) {
        return assignments.get(aid);
}

public HashMap<Long , IConvertible> getAssignments() {
        return assignments;
}

public HashMap<Long,IConvertible> getAssignmentsBy(Long cid) {
        HashMap<Long,IConvertible> assignmentsBy = new HashMap<Long,IConvertible>();
        ArrayList<Long> aids = coursesAssignments.get(cid);
        if (aids != null) {
                for (Long aid : aids) {
                        assignmentsBy.put(aid, assignments.get(aid));
                }
        }
        return assignmentsBy;
}

public void putAssignment(IConvertible assignment) {
        this.assignments.put(assignment.getOid(), assignment);
        setChanged();
        notifyObservers();
}

public void putAssignments(HashMap<Long,IConvertible> assignments) {
        this.assignments.putAll(assignments);
        setChanged();
        notifyObservers();
}

public void putAssignmentsBy(Long cid, HashMap<Long,IConvertible> assignments) {
        this.assignments.putAll(assignments);
        ArrayList<Long> aids = new ArrayList<Long>();
        for (Map.Entry<Long,IConvertible> entry : assignments.entrySet()) {
                aids.add(entry.getKey());
        }
        this.coursesAssignments.put(cid, aids);
        setChanged();
        notifyObservers(ASSIGNMENTS_TAG);
}

public void deleteAssignment(IConvertible ic) {
        this.assignments.remove(ic.getOid());
        Assignment a = (Assignment) ic;
        Long cid = a.getCourseid();
        ArrayList<Long> aids = this.coursesAssignments.get(cid);
        aids.remove(ic.getOid());
        setChanged();
        notifyObservers();
}

public void clearAssignments() {
        assignments.clear();
}

public void clearCoursesAssignments() {
        coursesAssignments.clear();
}
```

- Crear la clase *DBconverter* pertinente, en este caso *AssignmentDBConverter.java*. Esta clase es la responsable de mapear la relación entre las clases de dominio y las tablas de bases de datos, o la memoria caché en su caso. Hereda de la clase *AbstractDBConverter.java,* e implementa sus métodos.

```
public class AssignmentDBConverter extends AbstractDBConverter{
…
        protected String getTable() {
                return DBH.ASSIGNMENT_TABLE;
        }

        …

        protected IConvertible getRecordFromStorage(Cursor cur) {
                Assignment assignment = new Assignment();
                assignment.setState(OldCleanState.getInstance());
                assignment.setTimecached(cur.getLong(cur.getColumnIndex(DBH.ASSIGNMENT_TIMECACHED)));
                assignment.setOid(new Long(cur.getLong(cur.getColumnIndex(DBH.ASSIGNMENT_OID))));
                assignment.setId(new Long(cur.getLong(cur.getColumnIndex(DBH.ASSIGNMENT_ID))));
                assignment.setName(cur.getString(cur.getColumnIndex(DBH.ASSIGNMENT_NAME)));
                assignment.setIntro(Html.fromHtml(cur.getString(cur.getColumnIndex(DBH.ASSIGNMENT_INTRO))));
                …
                return assignment;
        }
        protected final ContentValues createTableCV(IConvertible ic) {
                ContentValues values = new ContentValues();
                Assignment assignment = (Assignment) ic;
                values.put(DBH.ASSIGNMENT_TIMECACHED, assignment.getTimecached());
                values.put(DBH.ASSIGNMENT_OID, assignment.getId());
                values.put(DBH.ASSIGNMENT_ID, assignment.getId());
                values.put(DBH.ASSIGNMENT_NAME, assignment.getName());
                values.put(DBH.ASSIGNMENT_INTRO, Html.toHtml(assignment.getIntro()));
                …
                return values;
        }
        …
}
```

Todos los *DBConverter* tienen prácticamente los mismos métodos de acceso a base de datos, pero con implementaciones distintas según el tipo que reciban. Como ya se comentó anteriormente, todas las llamadas de acceso a la base de datos ya están implementadas en la clase *AbstractDBConverter.java*. Éstas llamadas son estándar, y se pueden customizar en el caso de necesitar más funcionalidades de las ya existentes.

- Añadir el nuevo *DBConverter* a la clase *ConverterFactory.java*, que es una especie de generador de converters. A esta factoría se le llama cuando hace falta acceder a alguna tabla de la base de datos a través de un *DBConverter*.

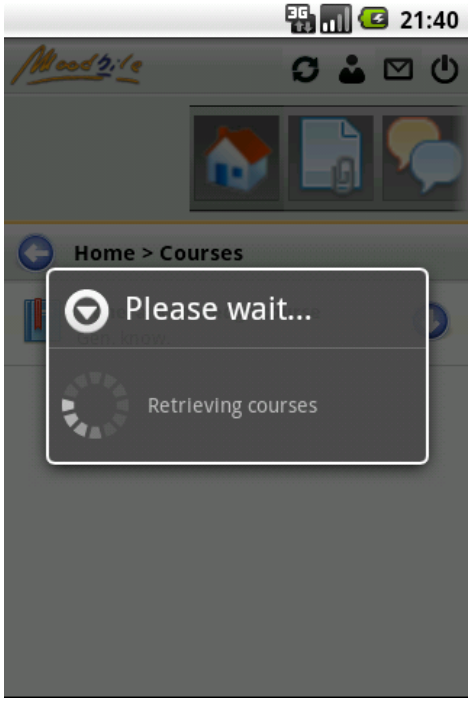
```
converters.put(Assignment.class, new AssignmentDBConverter());
```

6.2.6. Operaciones en segundo plano:

A continuación, explicaremos la manera de la cual se han tratado las operaciones en segundo plano o *background* en nuestro cliente.

Android está hecho para teléfonos donde es vital que no se bloquee el terminal con operaciones costosas. Por este motivo, a los pocos segundos de iniciarse un acceso a la base de datos o a los servicios web, si no se realizan en un hilo a parte, Android cierra la aplicación mostrando el mensaje de que la aplicación no responde, porque son operaciones que pueden llegar a tardar en llevarse a cabo.

La idea principal, por lo tanto, es mostrar un *Dialog* que informe al usuario que la tarea se está ejecutando, y conseguir de esta manera que la aplicación no se bloquee.



Para ello, se ha extendido la clase AsyncTask que nos gestiona todo el proceso en un hilo a parte mediante tareas asíncronas, realizando la acción y publicando los resultados posteriormente en el hilo principal.

6.2.7. Seguridad

Nuestro cliente, como ya se ha explicado anteriormente, tiene dos sistemas de validación; uno basado en usuario y contraseña, y otro basado en OAuth en su versión 1.0a. Esto se debe a decisiones que se tomaron durante el desarrollo de la versión anterior de este proyecto, y que llevaron, después de estudiar una serie de factores sobre la autenticación vía usuario-contraseña, a considerar el modo de validación Oauth más seguro.

Oauth es un protocolo abierto que permite al cliente acceder de un modo seguro, previa autorización, a datos del usuario albergados en el servidor Moodle

mediante una API que soporta Oauth y que pone a Moodle, a través de los servicios web comentados, a disposición de nuestro cliente.

La ventaja de este sistema es que el cliente conecta con Moodle a través de un navegador web, el usuario introduce sus credenciales en una ventana de Moodle y acepta o no conceder autorización al cliente a acceder a sus datos necesarios para ser mostrados.

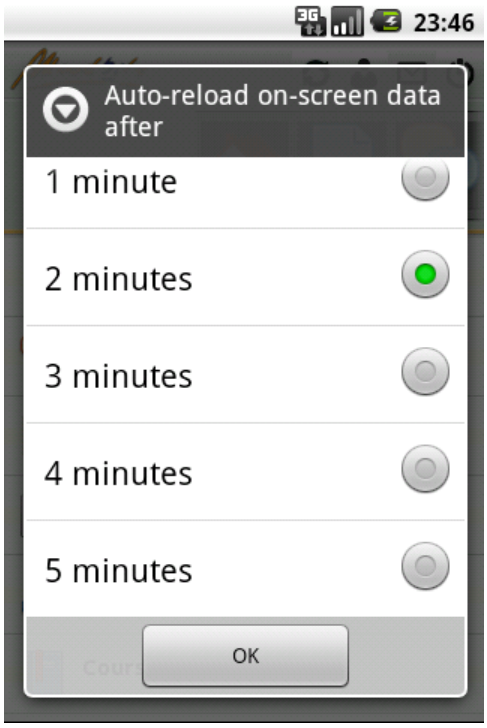
6.2.8. Sincronización y modos de conexión

Existen tres estados posibles del cliente en cuanto a conectividad: sin conexión, Wifi y 3G. Para el modo sin conexión, nos apoyamos en la caché, el modo wifi es ideal para actualizar contenidos y el modo 3G es un modo que debe permitir al usuario escoger debido al gasto económico que puede provocar.

A continuación, explicaré brevemente cómo funcionan el botón *actualizar* y la sincronización automática en la aplicación. Cabe destacar que todas las decisiones relativas a la gestión de los modos de conexión son decisiones que ya estaban tomadas por lo que nos hemos limitado a aplicarlas a nuestro módulos dónde oportuno.

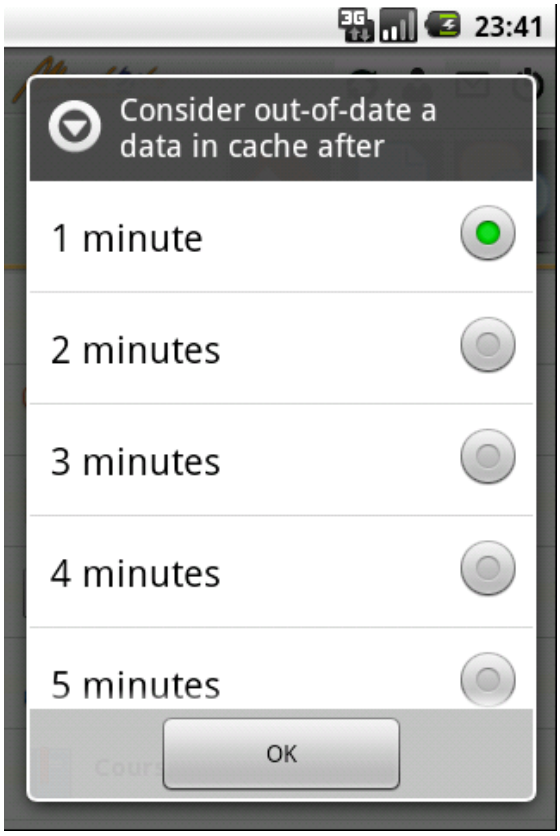
El cliente monitorea los cambios de la red mediante la clase BroadcastReceiver, que permite saber en todo momento el estado de la red. En función de este estado habilitamos/deshabilitamos el botón de *actualizar* del menú de opciones. De existir conexión, tanto si es wifi como 3G, el botón estará habilitado y permitirá actualizar los contenidos mostrados a través de los servicios web. Al ser una petición expresa del usuario dejamos bajo su responsabilidad el uso del 3G.

Por otra parte, la aplicación permite habilitar un refresco automático de los contenidos en pantalla cada cierto periodo de tiempo configurable.



Se puede desactivar el refresco automático de los contenidos mostrados con cero minutos. En caso contrario se ejecutará un intento de actualizaciones cada vez que pase el tiempo indicado. Si está desactivado o nos encontramos usando 3G y la opción de 3G está desactivada usaremos los datos en caché.

Por otro lado, el uso de la caché aún es posible si se da un acierto de caché lo cual sucede cuando los datos han sido introducidos recientemente. Esto es importante para aprovechar el uso de la caché para reducir el número de llamadas a los servicios web y el de accesos a la base de datos, ambos muy costosos. El hecho de que se considere un acierto de caché depende del tiempo indicado en la opción de menú y la marca de tiempo de entrada a caché del objeto concreto consultado.



Resumiendo, el botón actualizar es independiente de estas últimas opciones y permitirá actualizar los contenidos si alguna red está conectada.

6.3. Patrones

En este apartado, haremos una recapitulación de los patrones de diseño que se han escogido para la implementación del cliente. No hemos innovado en nada, ya que todos estos patrones ya estaban en la aplicación original, por lo que hemos vuelto a aplicar los mismo sobre el desarrollo de los nuevos módulos.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

En este cliente, se han aplicado cuatro patrones de diseño, de los cuales algunos (como MVC), se han explicado anteriormente. A continuación, daremos la definición de cada uno de estos patrones, sin entrar por tanto en por qué se han elegido estos y no otros, ya que esta decisión no ha sido nuestra.

6.3.1. GRASP

Grasp, General Responsibility Assignment Software Patterns, son patrones generales de software para la asignación de responsabilidades al diseñar objetos, como por ejemplo la responsabilidad de crear otro objeto.

- Experto en información:

Indica que la responsabilidad de crear un objeto o implementar un método se debe asignar a la clase que tiene toda la información para hacerlo.

- Creador:

Identifica el responsable de la creación e instanciación de nuevos objetos o clases.

- Controlador:

Es un patrón que sirve de intermediario entre una interfaz y su implementación. El objetivo principal es que la lógica de negocios esté separada de la capa de presentación para aumentar la reusabilidad del código y tener un mayor control.

- Alta cohesión:

La información contenida en una clase así como las responsabilidades que se le asignan deben ser coherentes.

- Bajo acoplamiento:

Las clases deben ser lo menos ligadas entre sí que se pueda. De tal manera que si se modifica alguna de ellas, se tenga la mínima repercusión posible en el resto de clases.

- Polimorfismo:

Hacer uso del polimorfismo. Cuando los comportamientos según el tipo de clase, se debe asignar la responsabilidad para este comportamiento a la misma clase utilizando operaciones polimórficas.

- Fabricación pura:

Es el principio de crear una clase “inventada” que no forma parte del dominio, para conseguir una mejora estructural del sistema.

- Variaciones protegidas:

Protegerse del cambio creando interfaces estables alrededor de puntos propensos a cambios o inestables. De esta forma, cuando se produzca la variación, repercuta lo mínimo sobre el software.

6.3.2. GOF

Gof, Gang Of Four es una referencia a los autores del libro *Design Patterns*, que recoge patrones útiles y prácticamente necesarios para el diseño correcto de objetos.

Los patrones que se han aplicado en la aplicación son los siguientes:

- Adaptador:

Se encarga de convertir la interfaz original de una clase en otra interfaz que el cliente espera.

- Factoría:

Este objeto maneja la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la diversidad de casos particulares que se pueden prever, para elegir el subtipo que crear.

- Singleton:

El objetivo es garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso único a ella. Se devuelve la propia clase mediante un método estático y se crea solo si no existe ya.

- Estrategia:

Determina cómo se debe realizar la comunicación entre diferentes objetos para llevar a cabo una tarea. Permite mantener un conjunto de algoritmos en una interfaz común

pero con implementaciones diferentes; y cada objeto cliente puede elegir aquel que le conviene.

- Fachada:

Define un punto un de acceso único a una interfaz o grupo de interfaces de un subsistema.

- Observador:

Define una dependencia entre objetos, de tal manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos sus suscriptores.

- Proxy:

El objetivo es proporcionar un intermediario de un objeto para controlar su acceso.

- Factoría abstracta:

Cuando se quiere crear familias de clases que implementen una interfaz común, se hace que extiendan de una clase abstracta que implemente una interfaz factoría abstracta.

6.3.3. Capas

El diseño por capas es una arquitectura en la que el objetivo principal es la separación de la lógica de negocios de la lógica de diseño. Por lo tanto, y como se ha explicado anteriormente, se organiza la estructura lógica del sistema en varias capas con responsabilidades distintas de forma que las capas superiores colaboren con las inferiores pero no al revés (sino a través del observador).

6.3.4. Modelo-Vista-Controlador

Es un patrón para separar los datos de una aplicación, la interfaz de usuario y la lógica de negocio.

En la siguiente figura, las líneas sólidas indican una asociación directa, y las punteadas una indirecta.

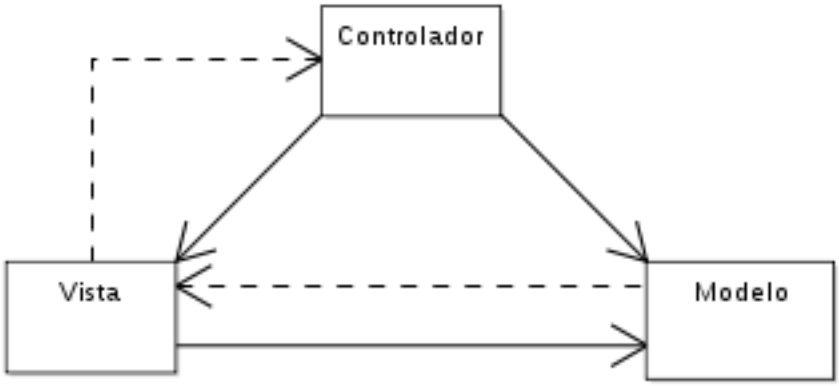


Figura 15 - Esquema del patrón Modelo Vista Controlador

- Modelo: Objeto que representa información sobre los objetos del dominio.
- Vista: La interfaz de usuario.
- Controlador: Responde a eventos e invoca peticiones al modelo y a la vista.

El objetivo de este patrón es evitar que el modelo envíe mensajes a la vistas de manera directa. Para evitarlo, se usa el patrón *Observador.*

7. Planificación y coste

En este capítulo, hablaremos sobre la planificación temporal y el coste material de nuestro proyecto. Comentaremos las distintas iteraciones que se han llevado a cabo en el desarrollo, definiendo los objetivos iniciales de cada una, y los objetivos conseguidos.

7.1. Visión general:

7.1.1. Primera iteración

Durante esta etapa del proyecto, que duró unos seis meses aproximadamente, todos mis esfuerzos se centraron en el aprendizaje y la familiarización con el desarrollo en Android, con Moodle como plataforma, con la comunicación vía Servicios web, y todos los conceptos en general que me harían falta para llevar a cabo mi proyecto.

Empecé desarrollando un pequeño prototipo de aplicación Android, que me ha servido para tener los conceptos más claros, y estar lista para aplicarlo a un nivel más avanzado.

A continuación, dediqué unas cuantas semanas a estudiar detalladamente el cliente Android ya existente, tanto a nivel funcional como a nivel de código e implementación. Instalé la aplicación y me centré en entender los pasos y patrones a seguir en cada acción para conseguir lo que se tenía visualmente, con especial enfoque en las acciones que me interesaban en mi proyecto, cómo el listado de *Modules,* por ejemplo. Posteriormente, empecé a incluir algunas pocas funcionalidades del módulo de cuestionarios, empezando el diseño del modelo de datos, que continuaría durante la segunda iteración del proyecto.

Esta parte del proyecto, fue la más larga temporalmente ya que nunca había trabajado con Android, por lo que he necesitado empezar completamente de cero. Por otra parte, tenía que trabajar con un código y una estructura ya construidos, con las dificultades que esto implica. Un ejemplo de las dificultades, es la capacidad de reacción ante los errores que me pude encontrar durante toda la fase de desarrollo. Al no haber desarrollado yo misma todo el código, cada error era un mundo, y a veces era necesario investigar mucho para encontrar el fallo.

Por otra parte, cabe destacar que el código del compañero es altamente entendible, ya que aplica muchos patrones, explicados anteriormente, que lo hacen además muy fácil de reutilizar y manejar.

Durante esta misma iteración, el equipo de *Moodbile* dedicó muchas horas al desarrollo de los nuevos *WebServices* que me harían falta en etapas futuras, para acceder al servidor de Moodle.

7.1.2. Segunda iteración

Durante esta segunda y última iteración, empecé con el desarrollo "*real*" de los nuevos módulos. Lo primero que hice fue conseguir la comunicación con el servidor, y comprobar su correcto funcionamiento vía los servicios web que ya existían. A continuación, creé el modelo de datos definitivo de los módulos, y continué el proceso de creación de funcionalidades, a medida que se adaptaba el contenido obtenido de los servicios web, a las pantallas ya diseñadas de nuestra aplicación.

La dificultad de la implementación del módulo de cuestionarios residía en el hecho de que era muy distinto de los módulos ya desarrollados (módulo de foros por ejemplo), por lo que había que desarrollar sobre nuevas bases, sin perder por tanto la coherencia con el código ya existente. Por otra parte, se necesitaba mucha coordinación con el equipo de Moodbile para probar y depurar los errores de los nuevos

Webservices. Esta coordinación se nos hizo difícil por momentos, al trabajar ellos durante el día, mientras yo le dedicaba al proyecto, principalmente las noches y los fines de semana, y solo alguna tarde puntual.

Esta etapa fue de más logros a nivel funcional, ya que es dónde se empezaron a ver resultados materiales, pero también es cuando empezamos a encontrarnos con algún que otro impedimento.

A este nivel del proyecto fue cuando nos percatamos de que iba a ser prácticamente imposible implementar la parte de *Upload* de los cuestionarios al servidor. La razón es bien sencilla, y es que en un inicio, la idea era hacer un upload del cuestionario al servidor, y que sea éste el encargado de evaluar las preguntas y devolver los resultados. Sin embargo, los miembros del equipo Moodbile, después de estudiarlo, llegaron a la conclusión de que la complejidad de esta tarea era muy alta, y era más razonable hacer estas evaluaciones en nuestro cliente. Finalmente, vista la carga de trabajo que implicaba esta elección, decidimos de común acuerdo, dejar esta tarea para un proyecto futuro, y centrar el esfuerzo en la implementación de otro modulo más, el de *Assignment,* integrar ambos módulos en el cliente original, y explicar los pasos de la integración detalladamente, para futuras ampliaciones.

Por ultimo, dediqué las últimas semanas a testear mi aplicación completamente y a redactar la memoria del proyecto en paralelo, lo que me permitiría alcanzar el punto final de este proyecto.

7.2. Planificación temporal

La planificación en todo proyecto software es muy importante para poder alcanzar los objetivos. Nuestro proyecto sufrió una serie de retrasos a lo largo de su desarrollo, como es habitual en prácticamente todos los proyectos. Sin embargo, con empeño, hemos sido capaces de superar estos retrasos y poder llevar a cabo una gran mayoría de los objetivos que se propusieron inicialmente en el tiempo establecido.

A continuación, mostramos un diagrama de Gantt que detalla la planificación temporal del proyecto, definiendo todas las tareas que se llevaron a cabo así cómo su duración.

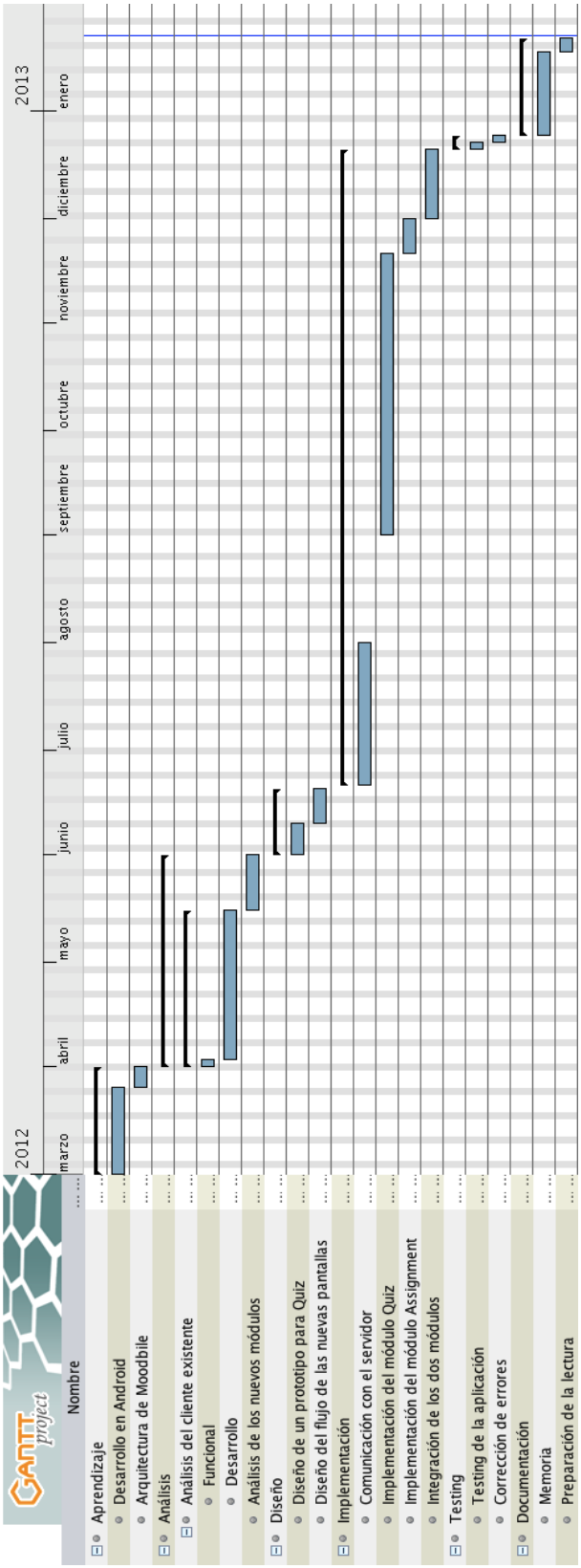


Figura 16 - Diagrama de Gantt

7.3. Coste

Este proyecto se inició en marzo del 2012, y alcanzará su punto final el día 22 de enero del 2013 con la presentación a los miembros del tribunal.

Ha durado once meses, incluyendo un parón de un mes en el verano (Agosto). Durante todo este período, he estado trabajando en paralelo 35 horas semanales, que se han convertido en 40 semanales los dos últimos meses, por lo que sólo le he podido dedicar alguna tarde, noches y fines de semana.

Se ha notado muchísimo la falta de tiempo, a la hora de querer profundizar en algún tema que me interesara, pero que estaba ligeramente margen del proyecto, o cuando he querido optimizar alguna funcionalidad, sin tener muy claro el coste temporal que podría llegar a tener. Por estas razones, y al tener tan poco margen, he tenido que tener mucho cuidado en el sentido que, no tenía que perder ni un segundo en alguna tarea, si no estaba muy segura de que podía aportar algo positivo al proyecto.

Para simplificar el cálculo del coste, asumiré que he trabajado en este proyecto unas 25 horas semanales aproximadamente, que se desglosarían en dos horas diarias de lunes a viernes, y siete horas diarias los fines de semana. Este valor es muy aproximativo, ya que no tiene en cuenta alguna tarde ó día libre que he tenido, y que he podido dedicar al proyecto, pero tampoco tiene en cuenta algún fin de semana que he estado fuera.

Por lo tanto, el coste temporal del proyecto nos queda como sigue:

Primera iteración, de marzo a julio : 22 semanas → 22 * 25 horas = 550 horas

Segunda iteración, de septiembre a enero: 21 semanas → 21 * 25 horas = 525 horas

Total de horas dedicadas = 1025 horas

Este proyecto ha sido realizado por una única persona y no tiene un coste material al ser un trabajo académico, sin embargo, nos encargaremos de calcular cuál sería el coste del mismo en caso de ser un proyecto del mundo profesional, de manera que podamos tener presente cuál sería el coste derivado de nuestro trabajo.

A continuación, basándonos en la planificación y dividiendo el trabajo por el tipo de perfil que lo llevaría a cabo en un caso real, calcularemos el coste del mismo en función del número de horas de dedicación.

Recurso	**Tareas**	**Horas**	**Precio/Hora**	**Coste**
Jefe de proyecto	Requerimientos	75h	60€	4500€
Analista	Análisis y diseño	50h	40€	2000€
Programador	Aprendizaje, desarrollo y testing	900h	30€	27.000€
Total				33.500€

8. Conclusiones

8.1. Conclusiones personales

Este proyecto por el contexto en el que se desarrolló, tiene mucho peso para mí, ya que es la culminación de muchos años de estudio y esfuerzo. En él, se han puesto en práctica muchos de los conocimientos adquiridos a lo largo de la carrera, y con él se pone un punto final a esta última.

Con este proyecto, he aprendido nuevas tecnologías, nuevas formas de trabajo y sobretodo me he enfrentado por primera vez a un proyecto de gran complejidad desde su inicio hasta su fin. He pasado por todos los estados por los que pasa un proyecto software estándar, desde la toma de requisitos para crear una aplicación siguiendo el modelo esperado por el cliente, pasando por su especificación, diseño, implementación y documentación.

Este proyecto me ha permitido adentrarme en el mundo del desarrollo para dispositivos móviles, mundo que hasta entonces no conocía. He aprendido a trabajar con Android, algo que deseaba hacer y que es la razón por la cual escogí este proyecto final de carrera y no otro. La experiencia me ha resultado muy enriquecedora, y no tengo ninguna duda sobre el hecho de que los conocimientos que he adquirido me van a ser de gran utilidad en el futuro.

Otro logro muy importante para mí, ha sido el conseguir trabajar sobre una aplicación y código ya existentes, y el introducir cambios en ellos sin por tanto trastornar la estructura general, ni el correcto funcionamiento del sistema.

Finalmente, he aprendido a adaptarme y reaccionar correctamente frente a los cambios de requisitos que nos hemos visto obligados a adoptar en mitad del proyecto.

8.2. Posibles ampliaciones

A continuación, propondremos brevemente algunas posibles ampliaciones del proyecto que nos parecen oportunas. Comentar que las mejoras que especificaremos son mejoras que corresponden a los dos módulos que desarrollamos, y no a la aplicación en general.

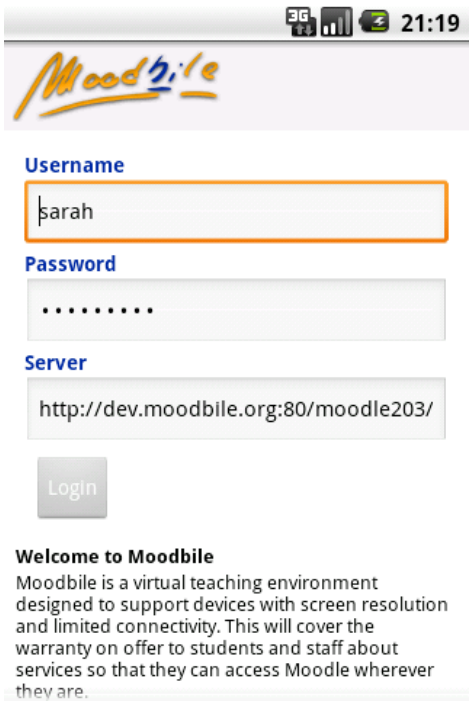
- Implementar un Webservice que permita obtener los quizzes de un curso concreto. De esta forma, podremos agregar los quizzes en el listado principal, y en el menú deslizante, cosa que no se ha podido hacer de momento, al no disponer del servicio web pertinente.

- Implementar la evaluación de un cuestionario en el cliente. Esta evaluación, como ya mencionamos, se iba a llevar a cabo en el servido, y a continuación éste devolvería una nota al cliente. Al ser esta evaluación de muy alta complejidad si se desarrolla en el servidor, se ha decidido hacerlo en el cliente. Por lo que el cliente, recibiría las preguntas con sus respuestas del servidor y a partir de las respuestas del alumno, generaría una nota que se enviaría al servidor.

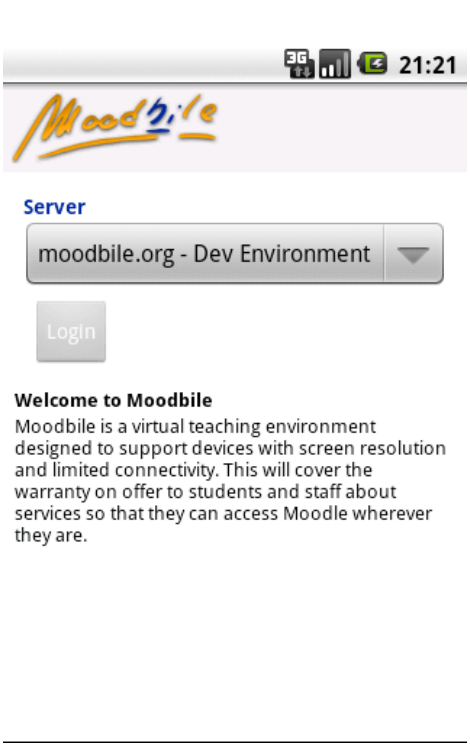
- Acabar la implementación del modulo de tareas. Mi objetivo en este proyecto fue conseguir agregar este modulo al resto de la aplicación, por lo que no implementé todos los tipos de tareas. Me basé en los Assignment de tipo Single Upload, pero faltaría agregar los demás tipos.

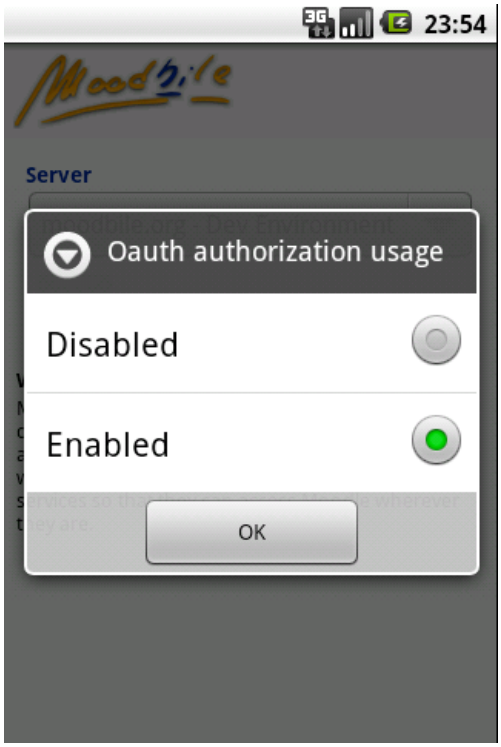
9. Manual de usuario

- Autenticación: La primera pantalla que se muestra cuando se abre la aplicación, es la pantalla de autenticación. Si escogemos la validación por usuario/contraseña, veremos la siguiente pantalla:

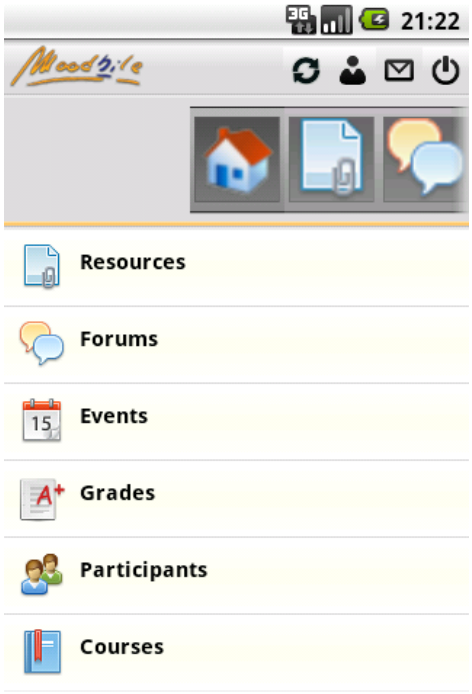


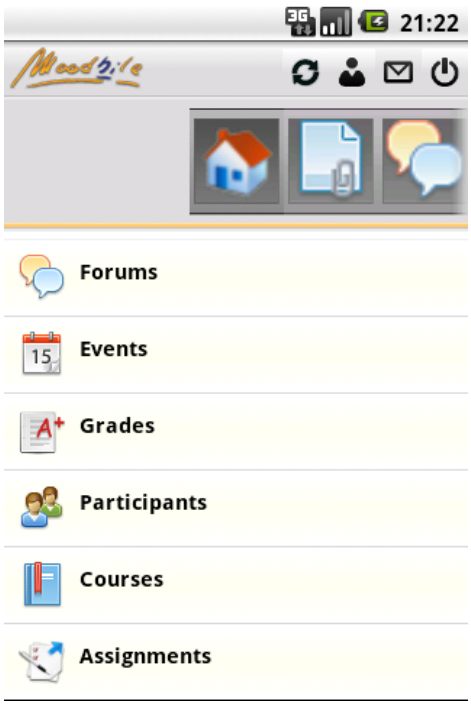
Podemos escoger en las opciones, la validación por OAuth como se muestra en las siguientes pantallas:



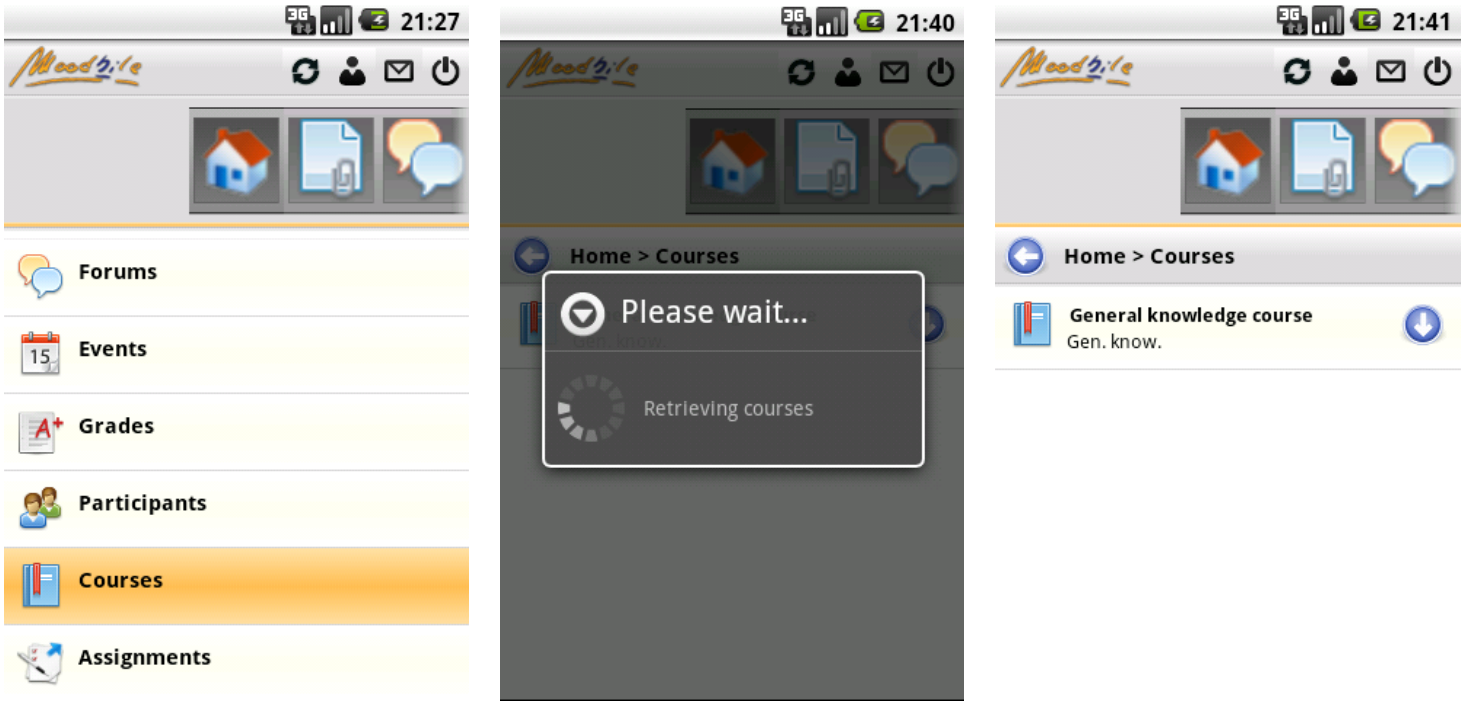


- Menú principal: A continuación, visualizamos el menú principal de la aplicación, desde dónde podremos acceder a los cursos, y diferentes módulos de Moodbile.

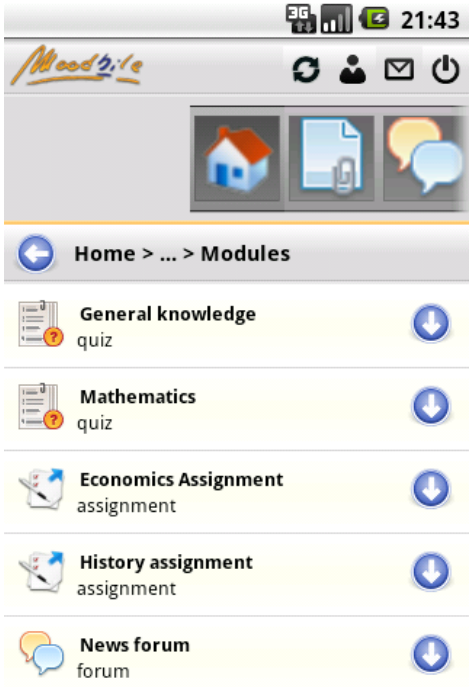




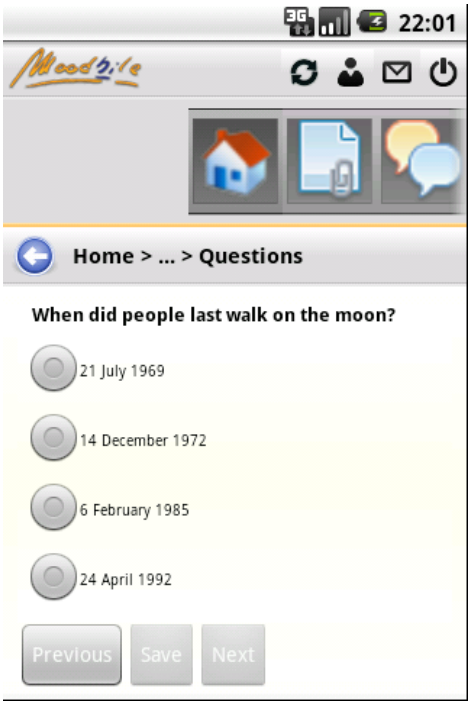
Módulo *Quiz:* Desde el menú principal, podemos seleccionar el componente *Courses,* para visualizar el listado de cursos del usuario conectado.

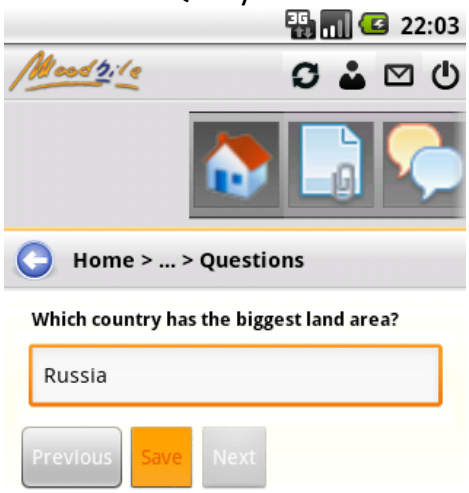


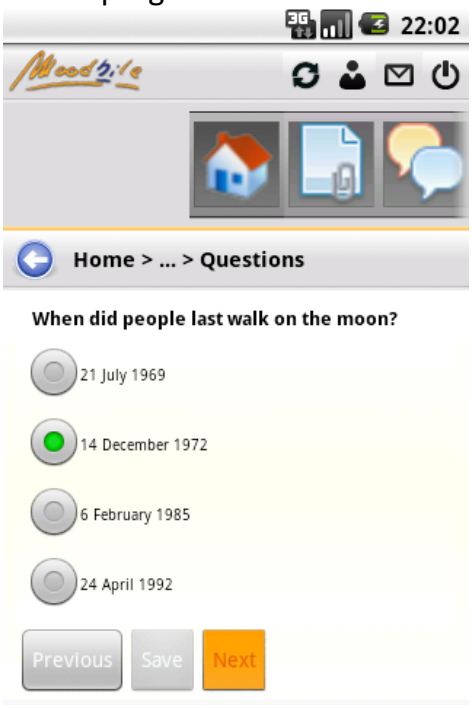
Al seleccionar un curso, se muestra una lista de todos los módulos asociados a éste.



A continuación, se selecciona un *Quiz* y se muestran todas sus preguntas:

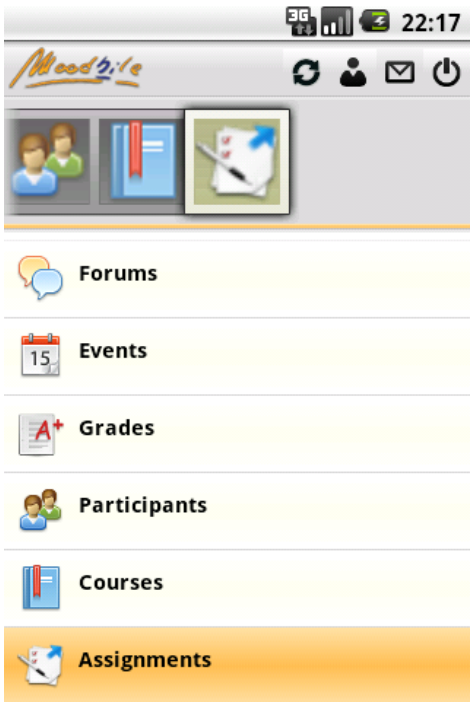


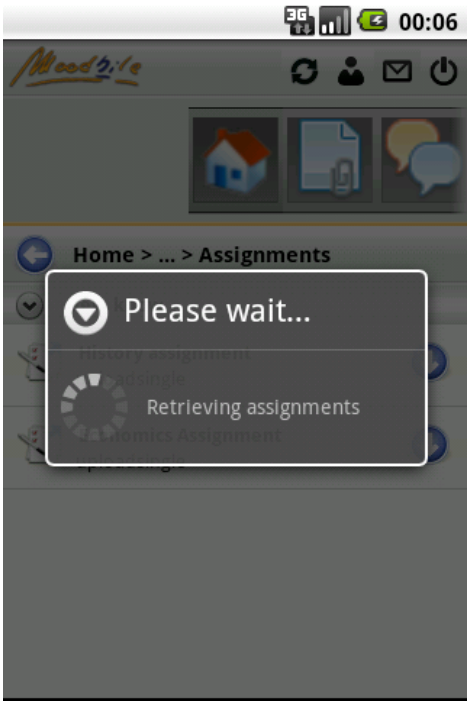


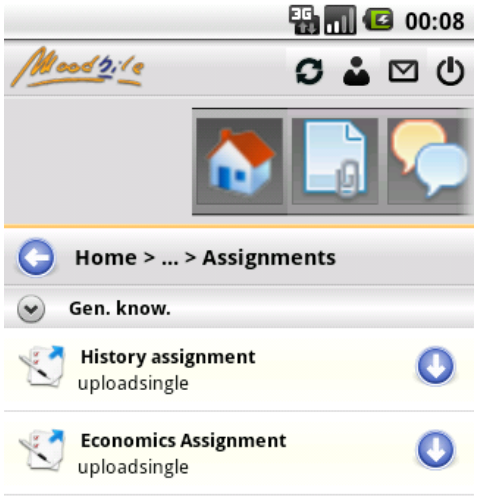


A partir de este punto, podemos recorrer todo el quiz pregunta a pregunta usando los botones *Next* y *Previous*. Y en cualquier momento, podemos decidir salvar todas las preguntas contestadas hasta entonces, haciendo clic sobre el botón *Save.*

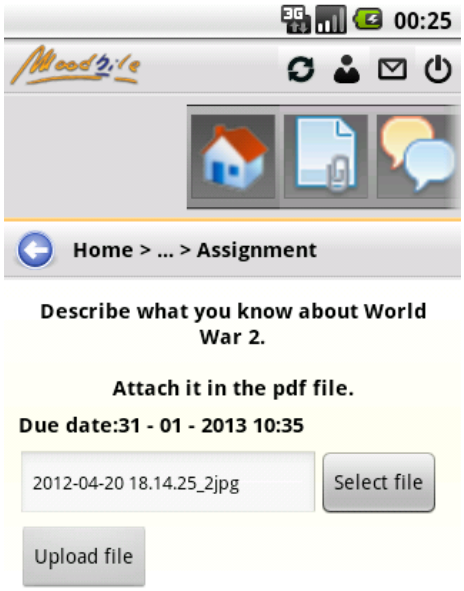
- Módulo de *Assignment:* Desde el menú principal, ó bien el menú deslizante, se puede acceder al listado de Assignments de un usuario conectado.





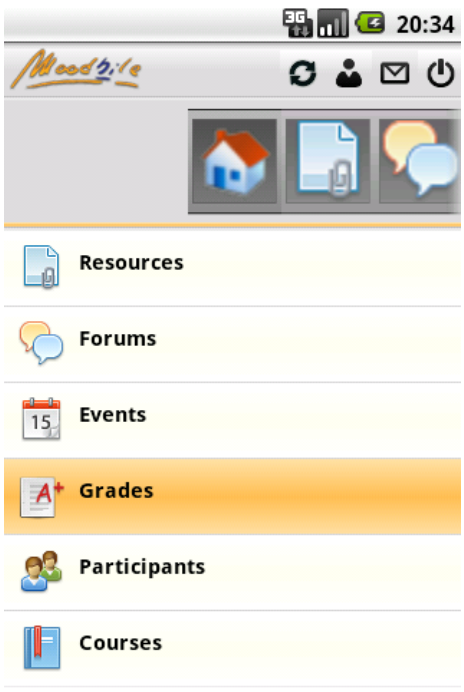


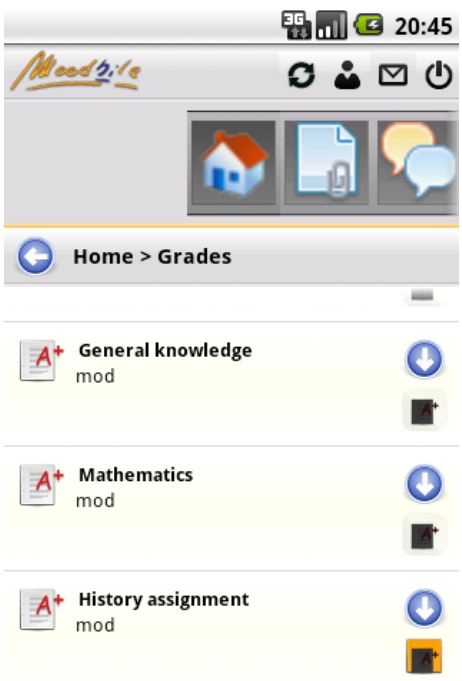
Al seleccionar un *Assignment* de la lista, pasamos a visualizar la pantalla que muestra los detalles de este Assignment, como se ve en la siguiente pantalla:

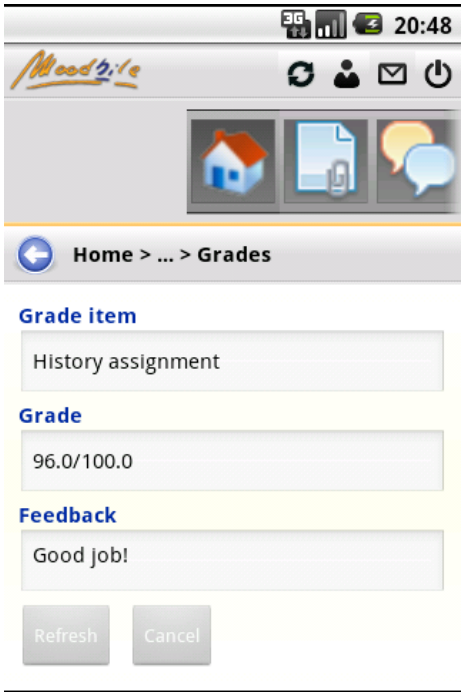


A partir de esta pantalla, podemos recorrer los ficheros del móvil, escoger uno y entregarlo en esta tarea. Cabe destacar que a los assignments también se puede acceder desde la lista de módulos de un curso.

Para acceder a la nota que se le ha concedido al usuario por la entrega de una tarea en concreto, accedemos a la lista de elementos *graduables*, desde el menú principal ó desde el icono *Grades* del menú deslizante, y seguidamente seleccionamos el elemento cuya nota queremos consultar. Se nos dirige a la pantalla que vemos, que muestra, además de la nota, un feedback opcional del profesor.


20:34
Resources
Forums
Events
Grades
Participants
Courses


20:45
Home > Grades
General knowledge
mod
Mathematics
mod
History assignment
mod


20:48
Home > ... > Grades
Grade item
History assignment
Grade
96.0/100.0
Feedback
Good job!
Refresh
Cancel

10. Bibliografía

10.1. Enlaces de interés

www.sgoliver.net

Curso de programación Android

www.tutorialandroid.com

www.vogella.com/articles/Android/article.html

www.androideity.com

Tutoriales de Android

http://developer.android.com

Site de los desarrolladores Android

http://moodbile.org

Moodbile

http://es.wikipedia.org/wiki/Portada

Wikipedia – La enciclopedia Libre

www.stackoverflow.com

Stackoverflow

10.2. Libros

Beginning AndroidTM 4 Application Development

WEI-MENG LEE

Desarrollo de aplicaciones para Android (Manual Imprescindible)

Joan Ribas Lequerica

10.3. Software

Moodle

http://moodle.org/downloads/

Android

http://developer.android.com/sdk/index.html

Navicat

http://www.navicat.com/en/download/download.html

OAuth

http://www.oauth.net/

Eclipse

www.eclipse.org/downloads/

Anexo A: Librerías

- **JDOM**

JDOM es una API desarrollada específicamente para Java y da soporte al tratamiento de XML: manipulación, parseo, búsquedas, modificación, generación y serialización. Es un modelo similar a DOM, pero no está creado ni modelado sobre DOM. La principal diferencia es que DOM fue desarrollado para que fuera independiente del lenguaje, mientras que JDOM está creado y optimizado específicamente para Java. Esto imprime a JDOM las ventajas inherentes a Java, lo que hace que sea una API más eficiente y más natural de usar para el desarrollador Java y por tanto requiere un menor coste de aprendizaje.

Al igual que DOM, JDOM genera un **árbol de nodos** al parsear un documento XML. En cuanto a lo tipos de nodos, son similares a los de DOM, aunque algunos cambia el nombre ligeramente. La jerarquía de clases también se ve algo modificada.

Mientras que en DOM todo hereda de la clase *Node*, en JDOM casi todo hereda de la clase *Content*, que se encuentran en el paquete *org.jdom*. La razón es que *Document* no se considera un contenido, sino más bien el continente y *Attribute* tampoco se considera un nodo, sino una propiedad de los nodos tipo *Element*.

Heredan de *Content*: *Element, Comment y Text.* De este último, *Text*, hereda *CDATA* (*CDATASection* en DOM). Por supuesto, hay más clases que heredan de *Content* y que forman el árbol de nodos, pero no las veremos en estos ejemplos ya que su uso es menos frecuente: *DocType, EntityRef* y *ProccessingInstruction.* Estas o sus equivalentes también existen en DOM.

Otra diferencia entre DOM y JDOM es que mientras en DOM todos estos tipos de nodos eran interfaces, en JDOM son clases y por tanto para crear un objeto de cualquier tipo basta con usar el operador new.

JDOM no es un parser de XML, es una capa de abstracción que se sitúa entre un parser de XML y la lógica de la aplicación, dándole al programador una serie de clases y métodos para tratar con XML fácilmente. Por lo tanto, lo principal será llamar al constructor de un parser de XML y posteriormente eso pasarlo por las clases que nos da JDOM.

- Ventajas de JDOM:

- Pensada especialmente para Java y por tanto mejor integrada en este lenguaje.
- Facilidad de uso (para programadores que conozcan Java)
- Representación de documentos como árboles, lo que posibilita el acceso directo a cualquier parte del documento.

- Desventajas de JDOM:

- Mayor consumo de memoria que SAX (puesto que usualmente se debe procesar y mantener en memoria el árbol del documento XML o una porción del mismo).
- El carecer del concepto de Nodo dificulta en cierta medida la navegación a través del árbol del documento. Por ejemplo, al obtener los hijos de un elemento, el resultado es una lista (java.util.List) de objetos genéricos, y debemos comprobar qué representan esos objetos (por ejemplo usando instanceof) para saber si son elementos, atributos, etc.

Para incluir JDOM en un proyecto. Lo primero es descargar la última versión de JDOM, descomprimir el fichero y simplemente copiar el fichero *jdom.jar* al directorio *libs* del proyecto.

- **Signpost (OAuth)**

Signpost es una solución fácil e intuitivo para la firma de mensajes HTTP en la plataforma Java en conformidad con el estándar Core OAuth 1.0a. Signpost sigue un diseño modular y flexible, lo que le permite combinarlo con diferentes capas de HTTP. Signpost puede ser descargado, modificado y redistribuido bajo los términos de la Licencia Apache versión 2.

Objetivos

Signpost se ha diseñado para cumplir los siguientes objetivos:

- Simplicidad:

El uso de Signpost es de lo más simple posible, todas las acciones se ejecutan con tan sólo unas pocas líneas de código. A continuación, mostramos un ejemplo de firma de una mensaje http usan Signpost (suponiendo que ya se han creado los objetos HttpClient y OAuthConsumer):

```
// create an HTTP request to a protected resource
HttpGet request = new HttpGet("http://example.com/protected");
// sign the request (consumer is a Signpost OAuthConsumer)        consumer.sign(request);
// send the request
HttpResponse response = httpClient.execute(request);
```

Signpost tiene una API minimalista diseñada con dos objetivos: firmar los mensajes HTTP y solicitar tokens de un proveedor de servicios de OAuth. Todo lo demás está fuera del alcance de la especificación OAuth, y por lo tanto se deja a la capa de mensajería HTTP.

- Discreción:

Signpost trata de ser lo más discreto posible. Signpost no envuelve toda la capa de HTTP y oculta sus propiedades al cliente. Simplemente se le pasa un objeto HttpRequest, y de Signpost se encarga de firma el mensaje utilizando las credenciales con las que se ha configurado.

Para incluir la librería en el proyecto, el primer paso consiste en descargar las librerías de SignPost. Para ello nos dirigimos a la página de OAuth-SignPost, y descargar tanto signpost-core.jar como signpost-commonshttp.jar, y seguidamente copiamos los jars en la carpeta libs.

- **Guice**

RoboGuice elimina la mecánica del desarrollo en Android de nuestro código. Inyectar su vista, recursos, servicios del sistema, o cualquier otro objeto, y que RoboGuice se encargue de cuidar los detalles. Solo tenemos que centrarnos en la lógica real de la aplicación.

Muchas aplicaciones destacadas, como **Facebook Messenger**, **Pulse**, **Google Docs** o **SwiftKey** ya usan esta librería.

Para usar RoboGuice tendremos que descargarnos estas dos librerías:

RoboGuice

guice-2.0-no_aop.jar

Una vez descargadas añadirlas a la carpeta libs de nuestro proyecto.

Como último paso deberemos crear una clase que extienda de RoboApplication y configurarla correctamente en el AndroidManifest.xml.

Con esto ya tendríamos listo nuestro proyecto para usar RoboGuice. A continuación, mostramos un ejemplo de su uso.

Así sería nuestro código sin usar RoboGuice:

```
class AndroidWay extends Activity {
   TextView name;
   ImageView thumbnail;
   LocationManager loc;
   Drawable icon;
   String myName;

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      name      = (TextView) findViewById(R.id.name);
      thumbnail = (ImageView) findViewById(R.id.thumbnail);
      loc       = (LocationManager) getSystemService(Activity.LOCATION_SERVICE);
      icon      = getResources().getDrawable(R.drawable.icon);
      myName    = getString(R.string.app_name);
      name.setText( "Hello, " + myName );
   }
}
```

Y así usándolo:

```
class RoboWay extends RoboActivity {
   @InjectView(R.id.name)          TextView name;
   @InjectView(R.id.thumbnail)     ImageView thumbnail;
   @InjectResource(R.drawable.icon)  Drawable icon;
```

```
    @InjectResource(R.string.app_name) String myName;
    @Inject                    LocationManager loc;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        name.setText( "Hello, " + myName );
    }
}
```

Como podemos comprobar a simple vista el código queda mucho mas limpio y se nos facilita bastante el trabajo de implementación.

- **JEval**

JEval es una librería avanzada para el análisis y la evaluación de operaciones matemáticas, con lenguaje Java. JEval se autoriza y se distribuye bajo la licencia 2.0 de Apache.

A continuación, algunas características de JEval:

· Analiza y evalúa expresiones dinámicas y estáticas.

· Soporta variables

· Soporta valores matemáticos, booleanos, secuencias y funciones.

· Soporta los operadores matemáticos y booleanos más importantes.

· No depende de ninguna otra biblioteca no estándar de Java.

· Soporta pruebas de JUnit.

Hemos hecho uso de esta librería, para la evaluación de variable y ecuaciones en las preguntas de tipo Calculated, Multichoice, Calculated, etc…

- **Android-File-Explorer**

Esta es una librería que nos permite disponer de un explorador de ficheros del dispositivo móvil.

Las características son las siguientes:

- Posibilidad de navegar a través de múltiples niveles jerárquicos
- Posibilidad de seleccionar un archivo ó un directorio
- En la actualidad no permite agregar o eliminar archivos

Para poder usarla dentro del proyecto:

- Copiar el archivo FileExplore.java en el proyecto
- Copiar los elementos gráficos en la carpeta res.
- Agregar la funcionalidad que nos interesa en el fichero FileExplore.java