Escuela Politécnica Superior

Cliente de MirBot para Android

Máster Universitario en Desarrollo de Software para Dispositivos Móviles

Trabajo Fin de Máster

Autor:
David Antolín González

Tutor/es:
Antonio Javier Gallego Sánchez

Septiembre 2016

Universitat d'Alacant
Universidad de Alicante

Justificación y Objetivos

Este trabajo tiene como motivación crear y publicar un cliente para el sistema operativo Android del proyecto MirBot desarrollado por el departamento de Ciencia de la Computación e Inteligencia Artificial de la Universidad de Alicante, creando un producto equivalente a la app publicada para el sistema operativo iOS, añadiendo funcionalidades que la hagan más atractiva para el usuario, como un sistema de gamificación basado en puntuación, ranking y medallas.

La aplicación permitirá al usuario enviar imágenes al servicio para su almacenamiento y clasificación. Para cada imagen la inteligencia artificial procesará y propondrá una clase en la que clasificar dicha imagen. En caso de no ser correcta se le ofrecerá una lista de sugerencias con más clases y en última instancia el usuario debe elegir la clase de un diccionario basado en la base de datos lingüística WordNet. La inteligencia artificial utiliza el feedback del usuario para mejorar sus capacidades de clasificación, siendo cada vez más capaz de clasificar correctamente las imágenes enviadas.

La aplicación permitirá al usuario elegir entre dos modos de entrenamiento: utilizando sólo las imágenes subidas por el usuario o utilizar las imágenes subidas por todos los usuarios del servicio.

La aplicación amenizará el proceso de clasificación de manera que parezca un diálogo entre MirBot y el usuario, utilizando para ello la síntesis de voz junto con una serie de oraciones e imágenes predefinidas en función de los resultados de la clasificación.

La aplicación también permitirá al usuario gestionar la colección de imágenes que ha ido subiendo, pudiendo filtrar las imágenes por término de clasificación, eliminar las imágenes que haya clasificado incorrectamente o etiquetar ciertas imágenes con nombres concretos.

Agradecimientos

A mi tutor del trabajo de fin de máster Antonio Javier Gallego Sánchez por la paciencia que ha tenido, preocupándose por el estado del proyecto y siempre ofreciendo feedback y ayuda.

A Miguel Ángel Lozano Ortega por ser el organizador del máster.

A todos los profesores que a lo largo del máster han conseguido transmitir sus conocimientos a los alumnos.

A Higinio Mora Mora por conseguirme el primer trabajo durante el máster.

A mi madre, que por fin podrá respirar tranquila cuando esto esté presentado. Se lo debo.

A mis amigos de toda la vida que siguen ahí: Alejandro Gascón, David Marín, Fanyu Xu, Alejandro Navarro y Manuel Radu.

A las personas que he conocido en el baile, con las que he compartido muchas cosas en los últimos años, entre ellas quiero citar a: Mirta, Sonia, Fran, Tamara, Iván, Silvia, Javi Rosillo, Javi Tocho, Juanfran, Manu, Natalia y Cristina. He reído, he llorado, he amado, he disfrutado, he sufrido, he bailado, he hecho el tonto, he aprendido, he hecho daño y me han hecho daño. Gracias y perdón a partes iguales. Y lo que os queda todavía.

Dedicatoria / Citas

“Ni tu peor enemigo puede hacerte tanto daño como tus propios pensamientos”

Buda

“Solo existen dos días en el año en que no se puede hacer nada. Uno se llama ayer y otro mañana. Por lo tanto, hoy es el día ideal para amar, crecer, hacer y principalmente vivir.”

Dalai Lama

Índices

JUSTIFICACIÓN Y OBJETIVOS **3**

AGRADECIMIENTOS **5**

DEDICATORIA / CITAS **5**

ÍNDICES **6**

INTRODUCCIÓN **8**

OBJETIVOS 9

MARCO TEÓRICO O ESTADO DEL ARTE **10**

GOOGLE GOGGLES [1] 10

Reconoce *10*

Resultados *10*

Ventajas *10*

Desventajas *11*

CAMFIND [2] 11

Reconoce *11*

Resultados *11*

Ventajas *11*

Desventajas *11*

BLIPPAR [3] 11

Reconoce *11*

Resultados *12*

Ventajas *12*

Inconvenientes *12*

SHOT & SHOP [4] 12

Reconoce *12*

Resultados *12*

Ventajas *12*

Desventajas *12*

CUERPO DEL TRABAJO **13**

DISEÑO DE LA APLICACIÓN 13

BACKEND 18

Web *18*

API REST *18*

Clasificación *19*

TECNOLOGÍAS 20

Android SDK *20*

Android Studio [8] *21*

Android Support Library [9] *22*

Google Play Services Client Library [10] *24*

Glide [11] *25*

Dagger2 [12] *26*

Butter Knife [15] *28*

RxJava y RxAndroid *29*
Retrofit [17] *30*
MosbyMVP [18] *31*
Crashlytics [19] *32*
ARQUITECTURA 34
Clean architecture *34*
Patrón Model-View-Presenter *35*
Estructura de la aplicación *35*
IMPLEMENTACIÓN 37
Elección del nivel de API mínimo *37*
Inyección de dependencias *37*
Clases base de componentes Android *38*
Vector Drawables y Glide *39*
Imágenes y memoria *40*
Envío de las imágenes utilizando Retrofit *40*
Aspecto final de la aplicación *41*
CONCLUSIONES Y TRABAJO FUTURO 51
BIBLIOGRAFÍA Y REFERENCIAS 53
ANEXOS 55
API DE MIRBOT 55
GET /api/v1/user/{hash}/stats *55*
GET /api/v1/user/{hash}/ranking *56*
GET /api/v1/user/{hash}/badges *56*
GET /api/v1/user/{hash}/robot *57*
PUT /api/v1/user/{hash}/robot *57*
GET /api/v1/user/{hash}/labels/{class?} *58*
GET /api/v1/user/{hash}/images *58*
GET /api/v1/user/{hash}/images/{imgid} *59*
PUT /api/v1/user/{hash}/images/{imgid}/label *60*
DELETE /api/v1/user/{hash}/images/{imgid} *60*
POST /api/v1/user/{hash}/images/classify *60*
POST /api/v1/user/{hash}/images/{imgid}/confirm *61*
GET /api/v1/wordnet *62*
GET /api/v1/wordnet/lemma/{lemma} *63*
GET /api/v1/wordnet/class/{class_id} *63*
METADATOS SOPORTADOS PARA LA CLASIFICACIÓN 65

Introducción

Este trabajo fue propuesto por el profesor Antonio Javier Gallego Sánchez en el curso 2014-2015 y su objetivo es llevar a la plataforma Android una aplicación oficial del proyecto MirBot, ya que actualmente sólo está disponible para dispositivos iOS.

La aplicación permite al usuario enviar al servicio imágenes tomadas con la cámara del dispositivo. El servicio dispone de una inteligencia artificial, que explicaremos en la sección de metodología, dedicada a la clasificación de imágenes que devuelve una lista de clases en las que ha clasificado la imagen y el usuario debe confirmar una de las clases propuestas o aportar una en caso de que ninguno sea correcto. La inteligencia artificial es entrenada con el feedback que proporciona el usuario de manera que será más capaz de clasificar imágenes en posteriores ocasiones.

El proceso de confirmación se realiza mediante una interfaz que simula un diálogo entre el simpático robot, mascota del proyecto, y el usuario utilizando imágenes, texto escrito y sintetización de voz (desactivable por el usuario en los ajustes). El robot se expresará mediante diferentes imágenes y oraciones predefinidas en función de los resultados de la clasificación obtenidos por el servicio y la clase que confirme el usuario.

La aplicación proporciona, además de la imagen tomada por la cámara, toda una serie de metadatos que ayudan en la clasificación de la misma: el área de la imagen que se debe tomar en cuenta para la clasificación, la orientación espacial del móvil en el momento de tomar la fotografía utilizando los sensores integrados del móvil como el acelerómetro y el sensor de campo magnético, la ubicación del usuario mediante el GPS (opcional) y metadatos EXIF de la imagen.

La aplicación permite al usuario dos modos de utilización intercambiables en cualquier momento: entrenar a la inteligencia artificial utilizando sólo las imágenes subidas por el usuario o utilizar las imágenes subidas por el resto de usuarios del servicio. Gracias a esto, cada usuario puede entrenar una inteligencia artificial especializada solo en reconocer ciertos tipos de entidades en las imágenes.

La aplicación a su vez cuenta con una sencilla capa de gamificación. A cada robot se le asigna un valor de IQ que aumenta progresivamente según va aprendiendo. En la aplicación se puede consultar un ranking en el que se muestran los 10 mejores robots en términos de IQ y la posición del robot del usuario con respecto al top10, y una pantalla de estadísticas para comparar las estadísticas de tu robot con respecto a las globales del servicio.

Otro elemento de gamificación es la consecución de medallas al superar ciertas metas. Al finalizar una clasificación la aplicación alertará al usuario si ha conseguido nuevas medallas mediante un diálogo. Actualmente las metas están basadas en conseguir superar cierto número de imágenes de cada una de las principales categorías.

El usuario tiene a su disposición todas las imágenes que ha subido al servicio en la aplicación, de manera que puede navegar por ellas, buscar las imágenes que corresponden a una determinada clase, ver información extendida de la clase de una imagen, como los sinónimos y la jerarquía,

eliminar imágenes cuya clasificación haya sido incorrecta, etiquetar imágenes para facilitar su búsqueda, etc.

Objetivos

La aplicación se ha desarrollado para ser equivalente en funcionalidad a su homóloga de iOS, por lo que debe ser capaz tanto de tomar fotos y realizar el proceso de clasificación como de poder gestionar la colección de imágenes del usuario. En concreto:

- Permitir al usuario capturar imágenes a través de la cámara del dispositivo.
- Permitir al usuario seleccionar el área de la imagen que se utilizará en la clasificación.
- Recabar metadatos de la captura: aceleración, orientación, gps, metatados EXIF de la imagen, etc.
- Enviar la imagen al servicio para su clasificación.
- Proporcionar una interfaz para que el usuario interactúe con los resultados de la clasificación, simulando una conversación entre el usuario y el robot, donde se permite al usuario elegir una de las clases sugeridas por el robot o elegir una clase del diccionario WordNet.
- Permitir al usuario gestionar su colección de imágenes: navegar por la colección, filtrar, ver detalles, etiquetar, eliminar, etc.
- Permitir al usuario revisar su progreso y compararlo con respecto al resto de usuarios: consultar IQ del robot, ranking de robots, estadísticas, etc.

Marco teórico o estado del arte

La aplicación de Mirbot para Android tiene como característica distintiva el reconocimiento de objetos en imágenes y obtención de información de ese objeto (clase del objeto, definición, sinónimos, jerarquía).

La búsqueda de aplicaciones en Play Store que atiendan a los criterios de reconocimiento de objetos en imágenes ha obtenido las siguientes:

Google Goggles [1]

Esta aplicación, desarrollada por Google, reconoce objetos mediante la toma de fotos y devuelve resultados de búsqueda e información relacionada con el mismo.

Reconoce

- Lugares del mundo
- Obras de arte
- Logotipos
- Monumentos
- Textos (en francés, inglés, italiano, español, portugués, turco y ruso)
- Vinos
- Portadas de revistas
- Anuncios dentro de revistas y periódicos.
- Libros
- Carátulas de CD y DVD
- Posters
- Tarjetas de visita
- Códigos de barras y QR
- Sudokus

Resultados

- Información del Knowledge Graph de Google.
- Imágenes similares a la proporcionada.
- Resultados de búsqueda en Google.
- Traducción automática del texto reconocido a cualquier idioma disponible en Google Translate.
- Información del producto correspondiente al código de barras, con productos similares, comparador de precios, etc.
- Extracción de la información del código QR, como la dirección de un sitio web, un contacto para almacenar en el móvil, etc.
- Extracción de un contacto para almacenar en el móvil a partir de una tarjeta de visita.
- Resolución del Sudoku.

Ventajas

- Backend de reconocimiento de objetos y resultados respaldado por la gran cantidad de información que una empresa como Google es capaz de proporcionar.
- Escáner de códigos de barras y QR que puede ser utilizado por otras aplicaciones. No se necesita tomar una foto si no que lo reconoce en tiempo real.

Desventajas

- No reconoce todo tipo de objetos.
- La aplicación no se actualiza desde mayo de 2014.
- La interfaz de usuario está desactualizada, no cumple con las reglas Material Design de la propia Google.

CamFind [2]

Aplicación desarrollada por la empresa CamFind Inc.

Reconoce

- Casi cualquier tipo de objeto
- Lugares del mundo
- Obras de arte
- Logotipos
- Monumentos
- Textos
- Carátulas de CD y DVD
- Posters
- Códigos de barras y QR

Resultados

- Conjunto de términos que describen el objeto, por ejemplo, black cordless computer mouse.
- Resultados de búsqueda web.
- Imágenes similares y/o relacionadas.
- Lugares relacionados.
- Comparador de precios de productos.
- Traducción de texto.
- Información de la película correspondiente al póster o carátula, como sinopsis, tráiler, horarios en cines, etc.

Ventajas

- Reconoce un mayor número de objetos.
- Feed social en el que los usuarios pueden compartir sus búsquedas.

Desventajas

- El reconocimiento de imágenes es muy lento.
- Interfaz de usuario basada en iOS.

Blippar [3]

Blippar es una aplicación para iOS y Android creada por la compañía homónima.

Reconoce

- Logos de marcas
- Portadas de revistas
- Posters
- Carátulas

- Envases
- Alimentos y productos de alimentación
- Libros
- Casi cualquier tipo de objeto

Resultados

- Definición.
- Artículos.
- Fotos y vídeos.
- Información nutricional.
- Recetas.
- Records mundiales.
- Guías.
- Búsquedas de otros usuarios.

Ventajas

- Reconocimiento de objetos interactivo, no es necesario tomar una foto si no que graba y procesa los fotogramas de la previsualización.
- Interfaz de usuario muy agradable de utilizar.
- Dispone de un modo de realidad aumentada para algunos de los objetos reconocidos.

Inconvenientes

- El reconocimiento es más lento y la colocación del objeto es importante para un correcto reconocimiento.

Shot & Shop [4]

Aplicación de compra de productos y calzado que cuenta con un modo especial de búsqueda basado en el reconocimiento de objetos.

Reconoce

- Productos de ropa y calzado

Resultados

- El producto buscado si se encuentra en su tienda.

Ventajas

- Permite comprar el producto desde la propia aplicación.
- Nos permite definir un área de reconocimiento mediante dibujo a mano alzada.
- Nos permite discriminar por forma y/o color.
- Nos permite encontrar otras prendas de ropa que combinen con la reconocida.

Desventajas

- Interfaz basada en iOS.
- El reconocimiento no es del todo preciso.

Cuerpo del trabajo

Diseño de la aplicación

En la aplicación de iOS se utilizó un diseño enfocado principalmente a la toma de imágenes para subir al servicio de esta forma la primera pantalla con la que el usuario puede interactuar es la de la cámara. Desde esta pantalla se puede acceder a la colección de imágenes y a los ajustes de la aplicación.

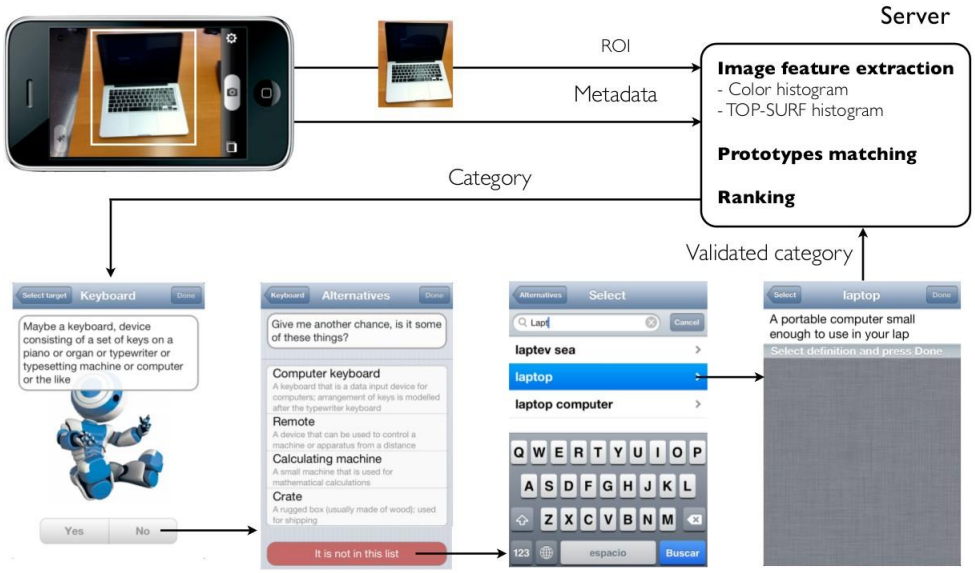

Ilustración 1 - Esquema de funcionamiento de la aplicación de iOS

En la aplicación Android se ha optado por dotar de más importancia a la colección de imágenes, de manera que esa será la pantalla principal de la aplicación (ilustración 2). En la pantalla se muestran las imágenes de la colección de cada una de las cinco principales categorías. Se puede navegar por estas categorías mediante desplazamientos laterales o tocando en el título de la categoría. Las imágenes están dispuestas en forma de cuadrícula de tres columnas en vertical y cinco en horizontal.

Para la navegación por las principales secciones de la aplicación se ha optado por aplicar el patrón de diseño de cajón de navegación (ilustración 3), accesible pulsando el icono de "hamburguesa" situado en la esquina superior izquierda o realizando un desplazamiento desde fuera hacia adentro desde el borde izquierdo de la pantalla. Desde aquí podemos acceder a las medallas, el ranking de robots, las estadísticas, los ajustes y la ayuda de la aplicación. También se muestra el nombre que le hemos dado al robot y su IQ, contando con una barra muestra el porcentaje de IQ con respecto al máximo que puede conseguir el robot.

Tomar imágenes para su clasificación sigue siendo la principal acción de la aplicación importancia por lo que se ha implementado en esta pantalla en forma de botón de acción flotante (FAB) siguiendo las guías de Material Design.

Desde esta pantalla también se accede a la búsqueda (icono de lupa en la parte superior derecha) que muestra una caja de búsqueda junto con la lista de resultados (ilustración 4).

Si tocamos una de las imágenes navegaremos a la pantalla de detalle (ilustración 5), donde podemos ver la definición de la clase, sinónimos y la jerarquía de WordNet hasta llegar a esa clase. La acción de etiquetar está presente en forma de FAB y la de eliminar en forma de icono en esquina superior derecha.

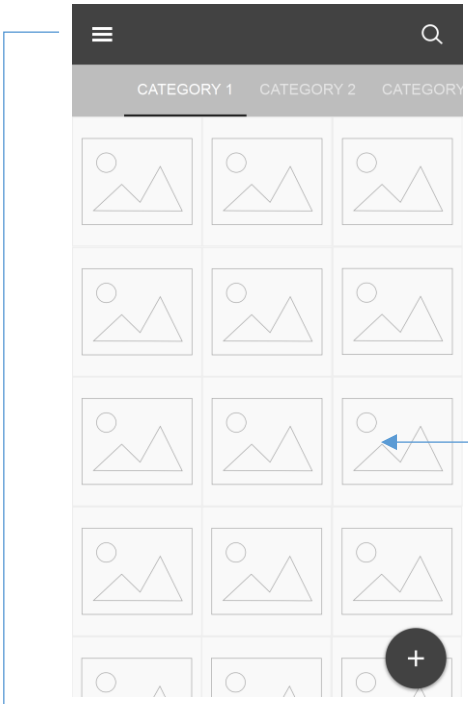

Ilustración 2 - Pantalla principal

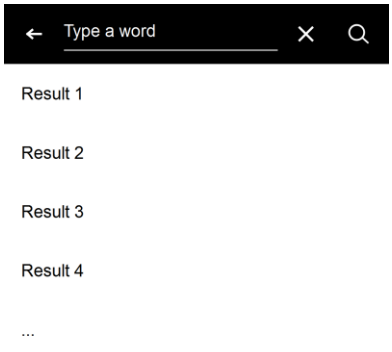

Ilustración 4 - Botón búsqueda

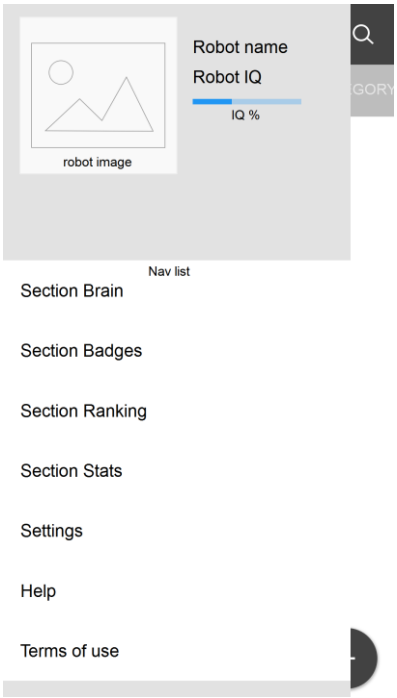

Ilustración 3 - Navegación

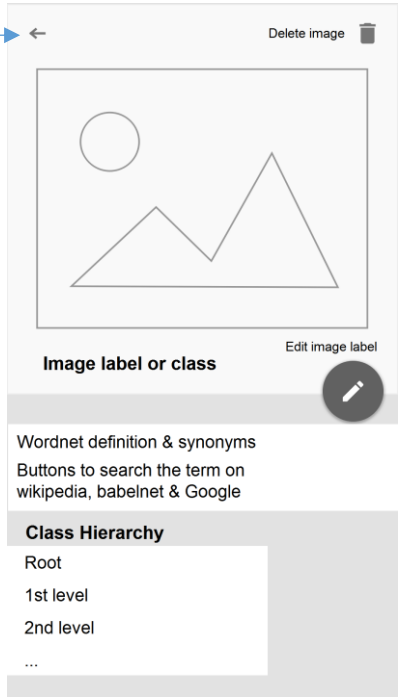

Ilustración 5 - Detalle imagen

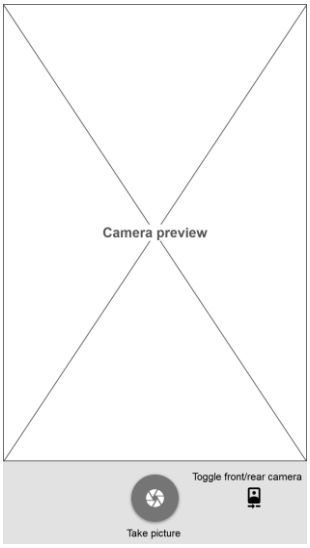

Ilustración 6 - Cámara

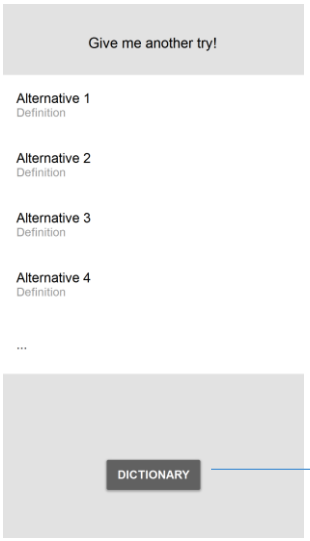

Ilustración 7 - Clases alternativas

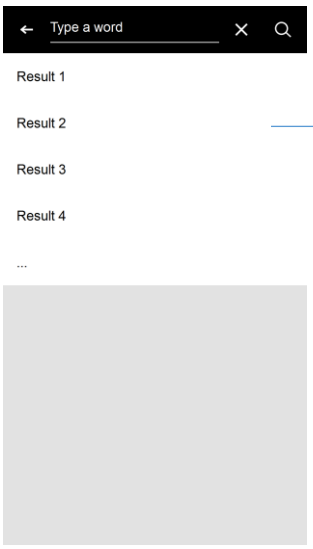

Ilustración 8 - Diccionario

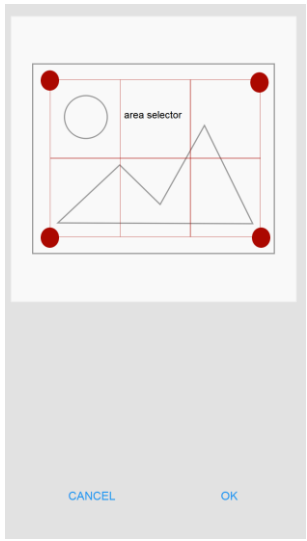

Ilustración 9 - Selección del ROI

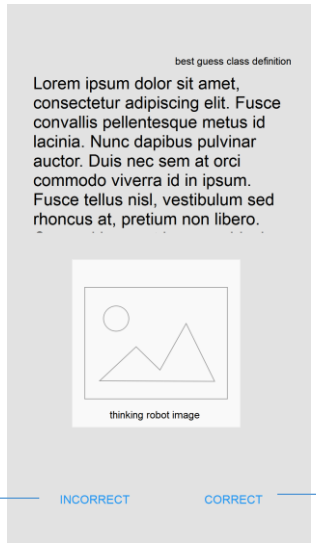

Ilustración 10 - Clase más probable

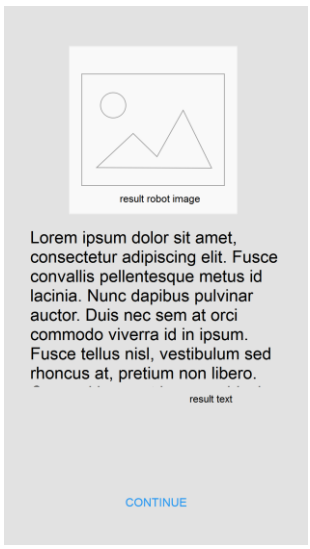

Ilustración 11 - Pantalla final

La pantalla de la cámara (ilustración 6) es muy similar a la de la aplicación de iOS. Se muestra en pantalla una previsualización de lo que está captando la cámara en estos momentos. Cuenta con un botón para tomar la fotografía y, como añadido, un botón para cambiar entre la cámara trasera y la frontal (por si quieres que el robot se quede con tu cara).

Una vez tomada la fotografía pasamos a la pantalla de selección del área de interés (ROI), que nos permite seleccionar el área de la imagen que queremos que se analice (ilustración 9). Por defecto toda la imagen queda seleccionada (en contraste con la aplicación de iOS en el que el usuario debe seleccionar algo antes de poder avanzar).

Al pulsar aceptar, se envía la imagen al servicio Mirbot, que analiza y nos devuelve unos resultados de clasificación.

Para interactuar con los resultados de la clasificación, se simula una conversación entre el usuario y el robot a través de una serie de pantallas. Estas pantallas serán similares a las de iOS, pero con un estilo basado en Material Design.

En la primera pantalla (ilustración 10), se coloca el texto con la descripción de la clase más probable encima del robot y las acciones del usuario debajo. Si se sitúa el dispositivo en horizontal está estructura es modificada con el robot en la parte izquierda y el texto y las acciones a la derecha para aprovechar mejor la pantalla.

En el caso de que la clase no sea correcta y haya clases similares se le mostrará al usuario la pantalla de sugerencias (ilustración 7), con un pequeño texto de conversación del robot instándonos a seleccionar una alternativa junto a la lista de alternativas y la acción de seleccionar del diccionario. En horizontal la lista de alternativas se muestra a dos columnas.

La pantalla del diccionario (ilustración 8) es la misma que la de búsqueda de la pantalla principal.

Por último, la pantalla final de clasificación (ilustración 11) es similar a la de primera opción, pero invirtiendo las posiciones del robot y el texto. La imagen del robot y el texto dependerán de cuan acertado estuvo el servicio. Al finalizar se nos devuelve a la pantalla principal.

La pantalla de la sección medallas (ilustración 12) muestra la lista de medallas (primero las ya obtenidas) en formato de una columna en vertical y dos columnas en horizontal. Al pulsar sobre una medalla se abre un diálogo con el detalle de la medalla.

La pantalla de la sección ranking (ilustración 13) muestra una lista con el top ten de robots. Si el robot del usuario no está en el top ten aparecerá un ítem adicional. El robot del usuario es fácilmente identificable porque es el elemento de la lista con la imagen del robot.

La pantalla de la sección estadísticas (ilustración 14) nos permite comparar las estadísticas del robot del usuario con las estadísticas globales. En la aplicación de iOS la estructura de esta pantalla es todas las estadísticas del usuario seguidas de todas las estadísticas globales. En la aplicación de Android irán en paralelo a dos columnas para una comparación más directa. La columna de la izquierda pertenece al usuario y la derecha al global.

La pantalla de ajustes es una pantalla de preferencias estándar de Android. En ella podemos:

- Cambiar el nombre a nuestro robot.
- Cambiar entre aprendizaje individual o colectivo (switch).
- Activar o desactivar la síntesis de voz (switch).

Por último, la pantalla de ayuda permite reproducir el vídeo de introducción y el vídeo de cómo tomar imágenes y contiene un enlace a la web del proyecto MirBot y la pantalla de términos de uso permite al usuario conocer los términos de uso del proyecto y la aplicación.

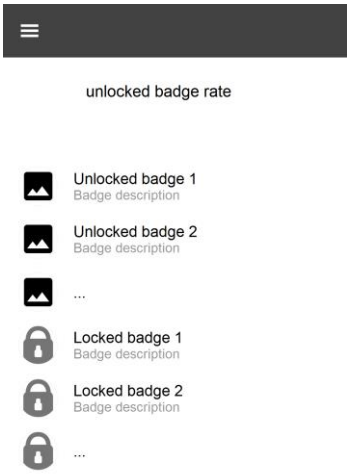

Ilustración 12 - Pantalla medallas

Ilustración 14 - Pantalla estadísticas

Ilustración 13 - Pantalla ranking

Backend

La parte servidor del proyecto MirBot está programada utilizando el framework PHP Laravel [5], creado por el desarrollador Taylor Otwell. Sobre él se construye el sitio web del proyecto y la API REST que utilizan las aplicaciones de iOS y Android para comunicarse con el servidor. El almacenamiento de datos se realiza en un sistema de gestión de base de datos MySQL, de tipo relacional.

Web

La Web del proyecto MirBot [6] cuenta con una parte pública dónde se encuentra información sobre el proyecto, investigación realizada y tecnologías utilizadas, enlaces a los clientes en las diferentes tiendas de aplicaciones, organigrama de desarrolladores y colaboradores y un formulario de contacto.

También cuenta con una parte privada (acceso mediante usuario y contraseña) para gestionar las imágenes enviadas por los usuarios.

Podemos navegar por el árbol de jerarquía de WordNet viendo las imágenes que han sido clasificadas en cada palabra mediante una interfaz de usuario similar a un explorador de archivos. Tenemos la opción de realizar búsquedas que nos devolverán las clases (carpetas) que coinciden con el término buscado.

Tenemos otra sección de revisión en el que vemos todas las imágenes subidas en orden descendente, pudiendo filtrar sólo las que se han marcado como clasificadas incorrectamente.

Para cada imagen tenemos las acciones de marcar como clasificada incorrectamente si no estamos de acuerdo con la clasificación hecha por el usuario, reasignar la imagen a otra clase o eliminar la imagen.

Si entramos en el detalle de una imagen veremos información del usuario que la envió, de la clase asignada, del proceso de clasificación y de los metadatos enviados por el cliente.

API REST

REST, REpresentational State Transfer, es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.

REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.

La API REST de MirBot, actualmente en la versión 1, permite a los usuarios enviar las imágenes para su clasificación, así como obtener y gestionar las imágenes enviadas.

Los datos se envían y se reciben en formato JSON, salvo algunos parámetros que se insertan en el query string y las imágenes que se envían como multipart.

La API está segmentada por usuario, es decir, todas las llamadas llevan un identificador de usuario, las imágenes enviadas quedan registradas como suyas y sólo puede obtener y gestionar sus imágenes enviadas.

Disponemos de los siguientes métodos:

- Obtener imágenes, filtradas por clase y con la posibilidad de adjuntar también las de sus descendientes en la jerarquía.
- Cambiarle la etiqueta a una imagen
- Eliminar una imagen
- Cambiar el nombre al robot
- Obtener las medallas obtenidas por el robot, así como las que todavía están por desbloquear.
- Obtener el ranking de los N mejores robots, así como el robot del usuario para comparar.
- Obtener estadísticas (del usuario y globales)
- Enviar una imagen a clasificar, junto con el conjunto de categorías en las que se quiere clasificar, si se desea utilizar para la clasificación sólo las imágenes del usuario o las de todos los usuarios y un conjunto de metadatos que proporcionan información extra que ayuda en la clasificación.

Una descripción detallada de la API se puede encontrar en los anexos.

Clasificación

WordNet [7]

El sistema de clases en las que MirBot clasifica las imágenes está basado en la plataforma WordNet de la universidad de Princeton.

WordNet es una base de datos léxica para el idioma inglés. Cada clase, o synset, corresponde a un concepto y tiene un conjunto de palabras que corresponden a ese concepto y son sinónimas entre sí. Las clases están interrelacionadas semánticamente formando una red cognitiva.

En MirBot esa red se ha simplificado en una estructura jerárquica de categorías.

Deep learning

El aprendizaje profundo (o deep learning en inglés) es un conjunto de algoritmos de aprendizaje automático que intenta modelar representaciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no-lineales múltiples. Se basa en la creación de topologías de redes neuronales que apilan múltiples capas de procesamiento de distintos tipos, en las cuales se aplican transformaciones que sucesivamente van extrayendo características de mayor nivel.

Mirbot utiliza una red neuronal con 21 capas convolutivas y una capa densa o fully connected final. En total la red tiene unos 7 millones de parámetros que han sido ajustados a partir de la base de datos ImageNet (1,2 millones de imágenes para entrenamiento clasificadas en 1000 clases).

Cada imagen se escala a una resolución de 224x224 píxeles y se realiza una pasada por la red. Con el vector de características obtenido se realiza una búsqueda del vecino más cercano en nuestra base de datos para encontrar el vector de características (previamente extraído y almacenado usando el mismo método) más similar. El sistema realiza un ranking de las clases a las que pertenecen las imágenes más similares y lo devuelve como resultado de la clasificación.

Tecnologías

Android SDK

Es el conjunto de herramientas proporcionado por Google que permite desarrollar aplicaciones para Android.

Proporciona herramientas de compilación, un emulador (basado en QUEMU) y una herramienta de comunicación (ADB) tanto con dispositivos reales como con emuladores, que permite instalar aplicaciones, grabar en vídeo la pantalla del dispositivo o conectar un depurador, entre otras cosas.

También proporciona plataformas de desarrollo de cada versión del sistema operativo. Cada plataforma contiene las APIs que esa versión es capaz de ejecutar, con su código fuente, documentación y ejemplos. También proporciona imágenes del sistema operativo que se ejecutan el emulador y para varias arquitecturas de procesador, como ARM, x86 o MIPS.

Por último, Google incluye en el SDK una serie de extras como la librería que permite acceso a los Google Play Services (ver sección), la Android Support Library (ver sección) o los pagos dentro de la aplicación utilizando Google Play.

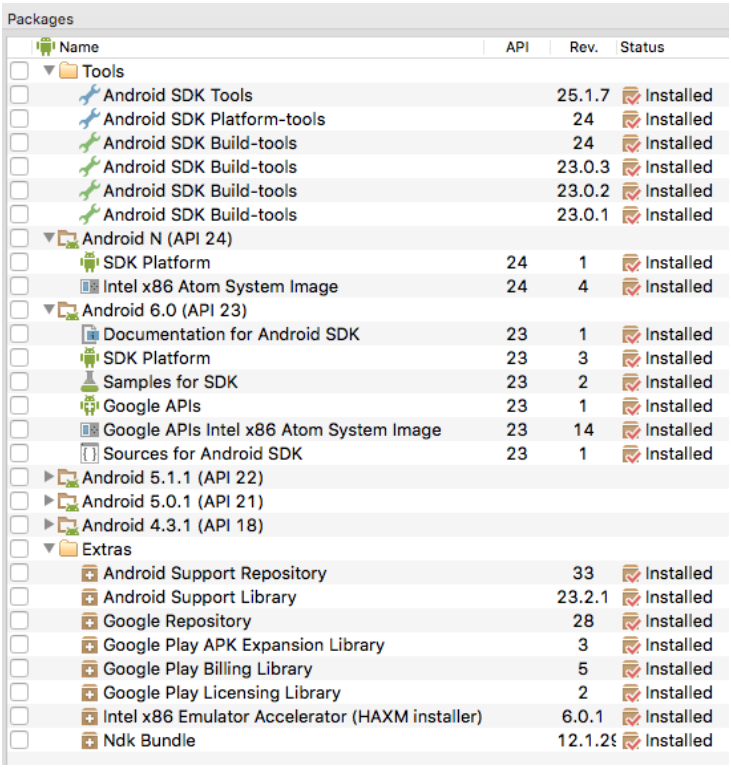

Ilustración 15 - Android SDK Manager

La mayoría de aplicaciones se ejecutan sobre una instancia de la máquina virtual de Android. Esta máquina virtual proporciona una abstracción y expone unas APIs que permiten a los desarrolladores acceder a las capacidades del sistema operativo.

El lenguaje principal para desarrollar aplicaciones sobre esta máquina virtual es Java, pero se puede hacer uso de cualquier otro lenguaje de programación que sea capaz de compilar al bytecode de la máquina virtual.

Si se desea, se pueden desarrollar aplicaciones que salgan de la máquina virtual utilizando el Native Development Kit (NDK) y el lenguaje de programación C++.

El SDK de Xamarin es una alternativa al Android SDK que permite programar utilizando las APIs de .NET para acceder al sistema operativo y el lenguaje C#. El runtime de Xamarin funciona en paralelo a la máquina virtual de Android y puede llamar a las APIs que expone esta máquina virtual mediante bindings.

Android Studio [8]

La aplicación ha sido desarrollada utilizando el entorno integrado de desarrollo (IDE) Android Studio. Publicado por Google, es la opción recomendada para nuevos desarrollos de aplicaciones para Android y sustituye a la combinación del IDE Eclipse de la Eclipse Foundation con el plugin Android Development Tools (ADT) de Google. Está basado en el IDE IntelliJ IDEA de la empresa JetBrains.

Ilustración 16 - Logo de Android Studio

Es un IDE relativamente joven, pero con un alto ritmo de desarrollo, siendo presentado oficialmente en el Google IO 2014 en versión beta (0.5.x) y llegando a su primera versión estable en diciembre del mismo año (1.0). Actualmente se encuentra en la versión 2.1.2 en el canal Stable y 2.2 en el canal Canary.

El IDE ofrece, entre otras, las siguientes características:

- Editores de para los lenguajes Java, Groovy, C++ y XML con resaltado de sintaxis, formateado y autocompletado de código.
- Editor visual de layouts.
- Depurador de Java y C++, con posibilidad de utilizar ambos de forma simultánea y transparente.
- Una vista de navegación los ficheros del proyecto personalizada para proyectos Android. Destaca la agrupación de los scripts de Gradle y configuración de los distintos módulos en un nodo de primer nivel, facilitando su gestión. También la agrupación de las distintas variantes de cada fichero fuente, manifest o de recursos en un mismo nodo.
- Actualizaciones automáticas, tanto del IDE como del Android SDK, NDK, herramientas y librerías varias.
- Administrador de dispositivos virtuales.
- Gestión de builds mediante Gradle.
- Emulador de terminal integrado en el IDE.
- Control de versiones mediante CVS, Subversion, Git o Mercurial y función de historial local para todos los ficheros de texto.
- Integración con la plataforma de desarrollo de aplicaciones Firebase de Google.

- Integración con Google Cloud Platform.
- Sistema de plugins para extender las capacidades del IDE, por ejemplo, el plugin de Crashlytics. Esta es una de las mejores características de IntelliJ IDEA y de hecho, la mayoría de las características mencionadas están disponibles para IntelliJ IDEA ya que han sido desarrolladas como plugins.

Android Support Library [9]

Cada nueva versión de Android ha ido añadiendo nuevas características, APIs, componentes, elementos de la interfaz, lenguajes de diseño de UX, etc. a la plataforma.

En Android, la responsabilidad de actualizar los dispositivos recae en los fabricantes, y ya sea por las limitaciones de los propios dispositivos a la hora de hacer funcionar las nuevas versiones o por decisiones de tipo económico, una gran parte de los dispositivos no tienen acceso a las últimas versiones de Android.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.9%
4.1.x	Jelly Bean	16	6.8%
4.2.x		17	9.4%
4.3		18	2.7%
4.4	KitKat	19	31.6%
5.0	Lollipop	21	15.4%
5.1		22	20.0%
6.0	Marshmallow	23	10.1%

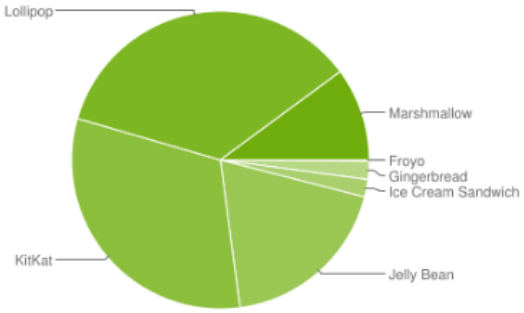

Ilustración 17 - Distribución de las versiones de Android en los dispositivos con acceso a Google Play Store

Debido a esto, los desarrolladores deben hacer frente al dilema de no poder aprovechar las nuevas capacidades de la plataforma para que su aplicación pueda llegar al máximo número de dispositivos.

En respuesta a esta situación de fragmentación de la plataforma, Google decidió publicar la Android Support Library, una librería que ofrece versiones compatibles hacia atrás de varios componentes de la plataforma.

Un ejemplo a destacar es el componente Fragment, que apareció en la versión de API 14, similar en funcionalidad a una activity y que facilita la composición de la aplicación para adaptase a diferentes factores de forma. Actualmente la support library permite utilizar este componente en dispositivos desde la versión de API 4. Incluso para las versiones que sí disponen del componente Fragment nativo, la support library ofrece las nuevas características que se le han ido añadiendo, como la posibilidad de anidar Fragments (API 17).

También proveen elementos de la UI que utilizan los estilos y patrones recomendados para Android. Por ejemplo, la librería appcompat proporciona una serie de componentes y widgets que permiten dotar de estilo Material Design a la aplicación y es compatible con dispositivos desde la versión de API 7. De hecho, algunos widgets clave hoy en día como ViewPager, RecyclerView, CardView, DrawerLayout, FloatingActionButton, SnackBar, etc. han sido desarrollados directamente para la support library y no tienen equivalente nativo.

Por último, también proporcionan utilidades para gestionar automáticamente la compatibilidad hacia atrás. Por ejemplo, los métodos de la librería relacionados con permisos pedirán al usuario que acepte cierto permiso si el dispositivo tiene una versión compatible con el nuevo sistema de permisos o comprobará si el permiso está definido en el manifest (y, por lo tanto, fue aceptado al instalar la app) en caso contrario.

No todas las nuevas características han podido ser implementadas. Por ejemplo, las llamadas a onActivityResult no se propagan en los fragments anidados de la librería de soporte, pero sí en los nativos.

La support library en realidad es un conjunto de librerías que están divididas por criterios de versión que soporta y las funcionalidades que incorpora, de manera que los desarrolladores solo tienen que incluir las que necesiten.

En la aplicación MirBot se han utilizado las siguientes librerías de la support library:

- Appcompat, para proporcionar estilo Material Design a la app. Es compatible con dispositivos desde la versión 7 del API. También ofrece soporte para utilizar vector drawables en dispositivos desde la versión 9 del API. Esta librería depende de la v4 support library, de la que se utilizan los Fragment, las clases de utilidad, y el widget DrawerLayout, el cajón que se despliega desde la parte izquierda de la app y contiene la navegación por las secciones principales de la misma.
- RecyclerView, elemento de la interfaz para dibujar eficientemente colecciones de objetos en forma de listas y grids sin tener que inflar widgets para los elementos no visibles en pantalla. Es compatible con dispositivos desde la versión 7 del API.
- CardView, elemento de la interfaz para mostrar información en forma de tarjetas. Es compatible con dispositivos desde la versión 7 del API.
- GridLayout, versión compatible hacia atrás del widget GridLayout utilizado para la tabla de la pantalla de estadísticas. Es compatible con dispositivos desde la versión 7 del API.
- Design, que ofrece un conjunto de componentes, widgets y layouts utilizados para implementar varios patrones de Material Design, como el FloatingActionButton.
- Preference, para proporcionar estilo Material Design a los ajustes de la aplicación. Es compatible con dispositivos desde la versión 7 del API. También posee una versión para utilizar desde la versión 14 del API utilizando fragments nativos.
- Percent, que incorpora una mejora de RelativeLayout y Framelayout en la que se puede definir el tamaño de los elementos contenidos en ellos mediante porcentajes.

Google Play Services Client Library [10]

Sobre la base open source de Android, Google aporta varias funcionalidades privativas, como mapas, almacenamiento en nube, autenticación, notificaciones push, etc. relacionadas con los servicios web de la empresa. Para poder disfrutar de estas funcionalidades los dispositivos deben tener instalado el APK de Google Play Services, que es necesario para hacer funcionar la mayoría de las Google Apps que están disponibles para Android.

De un tiempo a esta parte se ha acusado a Google de privatizar varios componentes de Android integrándolas en los Google Play Services, por ejemplo, las mejoras en el proveedor de ubicación. Posibles argumentos para estas acciones son el poder ofrecer dichas funcionalidades a pesar de la fragmentación de la plataforma y aumentar el control de Google sobre la misma, de manera que los fabricantes que no quieran suscribir un acuerdo con Google para instalar Google Play Store y las Google Apps no tengan acceso a determinadas funcionalidades clave.

Google provee a los desarrolladores de una librería cliente para comunicarse con el servicio en segundo plano de Google Play Services y utilizar sus funcionalidades en la aplicación. La librería es un conjunto de librerías divididas en funcionalidades o servicios determinados, por lo que para el desarrollador solo es necesario agregar las partes que utiliza en la aplicación.

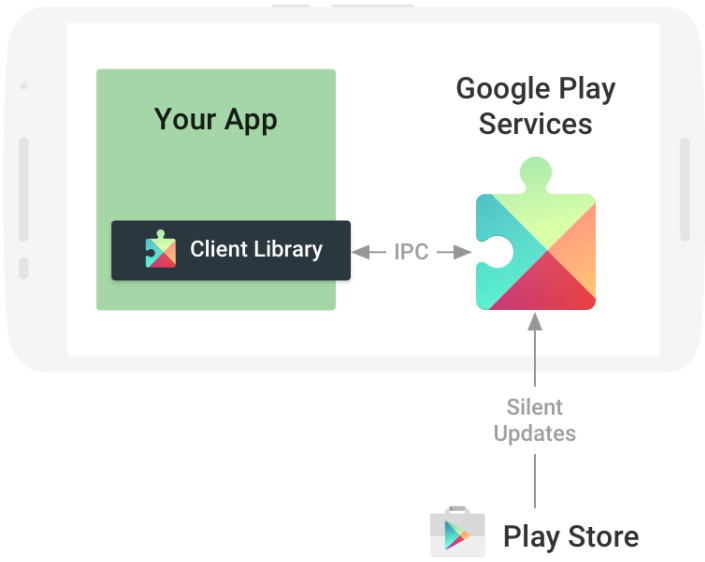

Ilustración 18 - Esquema de funcionamiento de Google Play Services

Para los servicios que lo requieren, permite al desarrollador obtener autorización del usuario para utilizar las funcionalidades de uno o varios servicios de Google en su nombre.

En MirBot se utiliza el proveedor combinado de ubicación de Google Play Services para obtener la ubicación donde se ha tomado la imagen y poseer más datos para su clasificación. El proveedor es compartido por varias aplicaciones por lo que cuenta con ventajas como que la frecuencia de actualización de la ubicación es alta y se ahorra batería al no haber múltiples aplicaciones accediendo al GPS de diversas maneras. El único requisito para utilizar esta API es contar con uno de los dos permisos de ubicación de Android: ACCESS_COARSE_LOCATION o ACCESS_FINE_LOCATION.

Glide [11]

Una imagen vale más que mil palabras, por lo que las imágenes son elementos comunes en la mayoría de las aplicaciones, ya que transmiten información y sensaciones que no se pueden conseguir de otra forma. Gran parte de las imágenes se encuentran almacenadas en internet, por lo que las aplicaciones deben ser capaces de obtenerlas y mostrarlas.

En el caso de los dispositivos móviles, que gran parte del tiempo están conectados a redes en las que se paga por los datos descargados, los desarrolladores deben ser especialmente sensibles a la hora de utilizar la menor cantidad de datos posibles para que los usuarios estén satisfechos con la aplicación.

También puede haber ocasiones en los que la red esté saturada y la velocidad de descarga se resienta, o la cantidad de imágenes a descargar sea muy grande, y la aplicación no debe quedarse bloqueada a la espera de que las imágenes se obtengan, es decir, las imágenes se deben obtener en segundo plano y mostrarse cuando estén listas, y en ese intervalo mostrar un indicador visual al usuario de que la descarga está teniendo lugar.

Por otra parte, los distintos factores de forma de los dispositivos móviles hacen que una imagen de tamaño fijo no se vea igual en todos, por lo que habrá que aplicar transformaciones (redimensionar, recortar, etc.) para su correcta visualización.

Ilustración 19 - Logo de la librería Glide

Glide es una librería que afronta todos estos desafíos. Está desarrollada por la empresa Bumptech y su uso está recomendado por Google. Ofrece:

- Descarga en segundo plano de imágenes, pudiendo mostrar en el contenedor de la imagen un placeholder mientras la descarga tiene lugar. En el caso de que la imagen no se pueda descargar, se puede aportar una imagen para indicar esa situación. También permite configurar una animación para ejecutar cuando la imagen se ha terminado de procesar y se va a pintar en el contenedor.
- Aplicar transformaciones a la imagen para ajustarla al tamaño del contenedor como redimensionado o recorte. También permite escribir transformaciones personalizadas para, por ejemplo, realizar un recorte circular de la imagen.
- Cacheo de imágenes en memoria/disco, de manera que si la aplicación solicita la imagen nuevamente no se tiene que volver a descargar de internet. En el caso de que se le hayan aplicado transformaciones a la imagen se puede configurar para cachear tanto la imagen original como el resultado de la transformación.

- Proporciona una API sencilla y fluida para realizar las peticiones, de manera que descargar y pintar una imagen resulta tan sencillo como:

```
Glide.with(context)
     .load(url)
     .centerCrop() // Recorta y centra
     .placeholder(R.drawable.loading_spinner) // Imagen mientras la descarga se está produciendo
     .crossFade() // Animación que se ejecutará al pintar la imagen descargada
     .into(myImageView) // Contenedor de la imagen
```

Glide no está limitado a la descarga de imágenes de internet y se puede utilizar para cargar imágenes que estén en los recursos de la propia aplicación o incluso en otras aplicaciones si se le proporciona un identificador de recurso con permisos de lectura para la aplicación.

En MirBot, Glide se utiliza para cargar todas las imágenes de la aplicación, tanto las que pertenecen al usuario y son descargadas de internet como las distintas imágenes del robot que están en los recursos locales. Las imágenes estáticas referenciadas como recursos en un layout son cargadas en memoria antes de aplicar las transformaciones pertinentes para ajustar su tamaño antes de pintar, en cambio Glide solo almacena en memoria la imagen transformada, por lo que el impacto en la memoria es significativamente menor.

Dagger2 [12]

La inyección de dependencias consiste en invertir el control en la resolución de dependencias, es decir, proveer las dependencias de un objeto en lugar de dejar que este las construya por sí mismo. Esto, junto a la declaración de las dependencias como interfaces, ofrece ventajas como mayor modularidad y testeabilidad, ya que se pueden cambiar la implementación de las dependencias sin tener que reescribir el código del objeto dependiente.

Existen múltiples frameworks dedicados a la inyección de dependencias, como Guice [13] de Google o Dagger [14] de la empresa Square. Dagger2 [12] es una adaptación de Dagger cuya principal característica es que es el primero que realiza todo el proceso en tiempo de compilación.

El compilador de dagger2 se ejecuta como una etapa más del proceso de compilación de la aplicación, analiza el código buscando las anotaciones correspondientes a la inyección de dependencias y genera código de manera muy a como lo escribiría un desarrollador si lo tuviera que hacer manualmente.

Esta aproximación resuelve gran parte de los problemas que tienen los otros frameworks:

- El análisis y creación del grafo de dependencias es un proceso costoso por lo que trasladarlo de tiempo de ejecución a tiempo de compilación redunda en un menor tiempo de arranque de la aplicación.
- El análisis en tiempo de compilación produce que se detecten errores como dependencias duplicadas o dependencias circulares, que en tiempo de ejecución producen el cierre forzado de la aplicación.
- La generación de código en tiempo de compilación evita tener que utilizar reflexión para inyectar los objetos, con el consecuente aumento de rendimiento.

Para declarar los objetos que Dagger2 debe generar código para construir y satisfacer sus dependencias se utiliza la anotación `@Inject`. Si se aplica sobre el constructor de una clase Dagger2 generará código para crear instancias de esa clase y proveer los parámetros del constructor.

En el caso de las activities de Android el constructor no es accesible para el desarrollador, en este caso la anotación se puede aplicar en miembros (no privados) de la clase, pero será necesaria hacer una llamada explícita para realizar la inyección, como explicaremos más adelante.

En los casos donde `@Inject` es insuficiente o no adecuada, por ejemplo, no se puede crear una instancia de una interface o clases de terceros a las que no podemos aplicar anotaciones, se debe declarar un método al que se le aplica la anotación `@Provides` para definir como se debe satisfacer dicha dependencia. En el caso de una interface, devolverá una instancia de una clase que implemente la interface. En el caso de una clase de terceros, por ejemplo, el contexto de la aplicación, se obtendrá a partir de la instancia de la aplicación. Los métodos proveedores deben estar declarados dentro de módulos, que son clases a las que se le aplica la anotación `@module`. En estos casos además se puede definir un ámbito de uso para la generación del objeto devuelto. Por ejemplo, aplicar la anotación `@Singleton` en un método proveedor hará que ese método devuelva siempre la misma instancia (se genera una vez y se cachea). Se pueden definir anotaciones personalizadas de ámbito de uso.

Los grafos de objetos se almacenan en componentes, que son interfaces a las que se le aplica la anotación `@Component`, que toma como atributos los módulos y/u otros componentes que son necesarios en el componente. Los componentes que utilicen módulos con métodos a los que se le haya aplicado una anotación de ámbito de uso deben anotarse con la misma anotación. Los objetos a los que se quiera hacer accesibles a través del componente se declaran como funciones sin parámetros cuyo tipo de retorno es el del objeto. Aquí también se declaran las llamadas explicitas mencionadas en el caso de las activities como métodos cuyo único parámetro es la activity. Dagger2 generará una clase que implementa la interfaz y que contiene todo lo necesario para generar los objetos y para inyectarlos (si se ha declarado un método en un módulo, pero no se utiliza en ningún punto de la aplicación no se generará código para ese método).

Los componentes se han de construir en el código de la aplicación, por lo que el ámbito del grafo de objetos está supeditado al objeto que contenga el componente. Un componente construido dentro de una activity existirá (y ocupará memoria) solo mientras exista dicha activity. Así, se puede configurar la inyección de dependencias con múltiples grafos de objetos que tengan solo las dependencias concretas para ese determinado ámbito de actuación con lo que se consigue un consumo de memoria eficiente.

En MirBot se ha creado una estructura de tres componentes correspondientes a los ámbitos de aplicación, activity y fragment, donde los objetos que se proveen con cada componente están adecuados a su ámbito; y 5 módulos basados en criterios de ámbito y similaridad de los objetos que proveen relacionados con la arquitectura de la aplicación, que veremos más adelante.

Butter Knife [15]

Una de las tareas más repetitivas como desarrollador de Android es obtener las referencias a los objetos de la jerarquía de vistas inflados en las activities, fragments, custom views, etc. Este proceso se realiza de forma individual para cada objeto que se quiera obtener mediante una llamada al método `findViewById`, que devuelve objetos de tipo View, por lo que se debe realizar una conversión explícita al tipo de View concreto sí queremos utilizar los métodos específicos de ese tipo.

Ilustración 20 - Logo de ButterKnife

Butter Knife es una librería que sirve para aligerar todo ese proceso y eliminar de nuestro código fuente todas esas llamadas que no aportan realmente valor a la aplicación. Para obtener la referencia a una vista basta con aplicar la anotación `@InjectView` a un miembro no privado de la clase, especificando como atributo de la anotación el identificador de recurso de la vista. Además de vistas, la librería se puede usar para inyectar recursos (`@BindString`, `@BindDrawable`, etc.) y algunos listeners, como `@OnClick`, entre otras cosas.

Para realizar la inyección se ha de llamar a una de las sobrecargas del método estático `ButterKnife.bind`. En el caso de que la clase no sea del tipo `Activity` se ha de proporcionar la vista a partir de la cual se quieren obtener el resto de vistas. En el caso particular del ciclo de vida de los `Fragment`, en los que la jerarquía de vistas se puede destruir sin necesidad de destruir el propio `Fragment`, se deben liberar las referencias a los miembros inyectados mediante el método `unbind` del objeto `Unbinder` que devuelve `ButterKnife.bind` en el método `Fragment.onDestroyView`.

Al igual que Dagger2, Butter Knife realiza el proceso de análisis y generación de código en tiempo de compilación. Para cada clase que llame al método `ButterKnife.bind`, se generará un objeto que realice las llamadas al método `findBiewById`, realice las conversiones de tipos y asigne las vistas a los miembros anotados con `@InjectView`.

En MirBot se utiliza Butter Knife para la inyección de vistas y listeners `@OnClick` en las activities, fragments y view holders de los RecyclerView.

RxJava y RxAndroid

Ilustración 21- Logo de ReactiveX

ReactiveX [16] es una API para programación asíncrona para la creación y manipulación de flujos de datos. Combina las mejores ideas del patrón Observer, el patrón Iterator y programación funcional.

La base de funcionamiento consiste en un observador se suscribe a un observable, para después reaccionar ante los elementos o notificaciones que emite ese observable. Un observable puede tener varios observadores al mismo tiempo.

Cada observador reacciona ante tres eventos:

- Cuando el observable emite un nuevo elemento de la secuencia (`onNext`).
- Cuando el observable se encuentra con un error que el impide seguir emitiendo objetos (`onError`).
- Cuando el observable ha terminado de emitir objetos (`onCompleted`).

Un observador recibirá cero o más eventos de tipo `onNext`, antes de recibir un evento de tipo `onError` u `onCompleted` con lo que finalizará su trabajo.

El API facilita que los elementos emitidos por el observable se puedan transformar en otro tipo de elementos o combinar con elementos emitidos por otros observables antes de llegar al observador.

Esto se consigue mediante el uso de operadores, funciones que toman un como entrada y devuelven un observable como salida, por lo que se pueden encadenar de forma sencilla.

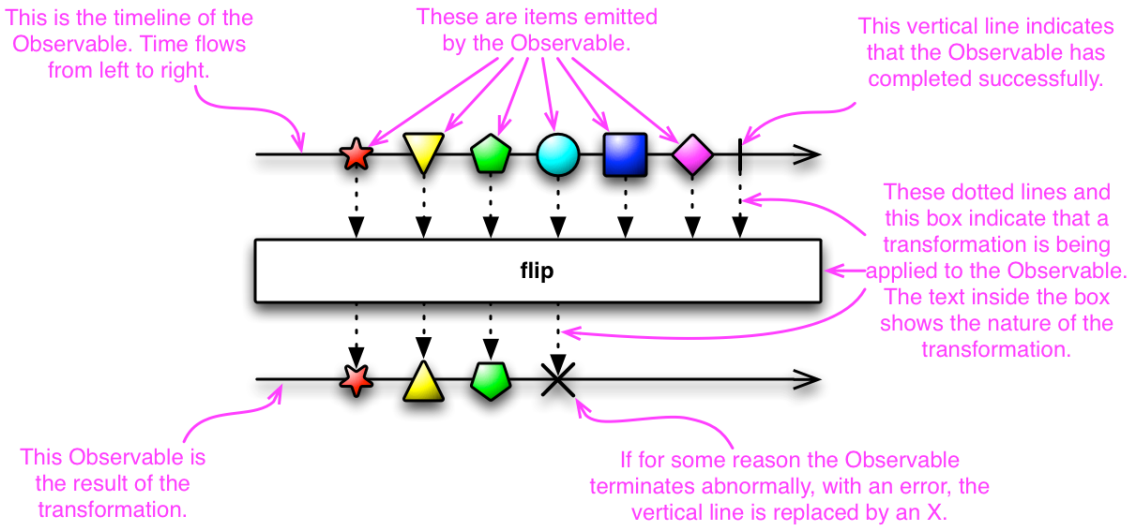

Ilustración 22 - Diagrama que representa un observable

Por defecto el observable se ejecuta y notifica a su observador en el mismo hilo en el que se realiza la llamada a subscribe. No obstante, mediante el uso de los operadores subscribeOn y observeOn, el observable puede ejecutarse en un hilo, cada operador en otro hilo distinto y finalmente notificar al observador en otro hilo distinto. El API permite al programador abstraerse de los problemas de creación de hilos, sincronización y concurrencia.

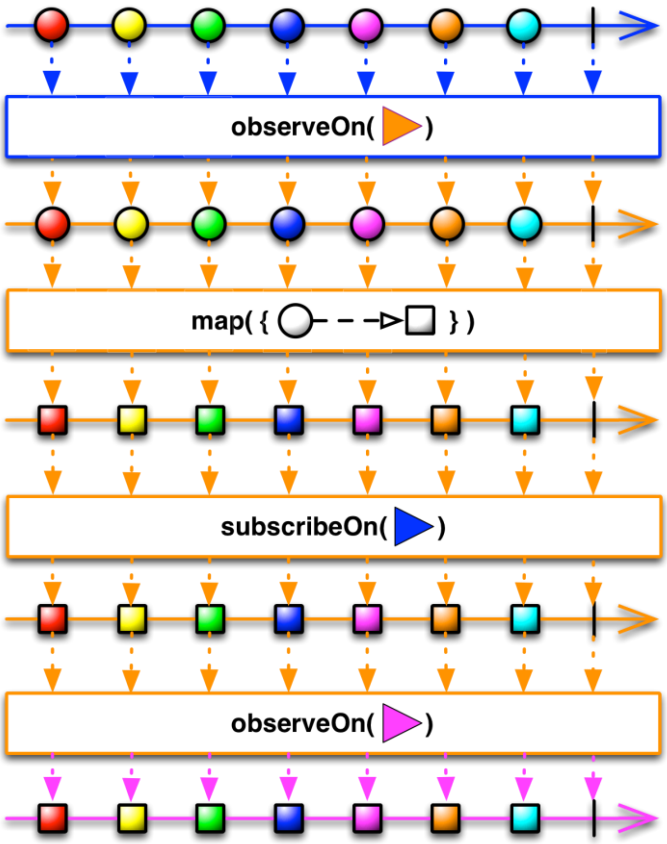

Ilustración 23- Diagrama de funcionamiento con varios hilos

RxJava es la implementación de ReactiveX en Java.

RxAndroid añade elementos específicos del funcionamiento de Android, como un acceso sencillo al scheduler del thread principal (o UI Thread) de la aplicación.

En Mirbot, se utiliza RxJava para la ejecución de los casos de uso, que conllevan una o varias llamadas a la API, en segundo plano que actualizan la UI, la cual no se bloquea, cuando reciben la respuesta.

Retrofit [17]

Muchos servicios web cuentan con una API REST para que usuarios u otros servicios puedan interactuar con ellos. Una API REST se basa en el uso del protocolo HTTP de manera que la URL define el recurso al que estamos accediendo y los verbos (GET, POST, PUT, DELETE, etc.) las acciones que realizamos sobre el recurso. Los servicios REST pueden retornar datos en múltiples formatos siendo los más utilizados JSON o XML. Es un sistema muy sencillo que sólo requiere de un cliente HTTP (disponibles en casi todos los lenguajes de programación y sistemas operativos) para su utilización.

Retrofit es una librería que facilita la creación de un cliente REST en Android. El desarrollador sólo debe definir una interface cuyos métodos representan los endpoints del servicio REST y la librería genera (en tiempo de ejecución) una implementación de esa interface que transforma los métodos en llamadas HTTP. Retrofit se encarga de:

- Generar la petición a la ruta adecuada.
- Convertir los parámetros del método al formato adecuado, por ejemplo, de Java a JSON.
- Realizar la llamada http de forma síncrona o asíncrona.
- Convertir la respuesta al formato adecuado, por ejemplo, de JSON a Java.

Los métodos de la interface se deben anotar con el verbo que se quiere utilizar en la petición (`@GET`, `@POST`, `@PUT`, `@DELETE`, etc.) y especificar la ruta (relativa o absoluta) del endpoint. Esta ruta puede contener parámetros, delimitados entre llaves, que se tomarán de los parámetros del método anotados con `@Path`. Se pueden añadir cabeceras HTTP a la petición utilizando `@Headers`. También se puede indicar que el cuerpo de la petición debe generarse en formato `@Multipart` o `@FormUrlEncoded`.

Los parámetros del método se anotan para definir su ubicación: `@Path` en la ruta, `@Header` en las cabeceras HTTP, `@Query` en el query string, en el cuerpo de la petición `@Body` en el cuerpo de la petición, `@Field` como campo de un `@FormUrlEncoded` o `@Part` como parte de un `@Multipart`.

Por defecto los parámetros deben ser de tipo RequestBody y el tipo devuelto ResponseBody. Se debe proveer un convertidor en el momento de generar el cliente para poder utilizar otro tipo de objetos. Retrofit provee varios conversores en forma de librerías adicionales, por ejemplo, el conversor a JSON utilizando Gson.

Por defecto los métodos de la interface deben retornar un objeto de tipo Call<T>, que es el encargado de realizar la petición de forma síncrona o asíncrona. Este comportamiento se puede modificar si proveemos un adaptador en el momento de generar el cliente. Retrofit provee varios adaptadores en forma de librerías adicionales, entre ellos uno compatible con RxJava que se utiliza en la aplicación MirBot para que las llamadas al servicio devuelvan observables.

MosbyMVP [18]

Es una librería escrita por el desarrollador Hannes Dorfmann cuya función es ayudar a construir aplicaciones Android utilizando el patrón MVP. El nombre de la librería fue elegido en honor del personaje Ted Mosby de la serie How I Met Your Mother (Como conocí a vuestra madre).

El núcleo de la librería está formado por las interfaces `MvpView` y `MvpPresenter<V extends MvpView>`, que implementarán las vistas y presenter de nuestra aplicación. También se proporcionan las clases abstractas `MvpBasePresenter<V extends MvpView>` y `MvpNullObjectBasePresenter<V extends MvpView>` en la que está encapsulada la funcionalidad básica de un presenter y de la que extenderán los presenter de nuestra aplicación.

A su vez, se proporcionan implementaciones por defecto de Activity, Fragment (de la support library) y ViewGroup con la lógica necesaria para gestionar el MVP, las cuales se pueden utilizar como clases base de las activities, fragments y viewgroups de nuestra aplicación.

La librería está basada en delegación, por lo que no es obligatorio utilizar las implementaciones por defecto, si no que podemos integrar la funcionalidad en nuestras propias clases base implementando las interfaces `BaseMvpCallback` y `ActivityBaseMvpCallback`, y delegando la funcionalidad en las clases que implementan `ActivityMvpDelegate`, `FragmentMvpDelegate` y `ViewGroupMvpDelegate`.

También ofrece unas implementaciones por defecto de Activity y Fragment para utilizar el flujo de trabajo Loading-Content-Error a.k.a. LCE, en el que se ejecuta una tarea en segundo plano que cargar contenido mientras se muestra un widget de progreso, para luego mostrar el contenido obtenido o una pantalla de error si la tarea falló. En este caso, nuestras vistas extenderán de `MvpLceView<M>`, debemos asignar unos ids específicos a las views correspondientes en el layout y llamar a los métodos `showLoading`, `showContent` y `showError` de la vista desde el presenter cuando corresponda.

La librería está preparada para tratar con los cambios de configuración, por ejemplo, un cambio de orientación, que provocan la recreación de la Activity o Fragment. El delegado utiliza los métodos `attachView` y `detachView` del presenter en el momento correcto del ciclo de vida por para que no se produzcan memory leaks por retención de la Activity y/o el Fragment. Además, los presenter pueden ser retenidos de forma que el estado del modelo no se pierda, tanto en activities como en fragments. En el caso de no ser retenidos el método detachView se puede sobrescribir para detener las tareas en segundo plano.

Además, la librería ofrece un módulo extra llamado ViewState cuya función es retener el estado de la vista ante los cambios de configuración, para, por ejemplo, pintar de nuevo el contenido ya cargado sin tener que solicitarlo nuevamente al presenter al girar la pantalla.

Crashlytics [19]

Crashlytics es un SDK para crash reporting para las plataformas móviles iOS y Android. Forma parte de la plataforma de desarrollo de aplicaciones móviles Fabric de la empresa Twitter, la cual es modular, por lo que se puede usar Crashlytics sin tener que integrar el resto de SDKs que ofrecen.

La integración de crashlytics es sencilla, añadir la dependencia en el build.gradle, establecer el Api Key en los metadatos del manifest e inicializar el SDK en el método onCreate de una clase que herede de Application o en la Activity que sirve como punto de entrada a la aplicación. Además, provee un plugin para Android Studio para que este proceso de integración se automatice.

Su funcionamiento en Android se basa en cambiar el manejador de excepciones no capturadas por defecto para todos los hilos por uno personalizado que genera un informe de la excepción y lo envía a los servidores de Crashylitics.

En este informe se incluye la excepción, la pila de excepciones anidadas y el stacktrace de todos los hilos de la aplicación. Además, se incluyen una serie de metadatos:

- Fecha y hora
- Información del hardware
    - Fabricante

    - Modelo
    - Orientación
    - Estado de la batería
    - Sensor de proximidad activado
- Información del software
    - Versión del sistema operativo
    - Versión de la aplicación
    - Cantidad de espacio libre en memoria
    - Cantidad de espacio libre en almacenamiento
    - Conectividad
    - Procesos en ejecución
    - Dispositivo rooteado
    - Aplicación en primer plano en el momento del crash

En la web disponemos de un dashboard donde gestionar todos los crashes, pudiéndolos filtrar por versión de la aplicación, tipo (fatales o no fatales) y periodo de tiempo. Podemos ver el detalle de cada error y marcarlo como cerrado una vez hayamos distribuido la corrección. Se nos ofrecen gráficas que nos permiten analizar qué errores están afectando más a los usuarios para poder tomar decisiones sobre la prioridad a la hora de resolverlos.

Arquitectura

Clean architecture

La arquitectura empleada para la aplicación es una adaptación de la Clean Architecture descrita por el desarrollador Uncle Bob en su blog [20].

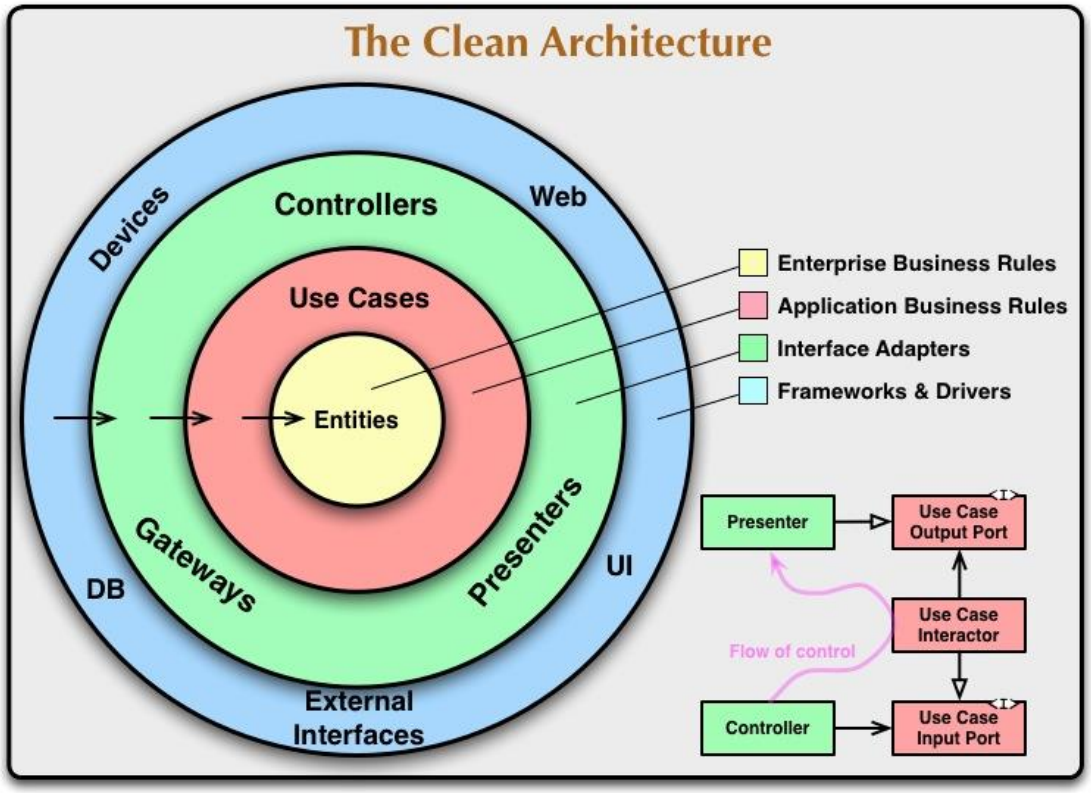

Ilustración 24 – Diagrama de la clean architecture

El objetivo de esta arquitectura es la separación de intereses o principios. Está separación se cumple dividendo el software en capas, que dotan a los sistemas construidos con esta arquitectura de:

- Independencia de frameworks: la arquitectura no es dependiente de ningún producto de terceros cuyas características nos aten a sus requisitos, por lo que podemos usar las librerías como herramientas, adaptándolas a nuestra aplicación y no al contrario.
- Testeabilidad: las capas inferiores son testeables independientemente de las capas superiores, por la lógica de negocio se puede testear sin necesidad de la interfaz de usuario, persistencia de datos, servidor web o cualquier otra herramienta externa.
- Independencia de la interfaz de usuario: podemos cambiar fácilmente la interfaz de usuario sin cambiar el resto del sistema, o utilizar varias interfaces de usuario a la vez, sin tener que cambiar la lógica de negocio.
- Independencia de la persistencia: las reglas de negocio no están atadas a la persistencia de los datos, por lo que podemos usar SQLServer, MySQL, SQLite, MongoDB o cualquier otro sistema de persistencia de datos, incluso combinarlos.

El nivel de abstracción del sistema es mayor cuanto más interior es la capa. Las capas interiores corresponden a las entidades de negocio y los casos de uso que operan con esas entidades, mientras que las más exteriores corresponden a las implementaciones concretas de la interfaz de usuario, la persistencia o las conexiones a servicios web, habiendo capas intermedias encargadas de convertir desde el formato adecuado para los casos de uso al formato concreto de la interfaz de usuario o la persistencia, y viceversa.

La regla de la dependencia es la que consigue que esta arquitectura funcione. Ésta dictamina que las dependencias a nivel de código fuente solo pueden apuntar hacia dentro. Nada que se encuentre en un círculo interior puede saber algo sobre lo que hay en un círculo exterior. Particularmente, algo declarado en un círculo externo no puede ser mencionado desde el código situado en un círculo interno. Eso incluye funciones, clases, variables o cualquier otra entidad software.

La comunicación entre capas se consigue mediante la aplicación del principio de inversión de dependencias: un circulo interior define una serie de interfaces (de entrada y de salida), de las que depende, y que son implementadas en círculos exteriores, por lo que el circulo interior no queda acoplado a los exteriores y se cumple la regla de la dependencia.

Como desventaja la implementación de esta arquitectura produce una cantidad superior de código fuente dedicado a lidiar con la comunicación entre capas y mantener la regla de la dependencia, y en ciertos casos puede resultar contraproducente.

Patrón Model-View-Presenter

Para la capa de presentación se ha elegido utilizar un patrón Model-View-Presenter a.k.a. MVP. Es un patrón de arquitectura utilizado para construir interfaces de usuario que, como en el caso de la clean architecture, tiene como objetivo la separación de intereses. La lógica de presentación es separada en tres elementos:

- Modelo: son los datos que se representarán en la vista y la lógica de negocio necesaria para su obtención y/o gestión.
- Vista: interfaz de usuario que representa los datos del modelo y enruta los comandos de usuario hacia el Presenter.
- Presenter: es el intermediario entre el modelo y la vista. Se encarga de procesar los eventos enviados por la vista, invocar la lógica de negocio necesaria para recuperar o manipular los datos del modelo y ordenar a la vista que se actualice en consecuencia.

En Android, se debe considerar a las activities y los fragments como elementos de la vista y no implementar la lógica del Presenter en ellos. Su responsabilidad en MVP es pintar la información en la interfaz de usuario y manejar los eventos de la UI, que desencadenarán llamadas a métodos del Presenter.

Estructura de la aplicación

En MirBot se ha tratado de adaptar la clean architecture a las particularidades del desarrollo Android, partiendo del ejemplo publicado por el desarrollador Fernando Cejas en su blog [X].

El proyecto consta de cinco módulos:

Módulo Domain

Este módulo implementa las dos primeras capas de la arquitectura. Contiene las entidades de negocio y los casos de uso a.k.a interactors.

Esté módulo no tiene constancia del resto de módulos de la aplicación, y es un módulo puramente Java, por lo que no tiene dependencia del SDK de Android tampoco. Se ha tomado la decisión de hacer una excepción a la regla con la librería RxJava, midiendo los pros y contras, tomándola como parte del core.

Se exponen dos tipos de fronteras para operar con otros módulos

- Las interfaces de los interactors. Cada interactor define una estructura de datos de entrada (o ninguna) en su método `execute()` y ofrece como salida un Observable de RxJava de elementos de alguna de las entidades de negocio.
- Interfaces definidas siguiendo un patrón repositorio que se usan dentro de los interactors para obtener y persistir cambios en las entidades de negocio.

Módulo Repository

Este módulo implementa las interfaces del patrón repositorio definidas en el módulo domain, al cual tiene como única dependencia. En un primer momento era parte del módulo data, pero se decidió separarlo para conseguir un módulo puramente java. A su vez, implementa una serie de interfaces para conectarse con las fuentes de datos.

Módulo Data

Este módulo implementa las interfaces de fuentes de datos definidas en el módulo repository, al cual tiene como dependencia. Es el encargado de comunicarse con el API de Mirbot, utilizando la librería Retrofit. Es un módulo de librería Android ya que es necesario utilizar el Context para obtener el id del usuario de las SharedPreferences o el ConectivityManager para obtener el tipo de red a la que está conectado el dispositivo.

Módulo Presentation

Este módulo contiene la lógica de presentación, formada por los presenter y las interfaces que definen las vistas del patrón MVP. Ejecuta los casos de uso definidos en el módulo domain, el cual tiene como dependencia. En un primer momento era un módulo puramente Java, pero cuando se decidió utilizar la librería MosbyMVP se cambió a módulo de librería Android.

Módulo App

Módulo principal de la aplicación. Se trata de un módulo de aplicación Android que contiene:

- La implementación de la interfaz de usuario. Activities, Fragments y CustomViews que implementan las vistas del MVP.
- La lógica del inyector de dependencias. Aquí se ubican los Component y Module de Dagger2 encargados de proveer las dependencias de todos los módulos.

Implementación

Elección del nivel de API mínimo

Al comenzar un nuevo proyecto Android Studio nos pregunta el nivel mínimo de API va a soportar nuestra aplicación.

En versiones recientes el IDE nos proporciona un diálogo de ayuda mediante un gráfico que muestra el porcentaje de dispositivos (según estadísticas de Google Play) compatibles con nuestra aplicación para cada nivel de API, así como todas las nuevas características que se añadieron a partir de ese nivel si pulsamos en uno de ellos.

Además, debemos tener en cuenta el nivel mínimo de API soportado por cada una de las librerías de terceros que vayamos a utilizar, ya que si es mayor al nuestro no las podremos utilizar directamente. Si bien es posible sobrescribir este comportamiento mediante el atributo `tools:overrideLibrary` en el Manifest de nuestra aplicación, nos obliga a estar comprobando el nivel de SDK del dispositivo cada vez que utilicemos la librería en nuestro código, ya que si se ejecuta en un nivel inferior lo más probable es que provoque que la aplicación deje de funcionar (crash).

El nivel de API mínimo elegido para la aplicación es el 15, el cual nos permite llegar al 97,4% de los dispositivos.

Inyección de dependencias

Como se mencionó en la sección de tecnologías y arquitectura, se ha empleado Dagger2 para implementar la inyección de dependencias. Las clases que se utilizan para componer el grafo de objetos están localizadas en el módulo App, que, al tener conocimiento del resto de módulos puede propagar la inyección a todos los módulos sin romper la regla de la dependencia.

Tenemos 5 módulos dónde se declaran las dependencias:

- `ApplicationModule`:
    - `Context` obtenido por el objeto `Application`. Se prefiere este contexto en operaciones que no afectan directamente a los widgets de la interfaz de usuario para no provocar memory leaks por mantener referencia a una Activity actuando como Context.
    - `SharedPreferences` de la aplicación. Para proveerlas a objetos que no disponen de `Context` sin tener que proveer el `Context`.
    - Un `ThreadPoolExecutor` personalizado utilizado para generar un `Scheduler` de RxJava donde ejecutar los casos de uso. Tiene un mínimo de 3 y un máximo de 5 hilos utilizables.
    - Un PostEjecutor que es una interfaz para pasar el Scheduler del hilo principal de la aplicación Android a módulos java.
    - Bus de eventos utilizado para pasar algunos eventos entre activities y fragments.
- `DataSourceModule`:
    - Implementaciones de las fuentes de datos que se inyectarán en las implementaciones de `Repository`.

    - `MirbotApiService`: el cliente REST para comunicarse con la API de MirBot creado mediante Retrofit.
- `RepositoryModule`: Implementaciones de `Repository` que se inyectarán en las implementaciones de `Interactor`.
- `InteractorModule`: Implementaciones de los casos de uso que se inyectarán en las implementaciones de `Presenter`.
- `PresenterModule`: Implementaciones de los `Presenter` que se inyectarán en las activities y fragments.

Todas las dependencias especificadas en los módulos son Singleton, es decir, se proveerá el mismo objeto siempre que se solicite por el mismo grafo.

Tenemos 3 componentes encargados de generar los grafos de objetos:

- `ApplicationComponent`: está atado al ciclo de vida de una `Application`. Los objetos obtenidos serán los mismos durante todo el tiempo de vida de la aplicación. Provee los objetos de los módulos Application, Repository y DataSource. Expone los objetos de los módulos Application y Repository a los componentes que extiendan este grafo.
- `ActivityComponent`: está atado al ciclo de vida de una `Activity`. Un nuevo grafo se generará en el caso de que la actividad sea recreada en un cambio de configuración. Extiende el grafo de `ApplicationComponent`, provee objetos de los módulos Presenter e Interactor y expone los `Presenter` que deben ser inyectados en activities.
- `FragmentComponent`: equivalente del anterior, pero atado al ciclo de vida de un `Fragment`. Si el `Fragment` se retiene no se generará un nuevo grafo.

Un único objeto `ApplicationComponent` se genera y puede ser obtenido desde el objeto de la clase `MirbotApp` que hereda de `Application`.

Se genera un objeto `ActivityComponent` o `FragmentComponent` en cada `Activity` o `Fragment` que necesite inyección de dependencias. La funcionalidad está encapsulada en las implementaciones de `ActivityInjectionDelegate` y `FragmentInjectionDelegate`. En estos delegados también se implementa la inyección de views de ButterKnife.

Se ha expuesto los `Presenter` en `ActivityComponent` o `FragmentComponent` para hacer más sencilla su integración con MosbyMVP, ya que el método que expone el `Presenter` será llamado dentro del método `createPresenter()` de la `MvpActivity` o `MvpFragment`. La alternativa sería generar métodos `inject()` y llamarlos en el método de ciclo de vida apropiado en cada `Activity` o `Fragment`.

Clases base de componentes Android

Se han generado una serie de subclases de Activity y Fragment para integrar la funcionalidad de MosbyMVP con la inyección de dependencias y views.

Así, tenemos `BaseActivity`, `BaseViewStateActivity` y `BaseLceViewStateActivity` que heredan de `MvpActivity`, `MvpViewStateActivity` y `MvpLceViewStateActivity`, respectivamente, obteniendo la funcionalidad de MosbyMVP. Estas clases contienen un `ActivityInjectionDelegate`, cuyo método `onContentChanged()` es llamado en el respectivo método de `Activity` para integrar la inyección de views de Butterknife; y cuyo método

`getActivityComponent()` es llamado en un método `getActivityComponent()` creado en estas activities.

En cuanto a los fragments, tenemos `BaseFragment` y `BaseLceViewStateFragment` que heredan de `MvpFragment` y `MvpLceViewStateFragment`. Contienen un `FragmentInjectionDelegate` cuyos métodos `onViewCreated()` y `onViewDestroyed()` son llamados en los respectivos métodos de Fragment para integrar la inyección de views de Butterknife; y cuyo método `getFragmentComponent()` es llamado en un método `getFragmentComponent()` creado en estos fragments.

Un caso especial es `BasePreferenceFragment` ya que no existe una clase por defecto de MosbyMVP para `PreferenceFragment`. No obstante, debido a que MosbyMVP se basa en composición y delegación, es tan simple con copiar el código de `MvpFragment` dentro de este nuevo Fragment que hereda de `PreferenceFragmentCompat`, además de lo realizado en el resto de fragments.

También se ha creado una clase `BaseViewHolder` que hereda de `RecyclerView.ViewHolder` y en cuyo constructor se llama a `Butterknife.bind()` para implementa la inyección de views.

Vector Drawables y Glide

`VectorDrawable` es un nuevo tipo de `Drawable` introducido en Android Lollipop (API 21) que permite utilizar imágenes construidas a través de gráficos vectoriales, cuyas ventajas son su correcta visualización en cualquier resolución a la vez que se ahorra en espacio de almacenamiento al no tener que proporcionar imágenes en distintas resoluciones para que se visualicen correctamente en los distintos dispositivos. La app de MirBot utiliza Material Icons [21] en su forma vectorial, pero ya que estos no son compatibles con nuestro nivel mínimo de API, Gradle posee una tarea que genera imágenes PNG a partir de los vectores durante la compilación para que la aplicación se ejecute sin problemas.

La versión 23.2 de la AppCompat Support Library trajo la compatibilidad con los vectores a la versión 7 del API. Para utilizarlos hay que utilizar el flag `vectorDrawables.useSupportLibrary = true` en el fichero build.gradle de los módulos en los que utilicemos gráficos vectoriales, de manera que la tarea de generar imágenes PNG es obviada.

Para referenciar una imagen vectorial en un layout debemos utilizar el atributo `srcCompat` en los widgets que heredan de `AppCompatImageView` o `AppCompatImageButton`. En los ficheros de los menús no es necesario hacer ningún cambio.

Glide no es capaz de obtener un `Drawable` a partir de un identificador de recurso de una imagen vectorial en versiones anteriores a la 21, por lo que en lugar de identificadores de recursos pasaremos directamente drawables de tipo `VectorDrawableCompat` mediante la llamada al método estático `VectorDrawableCompat.create()` con el identificador de recurso.

Imágenes y memoria

La memoria disponible para cada aplicación Android es limitada, y uno de los elementos que más memoria utiliza son las imágenes. En una aplicación que maneja un gran número de imágenes es fácil que se llegue a desbordar el heap y producir un OutOfMemoryError, lo que provoca que la aplicación deje de funcionar.

Cada imagen ocupa en memoria 4 (rgba) x altura (píxeles) x anchura (píxeles) bytes.

Las imágenes del usuario obtenidas por la aplicación son cargadas a través de Glide, a las cuales aplicamos una transformación para ajustarse al tamaño del View que la contiene antes de cargarlas en memoria, por lo que el impacto es moderado, ya que en la mayoría de los casos su tamaño se ve reducido al del contenedor.

Además, al utilizar `RecyclerView`, cada vez que uno de los hijos es reciclado la referencia a esa imagen queda liberada por lo que el recolector de basura puede eliminarla si necesita memoria.

En cambio, cuando referenciamos una imagen PNG o JPEG desde los recursos de la aplicación en el atributo `android:src` de un `ImageView`, este cargará la imagen original en memoria, incluso si le aplicamos alguna transformación mediante el atributo `android:scaleType`. Esto puede suponer un gran impacto en memoria dependiendo de la resolución original de las imágenes, llegando a provocar el temido `OutOfMemoryError`.

Para solucionar este problema, debemos evitar cargar las imágenes (excepto si son vectoriales) en los layouts, y en su lugar cargarlas desde código, aplicando una transformación para ajustarla al tamaño del contenedor antes de cargarla en memoria. En la aplicación de Mirbot se utiliza Glide en estos casos también por simplicidad y homogeneidad.

Envío de las imágenes utilizando Retrofit

Como hemos visto en la sección de tecnologías, es muy sencillo generar un cliente REST del API de Mirbot utilizando Retrofit. Basta con definir una interface con métodos que corresponden a las llamadas del API con las anotaciones correspondientes. Además, utilizamos el adaptador de RxJava para poder utilizar Observable como tipos de retorno, siendo el origen de los observables que devuelven los Interactor.

Los tipos devueltos se han modelo utilizando objetos Java que replican el formato del objeto JSON correspondiente, por lo que la asignación es instantánea para, a continuación, aplicarle un operador `map()` de RxJava que los convierte a las entidades de negocio correspondientes.

El único caso más complejo de modelar es el envío de imágenes, ya que si no se hace correctamente el servidor no será capaz de interpretar la imagen enviada.

La definición de la llamada en la interface es la siguiente:

```
@Multipart
@POST("user/{hash}/images/classify")
Observable<RestClassifyImageResult> classifyImage(@Path("hash") String user,
                                    @Part MultipartBody.Part image,
                                    @Part("onlyuser") RequestBody onlyUser,
                                    @Part("categories") RequestBody categories,
                                    @Part("metadata") ClassifyMetadata metadata);
```

Una llamada anotada con `@Multipart` generará un objeto `MultipartBody`, que hereda de `RequestBody`, a partir de los parámetros anotados con `@Part`. Estos parámetros pueden ser de tipo `MultipartBody.Part`, `RequestBody` o un objeto que los conversores utilizados en la instancia de Retrofit, en nuestro caso Gson, sean capaces de transformar en un `RequestBody`.

El método `RequestBody.create(final MediaType contentType, final File file)` nos proporciona un objeto `RequestBody` a partir de la imagen. Sin embargo, esto no es suficiente y si utilizamos este objeto como parámetro la imagen no se subirá correctamente.

El método `Part.createFormData(String name, String filename, RequestBody body)` se encarga de marcar esta parte como si estuviéramos enviando la imagen mediante un formulario HTML, de manera que pasándole el nombre del fichero y el objeto `RequestBody` obtenido con el método anterior, la imagen se envía con un formato que el servidor es capaz de interpretar.

Aspecto final de la aplicación

A continuación, se incluyen algunas capturas de las distintas pantallas de la aplicación donde se puede comprobar el resultado final obtenido.

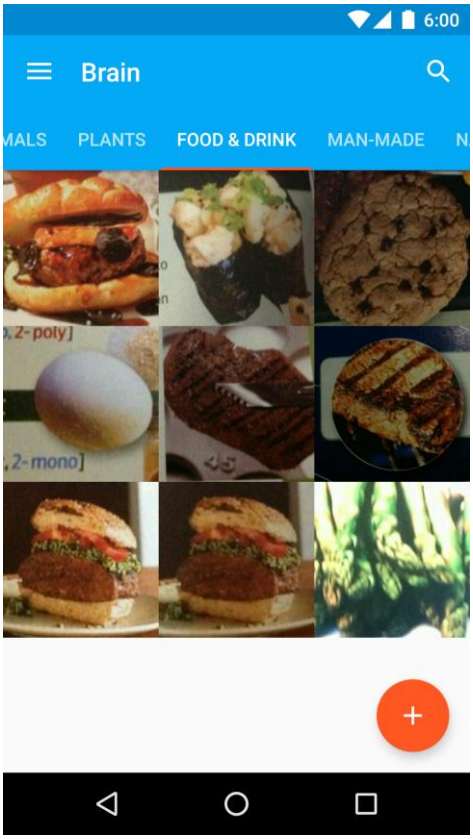

Ilustración 25 - Pantalla principal de la aplicación

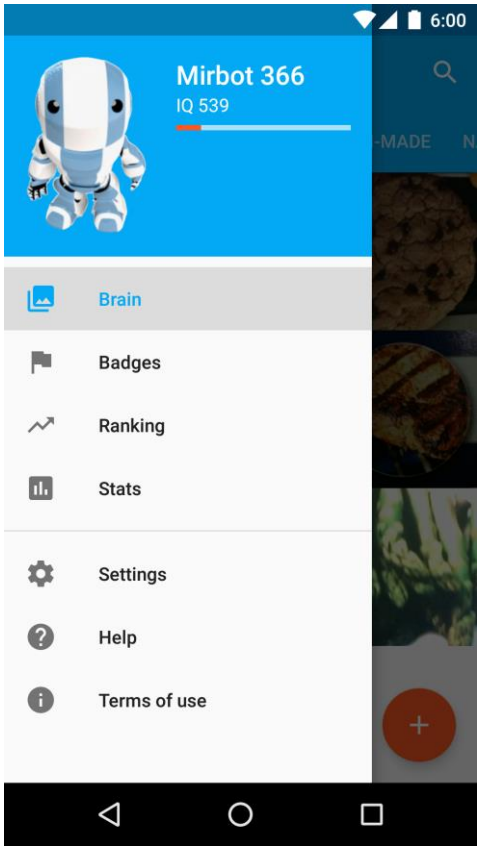

Ilustración 26 - Cajón de navegación abierto

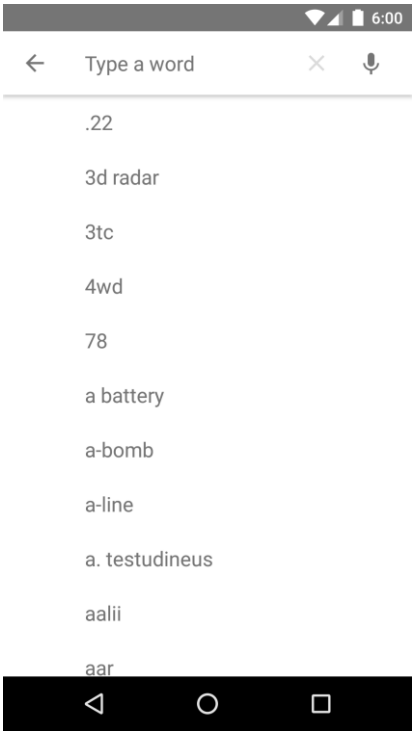

Ilustración 27 - Pantalla de búsqueda (estado inicial)

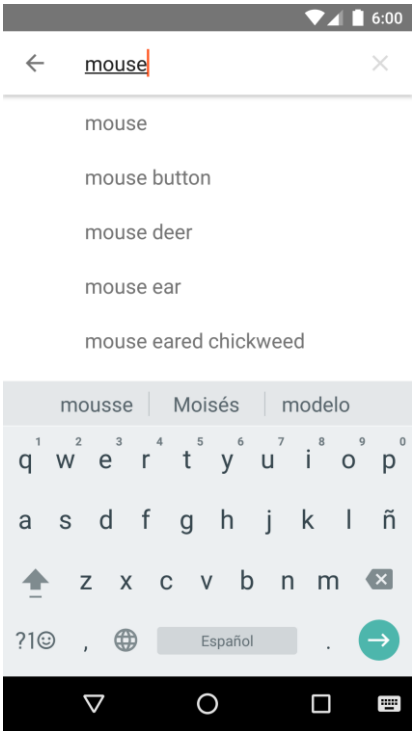

Ilustración 28 - Pantalla de búsqueda (término de búsqueda mouse)

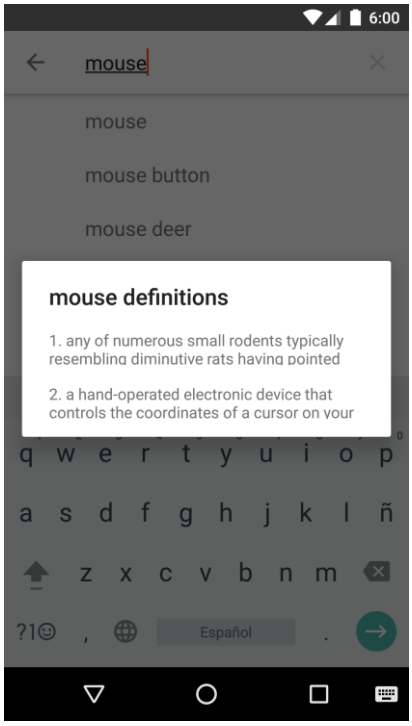

Ilustración 29 - Pantalla de desambiguación

Ilustración 30 - Pantalla de resultados de búsqueda

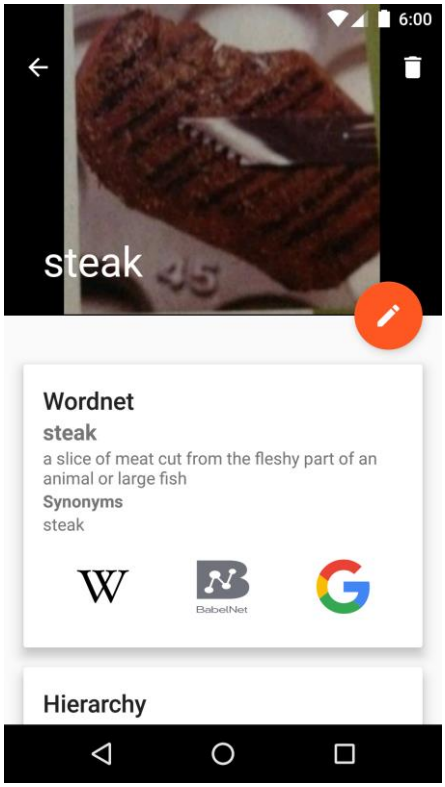

Ilustración 31 - Pantalla detalle

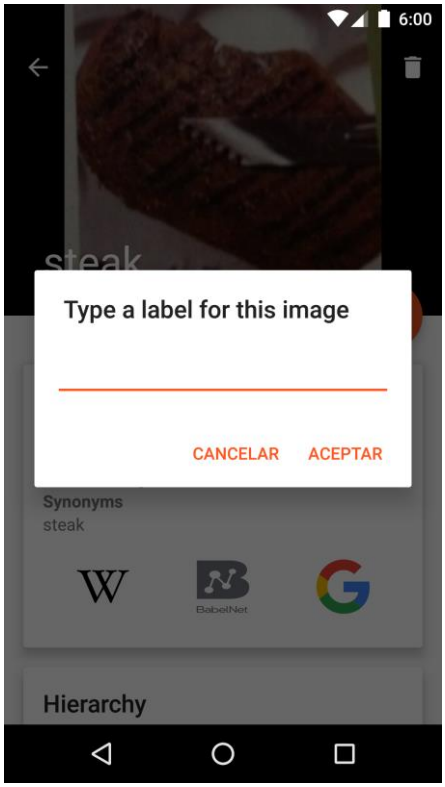

Ilustración 32 - Dialogo etiquetar

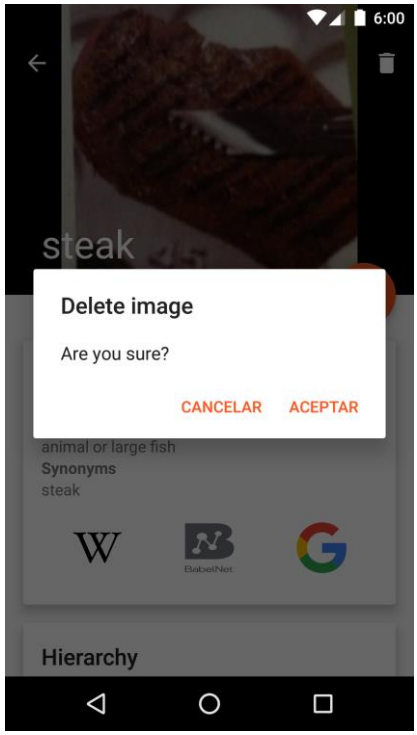

Ilustración 33 - Diálogo de eliminar

Ilustración 34 - Pantalla de la cámara

Ilustración 35 - Pantalla de selección del área de interés (ROI)

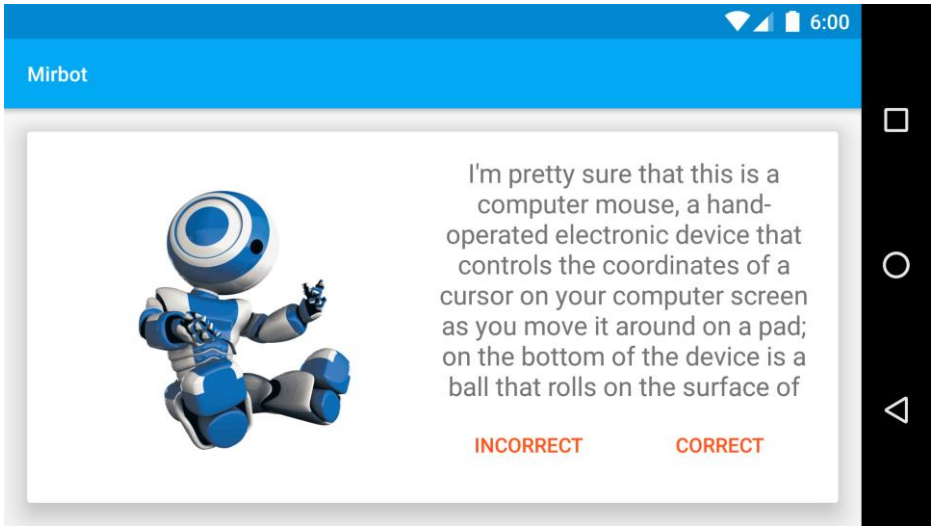

Ilustración 36 - Pantalla de primera opción (horizontal)

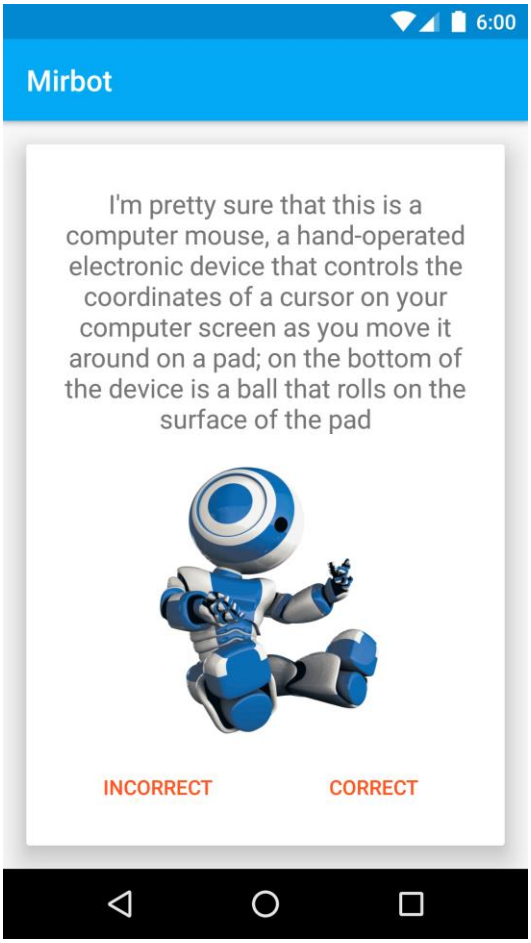

Ilustración 37 - Pantalla de primera opción (vertical)

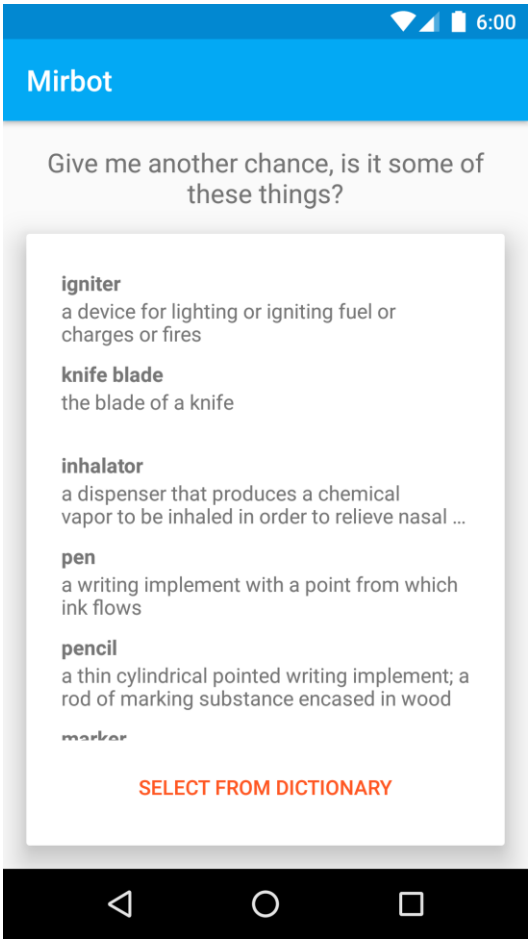

Ilustración 38 - Pantalla de alternativas (vertical)

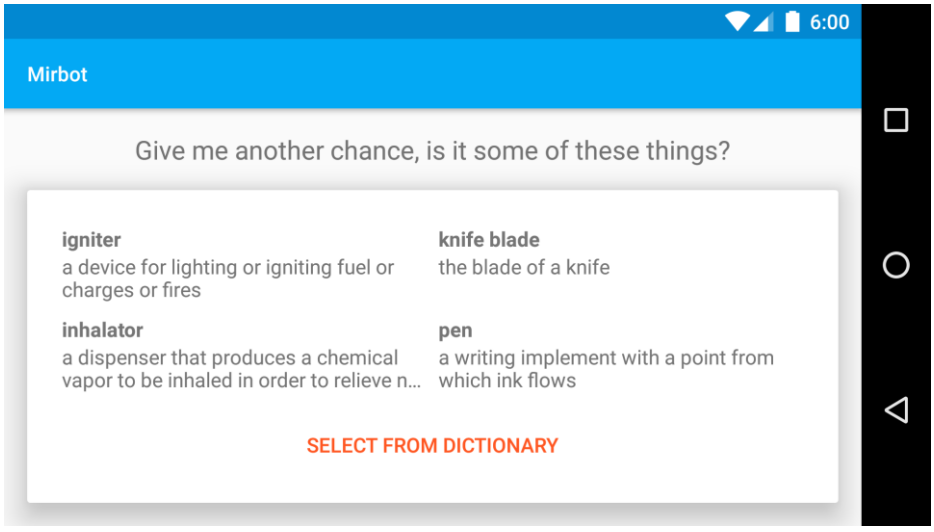

Ilustración 39 - Pantalla de alternativas (horizontal)

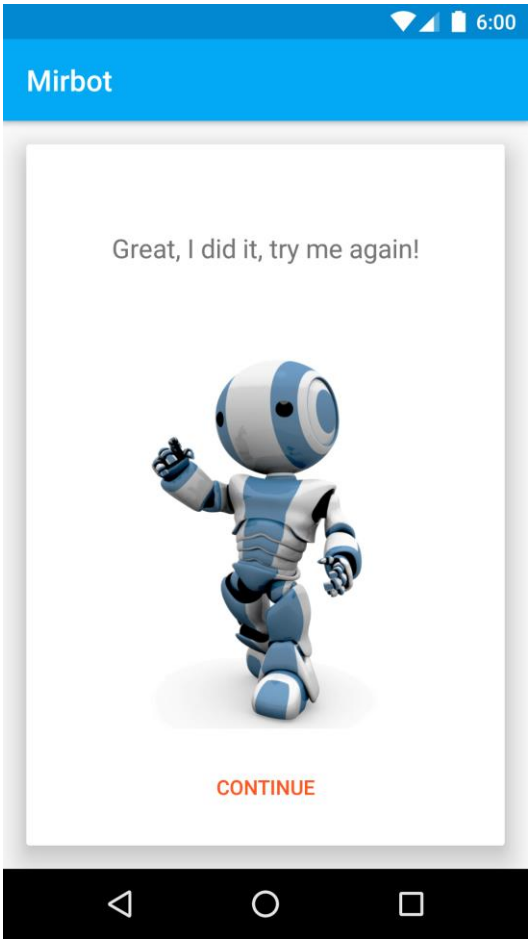

Ilustración 40 - Pantalla final de clasificación (vertical)

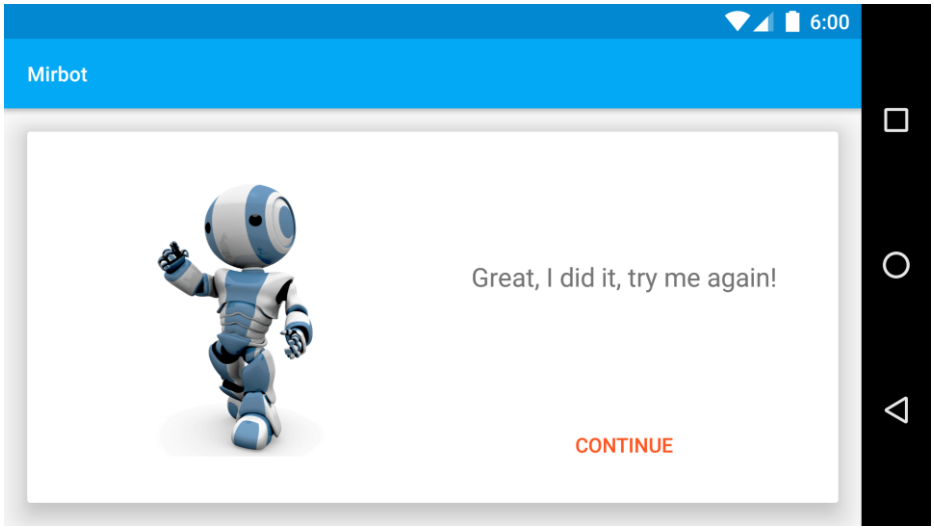

Ilustración 41 - Pantalla final de clasificación (horizontal)

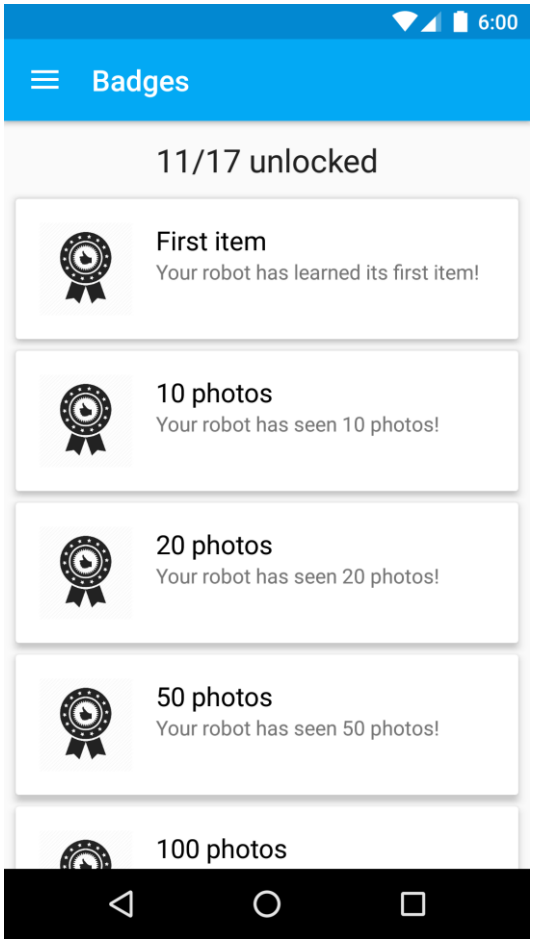

Ilustración 42 - Pantalla de medallas (vertical)

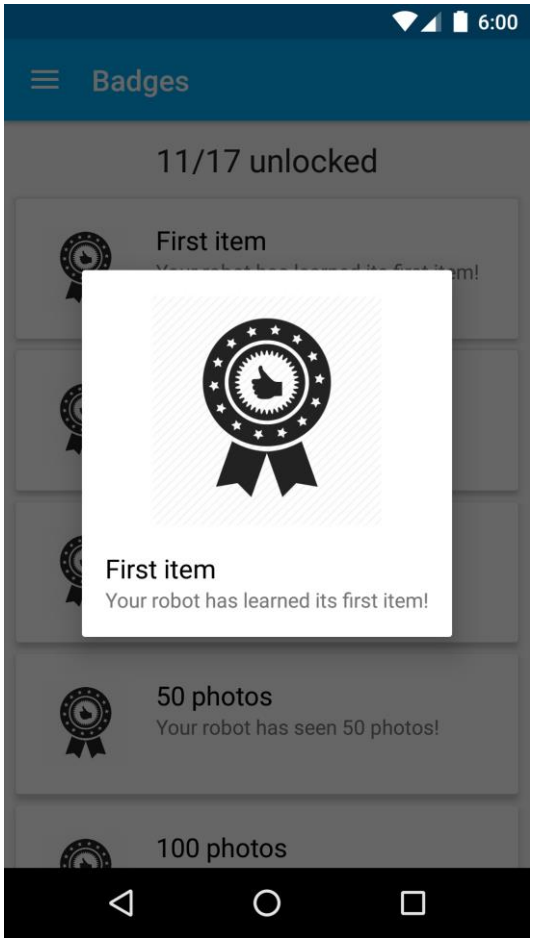

Ilustración 43 - Diálogo detalle de medalla

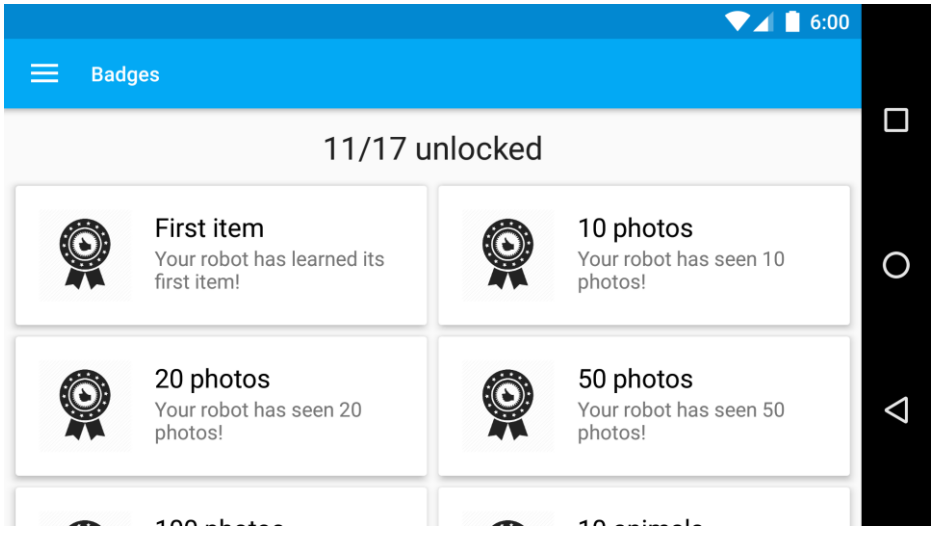

Ilustración 44 - Pantalla de medallas (horizontal)

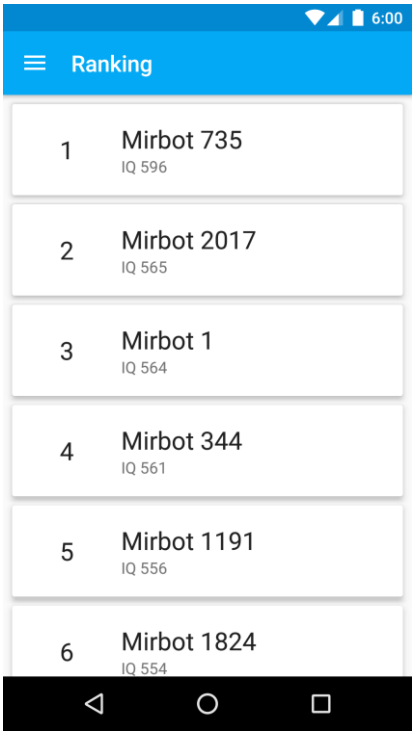

Ilustración 45 - Pantalla de ranking

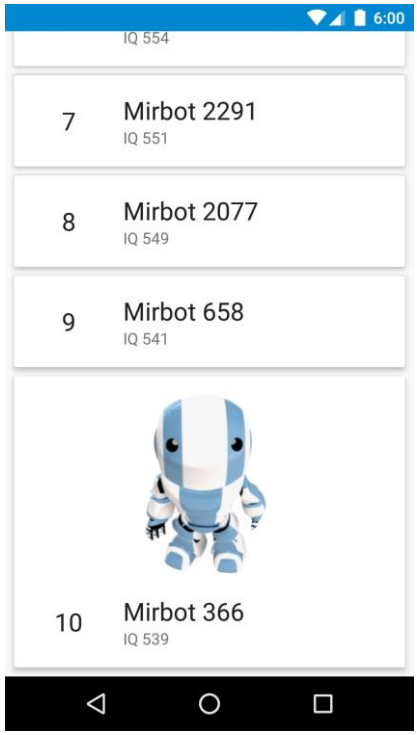

Ilustración 46 - Robot del usuario en la pantalla de ranking

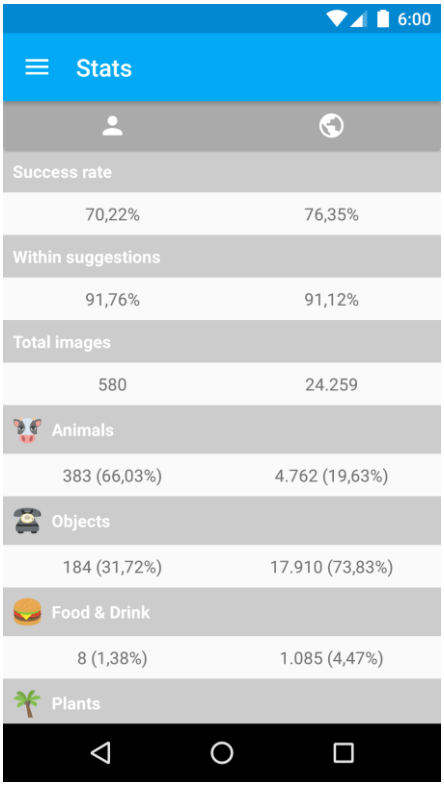

Ilustración 47 - Pantalla de estadísticas

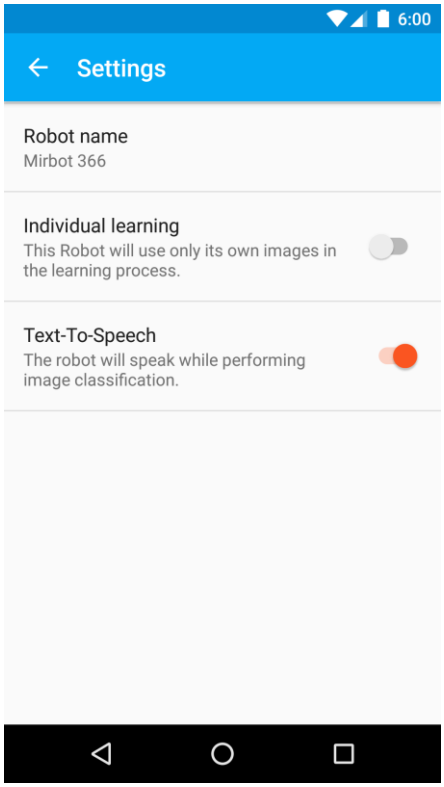

Ilustración 48 - Pantalla de ajustes de la aplicación

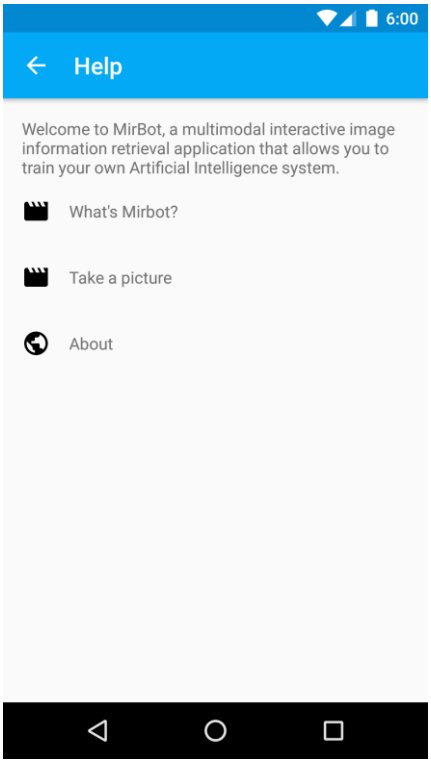

Ilustración 49 - Pantalla de ayuda

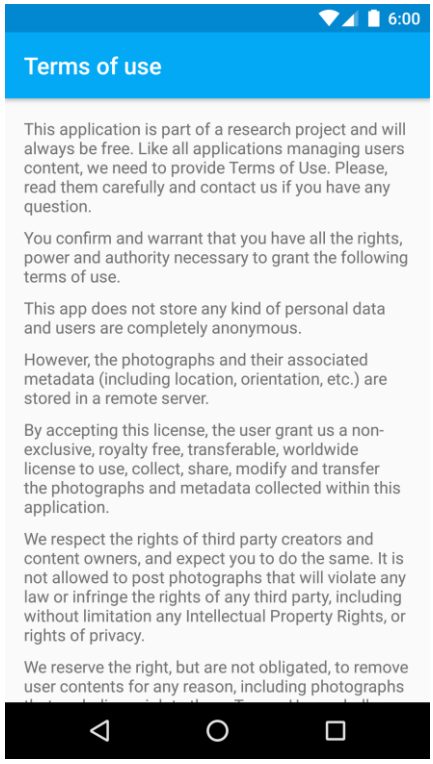

Ilustración 50 - Pantalla términos de uso

Conclusiones y trabajo futuro

Se han cumplido los objetivos propuestos en la sección de introducción. Hemos obtenido una aplicación para Android que reproduce toda la funcionalidad del cliente de iOS, que son clasificar nuevas imágenes, gestionar la colección de imágenes y ver las estadísticas, además de añadir los componentes de gamificación basados en progresión, medallas y ranking.

El ciclo de desarrollo finalizará con la publicación de la app en Google Play en los próximos días.

La relación de objetivos cumplidos se puede ver de una forma más detallada en la siguiente tabla:

Objetivo	Cumplido
Permitir al usuario capturar imágenes a través de la cámara del dispositivo.	Sí
Permitir al usuario seleccionar el área de la imagen que se utilizará en la clasificación.	Sí
Recabar metadatos de la captura: aceleración, orientación, gps, metatados EXIF de la imagen, etc.	Sí
Enviar la imagen al servicio para su clasificación.	Sí
Proporcionar una interfaz para que el usuario interactúe con los resultados de la clasificación, simulando una conversación entre el usuario y el robot, donde se permite al usuario elegir una de las clases sugeridas por el robot o elegir una clase del diccionario WordNet.	Sí
Permitir al usuario gestionar su colección de imágenes: navegar por la colección, filtrar, ver detalles, etiquetar, eliminar, etc.	Sí
Permitir al usuario revisar su progreso y compararlo con respecto al resto de usuarios: consultar IQ del robot, ranking de robots, estadísticas, etc.	Sí

El desarrollo de este proyecto ha servido de aprendizaje de la magnitud y complejidad de desarrollar una aplicación completa, desde sus esbozos hasta la publicación.

También para descubrir, analizar y utilizar una serie de tecnologías y librerías de terceros que facilitan el desarrollo, permitiendo poner el foco en las características de la aplicación, y patrones de arquitectura y diseño que repercuten en que el código de la aplicación esté bien estructurado, sea legible y facilite su mantenimiento y ampliación.

De cara a futuro se plantean las siguientes mejoras:

- La selección de la clase correcta durante la clasificación se podría hacer más interactiva con un sistema de tarjetas apiladas, que contienen las clases con su definición, en el que se acepta o se descarta haciendo swipe hacia los bordes de la pantalla. En caso de

descartar todas se le enviará al usuario a elegir una al diccionario (que estará accesible a partir del primer descarte).

- Buscar las imágenes con una etiqueta concreta.
- Etiquetas en el cajón de navegación como en la app de Gmail.
- Utilizar la API Camera2 en dispositivos con API 21 o superior.
- Añadir transiciones entre pantallas y animaciones para mejorar la experiencia de usuario.
- Utilizar un carrusel para poder navegar por las imágenes desde la pantalla de detalle sin tener que volver a la pantalla de la lista.
- Mejoras internas de arquitectura.

Bibliografía y referencias

[1] «Google Goggles,» [En línea]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.unveil&hl=es.

[2] «CamFind,» [En línea]. Available: https://play.google.com/store/apps/details?id=com.msearcher.camfind&hl=es.

[3] «Blippar,» [En línea]. Available: https://play.google.com/store/apps/details?id=com.blippar.ar.android&hl=es.

[4] «Shot & Shop,» [En línea]. Available: https://play.google.com/store/apps/details?id=com.shotnshop.android&hl=es.

[5] «Laravel, the PHP framework for web artisans,» [En línea]. Available: https://laravel.com.

[6] «Mirbot, a multimodal image information retrieval project,» [En línea]. Available: http://www.mirbot.com.

[7] «WordNet, a lexical database for English,» [En línea]. Available: https://wordnet.princeton.edu.

[8] «Android Studio, IDE para el desarrollo de aplicaciones Android,» [En línea]. Available: https://developer.android.com/studio/index.html.

[9] «Android Support Library,» [En línea]. Available: https://developer.android.com/topic/libraries/support-library/index.html.

[10] «Google Play Services,» [En línea]. Available: https://developers.google.com/android/guides/overview.

[11] «Glide, an image loading and caching library for Android focused on smooth scrolling,» [En línea]. Available: https://github.com/bumptech/glide.

[12] «Dagger2, a fast dependency injector for Java and Android,» [En línea]. Available: http://google.github.io/dagger/users-guide.html.

[13] «Guice, a lightweight dependency injection for Java 6 and above,» [En línea]. Available: https://github.com/google/guice.

[14] «Dagger, a fast dependency injector for Java and Android,» [En línea]. Available: http://square.github.io/dagger/.

[15] «Butter Knife, field and method binding for Android views,» [En línea]. Available: http://jakewharton.github.io/butterknife/.

[16] «ReactiveX, an API for asynchronous programming with observable streams,» [En línea]. Available: http://reactivex.io/.

[17] «Retrofit, a type-safe HTTP client for Android and Java,» [En línea]. Available: http://square.github.io/retrofit/.

[18] «Mosby, Model-View-Presenter library for Android,» [En línea]. Available: http://hannesdorfmann.com/mosby/.

[19] «Crashlytics, the most powerful, yet lightest weight crash reporting solution,» [En línea]. Available: https://fabric.io/kits/android/crashlytics.

[20] «The Clean Architecture,» [En línea]. Available: https://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html.

[21] «Material Icons,» [En línea]. Available: https://design.google.com/icons/.

[22] «Architecturing Android… The clean way?,» [En línea]. Available: http://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/.

Anexos

API de MirBot

En este anexo se incluye una descripción de las llamadas de la API REST proporcionada por el servidor de MirBot.

GET /api/v1/user/{hash}/stats

Estadísticas de un usuario. Devuelve las estadísticas del propio usuario y las estadísticas globales.

Respuesta


```
{
	"user_stats":
	{
		"total_num_images": 22,
		"success_rate": 89.05,
		"between_suggestions_rate": 95.71,
		"type_animal": 3,
		"type_object": 12,
		"type_food_drink": 4,
		"type_plant": 3,
		"percentage_animal": 13.64,
		"percentage_object": 54.55,
		"percentage_food_drink": 18.18,
		"percentage_plant": 13.64
	},
	"global_stats":
	{
		"total_num_images": 1380,
		"success_rate": 86.53,
		"between_suggestions_rate": 90.41,
		"type_animal": 280,
		"type_object": 570,
		"type_food_drink": 253,
		"type_plant": 277,
		"percentage_animal": 20.29,
		"percentage_object": 41.3,
		"percentage_food_drink": 18.33,
		"percentage_plant": 20.07
	}
}
```

GET /api/v1/user/{hash}/ranking

Ranking a partir del IQ de los robots. Devuelve un objeto con el ranking de los N primeros (en la variable “top”) y el ranking del robot del usuario (en la variable “user”).

Parámetros

- rank_size: parámetro opcional para indicar el tamaño (N) del ranking. Por defecto se devuelven los 10 primeros. Como mínimo se pueden solicitar 5 elementos y como máximo 100. En caso de superar el máximo o el mínimo no se obtendrá un error, sino que se devolverá el máximo del rango permitido.

Respuesta


```
{
	"top": [
		{
			"robot_name": "name550c05b4992dd",
			"robot_iq": "215",
			"rank": "1"
		},
		{
			"robot_name": "name550c05b4c9e46",
			"robot_iq": "209",
			"rank": "2"
		},
		{
			"robot_name": "name550c05b4ae890",
			"robot_iq": "203",
			"rank": "3"
		}
	],
	"user":
	{
		"robot_name": "name550c05b4e2ed6",
		"robot_iq": "120",
		"userid": "550c05b4e2e4b",
		"rank": "26"
	}
}
```


GET /api/v1/user/{hash}/badges

Listado con las medallas o premios obtenidos por el usuario. Devuelve un objeto con el listado de todas las medallas posibles (en la variable “badges_list”) y un array con las medallas que ha obtenido el usuario (variable “user_badges”).

Respuesta


```
{
	"badges_list": [
		{
			"key": "first_item",
			"title": "First item",
			"description": "First item classified",
			"description_disabled": "Take a photo to ...",
			"image": "http://url_to_image",
			"image_disabled": "http://url_to_image_disabled"
		},
		{
			"key": "10_pictures",
			"title": "10 pictures",
			"description": "Description...",
			"description_disabled": "Description disabled...",
			"image": "http://url_to_image",
			"image_disabled": "http://url_to_image_disabled"
		},
		{
			"key": "20_pictures",
			"title": "20 pictures",
			"description": "Description...",
			"description_disabled": "Description disabled...",
			"image": "http://url_to_image",
			"image_disabled": "http://url_to_image_disabled"
		}
	],
	"user_badges": [ "first_item",  "10_pictures" ]
}
```


GET /api/v1/user/{hash}/robot

Información sobre el robot del usuario: nombre, IQ y max IQ posible.

Respuesta


```
{
"robot_name": "Mirbot",
"robot_iq": 50,
"robot_iq_max": 700
}
```


PUT /api/v1/user/{hash}/robot

Actualizar el nombre del robot. En caso de no existir todavía el usuario se creará e inicializará con el nombre de robot indicado.

Parámetros

- name: String con el nuevo nombre a asignar.

Respuesta

true

GET /api/v1/user/{hash}/labels/{class?}

Devuelve una lista de todas las etiquetas del usuario (a cada imagen se le puede asignar una etiqueta). Además, permite pasar un parámetro opcional "class" para obtener solamente las etiquetas de una determinada clase.

Respuesta

["label1", "label2", "label3", "label4"]

GET /api/v1/user/{hash}/images

Devuelve un listado paginado con todas las imágenes del usuario.

Parámetros

- page: número de página a obtener, desde 1 hasta N. Si no se le pasa ninguna página por defecto se devolverá la primera.
- class: parámetro opcional para filtrar las imágenes de una sola clase.
- subtree: parámetro booleano opcional que ha de usarse de forma conjunta con el parámetro “class”. Si se indica como valor “true” devolverá como respuesta todas las imágenes del subárbol con raíz en la clase indicada por “class”. Por ejemplo, si se filtra por la clase correspondiente a perro se devolverían todas las imágenes de esa clase y de todas sus subclases.
- direction: parámetro opcional que permite indicar el orden de los resultados. Admite dos valores: “asc” o “desc”, que devolverían el listado de imágenes ordenado por fecha de forma ascendente o descendente respectivamente. Valor por defecto: “desc”.
- page_size: parámetro opcional para indicar el número de imágenes a devolver en cada página. Por defecto se devuelven 100 elementos. Como mínimo se pueden solicitar 30 elementos y como máximo 150. En caso de superar el máximo o el mínimo no se obtendrá un error, sino que se devolverá el máximo del rango permitido.

Respuesta


```
{
"page": "1",
"from": 0,
"to": 100,
"images": [
{
"id": "217",
"label": "label1",
"url": "http://mirbot.com/image217.jpg",
"class_id": "109214760"
},
{
"id": "218",
"label": "label2",
"url": "http://mirbot.com/image218.jpg",
"class_id": "111780589"
},
{
"id": "219",
"label": "label3",
"url": "http://mirbot.com/image219.jpg",
"class_id": "112307076"
},  ...
]
}
```


Notas

- Al consultar la última página (o al pedir una página que no existe) devolverá un array vacío en “images”.

Ejemplos

- Consulta de la página 7 sin filtrar por clase: /api/v1/user/{hash}/images?page=7
- Consulta de la página 5 indicando el orden y el número de elementos por página: /api/v1/user/{hash}/images?page=5&page_size=75&direction=asc
- Página 2 filtrando por una clase: /api/v1/user/{hash}/images?class={class}&page=2
- Obtener todas las imágenes del subárbol con raíz en la clase “class”: /api/v1/user/{hash}/images?class={class}&subtree=true

GET /api/v1/user/{hash}/images/{imgid}

Información sobre una imagen.

Respuesta


```
{
"id": "1191",
"class_id": "107790400",
"label": "label550c061cbb3ee",
"url": "http://mirbot.com/image1191.jpg"
}
```

PUT /api/v1/user/{hash}/images/{imgid}/label

Actualizar la etiqueta de una imagen.

Parámetros

- label: String con la nueva etiqueta.

Respuesta

true

DELETE /api/v1/user/{hash}/images/{imgid}

Elimina la imagen indicada (y todos sus datos asociados).

RESPONSE

true

POST /api/v1/user/{hash}/images/classify

Clasificar una nueva imagen. Este método permite clasificar una imagen buscándola entre las imágenes del propio usuario o entre todas las imágenes de la base de datos. Además, también permite filtrar por categoría.

Parámetros

- image: fichero con la imagen en formato JPG a clasificar.
- onlyuser: Posibles valores: “YES”, “NO”. Permite indicar si la clasificación se ha de realizar solamente entre las imágenes del usuario (onlyuser=“YES”) o con todas las imágenes de la base de datos (onlyuser=“NO”).
- categories: Posibles valores: 5, 13, 6, 17, 20. Permite realizar la búsqueda filtrando por categorías. Si se desea buscar dentro de todas las categorías se tendría que enviar categories=“5,13,6,17,20”; o si por el contrario se desea buscar dentro de una o dos categorías por separado podemos utilizar categories=“5”; o categories=“5,20”. Estos identificadores de categorías se refieren a:
    - 5 – Animal
    - 6 – Objetos
    - 13 – Comida o bebida
    - 17 – Objetos naturales
    - 20 – Plantas
- metadata: array con todos los metadatos asociados a la imagen y el estado del dispositivo y sus sensores en el instante de la captura de la imagen. Este campo se puede enviar como un array de datos por POST normal o como un objecto JSON (ejemplo: {"osversion":"500", "model":"Android", "mirbotversion":"1", …}). Para más información consultad la sección “Metadatos soportados”.
- num_results: parámetro opcional que nos permite indicar el número máximo de resultados que nos devolverá el algoritmo.

Respuesta


```
{
	"imageID": 29954,
	"sure_threshold": "0.20",
	"known_threshold": "0.30",
	"classification": [
		{
		"image": "-1",
		"class": "-1",
		"score": "-1",
		"label": "",
		"lemma": ""
		}
	]
}
```


Donde

- imageID: Identificador temporal de la imagen enviada a clasificar. Este ID se ha de utilizar en la llamada al método "confirm" de la API para confirmar la clase de la imagen.
- sure_threshold: Umbral a partir del cual se considera que el sistema está seguro de la respuesta.
- known_threshold: Umbral a partir del cual se considera que las respuestas son similares a la imagen buscada.
- classification: Listado con las 10 imágenes más similares a la enviada a clasificar. Para cada resultado se incluyen los siguientes valores:
    - image: identificador de la imagen encontrada como más similar a la query.
    - class: clase sugerida por el sistema como similar a la imagen de búsqueda.
    - score: distancia entre el resultado y la imagen de búsqueda. Los posibles valores de distancia son [0, 1], siendo 0 imágenes completamente iguales.
    - label: En caso de que el resultado contuviese una etiqueta especificada por el mismo usuario se devolverá también. Esto nos permitirá decir, por ejemplo: Creo que el objeto de la imagen es un libro, similar a tu libro "Don Quijote de la Mancha".
    - lemma: Lemma de la clase surgerida. Nota: en caso de no encontrar resultados similares se devolverá un resultado con valor "-1", como en el ejemplo de respuesta.

POST /api/v1/user/{hash}/images/{imgid}/confirm

Confirmar la clase de una imagen. Al enviar una imagen a clasificar esta se almacena en una tabla temporal a la espera de que el usuario valide la categoría. Tras confirmar la clase de una imagen esta se añadirá a la base de datos global y a las imágenes del usuario.

Parámetros

- class: String con el identificador de la clase validada por el usuario.

Respuesta

```
{
	“similarity”:
	{
		"similarity_level": 60,
		"similarity_lemma": "Animal"
	},
	“new_badges”: [
		{
		"key": "10_pictures",
		"title": "10 pictures",
		"description": "Description...",
		"image": "http://url_to_image"
		}
	]
}
```

Donde

- similarity_level: porcentaje de similitud (entre 0 y 100) entre la sugerencia del sistema y la clase correcta validada por el usuario.
- similarity_lemma: mayor lemma o clase común en la jerarquía del árbol entre la sugerencia y la clase validada.
- new_badges: array con las nuevas medallas obtenidas tras la clasificación.

GET /api/v1/wordnet

Devuelve el diccionario Wordnet paginado.

Parámetros

- page: número de página a obtener, desde 1 hasta N. Si no se le pasa ninguna página por defecto se devolverá la primera.
- search: parámetro opcional que permite realizar búsquedas en el diccionario.
- page_size: parámetro opcional para indicar el número de imágenes a devolver en cada página. Por defecto se devuelven 100 elementos. Como mínimo se pueden solicitar 50 elementos y como máximo 250. En caso de superar el máximo o el mínimo no se obtendrá un error, sino que se devolverá el máximo del rango permitido.

Respuesta

```
{
	"page": "190",
	"from": 18900,
	"to": 19000,
	"lemmas": [ "garden spider", "garden strawberry", "garden symphilid",
	"garden tool", "garden trowel", "garden truck", "garden violet",
	"garden webworm", "gardener's delight", "gardener's garters",
	"gardenia", "gardenia augusta", "gardenia jasminoides", "garfish", ...
	]
}
```

Notas

- Al consultar la última página (o al pedir una página que no existe) devolverá un array vacío en “lemmas”.
- Para realizar búsquedas con espacios se ha de utilizar el símbolo “+” en lugar del espacio, por ejemplo “human+being”.

Ejemplos

- Obtener la página 10: /api/v1/wordnet?page=10
- Búsqueda: /api/v1/wordnet?search=cat
- Búsqueda paginada: /api/v1/wordnet?search=cat&page=2&page_size=150

GET /api/v1/wordnet/lemma/{lemma}

Devuelve un listado de definiciones para un lemma dado. Para cada definición además se incluye su identificador o "class_id", categoría y ruta o jerarquía.

Parámetros

- lemma: Lemma del cual se quieren obtener el listado de definiciones.

Respuesta


```
[
	{
		"class_id": "102121620",
		"definition": "feline mammal usually having [...]",
		"category_id": "5",
		"path": "5/101466257/101471682/.../102121620"
	 },
	 {
		"class_id": "102983507",
		"definition": "a large tracked vehicle that [...]",
		"category_id": "6",
		"path": "6/103575240/103094503/.../102983507"
	},
	{
		"class_id": "102985606",
		"definition": "a whip with nine knotted cords",
		"category_id": "6",
		"path": "6/103575240/103183080/.../102985606"
	}
]
```


Notas

- Si el lemma contiene espacios no es necesario sustituirlos por ningún carácter ya que pertenece a la URL. Sin embargo, el método también funcionará correctamente si se sustituyen por el símbolo "+", por ejemplo: "human+being".

GET /api/v1/wordnet/class/{class_id}

Devuelve toda la información sobre la clase indicada.

Parámetros

- class_id: clase o synset_id de la cual se desea obtener la información. De forma opcional el sistema permite enviar una lista de synset_id separados por coma. En este último caso devolverá en un array la información de todos los synsets solicitados.

Respuesta


```
{
	"class_id": "102084071",
	"parent_class_id": "101317541",
	"category_id": "5",
	 "lemma": "dog",
	"definition": "a member of the genus Canis [...]",
	"synonyms": "canis familiaris, dog, domestic dog",
	"path": "5/101317541/102084071",
	"hierarchy_path": "Animal/domestic animal/dog",
	"hierarchy": [
		{
			"class_id": "5",
			"lemma": "Animal",
			"definition": "Animal"
		},
		{
			"class_id": "101317541",
			"lemma": "domestic animal",
			"definition": "any of various animals that have [...]"
		}
	]
}
```


Notas

- En la jerarquía no se envía el último elemento ya que es la propia clase consultada.

Metadatos soportados para la clasificación

En el mismo instante en el que se capture la imagen se tienen que almacenar todos los metadatos posibles sobre la misma, como: información del dispositivo, datos de todos los sensores (aceleración, GPS, etc.), e información EXIF de la imagen.

A continuación, se adjunta una tabla con el detalle de todos los datos esperados (aunque la mayoría son opcionales) en servidor:

Nombre	Tipo	Tamaño	Descripción
osversion	string	32	OS versión Example: 6.0.1
model	string	64	Device model Example: Android Google Nexus 5
mirbotversion	string	8	App versión Example: 1.0.8
timestamp_picture	timestamp		Timestamp of picture
wifi	enum	YES, NO	The data was sent using a WIFI network
flash	enum	YES, NO	Flash enabled
Localización			
name	string	200	Main name of the feature
locality	string	200	Locality
sublocality	string	200	Sublocality
pc	string	16	Postal Code
admin_area	string	200	Name of administrative division of level 1
thoroughfare	string	200	Name of administrative division of level 2
subthoroughfare	string	200	Name of administrative division of level 3
country	string	3	Country code (ISO-639-1)
ocean	string	16	
numaoi	smallInt		Number of close areas of interest

closestaoi	string	200	Name of the closest area of interest
reliablelocation	enum	YES, NO	Location acquired with adequate precision
horizontalerror	float		Approximate horizontal error in meters
verticalerror	float		Approximate altitude error in meters
Sensores			
lat	float		Latitude
lng	float		Longitude
altitude	float		Altitude
yaw	float		Rotation (yaw) in radians
pitch	float		Rotation (pitch) in radians
roll	float		Rotation (roll) in radians
angle	float		Angle with respect to horizontal in degrees
accelerationx	float		Acceleration (x)
accelerationy	float		Acceleration (y)
accelerationz	float		Acceleration (z)
globalacceleration	float		Global acceleration
Exif			
exif_pixelxdimension	integer		
exif_pixelydimension	integer		
exif_subjectarea	string	64	
exif_lensspecification	string	64	
exif_lensmake	string	64	
exif_lensmodel	string	64	
exif_aperturevalue	float		
exif_brightnessvalue	float		
exif_shutterspeedvalue	float		
exif_focallength	float		
exif_focallenin35mmfilm	smallInteger		
exif_scenetype	smallInteger		

exif_isospeedratings	string	64	
exif_fnumber	float		
exif_whitebalance	smallInteger		
exif_digitalzoomratio	float		
exif_flash	smallInteger		
exif_sensingmethod	smallInteger		
exif_meteringmode	smallInteger'		
exif_colorspace	smallInteger		
exif_sharpness	smallInteger'		
exif_exposuremode	smallInteger		
exif_exposurebiasvalue	float		
exif_exposureprogram	smallInteger		
exif_exposuretime	float		
exif_datetimeoriginal	timestamp		
exif_datetimedigitized	timestamp		
exif_subsectimeoriginal	integer		
exif_subsectimedigitized	integer		

Además de todos estos datos en servidor también se extraen la información GIS utilizando geolocalización inversa a partir de los datos del GPS recibidos. Estos datos NO hay que enviarlos desde el móvil, sino que se calcula en servidor.

Nombre	Tipo	Tamaño	Descripción
gis_id	integer	unsigned	Unique id to identify the feature
gis_feature_id	integer	unsigned	Unique id to identify the feature
gis_name	string	MAX: 200	Name of the feature
gis_adm1	string	MAX: 20	Code for administrative division of level 1
gis_adm2	string	MAX: 80	Code for administrative division of level 2
gis_amd3	string	MAX: 20	Code for administrative division of level 3
gis_adm4	string	MAX: 20	Code for administrative division of level 4
gis_adm1_name	string	MAX: 200	Name of administrative division of level 1

gis_adm2_name	string	MAX: 200	Name of administrative division of level 2
gis_adm3_name	string	MAX: 200	Name of administrative division of level 3
gis_adm4_name	string	MAX: 200	Name of administrative division of level 4
gis_feature_class	string	MAX: 4	The feature class
gis_feature_code	string	MAX: 10	The feature code
gis_country_code	string	MAX: 3	ISO 3166 country code
gis_population	integer	unsigned	How many people lives in that place
gis_elevation	integer		Elevation in meters
gis_gtopo30	integer		Average elevation of 30’x30’ area in meters
gis_distance	float		Distance to the feature in meters