UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de un cliente para dispositivos móviles para la "Ruta de la tapa"

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Gaona Jiménez, Marta

Tutor: Sáez Barona, Sergio

2014-2015

Resumen

Elaboración de una aplicación acerca de las rutas de la tapa existentes en España; específica para dispositivos móviles y basada en el sistema operativo Android. Esta aplicación permite el control de usuarios – registro-inicio de sesión – mediante un servidor en node.js, así como la obtención de los datos relativos a las rutas mediante el uso de una API externa. Entre la funcionalidad destaca la localización de las rutas más cercanas – a través del uso de geolocalización –, la votación de las tapas participantes mediante lectura de códigos QR y la posibilidad de crear *tweets* desde la propia *app* a través de Twitter. Asimismo, hace uso de la API de Google Maps para posicionar los bares sobre un mapa y señalar el camino de llegada a estos desde la posición inicial del usuario.

Palabras clave: Ruta de la tapa, Android, node.js, API, QR, mapas, Twitter.

Abstract

Elaboration of an application about the different *rutas de la tapa* existing in Spain; it is specific for mobile devices and it is based on the operating system called Android. This application allows for controlling users — registration–login — by a server in node.js, as well as obtaining data related to the routes with the use of an external API. Between its functionality stands out the localization of the nearest routes — by the use of geolocalization —, the vote of the participating bar snacks through QR code-scanning and the possibility of typing tweets from the application through Twitter. Likewise, it uses the API of Google Maps to position the different bars on a map and mark the way.

Keywords : Ruta de la tapa, Android, node.js, API, QR, maps, Twitter.

Resum

Elaboració d'una aplicació sobre les rutes de la tapa existents a Espanya; específica per a dispositius mòbils i basada en el sistema operatiu Android. Aquesta aplicació permet el control d'usuaris — registre-inici de sessió — mitjançant un servidor en node.js, així com l'obtenció de les dades relatives a les rutes mitjançant l'ús d'una API externa. Entre la funcionalitat destaca la localització de les rutes més properes -a través de l'ús de geolocalització-, la votació de les tapes participants mitjançant lectura de codis QR i la possibilitat de crear tweets des de la pròpia *app* a través de Twitter. Així mateix, fa ús de l'API de Google Maps per posicionar els bars sobre un mapa i assenyalar el camí d'arribada a aquests des de la posició inicial de l'usuari.

Palabras clave: Ruta de la tapa, Android, node.js, API, QR, mapes, Twitter.

A mi madre, esa gran cocinera, a mi padre y hermanos.

Tabla de contenidos

1. Introducción

Como es un lenguaje, muchas veces no lo entendemos. Comer y respirar es lo único que hacemos desde que nacemos.

— Ferran Adrià —

Nos encontramos actualmente en una época donde la información y las redes sociales tienen una relevancia enorme en nuestro día a día. Cada vez es más habitual que cualquier persona disponga de un dispositivo móvil de nueva generación, los llamados *smartphones*, con la posibilidad de hacer llegar a los usuarios productos y eventos de forma masiva. A través de las aplicaciones para móviles se ofrece un gran abanico de posibilidades a los clientes que, además, realizan una contribución cada vez mayor a la repercusión que pueda tener un producto gracias a la interacción mediante las redes sociales.

Por otro lado, si algo nos gusta en España, es el buen comer. No hay nada más típico que ir a “*tomar unas tapitas”* acompañadas de una cerveza o un vasito de vino, lo que provoca que, cada vez de forma más asidua, se realicen eventos de rutas de la tapa. Estos actos promueven el comercio de los bares ofreciendo, durante un tiempo determinado, precios bajos sobre un plato en concreto que entra a concurso. También tienen un carácter social, ya que se propone la participación activa de los consumidores, a los que se invita a ir visitando distintos bares de forma que prueben las distintas tapas que entran a concurso y así puedan compartir sensaciones y gustos con otros visitantes de la ruta. Además, es habitual que los clientes que consumen dichas tapas puedan votarlas para, finalmente, proclamar un ganador.

Por tanto, ¿Por qué no unir las ventajas que nos ofrecen los medios tecnológicos actuales con un aspecto tan característico de España como es la costumbre de las tapas? De esta idea surge la creación de la aplicación para dispositivos móviles *La ruta de la tapa*. Esta *app* pretende unir los aspectos más relevantes que nos ofrece la aparición de los *smartphones* con características esenciales de una ruta de la tapa. En una primera implementación, será desarrollada para dispositivos bajo el sistema operativo Android.

Nuestra aplicación permitirá el acceso a un mundo completo de rutas de la tapa en España, permitiendo además, mostrar al usuario las que se encuentran más cercanas a su situación actual geolocalizando su posición. De este modo, en cualquier momento, un usuario, haciendo uso de su propio móvil, podría consultar si existe alguna ruta que se encuentre activa y cerca de su localidad.

Asimismo, gracias a las tecnologías de mapas que se encuentran disponibles actualmente, se pretende poder mostrar todos los bares participantes sobre un mapa, así como, poder obtener el trayecto vía GPS hacia dichos establecimientos. Gracias a todo ello, se podría lograr que rutas poco conocidas obtuvieran un mayor número de visitantes ya que se facilitaría el conocimiento de su existencia así como el trayecto a realizar para poder visitarlas.

Por supuesto, la aplicación ofrecerá toda la información que de normal podríamos encontrar en los típicos folletos que se reparten habitualmente. Se mostrarán todos los bares participantes, las tapas que presentan, una breve descripción de éstas, su foto… y todo los datos que sean relevantes para los comercios.

Por último cabe destacar la relevancia de las redes sociales. Sin duda alguna, una de las más destacadas es Twitter. Mediante el uso de mensajes cortos y los *hashtag* se puede llegar a millones de personas realizando una publicidad mayor y gratuita del evento. Por ese motivo, en nuestra aplicación, se pretende incorporar la creación de *tweets* de forma que, cualquier cliente que haga uso de nuestra aplicación, pueda compartir con sus seguidores la experiencia tanto de la ruta como de nuestra aplicación.

A continuación, en próximos capítulos, se asentaran las bases de las necesidades que debe cumplir el sistema para conseguir nuestro propósito. Se realizará una descripción completa tanto de los objetivos como de los requisitos específicos de la aplicación, así como el mono en el que se ha llevado a cabo el desarrollo completo de la implementación y sus pruebas.

Bon profit![1]

[1] *Bon profit!:* expresión valenciana equivalente a ¡Qué aproveche!

2. Motivación y objetivos

Algunas personas quieren que algo ocurra, otras sueñan con qué pasará, otras haces que sucedan.

— Michael Jordan —

En esta sección, motivación y objetivos, se tratan las razones de elección del proyecto. Se hace una breve introducción de la importancia de las rutas de la tapa, así como de la gastronomía en España. Seguidamente, se justifica la selección del proyecto en base a dicha magnitud. A continuación, se aborda el alcance del proyecto base así como los objetivos a alcanzar.

2.1 Justificación

La experiencia culinaria es uno de los elementos clave en el que el turista se basa a la hora de elegir destino de viaje. Teniendo en cuenta que, según el informe 'Food Tourism 2014', España lidera el ranking a nivel europeo como destino atractivo por su oferta gastronómica, — por delante de Reino Unido (73%), Italia (61%), Francia (60%) y Alemania (55%) —, no es de extrañar que la gastronomía reciba un gran peso en la vida diaria del país, tanto a nivel sociocultural como económico; considerando que el sector terciario es el más importante de la economía de este país a año 2015.

Asimismo, la cocina española se ve enriquecida por las aportaciones de cada región. Si observamos por comunidades, en todas apreciamos el predominio de un estilo y alimentos autóctonos; como podría ser la paella en Valencia, el *pescaito frito* en Andalucía o los *pintxos* del País Vasco. Por tanto, se considera fundamental la relación entre la gran variedad de platos típicos — como el gazpacho, la paella, el cocido madrileño, el jamón ibérico o el marisco —, con el patrimonio turístico; fundamental para el desarrollo turístico y económico español. Por ejemplo, bien se puede apreciar este hecho con el valor que

recibe el aceite de oliva de España fuera de las fronteras, y que se emplea en una gran variedad de platos de nuestra gastronomía.

Por ese motivo, no es de extrañar que las rutas de la tapa estén en auge. Una ruta de la tapa consiste en la participación por parte de bares y restaurantes en una especie de concurso. Los participantes exponen una tapa — habitualmente acompañada de una bebida — que se somete a votación por parte de los clientes que la consumen. El aspecto de ruta viene definido por la proximidad de los establecimientos asociados — suelen celebrarse en zonas concretas de una ciudad o municipio — de forma que los usuarios no necesiten recurrir a medios de transporte para desplazarse de un local a otro.

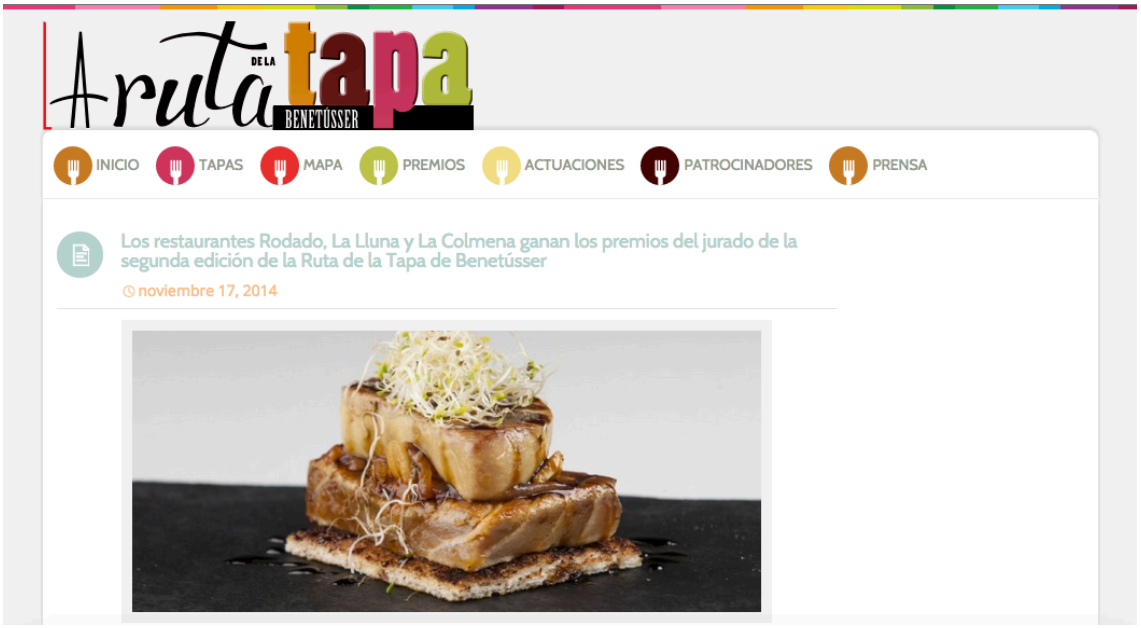


Figura 1. Ejemplo página web: La ruta de la tapa Benetússer, recuperada de: rutatapabenetusser.es

2.2 Motivación

Valiéndonos de todo lo expuesto y del gran crecimiento e interés hacia este movimiento social entorno a las rutas de la tapa, se evidencia la posibilidad de creación de una aplicación que responda ante la demanda de estos acontecimientos. De este modo, nace la aplicación *La ruta de la tapa*.

Los principales motivos de su creación vienen dados por la oportunidad de negocio que se evidencia de la oferta cada vez mayor de rutas existentes. En primer lugar, es un aspecto relevante en la economía del país, promoviendo el consumo de platos y bebidas de

diferentes bares y restaurantes, de los municipios que realicen estos eventos. Además es una forma de otorgarles publicidad y reputación a los concursantes. Gracias a nuestra aplicación, una ruta en una pequeña comarca consigue una trascendencia que, de otro modo, no sería capaz de obtener, promoviendo un mayor número de visitantes y consumidores que repercuten en los beneficios logrados por la ruta.

En segundo lugar, dada la envergadura social que posee, parece adecuado realizar una aplicación que fomente el conocimiento de las rutas entre usuarios. Un cliente puede aconsejar a amigos o familiares la visita de una ruta concreta, indicándoles que pueden ver la distribución de los locales en el mapa así como los platos que se ofertan a través de nuestra aplicación. Además una de las funcionalidades añadidas es el sistema de logros — se obtienen visitando rutas, consumiendo tapas o votando éstas — mediante el cual, en un futuro, se podría traducir en descuentos y ofertas para el consumo en las rutas.

También la incorporación de Twitter potencia la característica de red social, en tendencia actualmente. Desde la propia aplicación se permiten realizar *tweets* en relación a la ruta, fomentando la repercusión de estos eventos en Twitter, red usada por millones de usuarios.

Por otra parte, y relativo a las votaciones, permite una informatización del recuento de votos. A través de nuestro producto, *La ruta de la tapa*, se permite la votación de las tapas participantes si han sido consumidas por los usuario. Este voto se envía a un servidor que, una vez finalizado el periodo de concurso, calcula el resultado final de la competición. De este modo, el recuento se produce de forma automática sin tener que realizar un esfuerzo extra por parte de los ayuntamientos.

Por último cabe destacar la carencia de aplicaciones de este tipo en el panorama actual. Si hacemos una búsqueda web, o en alguna de las tiendas que incorporan los dispositivos móviles, podemos observar que no se encuentra prácticamente ningún producto que ofrezca este tipo de servicios — la gran mayoría son *apps* dedicadas a una ruta concreta —, de forma que, cubriría el vacío existente en la actualidad.

Así pues, podemos concluir con la idea fundamental de que existe una posibilidad de mercado a explotar con la aplicación *La ruta de la tapa*. En el siguiente capítulo se tratará como se encuentra el panorama actual en cuanto a la tecnología escogida, Android, así como la posible competencia a nuestra aplicación.

3. Estado del arte

Fíjate en los niños pequeños. No tienen límite. Su mente es un enorme panorama de posibilidades.

— Robin Sharma —

Este capítulo sirve como base teórica del proyecto que se ha desarrollado. En él, se ofrece una visión general del área en el que ha sido enmarcado, más concretamente, en los dispositivos móviles bajo el sistema operativo Android.

Se comienza hablando sobre el propio SO, sus características principales y su arquitectura. También se hará una comparativa de mercado para sustentar la decisión de realizar la aplicación Ruta de la tapa sobre dispositivos de tipo Android.

Por último, se hace una comparativa con aplicaciones encontradas en la propia tienda de Google relacionadas con el proyecto.

3.1 Android

Basándose en el núcleo Linux, surgió el sistema operativo Android de la mano de la empresa Android Inc., aunque su auge llegó al ser comprado por Google en 2005 y su posterior lanzamiento oficial en 2008.

Inicialmente fue diseñado para dispositivos móviles como *smartphones* o tabletas electrónicas pero, en la actualidad, se ha extendido a Smart TVs, tecnologías para automóviles o dispositivos *wereables* — se llevan sobre, bajo o incluido en la ropa — como relojes o las Google Glass.

Su desarrollo está liderado por la Open Handset Alliance, que como en su propia web muestra, a año 2015, se trata de un grupo compuesto por un total de 84 empresas de

tecnologías y móviles, unidas para crear una plataforma completamente abierta y de código libre.

Como dato anecdótico comentar que cada versión nueva lleva como nombre algún tipo de dulce, como el actual Lollipop, o otras anteriores como Icecream, Jellybean o KitKat. Esto se debe a que e*stos dispositivos hacen nuestra vida más dulce, por lo que el nombre de cada versión de Android corresponde a un dulce diferente*[2].

3.1.1 Características

Como características generales de los dispositivos basados en el sistema operativo Android, podemos sintetizar las que siguen:

- Proporciona un *framework* en el cual desarrollar aplicaciones y juegos en lenguaje Java. Su posterior compilación se realiza mediante las herramientas SDK de Android.
- La aplicación final tiene una extensión APK; éste es un paquete Android en el cual está todo el contenido de la misma. Este archivo es el instalador final.
- Dispone de múltiples puntos de entrada. Una aplicación está compuesta por componentes, como los *activities*, que interactúan entre sí mediante *intents* —mecanismo de comunicación en Android—. Esto provoca que desde una aplicación se pueda iniciar otra, de ahí que exista esa multiplicidad.
- Multiplataforma. Es capaz de desplegarse en diversos dispositivos: puede adaptarse a distintos tamaños de pantalla y resoluciones o hacer consultas sobre requisitos hardware específicos para permitir o limitar la funcionalidad de la aplicación.
- Entorno limitado de seguridad. Cada aplicación se ejecuta en su propio proceso Linux y queda identificada por un ID. De esta forma, el sistema es capaz de proporcionarle los privilegios mínimos necesarios para su correcto funcionamiento.

Por otra parte, como características específicas, que se han empleado en la puesta en práctica de este proyecto, destacamos las siguientes; clasificándolas según componentes, multimedia y sensores, conectividad y almacenamiento.

[2] Android – Historia, https://www.android.com/history [Mayo 2015]

- Componentes:
  - *Intent*: objeto tipo mensaje utilizado para solicitar una acción de otro componente.
  - *Activity*: proporciona una pantalla con la que los usuarios pueden interactuar para realizar acciones.
  - *Fragment*: representa un comportamiento o parte de una actividad. Una misma actividad puede tener diversos fragmentos coexistiendo en la misma pantalla o sustituyendo unos por otros.
  - *AsyncTask*: flujos a la espera de que una tarea asíncrona, como puede ser un petición de un recurso http, proporcione un resultado.
- Multimedia y sensores:
  - Captura de fotos: permite conectar de manera sencilla con la cámara del dispositivo. En la aplicación que se lleva a cabo se utiliza para la lectura de códigos QR.
  - Geolocalización: detecta el posicionamiento geográfico del dispositivo, en unas coordenadas latitud-longitud.
- Conectividad:
  - *Cliente HTTP: conexión por red con la que enviar y recibir datos.*
- Almacenamiento:
  - *Shared Preferences*: permite guardar valores de tipos de datos primitivos, como cadenas de caracteres, números o booleanos, en forma clave valor.
  - SQLite. Base de datos basada en SQL que implementa, sin servidor adicional, un motor de BD transaccional autónomo, con el que poder realizar consultas o inserciones de datos.

3.1.2 Arquitectura

Android posee una arquitectura en forma de pila. Cada capa de ésta y los elementos que contienen están integrados para proporcionar un desarrollo y un entorno de ejecución óptimo para la aplicación. Como queda representado en la figura 2, la arquitectura queda dividida en aplicaciones, framework, librerías, el run-time de Android y el kernel de Linux.

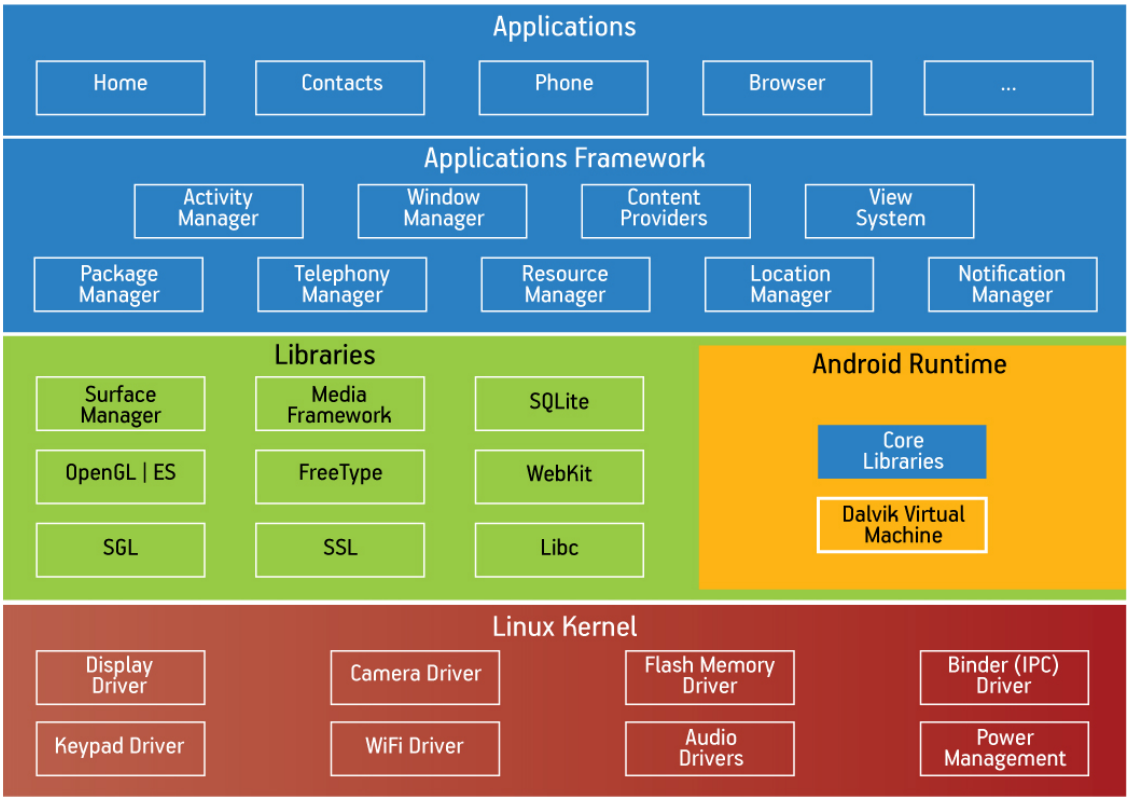


Figura 2. Arquitectura Android, recuperada de: www.techdesignforums.com

El **kernel de Linux** proporciona el nivel de abstracción entre el hw y las capas superiores de la pila. Basado en la versión 2.6 proporciona servicios básicos como multitarea, gestión de memoria, gestión de procesos, gestión de pila de tareas, dispositivos de red y controladores como los de la pantalla del dispositivo, Wi-Fi o audio.

Aunque el kernel de Linux proporciona su propio entorno de procesos, las aplicaciones Android se ejecutan en el entorno **run-time** — entorno de ejecución — empleando una instancia de máquina virtual propia de tipo Dalvik, creada por Google. De esta forma la aplicación queda aislada del propio sistema operativo evitando que interfiera de forma negativa con el correcto funcionamiento de éste.

Mediante la capa de **bibliotecas,** basadas en Java, obtenemos las funciones específicas para el desarrollo Android. Esta capa incluye aquellas bibliotecas que interactúan con el marco de la aplicación, la construcción de la interfaz y el acceso a base de datos. Éstas proporcionan una API de llamadas sobre las librerías del núcleo que realizan realmente trabajado y desarrolladas en C/C++.

Para enlazar dichas librerías con la actividad en la aplicación, el **framework**, o marco de la aplicación, suministra un conjunto de servicios que permiten que las aplicaciones

Android se ejecuten y se gestionen. Por ejemplo, el *manager activity* controla el ciclo de vida y la pila de la actividad; el *content provider* permite publicar y compartir datos de aplicaciones; y el *location manager* permite acceder a servicios de localización para recibir actualizaciones sobre cambios en la ubicación.

Tras todas las capas descritas anteriormente, se sitúa como más alta la de **aplicación**. Ésta se refiere a la aplicación final en sí misma, que podría ser tanto una propia del sistema, como el calendario, como una proporcionada por terceros, como *La Ruta de la tapa* desarrollada en este proyecto.

3.1.3 Comparativa de mercado

Para la comparativa de mercado se ha hecho uso de la web de IDC2, que a su vez ha obtenido los datos de Worldwide Quarterly Mobile Phone Tracker. En el primer trimestre del año 2015, se han venido un total de 334,4 millones de dispositivos móviles. Si observamos la figura 3, el mercado en este último trimestre, al igual que en anteriores, está dominado por los dispositivos Android. Esto también es favorecido porque multitud de compañías — Samsung, HTC, Alcatel, BQ, etc. — incorporan el sistema operativo de Google. En total, sobre un 78% de los dispositivos vendidos han incorporado como SO Android.

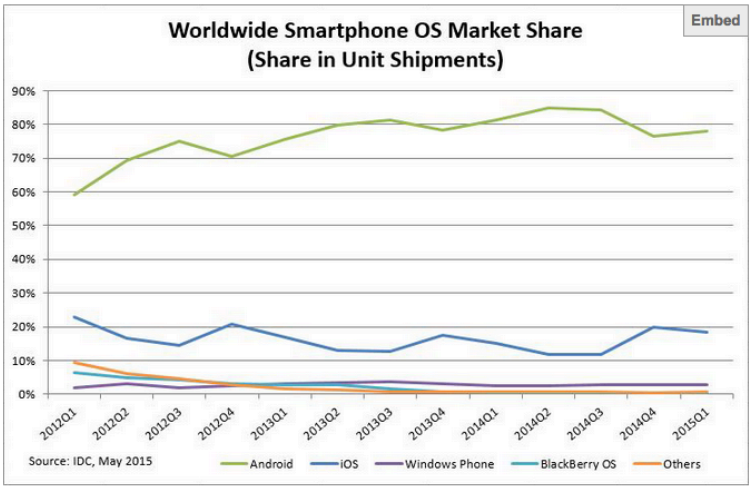


Figura 3. Comparativa de mercado: venta dispositivos móviles, recuperada de: www.idc.com

En conclusión, podemos afirmar que Android es un entorno adecuado para la realización de *La Ruta de la tapa*. Por una parte, sus características hacen que sea sencillo aplicar funciones necesarias de la aplicación como la localización de los bares en un mapa,

la lectura de códigos QR o un diseño amigable e intuitivo. Además, que el lenguaje de desarrollo sea Java, hace que el aprendizaje específico de funciones Android resulte rápido por la gran cantidad de información que se puede encontrar sobre dicho lenguaje en la Web. Por otra parte, la arquitectura resulta sólida para asegurarnos un correcto funcionamiento de nuestra aplicación sin interferir en ningún momento con el comportamiento del *smartphone* o tablet. Por último, y motivo de gran peso, el mercado Android es muy grande por lo que el número de usuarios que podrían hacer uso de la aplicación podría ser mayor.

3.2 Aplicaciones relacionadas

Tal y como se ha comentado anteriormente, existen muy pocas aplicaciones para dispositivos móviles Android dedicadas a gestionar y mostrar información sobre múltiples rutas de la tapa. A continuación se tratará una de las pocas aplicaciones disponibles que se han encontrado: *Rutappa*.

Con el fin de poder hacer una comparativa de nuestro proyecto con el panorama actual de aplicaciones, se realiza una búsqueda en Play Store, la tienda de aplicaciones para Android. Entre ellas, la única aplicación que contempla nuestro escenario es Rutappa. Con el fin de poder analizarla se ha decidido descargar la *app* que se muestra en la figura siguiente.

Figura 4. Pantalla de bienvenida de la *app* Rutappa, recuperada de la *app* Rutappa

Tras haber experimentado con ella se ha llegado a la conclusión de que cuenta con aspectos que se consideran básicos para el desarrollo de una aplicación de este tipo como son la situación en el mapa, la posibilidad de ver las tapas y la opción de poder votarlas.

Si bien es cierto que incorpora una opción de poder acceder al trayecto hacia el bar mediante Google Maps, no se ha podido probar. Al intentar realizar dicha acción, ha aparecido el mensaje “esperando GPS” y, tras observar los comentarios en la propia tienda de Android, es un error común en muchos usuarios, por lo que parece que no esté plenamente funcional.

Además en el menú aparecen otras acciones como bases o tapea y gana. En cuanto a bases, se trata de los requisitos propios de una ruta, más propios de una posible página web que de una aplicación final para un cliente que acuda a la ruta. Respecto a tapea y gana, específica únicamente el premio por haber realizado votos desde la aplicación; sería interesante que se hubiera añadido, al menos, un seguimiento de las participaciones que tiene el usuario. En la figura 10 se muestra el menú de la aplicación.

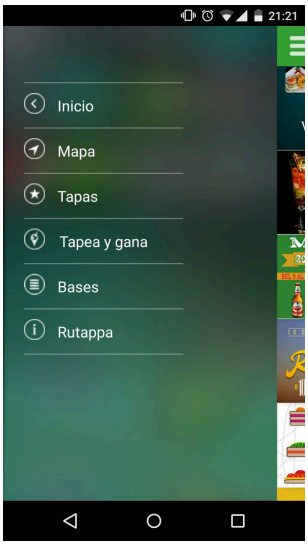


Figura 5. Menú de la *app Rutappa*, recuperada de la *app Rutappa*

Aunque la funcionalidad en sí misma es un poco básica, podría cubrir los aspectos necesarios para una aplicación que proporcione información de rutas por España. Sin embargo, un gran hándicap es el diseño y la usabilidad: los iconos se ven borrosos, la letra en algunos apartados — como en bases y tapea y gana — resulta excesivamente pequeña y el diseño es bastante pobre. Esto hace que la aplicación sea poco atractiva de cara a posibles usuarios.

En conclusión, tras el análisis tanto de Android, como de los protocolos de comunicación y la comparativa con otras aplicaciones ya existentes, se pretende desarrollar una aplicación estable y rápida, con un diseño minimalista a la par que cómodo, de forma que la navegación por la aplicación sea lo más intuitiva posible subsanando fallos de sus compañeras de área.

En la capítulo siguiente y base a los propósitos propuestos se realiza un análisis de los requisitos que debe cumplir la aplicación para satisfacer los objetivos que se han determinado.

4. Especificación de requisitos

Aquel que tiene un porqué para vivir se puede enfrentar a todos los cómos.

– Friedrich Nietzsche –

En este capítulo de especificación de requisitos se introduce una visión más cercana a lo que será el producto final. En primer lugar se realiza una descripción general donde se trata la perspectiva del producto, su funcionalidad, las características de los usuarios y la dependencia que tiene acerca de la API independiente a este proyecto.

En segundo lugar se analizaran los requisitos propios de la aplicación para determinar o satisfacer necesidades o características indispensables de nuestra aplicación. Por una parte se analizaran los necesarios para la *app* móvil; y por otra parte, los requisitos propios del servidor de gestión de usuarios.

4.1 Introducción

4.1.1 Propósito y alcance

El propósito de este proyecto es lograr un entorno donde se aúnen diversas rutas de la tapa para que posibles consumidores puedan recibir información sobre su localización, los comercios que participan o la clasificación final de éstos.

En éste punto es importante destacar dos factores. En primer lugar, la obtención de los datos es posible gracias al servidor de la aplicación web desarrollado por el alumno David Valero García. Dicho almacén de datos, hace de enlace con posibles ayuntamientos o eventos que deseen registrar sus rutas en nuestro entorno. En posteriores capítulos se hará referencia a este servidor como *proyecto Icarus*.

En segundo lugar, la aplicación *La Ruta de la tapa* — desarrollada en este trabajo — está enfocada al usuario final que desee visitar cualquiera de los recorridos que se encuentren disponibles.

En una primera implementación es necesario abarcar todos los aspectos funcionales básicos relacionados con una ruta en concreto. Debe proporcionar suficiente información para que cualquier consumidor interesado en visitar una ruta en concreto, sea capaz de consultar aquello relevante — como la distancia a la que se encuentra de su posición o si los bares y las tapas inscritas son de su agrado — y decidir si finalmente quiere participar en ella . Por tanto, es necesario que el alcance cubra las siguientes características:

- Capacidad de selección de ruta.
- Visualización y listado de los bares participantes.
- Valoración de las tapas.
- Incorporación de Twitter.
- Fidelización de clientes.

4.1.2 Objetivos

A continuación, basándose en el propósito y el alcance acordado, se describen los objetivos principales del proyecto.

El objetivo principal de este proyecto es proporcionar a los usuarios finales un entorno amigable donde poder consultar y localizar rutas de la tapa disponibles por toda España.

Dada la importancia que adquiere la gastronomía, este tipo de acontecimientos se realizan cada vez más asiduamente y de forma periódica — normalmente de forma anual — en multitud de municipios. Si hacemos una pequeña búsqueda podemos ver como existen rutas de la tapa en prácticamente toda España: en Cartagena, Torre Vieja, Granada, Valencia…

Por tanto, la meta final de la aplicación es poder concentrar toda la información de las rutas disponibles en toda España de forma clara y sencilla. Por una parte, poder acceder a las rutas más cercanas desde la posición actual donde se encuentre el usuario. Por otra parte, obtener toda la información sobre los bares participantes, su localización, la tapa que presentan y realizar votaciones sobre la tapa a través del propio dispositivo móvil. Por

último, y gracias a Google Maps, poder obtener el trayecto desde la posición actual del cliente hasta al bar participante que se desea visitar.

Asimismo, uno de los objetivos secundarios sería obtener una mayor repercusión social de los eventos. El uso de la aplicación ya promueve el conocimiento de rutas a través de la geolocalización pero, además, la incorporación de Twitter y la posibilidad de crear *tweets* desde ella, facilita una propagación mayor de la ruta ante usuarios potenciales.

Además cabe destacar que existe una carencia en dicho tipo de aplicaciones para dispositivos móviles. Si hacemos una búsqueda en Google, podemos obtener resultados de diversas rutas de la tapa de múltiples localidades. Sin embargo, no existe ninguna página web o aplicación que las concentre todas y las exponga para una consulta rápida y sencilla y tenga suficiente repercusión en el ámbito de aplicaciones para dispositivos móviles. Así que, además del propósito principal de obtener información sobre dichas rutas, se añadiría el objetivo de cubrir una carencia existente en el mercado, dando la opción a los usuarios de concentrar en un único portal todo un mundo de gastronomía. Esto, a su vez, proporciona una gran publicidad a todos los municipios inscritos en nuestra aplicación y un mayor número de visitantes en sus rutas.

Sintetizando, el objetivo de la aplicación queda claramente definido: crear un entorno de consulta eficaz donde se encuentren reunidas todas las rutas de la tapa disponibles para poder consultar su información principal y así aprovechar para explotar un mercado muy importante como es la gastronomía española.

4.1.3 Definiciones y acrónimos

A continuación se encuentra un listado sobre definiciones y acrónimos de términos que se encuentran en este proyecto. Se encuentran ordenados alfabéticamente.

Definiciones:

- **Framework:** marco de trabajo que sirve como base para el desarrollo software de determinado lenguaje de programación.
- **Geolocalización:** conocimiento de la posición actual de un dispositivo de forma automática.
- **Google maps:** servidor de aplicaciones de mapas en la web que pertenece a Google.

- **Hashtag:** etiqueta formada por una cadena de caracteres formada por una o varias palabras concatenadas y precedidas por una almohadilla. Tiene el fin de identificar un tema de forma rápida.
- **Kernel:** núcleo que hace de intermediario entre software y hardware.
- **oAuth:** protocolo que permite la autorización para sitios web o aplicaciones informáticas.
- **Play Store:** tienda de publicación de aplicaciones Android.
- **Smartphone:** dispositivo móvil con un avanzado sistema operativo.
- **Splash:** pantalla de bienvenida.
- **Twitter:** aplicación web gratuita de *microblogin* — mensajes cortos publicados en un muro —.
- **Tweet:** mensaje corto — 140 caracteres — que se utiliza en la aplicación Twitter.

Acrónimos:

- **API:** *Application Programming Interface,* interfaz de programación de aplicaciones.
- **App:** *Application,* aplicación para dispositivos móviles.
- **BD:** Base de datos.
- **HTTP:** *Hypertest Transfer Protocol* , protocolo de intercambio de información en internet.
- **MVC:** Modelo-Vista-Controlado, patrón de arquitectura software.
- **QR:** código *quick response* — de respuesta rápida — es un modulo donde almacenar información en una matriz de puntos.
- **REST:** *REpresentational State Transfer,* arquitectura de desarrollo web que se apoya en el estándar http.
- **SO:** sistema operativo.
- **TDD:** *Test-driven development*. Desarrollo guiado por pruebas de software.
- **TCP:** *Transmission Control Protocol,* protocolo de control de transferencia de datos en Internet.

Este listado está pensado a modo de guía para posibles términos desconocidos.

4.2 Descripción general

En este apartado se hace una descripción mas abstracta del producto, sin entrar en requisitos técnicos más concretos. Esto permite que se obtenga una perspectiva de lo que se va a realizar en el proyecto y se obtenga una vista general de las funciones que va a incluir y a los usuarios a los que va a estar dirigido.

4.2.1 Perspectiva del producto

El propósito de este proyecto es obtener una aplicación para dispositivos móviles donde consultar rutas de la tapa disponibles y realizar funciones básicas como consultar los bares, las tapas o su localización. Por tanto, se proyecta implementar un sistema que permita, para un usuario registrado de la aplicación, consultar las rutas disponibles, situarlas en un mapa y consultar sus bares y tapas.

Además, también debe tener aspectos de la Web 2.0[3]. Para potenciar esta característica de colaboración entre usuarios, se debe poder votar una tapa desde la propia aplicación móvil, así como, poder interactuar con Twitter mediante la creación de *tweets*.

Por último es interesante añadir una propiedad para la fidelización de clientes, que será llevada acabo mediante un sistema de logros.

4.2.2 Funcionalidad del producto

Los procesos o funciones que conforman el sistema son los siguientes:

Gestión de usuarios: permite el acceso a la aplicación una vez el usuario ha sido registrado. Las operaciones a realizar son: registro de usuario, autenticación de usuario.

Mediante esta funcionalidad, se podrá restringir el acceso a la *app* únicamente para usuarios que se hayan registrado con anterioridad.

Gestión de rutas: genera un listado de rutas disponibles para consulta en la aplicación. Las operaciones a realizar son: petición de datos, gestión de datos para su visualización y visualización de rutas disponibles para el usuario.

---

3 Web 2.0 : Sitios que facilitan compartir información, permitiendo a los usuarios interactuar y colaborar entre sí siendo ellos mismos generadores de contenido.

Se encargará de mostrar un filtrado de las rutas disponibles más cercanas desde la posición actual del dispositivo móvil.

Gestión de participantes: muestra la localización de los bares participantes sobre un mapa. Las operaciones a realizar son: petición de los datos de los bares, gestión de los datos para obtener la información relevante así como su localización y situar los establecimientos sobre un mapa.

Gracias a esta función se podrá obtener una vista general de la ruta con los bares participantes y su situación en el municipio.

Gestión de tapas: obtiene los datos de los platos que se presentan a concurso. Las operaciones a realizar son: petición de los datos de los bares, petición de los datos de las tapas que presentan, obtención de las fotos representativas de los platos, organización de la información para ver un listado con los datos y muestra de los datos concretos de cada uno de los visitantes.

Con ello, obtendremos toda la información precisa de cada uno de las tapas que entran a concurso de forma clara y organizada.

Gestión de la votación: permite la votación de una tapa cuando haya sido consumida por un cliente. Las operaciones a realizar son: comprobación del consumo por parte del usuario de la tapa, votación de la tapa y envío del resultado de la votación del cliente.

Se informatizará toda la gestión del cálculo de votos centralizándolo a través de la aplicación.

Gestión de la clasificación: posibilita obtener el estado actual la votación — si ha sido o no finalizada —. Si su estado es finalizado, muestra la clasificación de las tres tapas más votadas. Las operaciones a realizar son: consultar el estado de la votación, recuento de votos, recibir los datos de los bares ganadores, organizar los datos según su puesto de clasificación.

Con esta función, unida a la gestión de la votación, se completa la gestión de los resultados que se obtengan en dicha ruta.

Fidelización de clientes: crear un sistema que permita que los clientes sigan usando la aplicación. Para ello, se pretende un sistema de logros de forma que, al realizar ciertas acciones mediante la aplicación se obtengan premios directos para el usuario en un futuro.

Las operaciones a realizar son: obtener los logros disponibles, consultar el estado actual de los logros — conseguidos o no — y cuánto falta para su consecución y actualizar el recuento de acciones realizadas cada vez que se cumpla una de ellas.

Mediante esta característica se pretende el fomento del uso de la aplicación así como un crecimiento en el consumo por parte de los clientes de un ruta.

4.2.3 Características de usuario

Para nuestro sistema, se han detectado cuatro tipos de usuarios diferentes: el usuario no registrado, el usuario registrado, la ruta y el administrador. Por una parte, el usuario no registrado es aquel que ha tenido acceso al sistema pero que aún no ha hecho su registro por lo que su funcionalidad está muy restringida. Por otra parte, el usuario registrado además de su registro ha realizado el inicio de sesión con el sistema y puede acceder al conjunto completo de operaciones que proporciona la aplicación *La ruta de la tapa*. A su vez, el usuario ruta es el que proporciona toda la información que consume la *app* y es externo a nuestra organización; también recibe desde nuestro sistema las votaciones. Por último, el administrador, es el encargado de la gestión de los usuarios así como de los logros.

A continuación, en las tablas que se presentan, se ha identificado a partir de la funcionalidad que se ha descrito en el apartado anterior, qué procesos realizan cada uno de los usuarios. Se ha de tener en cuenta que se han omitido las acciones propias que realizará el sistema como son la organización, la visualización de los datos o posibles comprobaciones.

La primera tabla, gestión de usuarios, se encarga del registro y del inicio de sesión por parte de los usuarios:

Proceso	Usuario no registrado	Usuario registrado	Ruta	Administrador
Registro de usuario	X			X
Inicio de sesión de usuario	X			X

Tabla 1. Procesos gestión de usuarios

En segundo lugar, la tabla muestra la gestión de rutas con los procesos siguientes: la petición de los datos de la ruta y el filtrado de rutas para su posterior visualización.

Proceso	Usuario no registrado	Usuario registrado	Ruta	Administrador
Obtención listado de rutas		X	X	
Filtrado de rutas			X	

Tabla 2. Procesos gestión de rutas

E tercer lugar, tenemos la gestión de las tapas, en la que se involucran los procesos de petición con los procesos de petición de los datos de los bares, petición de los datos de las tapas que presentan y obtención de las fotos representativas del plato.

Proceso	Usuario no registrado	Usuario registrado	Ruta	Administrador
Obtención datos bares participantes		X	X	
Obtención datos tapas		X	X	
Obtención fotos tapas		X	X	
Obtención localización bares		X	X	

Tabla 3. Procesos gestión de tapas

La cuarta tabla, gestión de la votación, presenta los procesos de consumo de una tapa y de su posterior votación.

Proceso	Usuario no registrado	Usuario registrado	Ruta	Administrador
Consumir tapa		X		
Votar tapa		X	X	

Tabla 4. Procesos gestión de votación

En quinto lugar tenemos la tabla de la gestión de la clasificación que complementa a la tabla anterior en cuanto a la funcionalidad referida a los resultados obtenidos por los bares en la ruta. Los procesos son la consulta del estado de la votación, el recuento de votos y la petición de los bares ganadores.

Proceso	Usuario no registrado	Usuario registrado	Ruta	Administrador
Consultar estado de la ruta		X	X	
Recuento de votos			X	
Obtención datos participantes ganadores		X	X	

Tabla 5. Procesos gestión de clasificación

Por último se muestra la tabla de fidelización de clientes mediante el sistema de logros. Para ello, los procesos necesarios son: obtener la lista de logros, acceder a su estado actual y actualización de un logro.

Proceso	Usuario no registrado	Usuario registrado	Ruta	Administrador
Obtención logro		X		
Consultar estado de logros		X		X
Actualizar logro		X		X

Tabla 6. Procesos fidelización de clientes

4.2.4 Restricciones del sistema

La aplicación *La ruta de la tapa* obtiene la información de un servidor ajeno a este proyecto por lo que será necesario que éste se encuentre en funcionamiento y con datos cargados para el correcto funcionamiento de la *app*. Si hubiera un cambio en la dirección del servidor, el sistema está adaptado para que, mediante una actualización sencilla de la configuración, siga con su correcto funcionamiento. En el apartado siguiente se aborda con más detalle este tema.

Asimismo, para que un usuario pueda utilizar dicha aplicación, debe estar conectada a Internet ya sea vía datos o vía WiFi. También cabe tener en cuenta que si el dispositivo tiene el GPS en funcionamiento, la geolocalización del dispositivo será más precisa.

Finalmente, la aplicación solo se puede instalar en dispositivos que funcionen bajo el sistema operativo Android.

4.2.5 Dependencias del sistema

Como ya se ha comentado, el sistema depende de un servidor externo que proporciona todos los datos relacionados con las rutas disponibles. Mediante el acceso vía API — en el capítulo 7 se explican más detalladamente qué es una API y los mensajes que se intercambian entre sistemas — se obtienen tanto las rutas disponibles como los datos de los participantes de una ruta en concreto. Además la aplicación *La ruta de la tapa*, es la encargada de recopilar las votaciones de los clientes, sin embargo, es este servidor el que realiza el recuento final.

4.3 Especificación de requisitos

Mediante la especificación de requisitos se obtienen las necesidades y condiciones para el buen funcionamiento de la aplicación de *La ruta de la tapa*. Estos requisitos quedarán agrupados en requisitos de interfaz externa, funcionales y de bases de datos.

4.3.1 Requisitos de interfaz externa

En relación a los requisitos de interfaz externa, se detallarán todo lo relativo a la visualización de la aplicación móvil. A continuación se plantea el listado de requisitos:

- **IR-R01.** Idioma: El idioma principal debe ser el castellano; ES-es.
- **IR-R02.** Compatibilidad Android: El sistema debe poder funcionar con normalidad en cualquier versión de Android superior a la 4.1; ésta inclusive.
- **IR-R03.** *Portrait*/*Landscape*: Las vistas se deben mostrar en *portrait*; vista vertical.
- **IR-R04.** Extensibilidad de idiomas: El sistema debe estar preparado para ser multilenguaje en un futuro.
- **IR-R04.** Tema: El tema de la aplicación debe ser *Material Theme*. Ha de tener una correcta visualización en cualquier dispositivo soportado.

4.3.2 Requisitos funcionales

En esta sección se trata toda la funcionalidad completa que la aplicación lleva a cabo. A continuación se desarrolla el listado de requisitos por parte de la aplicación agrupando en funcionalidad de usuario — UR —, de ruta — RR —, de clasificación — CLR —, de logros — LR — y de comunicación — COR —; siguiendo el hilo del apartado de funcionalidad del producto.

Requisitos de usuario:

- **UR-R01.** Registrar usuario con datos básicos como nombre, apellidos, email y contraseña.
- **UR-R02.** Registrar usuario vía Twitter.
- **UR-R03.** Iniciar sesión.
- **UR-R04.** *Autologin*. Una vez el usuario haya iniciado sesión, iniciarla automáticamente.
- **UR-R05.** Cerrar sesión.

Requisitos de ruta:

- **RR-R01.** Elegir ruta filtrada por cercanía a la posición actual del dispositivo.
- **RR-R02.** Situación de los bares sobre un mapa.
- **RR-R03.** Trayecto a bar seleccionado.
- **RR-R04.** Listado de bares participantes.
- **RR-R05.** Visualización de la información del bar/tapa.
- **RR-R06.** Votación de la tapa.
- **RR-R07.** Salir de la ruta seleccionada.

Requisitos de clasificación:

- **CLR-R01.** Obtener estado de la votación.
- **CLR-R02.** Ver clasificación final.

Requisitos de logros:

- **RR-R01.** Listado de logros disponibles.
- **RR-R02.** Actualización de logros al realizar una acción en la aplicación.

Requisitos de comunicación:

- **COR-R01.** Capacidad de crear *tweets* desde la aplicación.

4.3.3 Requisitos de bases de datos

El único requisito de BD es que para la gestión de usuarios y logros se utilice una base de datos no relacional. Con ello aseguramos una mayor escalabilidad y velocidad.

Además permite ausencia de atributos en una misma colección, facilitando el polimorfismo, y la inserción de datos complejos como parámetros, evitando tener varias tablas que contengan la información relativa a un mismo objeto.

En resumen, se ha obtenido una vista general del producto, así como la funcionalidad que ofrece y sus restricciones y dependencias. De esta modo, a continuación, se realizará el análisis completo del producto partiendo de los requisitos obtenidos.

5. Análisis

El mayor misterio del mundo es que resulta comprensible.

— Albert Einstein —

A continuación, se realiza una análisis del sistema para ofrecernos la estructura general del producto final. En primer lugar obtendremos el diagrama de clases donde se ofrecen las clases principales empleadas así como sus atributos y métodos. En segundo lugar obtendremos los casos de uso a partir de los escenarios y la interacción que realizan los actores en el sistema. En tercer y último lugar, abordaremos el diagrama de flujo para representar de forma gráfica el flujo de las funcionalidades del sistema.

5.1 Diagrama de clases

Para el desarrollo del proyecto es necesario tener en cuenta tanto el lado cliente, la aplicación Android, como el servidor de usuarios. Por tanto en primer lugar se muestra el diagrama de clases de la *app* y en segundo lugar el del servidor.

En la figura 11 podemos ver el diagrama de clases de la aplicación para dispositivos móviles *La ruta de la tapa*. Como se puede observar, existen seis clases: rutas, bares, tapas, clasificación, usuario y logros. En ellas podemos ver las funcionalidades principales que presentan como es la obtención de sus datos; en el caso de usuario tenemos la creación — registro — y el inicio de sesión; para las tapas se hace uso del método vote; por último, para logros tenemos su actualización al realizar acciones predeterminadas. Es una primera aproximación, por lo que los atributos y métodos finales pueden tener ciertas variaciones.

Existe una relación entre las clases. Un usuario solo puede estar en una ruta. Sin embargo una ruta puede tener muchos bares — al menos una — y cada uno de éstos tiene asociada una tapa. Además, una vez obtenidos los resultados, se crea la clase clasificación

donde quedan agrupados un bar y una tapa. Para finalizar, los logros, pertenece a un usuario y pueden ser un número indeterminado.

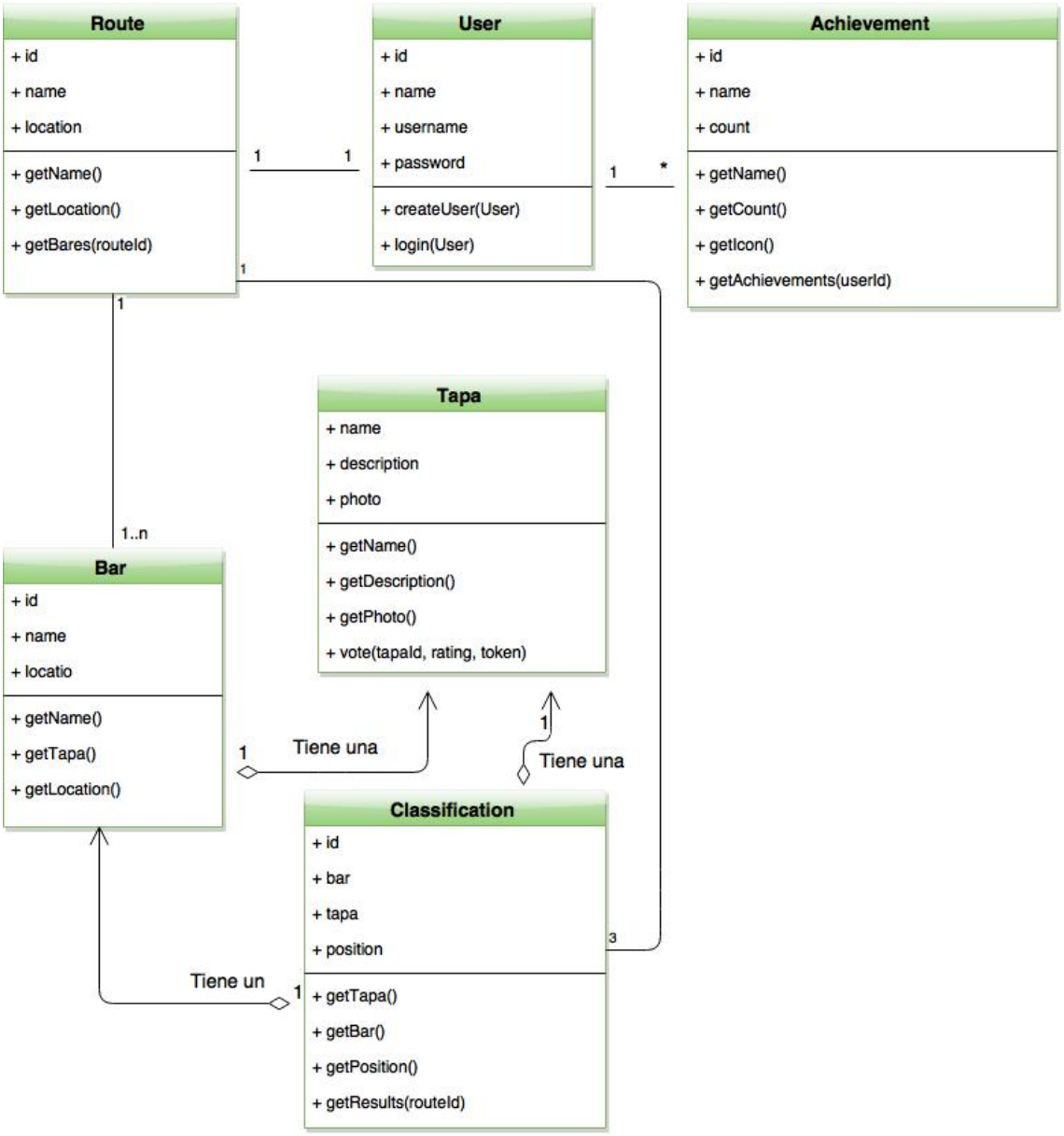


Figura 6. Diagrama de clases aplicación Android

En la figura 12 tenemos el diagrama del servidor. En esta ocasión únicamente existen dos clases: usuarios y logros. Como en el caso anterior un usuario puede tener un número indeterminado de logros.

Mientras que usuarios tiene los métodos relacionados con el registro y el inicio de sesión de usuarios, la clase logros realiza la gestión de los logros pertenecientes a un

usuario. Esta clase tiene los métodos de creación y obtención de logros así como una actualización automática de éstos mediante el paso de un id y el usuario al que pertenece.

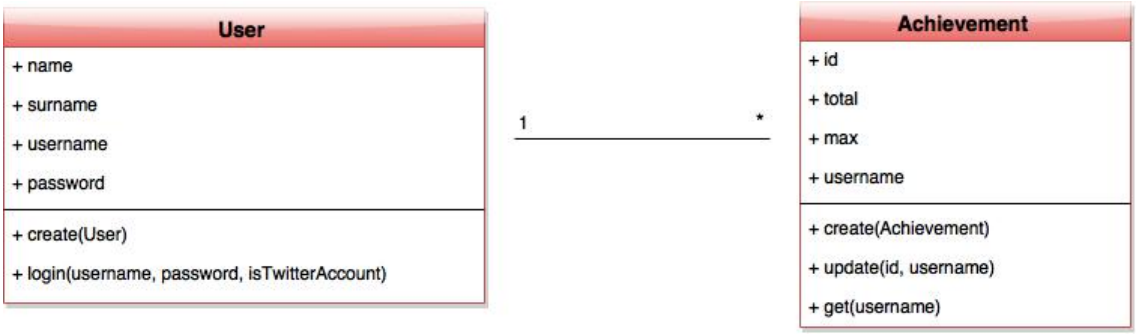


Figura 7. Diagrama de clases del servidor

5.2 Casos de uso

Mediante los casos de uso vamos a explicar las actividades que deben llevarse a caso para los procesos. Para comenzar se describe los escenarios relativos a acciones entorno a un elemento en común — por ejemplo todo lo relacionado con una ruta — y, tras ello, se muestra el diagrama caso de uso relacionado. Para todos los casos se omite los pasos iniciales de acceder al sistema y esperar a que cargue el *Splash*. Los usuarios empleados serán el cliente no registrado y el cliente ya que el sistema es el encargado de realizar los procesos— y hará, además, las funciones de administrador — y la ruta es un actor externo al desarrollo de nuestro proyecto.

Antes de poder acceder a la aplicación es necesario iniciar sesión y, anteriormente, haber realizado un registro. Para ello vamos a tener dos opciones: el registro manual o el registro por Twitter. El registro a su vez provoca que el sistema cree automáticamente los logros relacionados con dicho usuario iniciando su contador a cero. A continuación podemos ver los escenarios relacionados.

CU1 – Dar de alta un cliente

- Escenario Principal

El cliente no registrado accede a la actividad inicial del sistema. Tras ello se dirige a la vista de registro. Completa los campos requeridos y selecciona la opción registrar. El sistema verifica los datos de entrada y registra al cliente. Además crea los logros asociados a éste.

- Escenario Alternativo

El cliente no registrado accede a la actividad inicial del sistema. Tras ello selecciona la opción acceder con Twitter y autoriza el registro. El sistema obtiene los datos a través de Twitter y registra al cliente. Además crea los logros asociados a éste.

CU2 – Iniciar sesión

- Escenario Principal

El cliente accede a la actividad inicial del sistema. Introduce su usuario y contraseña y presiona “Entrar”. El sistema valida el usuario y éste inicia sesión.

- Escenario Alternativo

El cliente ya había iniciado sesión en el sistema o había sido registrado vía Twitter. El sistema, que ya contiene los datos de usuario realiza un *autologin*. El usuario ha iniciado sesión.

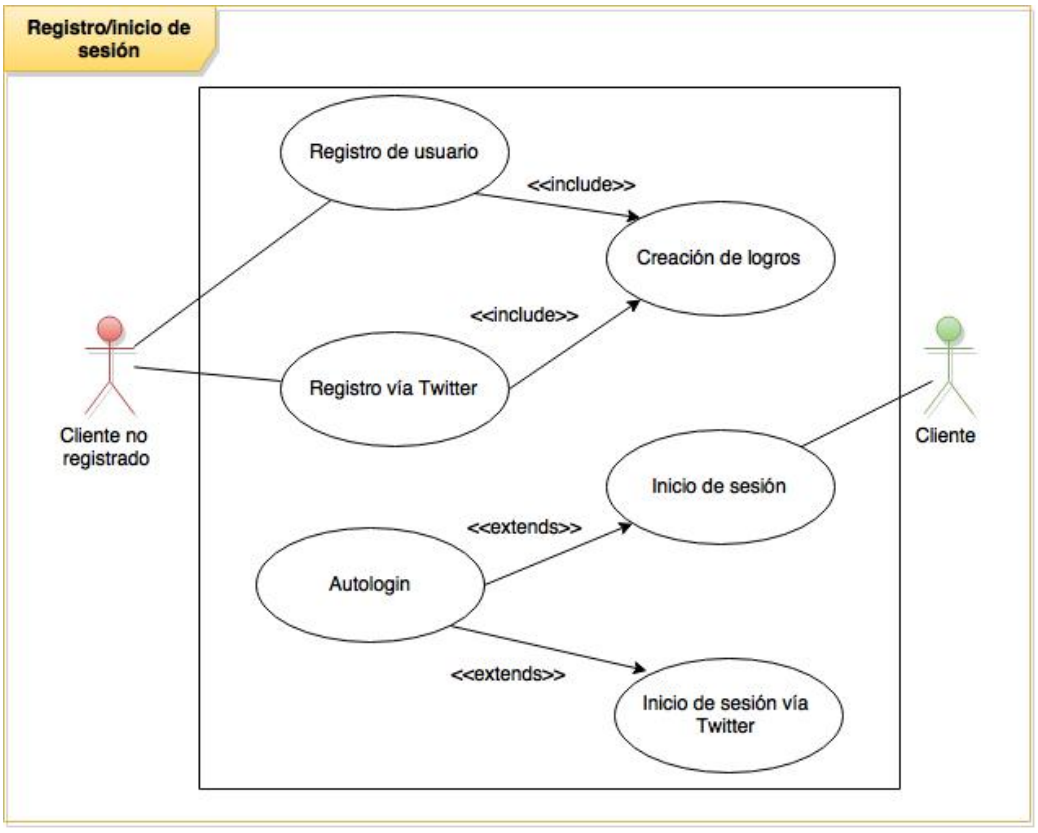


Figura 8. Proceso registro/inicio de sesión

Una vez se ha accedido a la aplicación, la funcionalidad disponible está enfocada entorno a una ruta en concreto. Para ello, inicialmente, se ha de elegir una ruta en concreto. Una vez seleccionada se podrán realizar acciones como ver sus bares o votar una tapa. El sistema también debe dar opción a salir de la ruta para poder escoger una nueva.

CU3 – Seleccionar una ruta

Precondiciones: haber iniciado sesión en el sistema.

- Escenario Principal

El cliente accede a la actividad principal del sistema. Aparece un listado de rutas para elegir. Selecciona una y, de este modo, el sistema queda asociado a una ruta en concreto.

- Escenario Alternativo

El cliente ya había seleccionado una ruta. Accede a la opción salir de ruta y tras ello vuelve aparece de nuevo el listado de rutas. El proceso a seguir a partir de aquí es el mismo que en el escenario principal.

CU4 – Ver bares de la ruta sobre un mapa

Precondiciones: haber iniciado sesión en el sistema, haber elegido una ruta.

- Escenario Principal

El cliente accede a la actividad principal del sistema. En ella se muestra un mapa con la localización de todos los bares.

- Escenario Alternativo

El cliente ya se encuentra en la aplicación en otra pantalla. Presiona la opción del menú “Ruta de la tapa”. La aplicación muestra la actividad principal donde se sitúa el mapa con los bares.

CU5 – Ver lista de bares participantes

Precondiciones: haber iniciado sesión en el sistema, haber elegido una ruta.

- Escenario Principal.

El cliente accede al sistema. Presiona la opción del menú “Bares y tapa”. El sistema muestra un listado con los bares participantes.

CU6 – Ver datos de un bar participante

Precondiciones: haber iniciado sesión en el sistema, haber elegido una ruta.

- Escenario Principal.

El cliente accede al sistema. Presiona la opción del menú “Bares y tapa”. El sistema muestra un listado con los bares participantes. El usuario presiona sobre uno de los bares disponibles. El sistema muestra a vista con los datos del bar.

- Escenario Alternativo.

El cliente se encuentra visualizando un bar. Presiona el botón atrás. El sistema muestra un listado con los bares participantes. El usuario presiona sobre uno de los bares disponibles. El sistema muestra a vista con los datos del bar.

CU7 – Votar una tapa

Precondiciones: haber iniciado sesión en el sistema, haber elegido una ruta, haber consumido una tapa en un bar para obtener su código QR.

- Escenario Principal.

El cliente accede a la vista de un bar. Presiona la opción “Votar”. El sistema accede a la cámara del dispositivo. El usuario debe leer el código QR con la cámara. Una vez leído el sistema da la opción de elegir una puntuación sobre 5. El consumidor califica la tapa. El sistema envía la votación y valida el voto.

CU8 – Crear un *tweet* de un bar.

Precondiciones: haber iniciado sesión en el sistema, haber elegido una ruta, tener iniciada la sesión en Twitter.

- Escenario Principal.

El cliente accede a la vista de un bar. Presiona la opción “#Tapéame”. Se abre la actividad para la creación del *tweet* con los *hashtag* #RutaDeLaTapa #NombreDelBar. El usuario escribe su *tweet* y lo *twittea*.

CU9 – Ver el trayecto a un bar

Precondiciones: haber iniciado sesión en el sistema, haber elegido una ruta.

- Escenario Principal.

El cliente accede a la actividad principal del sistema. En ella se muestra un mapa con la localización de todos los bares. Selecciona un bar del mapa. Presiona la opción de trayecto para que se abra la aplicación de mapas nativa del dispositivo Android.

- Escenario Alternativo, precondición añadida: estar en la vista bar.

El cliente accede a la vista de un bar en concreto. Presiona la opción localización. La aplicación le redirige a la vista de mapa de la ruta con el bar ya seleccionado. Presiona la opción de trayecto para que se abra la aplicación de mapas nativa del dispositivo Android.

CU10 – Ver clasificación final.

Precondiciones: haber iniciado sesión en el sistema, haber elegido una ruta, la votación está finalizada.

- Escenario Principal.

El cliente accede al sistema. Presiona la opción del menú “Clasificación”. El sistema muestra un listado con los bares ganadores.

- Escenario Negativo.

El cliente accede al sistema. Presiona la opción del menú “Clasificación”. La ruta no ha finalizado la votación, por lo que el sistema muestra un mensaje de votación en progreso.

CU11 – Salir/Cambiar de ruta.

Precondiciones: haber iniciado sesión en el sistema, haber elegido una ruta.

- Escenario Principal.

El cliente accede al sistema. Presiona la opción del menú “Salir de ruta”. El sistema borra los datos relacionados con la ruta y el usuario puede elegir una nueva.

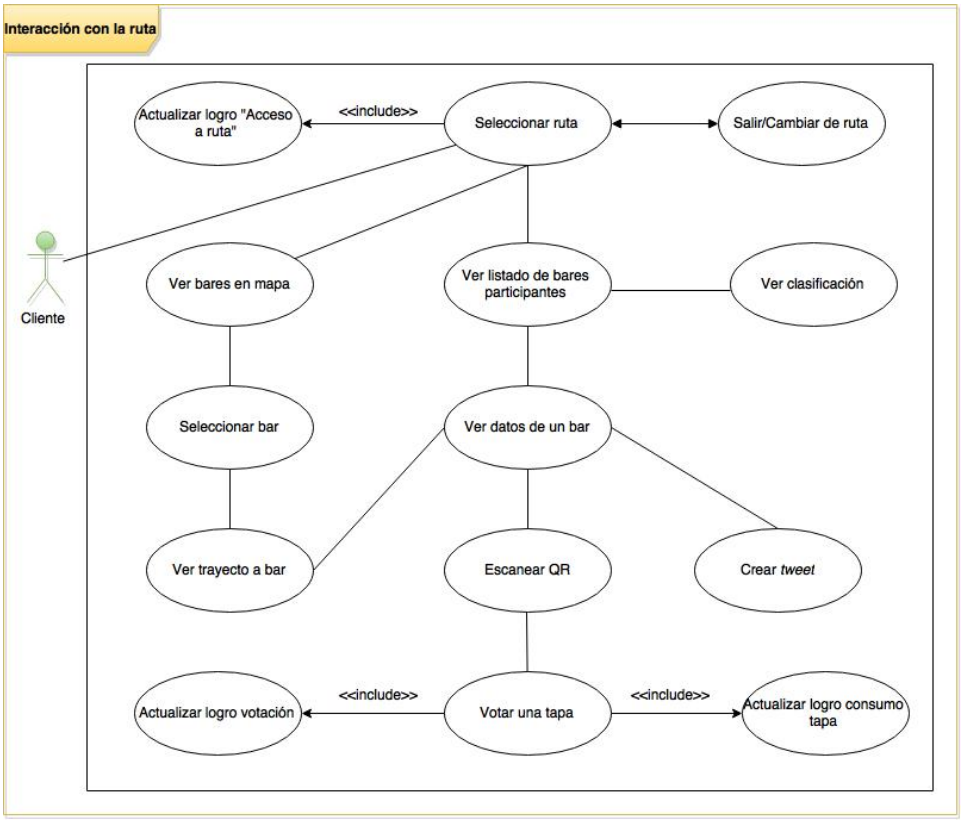


Figura 9. Proceso de interacción con una ruta

Por otro lado, también tenemos el sistema de logros. Éstos están asociados con funciones de la aplicación como acceder a una ruta. En la imagen anterior, la figura 13, se pueden observar los flujos que seguirá la aplicación para actualizar un logro. Por otra parte, la *app* permite ver el estado actual de dichos logros.

CU12 – Ver estado de los logros.

Precondiciones: haber iniciado sesión en el sistema.

- Escenario Principal.

El cliente accede al sistema. Presiona la opción del menú “Logros”. El sistema muestra un listado con los logros así como su estado actual. Si el logro ha sido conseguido se muestra en un color diferente.

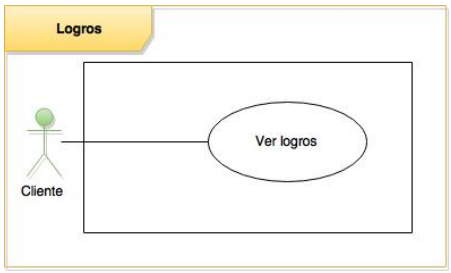


Figura 10. Proceso visualización de logros

Por último el usuario debe poder cerrar la sesión en curso.

CU13 – Cerrar sesión.

Precondiciones: haber iniciado sesión en el sistema.

- Escenario Principal.

El cliente accede al sistema. Presiona la opción del menú "Cerrar sesión". El sistema borra todos los datos guardados tanto del usuario como de la ruta actual en la que se encuentra. El sistema redirige a la actividad de iniciar sesión.

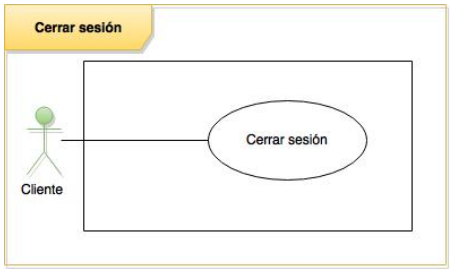


Figura 11. Proceso cierre de sesión

Finalmente, tras el análisis realizado en este aparatado, tenemos una vista de las entidades que participaran en el sistema así como las acciones que realizarán. En el siguiente capítulo se diseña el proyecto teniendo en cuenta todo el estudio realizado.

6. Diseño

Diseño no es solo lo que se ve o lo que se siente. Diseño es cómo funciona.

— Steve Jobs —

En este capítulo se contempla el diseño completo del proyecto. Para comenzar, se hace una introducción del esquema del producto que se va a desarrollar. Seguidamente, se hace un análisis por capas: presentación, persistencia y lógica, quedando definidos tanto la aplicación Android como el servidor.

6.1 Introducción

Como se puede observar en la figura 17, el proyecto queda dividido en dos partes diferenciadas. Por una parte tenemos los servidores con los que se comunica nuestra aplicación Android: el servidor de rutas y el servidor de control de usuario; y por otra parte las APIS: la Google Maps y la API de Twitter.

Figura 12. Estructura del sistema

En primer lugar tenemos el servidor externo al proyecto que se encarga de hacer las suscripciones de las rutas y sus bares. Está programado en lenguaje PHP. Anteriormente, en el apartado 4.1.5.1, han sido detallados los mensajes que se intercambian con dicho sistema para la obtención de los datos. Con él, obtenemos las rutas disponibles, los bares que las componen, sus posiciones… en definitiva toda la información relativa a una ruta en concreto.

En segundo lugar, tenemos nuestro servidor propio, desarrollado en node.js, que se encarga de la gestión de usuarios. Realiza las funciones tanto de registro como de inicio de sesión y, además, lleva a cabo la gestión de logros: desde su inicialización al crear un nuevo usuario, como el posterior seguimiento. Además también registra los posibles usuarios que accedan vía Twitter.

En tercer lugar, se encuentra la API de Google Maps. Mediante la comunicación con ésta, podemos gestionar todo lo relativo a los mapas en nuestra aplicación, así como, la geolocalización vía GPS, si se encuentra disponible, o mediante internet. Permite, además, centrar el mapa principal sobre la vista de la ruta y posicionar los bares en ella en sus respectivas localizaciones. Además si se selecciona un bar en el mapa, permite comunicar con la aplicación nativa de mapas de Android, con la cual es posible obtener el trayecto desde la posición actual del usuario hasta el bar.

Por último, también se hace uso de la API de Twitter. Desde ella es posible el registro automático siempre que se haya autorizado. De esta forma, estamos accediendo a la aplicación vía *oAuth* — autorización abierta — sin necesidad de hacer uso del registro manual de la aplicación. Con esta opción, se obtienen los datos para ser mostrados en la aplicación directamente del usuario de Twitter, como son el nombre o la foto de perfil. Asimismo, permite escribir *tweets* desde la propia aplicación de *La ruta de la tapa*, añadiendo además un *hashtag* propio — #RutaDeLaTapa — y uno personalizado para cada bar — su nombre —.

Otro aspecto a tener en cuenta es la estructuración del sistema. Ha de tenerse en cuenta tanto el lado del servidor como el lado cliente Android. Para ello partimos de un modelo clásico como es el MVC en el conjunto global del proyecto. En ambos lados del sistema se definen los modelos necesarios para su correcto funcionamiento así como los controladores para realizar las acciones necesarias y dotar al producto de la funcionalidad

deseada. Las vistas quedan definidas en la aplicación Android, ya que es con la que interactuará el futuro usuario.

Los modelos, hacen referencia a las representaciones creadas en las bases de datos. Por su parte, el controlador es el encargado de gestionar la lógica de la aplicación respondiendo a peticiones realizadas por el usuario. También permite la comunicación entre los distintos sistemas que interactúan con la aplicación. Por último, las vistas son la interfaz del usuario con la que puede realizar toda la funcionalidad ofrecida.

A continuación se descompone el producto en capas definiendo su comportamiento y la forma de mostrar la información.

6.2 Capa de presentación

Para el diseño de la capa de presentación se ha hecho uso de la herramienta Justinmind que permite un prototipado del sistema. Gracias a este instrumento, se pueden crear vistas muy aproximadas a su impresión final en el dispositivo al que va dirigido, en nuestro caso, dispositivos móviles de tipo Android. Además, permite simular el proyecto creado permitiendo interactuar entre las pantallas.

A continuación se muestran las pantallas que debe contener la aplicación *La ruta de la tapa*. Cabe destacar que pueden tener diferencias con el resultado final ya que no deja de ser un prototipo de cómo deberían estar distribuidas las pantallas y el aspecto que deberían presentar. Por ejemplo, los iconos, han sido elegidos entre los que ofrece esta herramienta. Además, en un primer diseño, se eligió como color secundario el verde — que es el mostrado a continuación—, sin embargo, finalmente este color ha cambiado a un tono grisáceo.

Al abrir la aplicación la primera pantalla a mostrar será el *Splash* donde se muestra el logotipo de la aplicación.

Figura 13. Pantalla de *Splash*

Una vez transcurrida la carga de la aplicación aparece la pantalla de inicio de sesión que a su vez deberá enlazar con las distintas posibilidades de registro.

Figura 14. Pantallas inicio de sesión / registro

Tras ser autorizado para acceder a la pantalla principal, vemos la vista del mapa.

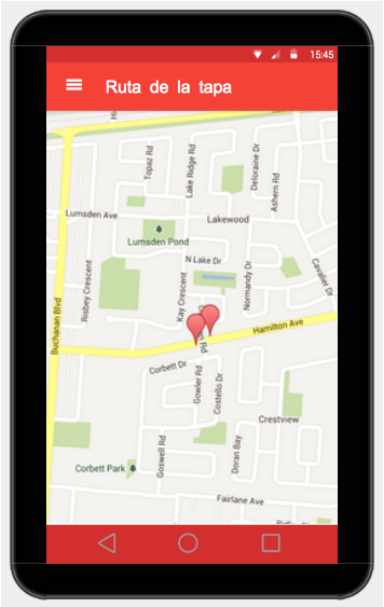


Figura 15. Pantalla principal: mapa

Para acceder al resto de las funcionalidades, se hace uso del menú. En el encontramos además de la opción de ver la ruta la de listar los bares, ver la clasificación, ver los logros y cerrar sesión. También será añadida la opción de salir de la ruta en la versión final de la aplicación. Además se puede observar que se muestra el nombre del usuario, el correo o la cuenta de Twitter con la que ha accedido y la foto si se ha registrado mediante Twitter.

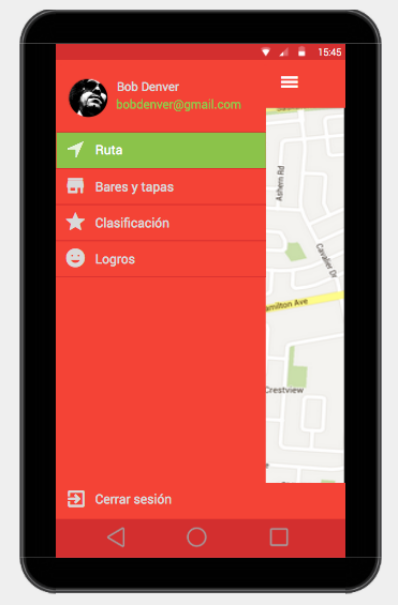


Figura 16. Menú de la aplicación

Tanto la sección de ver bares, como la de clasificación, como la de logros se muestra en forma de lista.

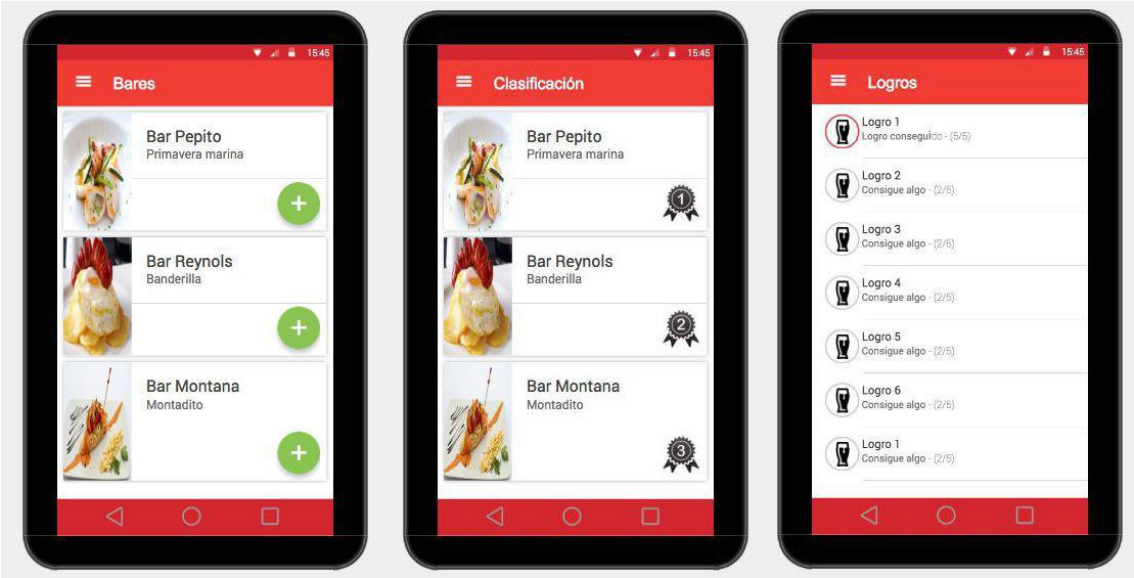


Figura 17. Vistas bares, clasificación y logros

Por último tenemos la pantalla de un bar. En ella además de ver la información del participante, también se encuentran las opciones de crear un *tweet* y de votar una tapa.

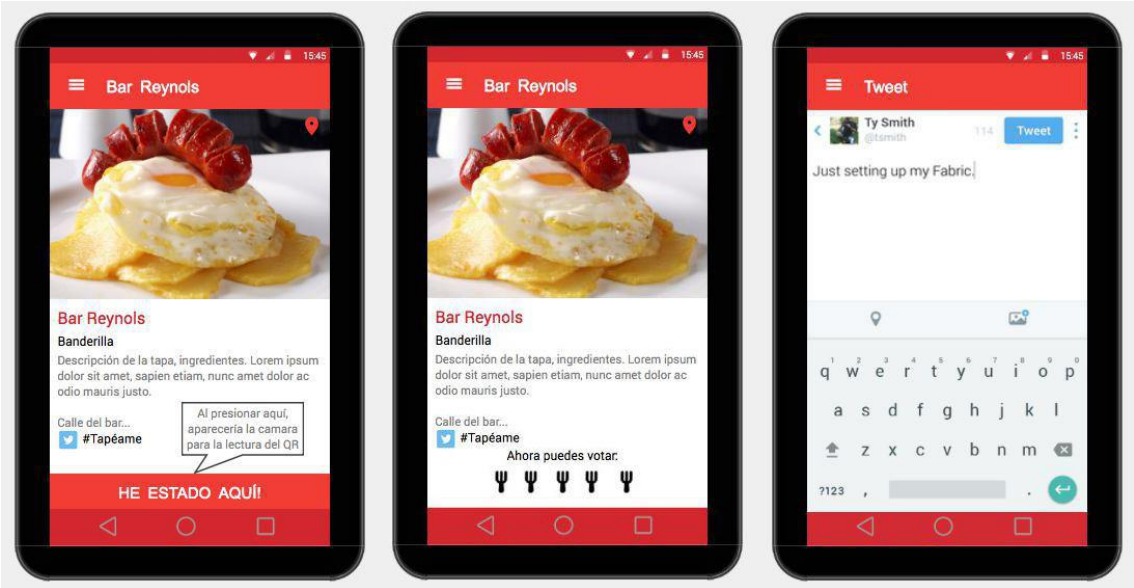


Figura 18. Pantalla bar participante

6.3 Capa de persistencia

La definición de la capa de persistencia se encuentra tanto en el servidor como en la aplicación Android y cada una cuenta con sus entidades propias. En cuanto al servidor, la

restricción de uso de una base de datos no relacional provoca que los datos queden relacionados de forma indirecta mediante un campo incluido en la entidad de logro, es decir, existen dos entidades, usuarios y logros, y un usuario tiene varios logros.

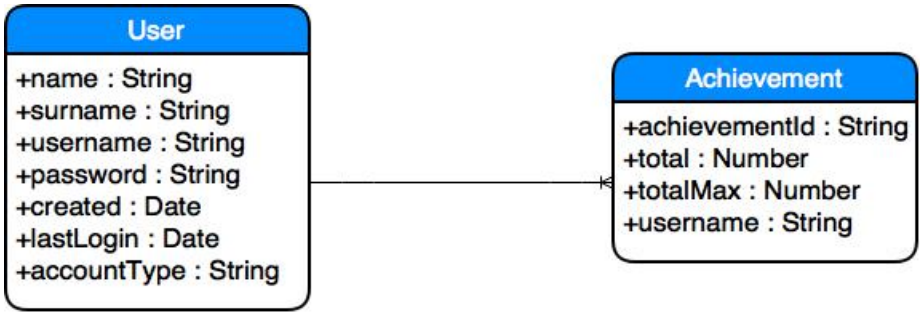


Figura 19. Entidades servidor

El usuario tiene los siguientes campos:

- *Name*: nombre del usuario
- *Surname*: apellido del usuario
- *Username*: email o cuenta de twitter, debe ser único en el sistema
- *Created*: fecha de registro
- *LastLogin*: fecha de último acceso
- *AccountType*: tipo de cuenta, puede ser ‘default’ en caso de registro habitual o ‘twitter’ si se ha accedido a través de esta red social.

En cuanto a los campos de la entidad logro:

- *AchievementId*: identificador de logro, coincide con el que contiene la aplicación Android
- *Total*: número actual de acciones conseguidad
- *TotalMax*: número de acciones a conseguir
- *Username*: usuario al que pertenece

También tenemos el lado cliente, la aplicación Android. A tratarse de un proyecto Java se hace uso de clases y métodos durante la implementación. Éstos completan las clases que ya se vieron en el apartado 5.1. La estructura final quedaría de la siguiente forma:

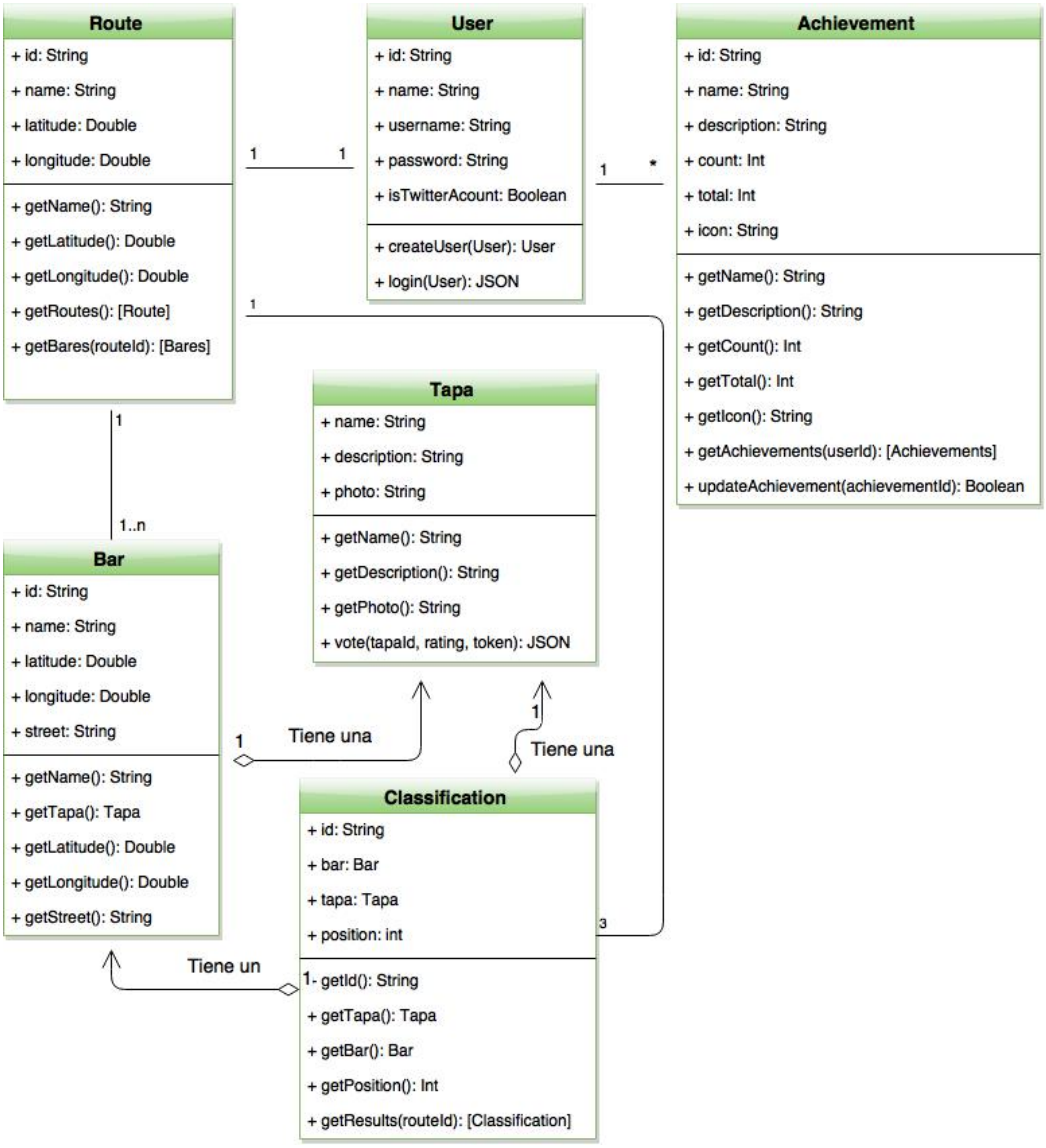


Figura 20. Entidades aplicación móvil

En ella interesa que la interacción sea lo más fluida posible por lo que los datos de usuario y de identificador de ruta serán guardados directamente en preferencias en forma clave-valor. Además, los datos de los logros vienen predefinidos en un fichero de tipo xml. Por tanto, la única tabla que se encuentra en el sistema es la de los bares, que a su vez contiene también la tapa propia del bar. Esta tabla contiene los siguientes campos, todos de tipo :

- *_ID*: identificador único de participante

- *Name*: nombre del bar
- *Latitude*: latitud de la localización
- *Longitude*: longitud de la localización
- *Street*: calle en la que está situado
- *TapaName*: nombre de la tapa
- *TapaDescription*: descripción del plato
- *TapaImage*: url de la foto de la tapa

6.4 Capa lógica

La capa lógica o de negocio define los flujos que debe seguir un proceso para el correcto funcionamiento del sistema. En ella, debe quedar recogida toda la funcionalidad definida mediante los casos de uso. Para ello, analizamos dichos flujos atendiendo a los indicadores que hemos asignado en el apartado 5.2. A continuación se explican las acciones que realiza el sistema para llevar a cabo cada acción.

*** Cuando hablemos del servidor Icarus, nos estaremos refiriendo al servidor externo al proyecto ***

CU1-Dar de alta un cliente: existen dos opciones. Si el cliente realiza el registro normal, la aplicación debe recoger los datos de la pantalla de registro, validarlos y enviarlos al servidor. Éste debe comprobar que no falta ningún campo, guardarlos — es decir, crear un usuario —, crear los logros e iniciar sesión. Si todo es correcto, la aplicación continuará a la pantalla principal. En cambio, si se registra vía Twitter, aparecerá la pantalla de autorización propia de Twitter. Si el usuario acepta se producirá el mismo proceso anterior tras obtener los datos de la cuenta de Twitter.

CU2-Iniciar sesión: si el usuario ya ha iniciado sesión anteriormente o se ha registrado vía Twitter, este proceso es automático. Si no, la app recoge los datos de la pantalla de inicio de sesión, los envía al servidor y éste valida si es un usuario registrado y la contraseña es correcta.

CU3-Seleccionar una ruta: la aplicación debe comunicarse con el servidor *Icarus* para obtener las rutas. Si se puede obtener la posición actual del dispositivo, enviará la localización para que las rutas vengan filtradas por cercanía. Una vez el usuario elija una ruta, vuelve a realizarle una petición para obtener todos los datos de los bares y las tapas que pertenecer a dicha ruta. Además se actualiza el logro visitar ruta.

CU4-Ver bares sobre mapa: se crea un mapa en la pantalla principal de la aplicación y con los datos anteriormente obtenidos se posicionan todos los bares mediante marcadores.

CU5-Listar bares: se crea una lista de los bares con los datos anteriormente cargados.

CU6-Ver datos bar participante: al presionar sobre un bar, aparece su vista. Se busca en la base de datos el bar por id y se distribuye la información en la pantalla. También se ha de descargar la foto para mostrarla.

CU7-Votar una tapa: en la vista de bar, al presionar el botón votar, la aplicación abre la aplicación de la cámara. Tras leer un QR, se permite que el usuario vote. Una vez calificada la tapa, se envía al servidor *Icarus* para su validación. Si ha sido correcta se muestra un mensaje de agradecimiento, sino uno de fallo.

CU8-Crear un *tweet*: en la vista de bar, al presionar el botón de creación de *tweet* se abre la pantalla de Twitter para escribirlo. Si se presiona enviar el *tweet* aparecerá en el muro del usuario.

CU9-Ver el trayecto: existen dos opciones. En la vista del mapa de la ruta al hacer presionar un marcador, éste se abre. Aparece un botón con la opción de abrir la aplicación nativa de mapas de Android para obtener el trayecto. La segunda opción, es, desde la vista propia del bar, presionar el botón de localización. Tras ello, la aplicación redirige directamente a la vista del mapa con el bar ya seleccionado.

CU10-Ver clasificación final: la aplicación se comunica con el servidor *Icarus* para obtener la clasificación. Si ha finalizado, se muestra un listado con las tres primeras posiciones de los bares ganadores. Si no, se muestra un mensaje de votación en curso.

CU11-Salir cambiar de ruta: al realizar dicha acción se eliminan todos los datos relativos a una rato tanto de base de datos como de preferencias. Se ha de mostrar de nuevo la selección de ruta.

CU12-Ver estado de logros: la aplicación se comunica con nuestro servidor para obtener el número de acciones realizadas por logro. El nombre, la descripción y la imagen las obtiene de un fichero xml dentro del propio proyecto Android. Tras recopilar los datos, los muestra en forma de lista.

CU13-Cerrar sesión: la aplicación elimina todos los datos guardados, tanto de ruta como de usuario y muestra la pantalla de inicio de sesión.

En síntesis queda definida la estructura del sistema, pudiendo pasar a su implementación. Han sido definidas tanto las pantallas, como los modelos que deben estar en las bases de datos, así como los flujos a seguir por parte de los procesos. En el capítulo siguiente, se explica las tecnologías utilizadas y como queda el directorio del proyecto. También se hace una pequeña explicación de los contenidos más relevantes.

7. Detalles de implementación

La mejor forma de predecir el futuro es implementarlo.

— David Heinemeier Hansson —

En este capítulo se tratan detalles de la propia implementación del proyecto. Para comenzar, se hablará de las tecnologías empleadas haciendo hincapié en node.js, la parte servidor, ya que el sistema Android se ha explicado con más detenimiento en el capítulo de estado del arte. También se detallan las tecnologías empleadas en las bases de datos. Seguidamente, se muestra el sistema de directorios de los proyectos y se hace una breve explicación de su uso así como de librerías o módulos necesarios para el desarrollo.

7.1 Tecnologías utilizadas

7.1.1 Node.js

En la actualidad, existen un gran número de lenguajes del lado de servidor entre los que destacan algunos como PHP, Perl o ASP.Net, sin embargo, nos hemos decantado por el uso de Node.js.

Node.js es una plataforma basado en el motor JavaScript de Chrome para construir aplicaciones de red altamente escalables desarrolladas en lenguaje JavaSript. En vez de crear hilos para cada una de las peticiones, con el gasto de memoria que eso conlleva, cada nueva conexión se realiza mediante la creación de un evento dentro del proceso del motor de node. Esto, unido al modelo de no bloqueo de peticiones entrada-salida que emplea, hace que, ante un gran número de conexiones, el sistema resulte más ligero y efectivo. Por tanto, para aplicaciones que necesitan distribuir datos en tiempo real en distintos dispositivos, resulta un dispositivo muy eficiente.

¿Y cómo maneja node estos eventos? Cuando una aplicación node necesita realizar una operación I/O envía una tarea asíncrona al *event loop*. Ese “bucle de eventos” atiende todas las peticiones, las gestiona de forma eficiente y las va solucionando sin realizar ningún bloqueo. Cuando alguna de estas acciones finaliza, este gestor envía un callback con los resultados y la tarea inicial recibe los datos que necesitaba. Este flujo se puede observar en la figura 25.

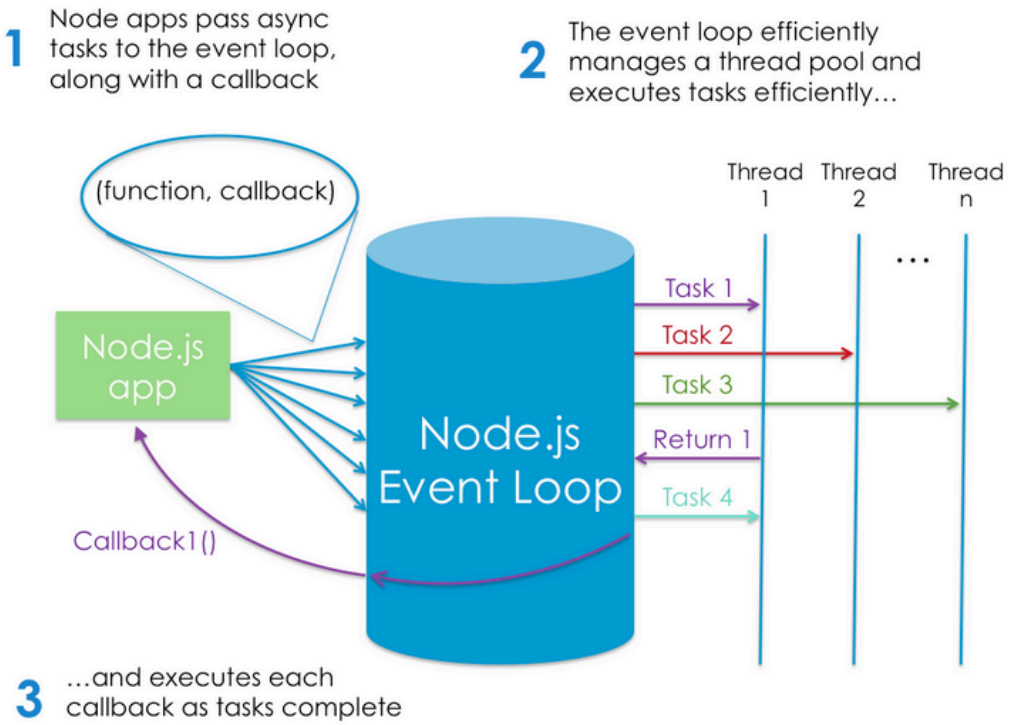


Figura 21. Event loop en node.js, recuperado del libro: *The node beginner book*

Dicha ejecución, permite manejar un gran número de procesos, así como un ahorro considerable de memoria, al no crear un hilo para cada una de las peticiones como hemos comentado anteriormente en este mismo apartado. El Event loop es el encargado de gestionar los subprocesos para que estos finalicen en el menor tiempo posible, lo que significa, que solo es necesario preocuparse por el correcto tratamiento del callback para un uso eficiente de los recursos.

Por último cabe destacar que en las últimas versiones de node en la actualidad, siendo la más reciente la v0.12.4, está soportado el uso de generadores incluido en la versión V8 JavaScript Engine. Estas funciones generadoras pueden llamarse múltiples veces reanudando su ejecución en varios puntos dentro de una misma función. Aunque el concepto puede resultar un poco complejo, en la práctica este tipo de funciones permiten escribir código de forma síncrona que realmente se está ejecutando de forma asíncrona. En

el código, esto se traduce como la sentencia yield delante de la llamada a la función, lo que produce que, aunque la llamada ejecute procesos asíncronos, el código no continúe con las operaciones siguientes hasta que no sea recibido el resultado de dicha operación.

7.1.1.1 Ventajas

Por una parte, como hemos comentado en la introducción, node es un sistema muy ligero y rápido para peticiones entrada salida, por lo que, para consultas en bases de datos — que es lo que se busca que ofrezca el servidor de nuestra aplicación — resulta altamente eficiente. Además, si el sistema creciera, node.js es en su definición altamente escalable.

Por otra parte, contiene un gran número de módulos que permiten agregar funcionalidad al servidor mediante una instalación muy simple con el uso del comando npm.

Uno de los más destacados, y usado en este proyecto, es el módulo express, con el cual manejar peticiones HTTP. También, existe una gran comunidad tras node.js que se encarga de documentar y desarrollar continuamente nuevos módulos o actualizando los existentes.

Otro elemento a destacar, es que el servidor es fácilmente testeable. Se puede simular llamadas erróneas para ver cómo reacciona el sistema, así como peticiones correctas para comprobar si el resultado es el esperado. Por ejemplo, uno de los módulos que nos permite hacer estas pruebas es Jasmine, que es usado en el capítulo 9 para probar nuestro propio servidor.

Y, por último, JavaScript es un lenguaje realmente sencillo e intuitivo. Al no tener tipos de objeto hace que el manejo de datos sea muy liviano. También tiene como ventaja que, al ser un lenguaje de tipo interpretado y no compilado, la independencia de la plataforma donde es ejecutada.

7.1.2 Protocolo de comunicación: HTTP

El protocolo de transferencia de hipertexto — HTTP, *HyperText Transfer Protocol* — es el protocolo empleado en la capa de aplicación de la Web y se encuentra definido en el documento RFC 723X. HTTP consta de dos partes: el programa cliente, que realiza peticiones de recursos; y el programa servidor que resuelve dichas peticiones y envía la

información que ha sido consultada. Este protocolo define cómo se realiza dicha comunicación.

HTTP emplea a su vez como protocolo de transporte TCP. En primer lugar el cliente debe iniciar la conexión TCP con el servidor. Una vez establecida, los procesos cliente y servidor se pueden comunicar. Asimismo, como TCP ofrece un servicio de transferencia de datos fiable, HTTP por herencia también lo tiene. Esto significa que los mensajes entre los pares no deben sufrir ningún tipo de modificación. Este proceso queda descrito con un gran nivel de abstracción en la figura 4.

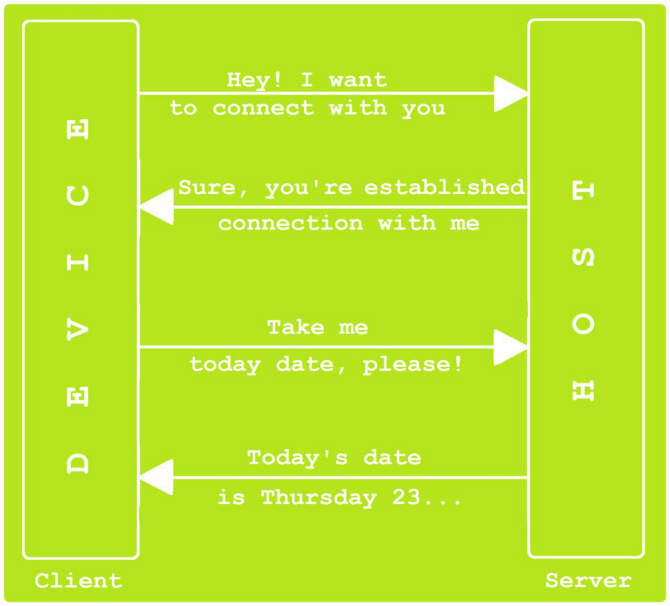


Figura 22. Proceso de intercambio de mensajes entre cliente y servidor

Otra de las características de este protocolo es que no tiene memoria de estado. Si un cliente realiza una petición y a continuación vuelve a realizar la misma, el servidor contesta ambas veces, por lo que no es necesario mantener ninguna información adicional de los distintos clientes.

La comunicación vía HTTP se realiza por el paso de mensajes que se describen a continuación.

7.1.2.1 Mensajes de solicitud HTTP

Un mensaje de solicitud queda dividido en cuatro partes como se observa en la figura 5: línea de petición, cabeceras, línea en blanco y cuerpo del mensaje.

Figura 23. Estructura mensaje de solicitud HTTP

En la línea de solicitud consta de tres campos: el método, el campo URI — identificador de recursos único — y la versión de HTTP. En cuanto a los métodos pueden ser de tipo GET, POST, PUT, HEAD o DELETE. Éstos adquieren significado cuando hacemos uso de APIs REST, como se explica en el apartado 3.3.1 de este mismo capítulo.

Las cabeceras son del tipo clave valor y se representan de forma. Éstas contienen información sobre la petición como el host al que van dirigidos, el tipo de contenido del cuerpo del mensaje o el lenguaje de respuesta que espera.

La línea en blanco, permite separar el cuerpo del mensaje del resto de la petición. Este cuerpo no siempre tiene contenido. Se rellena cuando es necesario enviar datos adicionales para la petición y, normalmente, se usa el método POST. A continuación, en la figura 6, se muestra un ejemplo de mensaje de solicitud HTTP.

```
Client request:

  GET /hello.txt HTTP/1.1
  User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
  Host: www.example.com
  Accept-Language: en, mi
```

Figura 6. Ejemplo mensaje de solicitud http, recuperada de: tools.ietf.org/html/rfc7230

7.1.2.2 Mensajes de respuesta HTTP

Seguidamente se puede ver en la figura 7 el esquema de un mensaje de respuesta. Se puede observar que la principal diferencia es el cambio de la línea de petición por la línea de estado.

Figura 24. Estructura mensaje de petición HTTP

En esta ocasión la primera línea es la de estado que es del tipo versión HTTP, código de estado y mensaje de estado. El código de estado es numérico y está asociado al mensaje. Los estados principales son:

- 200 *OK*. La solicitud ha sido procesada correctamente y se ha devuelto la información en el mensaje respuesta.
- 400 *Bad Request*. La petición no ha sido entendida por el servidor, es decir, la URI especificada ha sido atendida pero falta algún tipo de información adicional.
- 401 *Unauthorized*. Algunos accesos pueden ser restringidos. En caso de pedir información reservada sin tener el permiso de usuario adecuado, se devuelve este código de error.
- 404 *Not Found*. La petición no corresponde a ningún recurso ofrecido por el servidor.
- 500 *Internal Server Error*. Error producido internamente en el servidor que ha incapacitado el poder servir una respuesta correcta.

También destacar que el cuerpo del mensaje ahora contiene la información que había sido demandada. Un posible ejemplo de respuesta podría ser la que se muestra en la figura 8.

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.
```

Figura 25. Ejemplo mensaje de petición HTTP, recuperada de: tools.ietf.org/html/rfc7230

7.1.3 APIs REST

Una API REST es una interfaz de programación de aplicaciones que contiene un conjunto de subrutinas, funciones y procedimientos que son ofrecidos para ser consumidos por otra aplicación bajo la arquitectura REST.

Por su parte, REST es una arquitectura para desarrollo web, basada en HTTP y que permite prestar los servicios propios de una API mediante el intercambio de mensajes.

REST se basa en tres reglas principales: el uso correcto de las URIs, el uso correcto de HTTP e *hypermedia*. A continuación se explica a que se refiere cada una de estas reglas.

7.1.3.1 Reglas

En primer lugar, en cuanto a las **URIs** existen las siguientes reglas propias al respecto:

1. Implican un recurso, no una acción. Por ejemplo la URI /obtener/bar/pepito, sería incorrecta. La forma correcta sería /bar/pepito ya que la acción quedea especificada por el método de la petición. Las acciones por método son explicadas con más detenimiento en este mismo apartado más adelante.
2. Son únicas. La URI /bar/pepito siempre debe hacer referencia al mismo objeto, aunque esto no significa que el objeto no puede sufrir modificaciones.
3. Independencia del formato. Aunque el formato de recepción sea de tipo JSON, XML, pdf o cualquiera de los posibles, no debe estar implícita en la URI. Por tanto /bar/pepito.json sería incorrecto. Este aspecto se relaciona con la regla de hypermedia.
4. Orden jerárquico adecuado. Debe mantener un orden jerárquico lógico. Siguiendo el ejemplo de *La ruta de la tapa* un orden correcto sería /ruta/valencia/bar/pepito y, por el contrario, /bar/pepito/ruta/valencia sería incorrecto.
5. Filtrado y otras operaciones. Para realizar filtrados u ordenar listas, es más adecuado pasar las opciones por parámetros de la petición y no en la petición en sí misma.

En segundo lugar tenemos las reglas **HTTP**. Como ya se ha tenido en cuenta en el apartado de HTTP, los métodos están directamente relacionados con este tipo de APIs. Por tanto un uso correcto de los diversos métodos proporciona información adicional al tipo de

petición que se está realizando. Estos métodos se relacionan con acciones de la forma que sigue:

- GET, obtener un recurso.
- POST, crear un recurso.
- PUT, modificar un recurso.
- DELETE, borrar un recurso.

Además hay que tener en cuenta un correcto uso de los códigos de estado ante peticiones. Como ya se ha explicado en el apartado de HTTP anterior, los códigos de error se asocian a lo sucedido al procesar la petición. Por tanto, no sería lógico enviar un 404 — *Not found* — cuando la petición ha sido procesada correctamente. Además en caso de fallo, es aconsejable incluir un mensaje de error en el cuerpo del mensaje para una mayor explicación de lo acontecido.

En tercer y último lugar tenemos ***hypermedia***. Este término viene a asociarse con el tipo de recurso que queremos obtener. Al ser información añadida sobre el recurso, no debe estar incluida en la línea de solicitud sino que debe incluirse como cabeceras. En el caso de la petición, la cabecera *Accept* indica al servidor en qué formato desea obtener un recurso. Puede indicar un único formato o varios. De esta forma queda claro que se está haciendo la petición a un recurso único aunque su representación pueda estar dada en distintos formatos. Por parte del servidor, la cabecera *Content-Type*, indica el formato final en el que han sido enviados los datos.

Sintetizando, tanto el uso del protocolo HTTP como el de las APIs REST forman un buen conjunto para el intercambio de mensajes mediante nuestra aplicación y los servidores de los cuales obtiene información. En el apartado 4.1.5.1 se detallará la API empleada para recoger los datos de las rutas. En el X se explicará la usada para la gestión de usuarios.

7.1.3.2 API servidor proyecto Icarus

Para el uso de la API del servidor del proyecto Icarus, es necesario el intercambio de mensajes entre la aplicación Android y dicho servidor. Para ello vamos a mostrarlo en dos partes: por una parte, las peticiones necesarias para obtener los datos, por otra parte, los

mensajes que se intercambian a través de dichas peticiones. Por tanto vamos a organizar ver las peticiones de ruta, de bares y tapas y de votaciones y clasificación.

Peticiones al servidor

Para los diversos grupos anteriormente descritos vamos a incluir una tabla en la cual se muestran los campos de método, ruta relativa y la descripción. En cuanto al método, al ser peticiones de datos, será GET[4], más concretamente con la sección de reglas. En cuanto a la ruta, se va a mostrar únicamente la ruta relativa, es decir, no se mostrará la dirección completa del servidor para ocultar su IP. Por último la descripción aclara el uso con el que se relaciona la petición.

Método	Ruta	Descripción
GET	/api/routes	Obtiene todas las rutas disponibles. Según el contenido del mensaje, las rutas vienen filtradas por posición más cercana a la dada
GET	/api/routes/:idRuta	Recopilación de bares y tapas pertenecientes a una ruta en concreto
GET	/api/results/:idRuta	Consulta del estado de la votación. Si ha finalizado, además, contiene los participantes que han sido premiados
GET	/api/vote/:idRuta/:token/:puntuación	Envía la calificación otorgada por un usuario a una tapa. Es necesario el *token* para su posterior validación en el servidor.

Tabla 7. Peticiones al servidor de rutas

En cuanto al intercambio de mensajes, éstos son analizados en las siguientes tablas. La línea de cabecera numera los mensajes. En cuanto al formato define el contenido esperado del cuerpo del mensaje, en este caso JSON. Sobre el nombre del documento — Nombre Doc. —, damos titulo a las peticiones descritas anteriormente. Seguidamente tenemos el campo de parámetros, datos necesarios para el posterior tratamiento de la petición. Y, por último, tenemos el esquema de respuesta, donde se muestra el contenido del cuerpo de mensaje que recibe la aplicación Android.

** Los símbolos de [], corresponden a un vector — conjunto de objetos — y el de {} a un JSON.

4 Existe la excepción en la votación. Este método sería más correcto si fuera un POST.

M1			
Formato	JSON	**Nombre Doc.**	Obtener rutas disponibles
Parámetros	Opcionales, en el cuerpo del mensaje: {"latitude" : latitud, "longitude" : longitud}		
Esquema de respuesta	[{"id" : id ruta, "name" : nombre de la ruta, "latitue" : latitud, "longitude" : longitud}]		

Tabla 8. Mensaje de la petición para obtener las rutas

M2			
Formato	JSON	**Nombre Doc.**	Obtener datos de los bares
Parámetros	En la ruta, :idRuta, identificador único de ruta		
Esquema de respuesta	[{"id" : id bar, "name" : nombre del bar, "latitue" : latitud del bar, "longitude" : longitud del bar, "address" : dirección del bar, "tapaName" : nombre de la tapa, "tapaDescription" : descripción de la tapa, "tapaImage" : ruta de la imagen de la tapa}]		

Tabla 9. Mensaje de la petición para los datos de los participantes de una ruta

M3			
Formato	JSON	**Nombre Doc.**	Obtener resultados ruta
Parámetros	En la ruta, :idRuta, identificador único de ruta		
Esquema de respuesta	{"finished" : estado(puede ser *true* o *false*), "rewards" : [{"tapaId" : id tapa, "position" : clasificación}]}		

Tabla 10. Mensaje de la petición para los datos de los participantes de una ruta

M4			
Formato	JSON	**Nombre Doc.**	Votar una tapa
Parámetros	En la ruta, :idRuta, identificador único de ruta, :token, identificador único de voto válido, :puntuación, puntuación otorgada a la tapa		
Esquema de respuesta	{"status" : estado(puede ser *ok* o *failed*)}		

Tabla 11. Mensaje de la petición para la votación

7.1.3.3 API servidor proyecto desarrollado

Para el uso de la API del servidor que se ha desarrollado en este proyecto, es necesario, igual que en el caso anterior, el intercambio de mensajes entre la aplicación Android y dicho servidor. Por ello se seguirá la misma metodología para mostrar tanto las peticiones a realizar como los resultados obtenidos.

Peticiones al servidor

En cuanto a las peticiones, se pueden clasificar en dos grupos: las peticiones a usuario y las peticiones a logros. En la tabla que se muestra a continuación, tanto la petición de registro de usuario como la de inicio de sesión:

Método	Ruta	Descripción
POST	/api/user	Recibe los datos en relación a un nuevo usuario. Una vez validados, crea un nuevo usuario y sus logros asociados.
GET	/api/user	Comprueba que el usuario se encuentre registrado y, si es así, inicia sesión.

Tabla 12. Peticiones al servidor para usuarios

También tenemos las peticiones relacionadas con la gestión de logros. En esta ocasión tenemos tres tipos de llamadas: la de creación de un logro — solo puede realizarlo un sistema autorizado —, la petición de logros sobre un usuario y su actualización.

Método	Ruta	Descripción
POST	/api/achievement	Recibe los datos en relación a un nuevo logro. Una vez validados, crea un logro asociado a un usuario. Por ahora, esta opción solo la puede realizar el propio sistema.
GET	/api/achievement/:username	Obtiene los datos de los logros relacionados con un usuario concreto.
PUT	/api/achievement/:achievementID/user/:username	Actualiza el estado de un logro, añade una acción mas al contador actual del logro especificado.

Tabla 13. Peticiones al servidor para logros

En cuanto al intercambio de mensajes, los siguientes son los específicos de nuestra aplicación y serán analizados como en el apartado anterior. Solo se contemplan los casos en los que el mensaje se haya procesado correctamente, los mensajes de error se verán en el apartado de pruebas.

M1			
Formato	JSON	**Nombre Doc.**	Crear usuario
Parámetros	{“name”: nombre, “username”: cuenta usuario, “password”: contraseña} Opcionales: {“surname” : apellidos, “accountType” : por defecto es ‘default’}		
Esquema de respuesta	{“id” : id de usuario}		

Tabla 14. Mensaje de la petición para crear un usuario nuevo

M2			
Formato	JSON	**Nombre Doc.**	Iniciar sesión
Parámetros	{“username” : cuenta usuario, “password” : contraseña, “isTwitterAccount” : booleano }		
Esquema de respuesta	{“isAuth” : true, “id” : id de usuario,“name” : nombre del usuario, “username” : cuenta usuario}		

Tabla 15. Mensaje de la petición para iniciar sesión

M3			
Formato	JSON	**Nombre Doc.**	Crear logro
Parámetros	{"achievementId" : id de logro, "total" : acciones totales conseguidas, "totalMax" : total acciones a conseguir, "username" : usuario al que pertenece },		
Esquema de respuesta	El propio logro		

Tabla 16. Mensaje de la petición para crear un nuevo logro

M4			
Formato	JSON	**Nombre Doc.**	Obtener logros
Parámetros	En la ruta, :username, identificador único de usuario		
Esquema de respuesta	[{logros}], mismos parámetros que mensaje anterior		

Tabla 17. Mensaje de la petición de logros de un usuario

M4			
Formato	JSON	**Nombre Doc.**	Actualizar logro
Parámetros	En la ruta, : achievementID, id del logro a actualizar; :username, identificador único de usuario		
Esquema de respuesta	{logro}, mismos parámetros que mensaje anterior pero con la cuenta actualizada		

Tabla 18. Mensaje de actualización de logro

7.1.4 Bases de datos

En cuanto a las bases, se han empleado dos tecnologías distintas: en el lado del servidor se ha usado MongoDB y en Android SQLite.

MongoDB es una base de datos orientada a documentos. Esto quiere decir que los datos no son guardados en registros, sino que se almacenan en ficheros de tipo BSON[5]. Una de sus características principales es que no es necesario seguir un esquema. Una misma colección de datos puede tener objetos que no contengan los mismos atributos. Además permite almacenar como atributo un objeto complejo de forma que no es necesario crear tablas secundarias para almacenar elementos de este tipo. Además la velocidad de búsqueda en la BD es mayor que en otras de tipo relacional. Por tanto, es una base de datos muy efectiva para desarrollo web, en nuestro caso, para la creación de una API.

También tenemos una base SQLite en la parte cliente Android. La decisión es sencilla en este caso: Android incorpora de serie todas las herramientas necesarias para la creación y la gestión de bases de datos de este tipo. Esto se consigue porque SQLite implementa un

5 BSON: representación binaria de JSON

motor de base de datos empotrado en el propio SO, es decir, no inicia un servicio independiente sino que se encuentra enlazada a la propia aplicación mediante la librería de soporte.

7.1.5 Módulos para node.js

El uso de módulos complementa la capacidad de funcionamiento del servidor desarrollado en node.js. En nuestro caso, se han empleado los paquetes que seguidamente quedan detallados.

- **Bcrypt-node.js** realiza la encriptación de cadenas de texto. Ha sido empleado para guardar la contraseña en base de datos de forma cifrada.
- **Co** permite el control de flujo utilizando promesas lo que permite escribir código asíncrono sin bloqueos y de forma más sencilla.
- **Coffee-script** es un lenguaje de programación que se compila en JavaScript pero que aporta una sintaxis más parecida a la de Ruby. El código resulta más legible y rápido de programar.
- **Express** es un *framework* que facilita todas las interacciones web y permite una creación sencilla de una API.
- **Jasmine** permite realizar test unitarios y de integración para código JavaScript.
- **Mongoose** interacciona con la base de datos MongoDB, haciendo todas las acciones necesarias como conectar con ella, insertar datos o extraerlos.

7.1.6 Librerías Android

En el caso de Android, para añadir funcionalidad que no viene soportada por sus propias librerías se añaden adicionales lo cuál permite cargar sus clases y métodos. En nuestro caso han sido empleadas las siguientes:

- **JsonSimple** permite trabajar con JSON en Java.
- **MaterialDialogs** crea diálogos con el nuevo diseño de *Material Design* de Android compatibles con versión inferiores del sistema operativo.
- **zBarScanner** añade la funcionalidad que permite la lectura de códigos QR desde la aplicación.

- **Fabric.io** permite el desarrollo de Twitter en el dispositivo móvil. Añade toda la funcionalidad que se obtiene mediante llamadas a la API de Twitter pero empleando sus clases y métodos.
- **Google play services** añade más compatibilidad con librerías propias de Android, como el uso de Google Maps empleado en este proyecto.

7.1.7 Control de versiones

Por último en cuanto a tecnologías utilizadas, se ha empleado git para el control de versiones. Esta herramienta permite tanto trabajar en local, como subir ficheros a un repositorio. Esto permite que, mediante el uso de *commits*[6] se pueda volver a un estado anterior en un fichero, ver su evolución o revertir cambios que hayan producido un fallo. Además permite el uso de ramas lo que permite que no todo el código funcional tenga que encontrarse en producción ni sea visto por todos los participantes del proyecto.

En este caso, se ha empleado por llevar un control de los cambios realizados y poder, en caso de fallo, volver a un estado estable anterior.

7.2 Estructura de los ficheros y directorios

En esta sección se describe la estructura de directorios de los proyectos. Además se realiza la explicación de los aspectos relevantes a la implementación del proyecto. En primer lugar se analiza el servidor y, en segundo lugar, la aplicación Android.

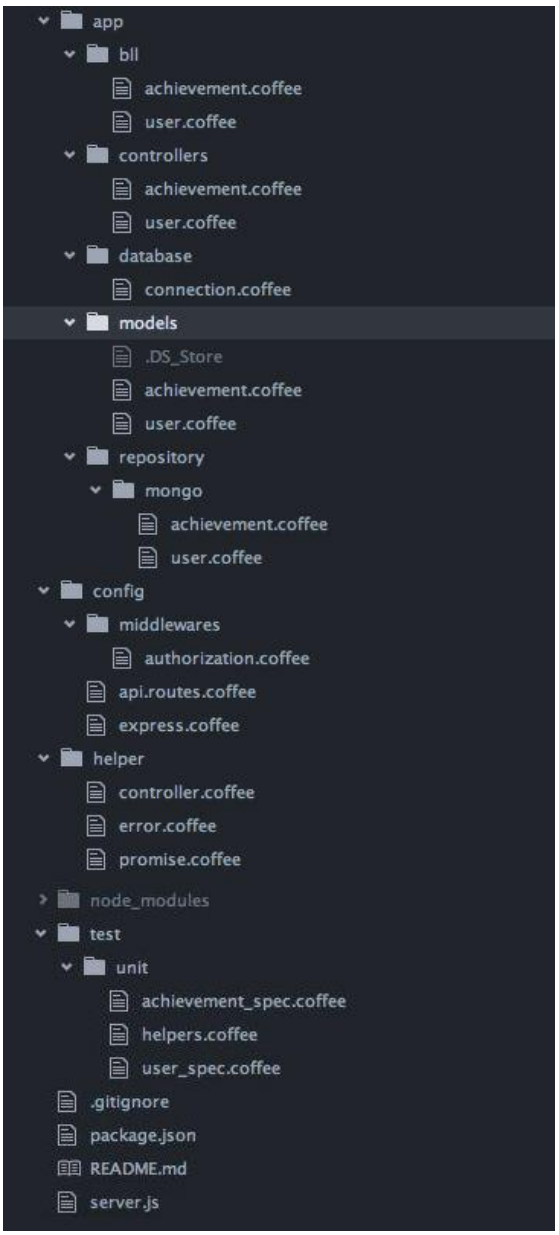
7.2.1 Directorio servidor

Como se puede observar en la figura 26, en la raíz del proyecto, existen cinco directorios y cuatro ficheros. En relación a los ficheros, en primer lugar, tenemos el *server.js*. Este fichero es el encargado de iniciar el servidor así como de su configuración. En segundo lugar, tenemos el *package.json*, donde se encuentran algunos datos relativos al proyecto y los módulos para su instalación mediante *npm*. A continuación, en tercer lugar, tenemos el .gitignore un fichero que contiene todo aquello que no queramos subir al repositorio, como los módulos, ya que ocupan un espacio considerable. Por último tenemos el README del

6 Commit: Confirmación de cambios realizados

proyecto donde se puede añadir una pequeña guía o consideraciones a tener en cuenta así como datos del autor.

Figura 26. Directorio servidor



En cuanto a la estructura de carpetas observamos cinco grandes grupos. *App* contiene todo lo relativo al desarrollo del proyecto en sí como funcionalidad o modelos de datos, más adelante se explica más en profundidad.

Config incluye todos los parámetros de configuración como las rutas en las que el servidor escucha o la autorización.

Helper contiene ficheros de ayuda como el controlador de promesas, el creador de errores y el manejador de estos últimos —*controller.coffee* —.

Node_modules es el directorio donde se instalan los módulos empleados.

Test es el directorio donde se incluyen los ficheros de pruebas. Como se puede observar existen dos, uno para cada tipo de modelo. El fichero *helpers* es usado para vaciar cualquier dato que puediera haber quedado guardado en la base de datos del entorno de pruebas.

Si vemos más a fondo la carpeta *app* vemos que ésta a su vez, contiene varios subdirectorios. *Bll* es una capa intermedia entre el controlador y el repositorio que permite realizar todas las validaciones pertinentes o modificaciones de los datos, como por ejemplo, en el caso de usuarios cifrar la contraseña. *Controllers*, como su propio nombre indica, son los controladores de los modelos. Estos recopilan los datos recibidos y realizan las acciones de creación o búsqueda. *Database* conecta con la base de datos y carga los modelos. *Models* contiene el modelo de datos, en forma de JSON. Y, por último, *Repository* contiene los métodos que realizan acciones con la base de datos de mongo. Si en un futuro se quisiera dar soporte a distintas bases de datos, se podrían añadir tantas subcarpetas como fuera necesario y que se usaran los métodos en función de la BD empleada.

Como se puede observar, la funcionalidad esta separada por modelos, es decir, tanto usuario como logro tiene su propio controlador que a su vez usa una BLL única la cuál llama a un repositorio propio que comunica con su colección concreta en la base de datos de MongoDB.

7.2.2 Directorio Android

El directorio del proyecto de la aplicación del cliente es más complejo por lo que lo analizaremos por partes. Como podemos observar en las siguientes figuras, el proyecto genera una serie de carpetas y ficheros de forma automática para una correcta configuración y funcionamiento de la app. Podemos observar que, igual que en el proyecto en node, tiene su propio gitignore. Sin embargo, el fichero que más destaca es el *build.gradle*. Gradle es una herramienta que automatiza la construcción del proyecto realizando las tareas, entre otras, de compilación y empaquetado del proyecto. En el se incluyen las dependencias con librerías o repositorios y el sdk utilizado para la compilación del proyecto así como su versión mínima soportada.

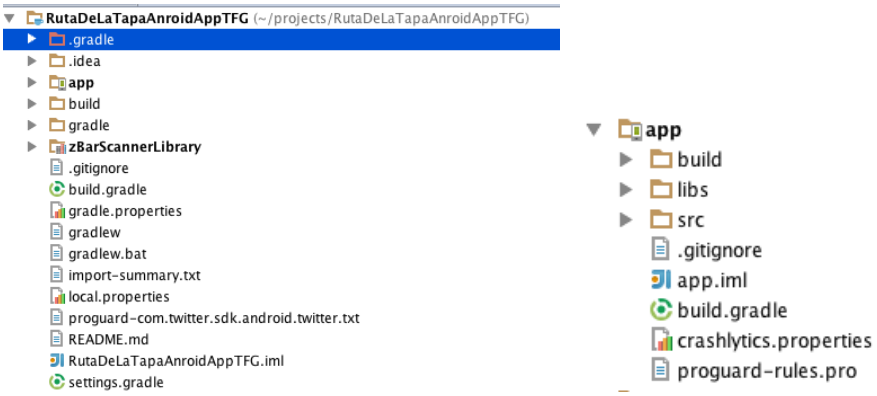


Figura 27. Directorio raíz aplicación Android

También podemos observar que en directorio *app* se encuentran la carpeta build, donde se autogenera el instalador de la aplicación; la carpeta libs que contiene las librerías que hemos añadido; y el directorio src, que pasamos a analizar a continuación puesto que en él se encuentran todos los ficheros que hacen posible la creación de la aplicación.

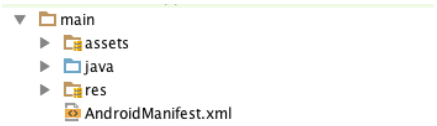


Figura 28. Carpetas directorio *src*

El *AndroidManifest* es nuestro fichero de configuración del proyecto. En el se encuentran los permisos que pedimos sobre el dispositivo móvil, en nuestro caso, acceso al gps, internet y cámara entre otros; asimismo están especificadas todas las actividades que contiene la aplicación y queda indicada cual es la que la inicia, el *Splash* en este caso. También se incluyen otros parámetros como el tema empleado, el icono de la aplicación y las apiKeys con los servicios de Google y Twitter. Sin ellas no se permite el uso de sus APIs.

También tenemos el directorio res, donde se encuentran todos los recursos que utiliza nuestra aplicación. En *drawables* se encuentran los iconos y en *layouts* la implementación de las vistas, los diálogos y otros elementos visuales. También tenemos la carpeta *values* donde se encuentran definidos tanto los colores que emplea la aplicación, como las cadenas que se muestran en las vistas — al tenerlas en un fichero el sistema soporta multi idioma[7] — y el tema empleado. Como se puede ver existe un *values-v21*, esto sucede porque el tema más actual no es compatible con modelos antiguos de Android — se soporta a partir de la versión 5 — por lo que es necesario que se definan los temas de forma compatible, en v21 para los últimos modelos y en el otro para dispositivos más anticuados.

[7] Multi idioma Android: Solo sería necesario añadir una carpeta *values* por cada idioma soportado con el prefijo adecuado, por ejemplo, *values-fr* para francés

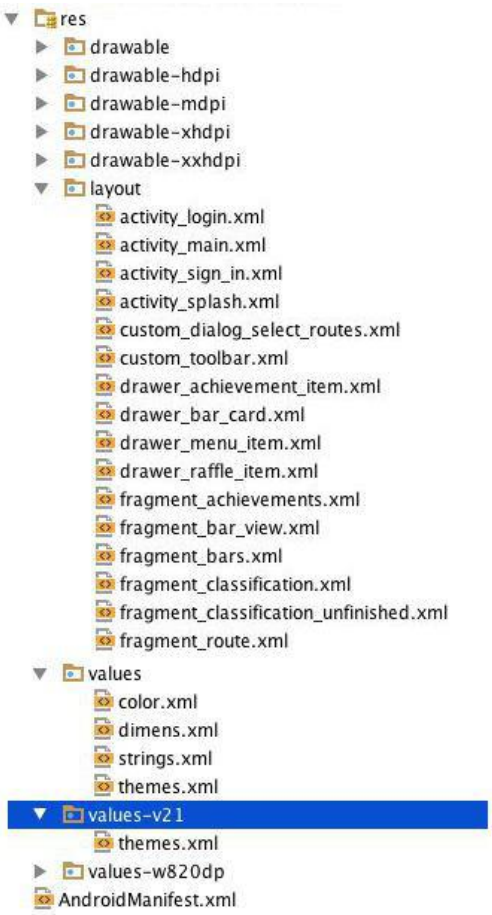


Figura 29. Carpetas directorio *res*

Por otra parte tenemos el directorio java, que analizaremos a continuación. Como ya hicimos con el proyecto en node, describiremos que acciones realizan los ficheros que contienen las distintas carpetas y, si es necesario, alguna aclaración concreta.

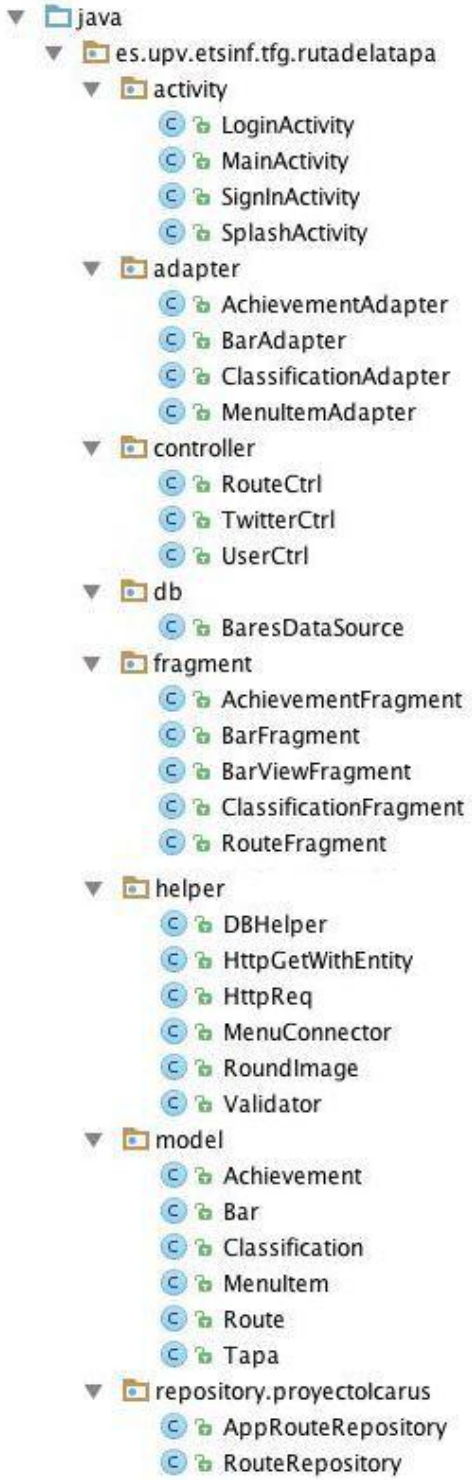
Activity contiene las actividades principales de la aplicación. *Splash* es la pantalla de bienvenida que es encargada de realizar autologin si el usuario ya ha iniciado sesión anteriormente. *Main* es la actividad principal, en la cual, se controla el fragmento que es mostrado, por defecto el de ruta, y se añade la funcionalidad del menú. *SingIn* es la actividad encargada de realizar el registro manual y *LogIn* la del inicio de sesión y el acceso vía Twitter.

Adapter contiene clases java para acoplar los datos de los modelos con las listas y el menú que se utilizan en la aplicación.

Controller contienen los métodos que realizan las acciones sobre los modelos de datos. *Route* gestiona todo lo relativo a las rutas, los bares y las tapas. *Twitter* contiene el registro y la obtención de datos mediante dicha red social. *User* permite el registro y el inicio de sesión de los clientes.

DB contiene el fichero de la tabla bares, donde se encuentran definidas las columnas de la tabla y la cadena de inserción de bares.

Figura 30. Carpetas directorio *java*



Fragment contiene las vistas parciales de la aplicación. La actividad principal, gestiona dichas vistas según la acción que el usuario realiza. Inicialmente se encuentra el fragmento *Route* que muestra el mapa con los bares marcados. Los fragmentos de *Achievement*, *Bar* y *Classification* muestran la lista de logros, bares y la clasificación respectivamente.

También se encuentra el fragmento *BarView*, que muestra la información concreta de un bar y su tapa.

Helpers contiene clases necesarias para el funcionamiento del proyecto. El fichero de base de datos conecta y obtiene los datos de SQLite. Los http son los encargados de crear clases genéricas para realizar peticiones HTTP. El fichero de menú gestiona las acciones de éste. *RoundImage* permite que la imagen de usuario del menú aparezca en forma de circulo y, por último, *validator* contiene validaciones de campos como la escritura correcta de un email o que un campo requerido no se encuentre vacío.

Model tiene las clases principales de los tipos de datos creados para la aplicación. Cada modelo tiene su constructor con los parámetros necesarios y métodos de obtención de datos a partir del objeto.

Repository, por último, contiene los ficheros que realizan peticiones a

los servidores API propios de *La ruta de la tapa*. App conecta con el servidor desarrollado en este proyecto mientras que el otro fichero conecta con el servidor externo a éste.

Para finalizar, los iconos de la aplicación se muestran gracias a la fuente *Font Awesome* que, mediante transformar una serie de cadenas identificadoras con iconos completamente escalables, como si fueran letras. Basta con, a la hora de visionar el elemento que contenga este tipo de icono, indicarle que debe hacerlo con el uso de esta fuente que ha de ser importada en el proyecto.

En conclusión, tras la implementación tanto de la parte servidora como la del cliente, se ha conseguido tener un proyecto estructurado que además cumple con toda la funcionalidad y los requisitos que se habían acordado. En el siguiente capítulo se explica como desplegar el proyector para poder hacer uso de la aplicación *La ruta de la tapa*.

8. Despliegue

Tres cosas hay en la vida… reset, reset y reset.

— Bill Gates —

Una de las últimas acciones que hay que realizar para finalizar el proyecto es el despliegue del mismo. En cuanto a la aplicación Android su instalación es sencilla. Mediante el apk que genera el propio entorno de desarrollo de Android Studio se permite la instalación de nuestra *app* en cualquier dispositivo Android con la versión soportada. El despliegue del servidor queda fuera del alcance del cliente pero parece interesante comentarlo con más detenimiento a continuación.

8.1 Despliegue del servidor

Para que el servidor siempre esté online y con acceso público se ha decidido subirlo a la nube. Para ello hemos hecho uso de OpenShift. Esta herramienta permite alojar un servidor de tipo node.js en una nube pública, así como su base de datos en mongo, de forma gratuita. Tras el registro previo en la plataforma, es necesario instalar el cliente[8]. Una vez instalado, se debe configurar mediante el comando `rhc setup` por consola.

Tras concluir la configuración — se encuentra bastante guiada por lo que omitimos los pasos intermedios —, se ha de crear la aplicación en OpenShift. Este paso se puede realizar tanto por línea de comandos como por la propia página web de la aplicación. En la misma sección de instalación del cliente se puede ver el proceso completo pero en resumen las acciones a realizar son:

- Iniciar la aplicación en OpenShift indicando el nombre que queremos usar la tecnología node.js.
- Añadir MongoDB a servidor, también disponible vía web.
- Clonar en un directorio el repositorio git remoto que se ha creado al iniciar la aplicación.

[8] Para ver más detalles consultar: https://developers.openshift.com/en/getting-started-osx.html#client-tools [Junio 2015]

- Importar nuestro servidor a dicho directorio, se ha de tener en cuenta que los ficheros de configuración con los parámetros de OpenShift. El repositorio inicial que se descarga puede servir de guía.
- Subir nuestro proyecto al repositorio.

Una vez configurado el servidor, podemos acceder a él vía ssh — podemos consultar la dirección en la propia web de OpenShift en nuestro panel de administración — o ver su log mediante el comando `rhc tail` con el nombre que le hayamos asignado al proyecto. También podemos consultar el estado en el propio panel de administración que nos ofrece esta herramienta, que se puede ver en la siguiente figura.

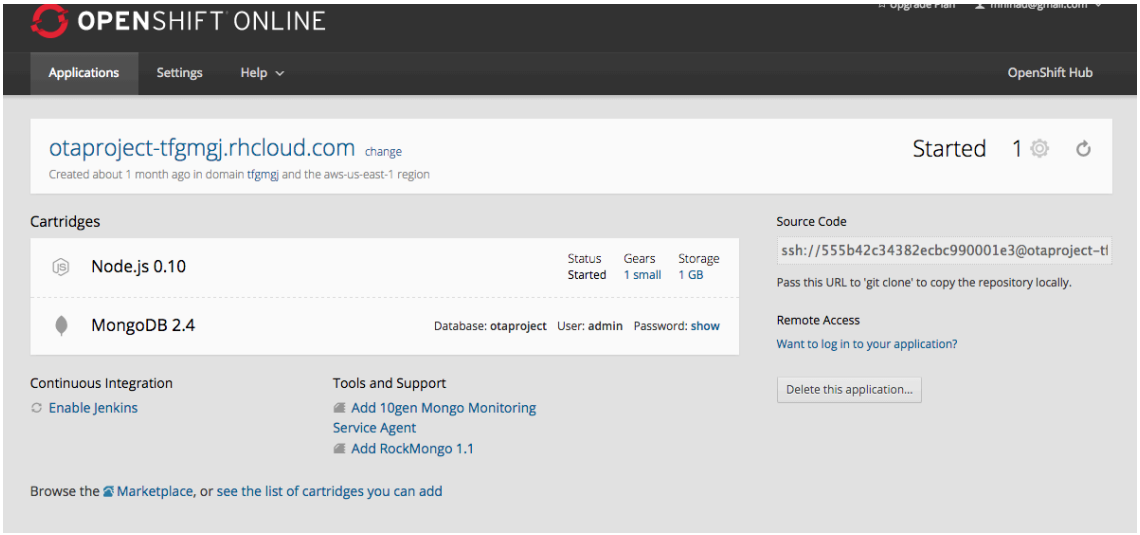


Figura 31. Panel de administración OpenShift

Como podemos observar, mediante OpenShift, podemos tener un servidor de pruebas alojado en la nube de forma sencilla. Sin embargo, para un entorno de producción real la capacidad ofrecida sería muy pequeña. En conclusión, en el estado actual que se encuentra el proyecto, se podría probar la aplicación *La ruta de la tapa* ya que los servidores se encuentran en funcionamiento y accesibles permanentemente. Bastaría con instalar la aplicación en un dispositivo compatible. En el capítulo 9 de pruebas, se muestran los test que se han pasado para comprobar que la funcionalidad del servidor funciona correctamente y se realiza un paso por toda la aplicación Android en forma de prueba.

9. Pruebas

Where's the any key? Frente a un mensaje "press any key".

– Homer J. Simpsons –

A continuación y para finalizar la implementación del proyecto se realizarán una serie de pruebas para comprobar el funcionamiento correcto del sistema. Para probar el servidor, se hará uso de TDD mediante pruebas unitarias. En el caso de la aplicación cliente, se hará una prueba completa de la *app* en un móvil para comprobar que tanto el aspecto visual como el funcional responden a los requisitos especificados.

9.1 Pruebas del servidor

Como se ha comentado anteriormente, para realizar un test del servidor se hace uso de TDD. Esta práctica involucra dos prácticas más adicionales *las pruebas primero* y *refactorizar*. Para llevar a cabo las pruebas unitarias se ha utilizado la herramienta Jasmine un *framework* que facilita la realización de pruebas mediante el uso de *expect*. Para poder explicar con detenimiento como se realizan dichas pruebas se muestra la figura siguiente.

```
apiUrl = "/api/user"

describe "User", ->
  beforeEach (done) ->
    helpers.cleanDataBase().then ->
      createDataTest (err) ->
        assert.ifError(new Error(err)) if err
        done()

  describe "API", ->

    describe "Create", ->
      it "Shouldn't create a user if not name", (done) ->
        request(app).post(apiUrl).end (err, res) ->
          return done err if err

          expect(res.statusCode).toBe 400
          expect(res.body.message[0]).toBe "NameNotFound"
          done()
```

Figura 32. Comienzo de un test escrito con Jasmine

Los *describe* indican que elementos vamos a probar. En los resultados finales permiten observar una estructura para observar más concretamente que ha podido fallar. Los *it* son las funciones concretas que se van a probar, es decir, cada uno de los tests unitarios a llevar a cabo. Mediante *request* se realizan las peticiones al propio servidor y los *expect* son aquellos aspectos que se pretenden probar, tanto respuestas negativas como positivas. Además la sentencia *beforeEach* permite realizar acciones previas a cada test unitario, en nuestro caso, antes de cada test se vacía por completo la base de datos de pruebas.

Los test que se han llevado a cabo y los resultados que se esperan obtener son los siguientes:

API Usuarios

No se puede crear un usuario sin nombre. Si el servidor recibe una petición para crear usuario y éste no tiene el atributo *name*, el código de estado de la respuesta debe ser 400 y el mensaje de error *NameNotFound*. Para su prueba se envía un objeto usuario sin nombre.

No se puede crear un usuario sin nombre de usuario. Si el servidor recibe una petición para crear usuario y éste no tiene el atributo *username,* el código de estado de la respuesta debe ser 400 y el mensaje de error *UsernameNotFound*. Para su prueba se envía un objeto usuario sin nombre de usuario.

No se puede crear un usuario si el nombre de usuario ya existe. Si el servidor recibe una petición para crear usuario y el nombre de usuario ya se encuentra en la base de datos, el código de estado de la respuesta debe ser 400 y el mensaje de error *UserExists*. Para su prueba en primer lugar, se crea un usuario en la base de datos. Seguidamente se envía un objeto usuario con el mismo nombre de usuario.

Se debe poder crear un usuario. Si el servidor recibe una petición para crear usuario y los datos recibidos son válidos, el código de estado de la respuesta debe ser 200. Para su prueba se envía un objeto con todos los atributos válidos.

No se puede iniciar sesión sin nombre de usuario. Si el servidor recibe una petición para iniciar sesión y éste no tiene el atributo *username,* el código de estado de la respuesta debe ser 400 y el mensaje de error *UsernameNotFound*. Para su prueba se envía un objeto usuario *username*.

No se puede iniciar sesión si la contraseña es incorrecta. Si el servidor recibe una petición para iniciar sesión y la contraseña es incorrecta, el código de estado de la respuesta debe ser 400 y el mensaje de error *InvalidPassword*. Para su prueba se envía un objeto con la contraseña incorrecta.

Se debe poder iniciar sesión. Si el servidor recibe una petición para iniciar sesión y los datos recibidos son válidos, el código de estado de la respuesta debe ser 200 y la respuesta debe tener un atributo *isAuth* a true. Para su prueba se crea en la base de datos de test un objeto usuario. A continuación se envían como atributos el *username* y el *password* correspondientes a dicho usuario.

API Logros

No se puede crear un logro sin el objeto logro. Si el servidor recibe una petición para crear logro y el cuerpo del mensaje no contiene ningún objeto, el código de respuesta debe ser 400 y el mensaje de error *AchievementNotFound*. Para su prueba se realiza la petición sin ningún cuerpo del mensaje.

No se puede crear un logro sin su identificador. Si el servidor recibe una petición para crear logro y el cuerpo del mensaje no contiene al atributo *achievementId*, el código de respuesta debe ser 400 y el mensaje de error *AchievementIdNotFound*. Para su prueba se realiza la petición con un objeto vacío.

No se puede crear un logro sin el número actual de acciones realizadas. Si el servidor recibe una petición para crear logro y el cuerpo del mensaje no contiene al atributo *total*, el código de respuesta debe ser 400 y el mensaje de error *TotalNotFound*. Para su prueba se realiza la petición con un objeto logro sin el atributo *total*.

No se puede crear un logro sin el total de acciones a cumplir. Si el servidor recibe una petición para crear logro y el cuerpo del mensaje no contiene al atributo *totalMax*, el código de respuesta debe ser 400 y el mensaje de error *TotalMaxNotFound*. Para su prueba se realiza la petición con un objeto logro sin el atributo *totalMax*.

No se puede crear un logro sin el usuario al que se asocia. Si el servidor recibe una petición para crear logro y el cuerpo del mensaje no contiene al atributo *username*, el código de respuesta debe ser 400 y el mensaje de error *UsernameNotFound*. Para su prueba se realiza la petición con un objeto logro sin el atributo *username*.

No se puede crear un logro si éste ya existe. Si el servidor recibe una petición para crear logro y éste ya se encuentra en la BD, el código de respuesta debe ser 400 y el mensaje de error *AchuievementExists*. Para su prueba, en primer lugar, se crea un logro en la base de datos de prueba. Tras ello, se envía una petición de creación de logros con el mismo objeto creado en el cuerpo de mensaje.

Se debe poder crear un logro. Si el servidor recibe una petición para crear logro y los datos recibidos son válidos, el código de estado de la respuesta debe ser 200. Para su prueba se envía un objeto con todos los atributos válidos.

Se debe poder obtener los logros de un usuario. Si el servidor recibe una petición para obtener usuario y los datos recibidos son válidos, el código de estado de la respuesta debe ser 200 y el cuerpo del mensaje debe tener los logros. Para su prueba se crea tanto un usuario como un logro asociado en la base de datos. Tras ello, se envía una petición de obtención de logros con el identificador de usuario.

Se debe poder actualizar la cuenta de un logro de un usuario. Si el servidor recibe una petición para actualizar un logro, el código de estado de la respuesta debe ser 200 y el cuerpo del mensaje debe tener el logro con el atributo *total* actualizado. Para su prueba se

crea tanto un usuario como un logro asociado en la base de datos. Tras ello, se envía una petición de actualización de logros con el identificador de usuario y el identificador de logro.

En la figura siguiente se observa el resultado que muestra por consola la ejecución de los tests que se han realizado. Podemos observar que todos se han ejecutado correctamente.

```
Achievement - 810 ms

    API - 810 ms

        Create - 744 ms
            Shouldn't create a achievement if not achievement - 512 ms
            Shouldn't create a achievement if not achievement id - 37 ms
            Shouldn't create a achievement if not total - 28 ms
            Shouldn't create a achievement if not total max - 33 ms
            Shouldn't create a achievement if not username - 29 ms
            Shouldn't create a achievement if achievement already exists - 78 ms
            Should create a achievement - 27 ms

        Get - 27 ms
            Should get achievements for username - 27 ms

        Update - 39 ms
            Should update achievements for username - 39 ms

User - 924 ms

    API - 924 ms

        Create - 490 ms
            Shouldn't create a user if not name - 90 ms
            Shouldn't create a user if not username - 92 ms
            Shouldn't create a user if username already exists - 149 ms
            Should create a user - 159 ms

        Login - 434 ms
            Shouldn't login a user if not username - 89 ms
            Shouldn't login a user if wrong password - 154 ms
            Should login a user - 191 ms

Finished in 1.741 seconds
16 tests, 31 assertions, 0 failures, 0 skipped
```

Figura 33. Resultados pruebas servidor

9.1 Pruebas de la aplicación

En el caso de la aplicación se procede a probar la aplicación completa en un entorno real, es decir, en un dispositivo físico. Para sus pruebas se ha instalado la aplicación en un móvil Nexus 5 con versión Android 5.1.1.

A continuación se realiza un vista guiada de todas las opciones disponibles en la aplicación. Para comenzar, la primera pantalla que se muestra es el *Splash*. Se aprovecha para cargar datos que ya se encontraran almacenados en la aplicación — como un usuario

que haya iniciado sesión o una ruta ya seleccionada — además de mostrar el logotipo de la aplicación. El usuario no necesita realizar ninguna acción adicional, simplemente debe esperar a que esta pantalla finalice de realizar sus funciones asociadas.

Figura 34. Pantalla *splash*

Tras tres segundos, pueden ocurrir dos sucesos. Si el usuario ya había iniciado sesión se pasará a la pantalla principal de la aplicación. Si no, se mostrará la vista de inicio de sesión. Comenzaremos analizando el segundo caso. El inicio de sesión contiene los campos para introducir un usuario y una contraseña. Además también contiene los enlaces al registro vía Twitter y al registro manual.

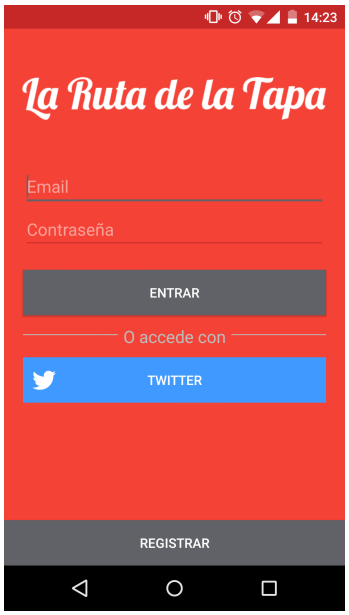


Figura 34. Pantalla inicio de sesión

Desde esta pantalla se pueden realizar varias acciones. Si se presiona el botón atrás del dispositivo, la aplicación se cerrará. Si se presionan los botones *TWITTER* o *REGISTRAR* se mostrarán las vistas asociadas a dichas acciones. También se puede acceder directamente a la aplicación introduciendo el email y la contraseña y presionando el botón *ENTRAR*.

Veamos las distintas opciones. En primer lugar el registro por Twitter mostrará la siguiente pantalla:

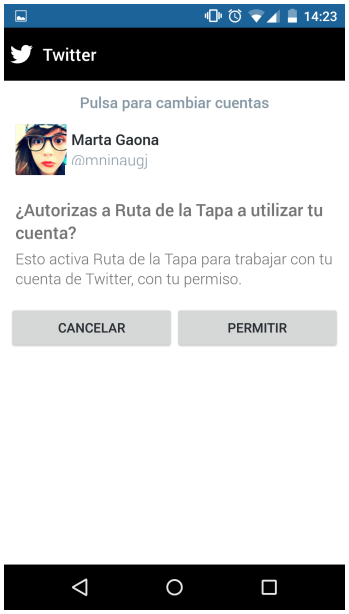


Figura 35. Pantalla registro vía Twitter

Como se puede observar, esta acción muestra una ventana propia de la aplicación Twitter donde se pide la autorización del usuario para acceder mediante la cuenta asociada al dispositivo. Si se presiona el botón *PERMITIR* se accedería a la pantalla principal de la aplicación.

También podemos realizar el registro de forma manual. Al presionar el botón *REGISTRAR* obtenemos la siguiente vista:

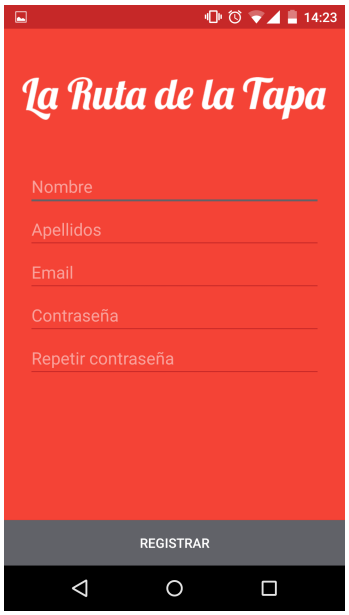


Figura 36. Pantalla registro manual

Si todo el formulario se rellena correctamente se accederá a la actividad principal tras presionar de nuevo el botón *REGISTRAR*. En caso contrario, se mostrará algún mensaje de error en los campos inválidos.

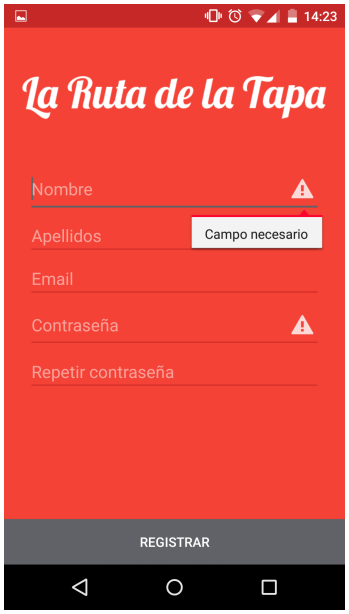


Figura 37. Pantalla registro con datos inválidos

Un procedimiento similar se produce al iniciar sesión, esta vez, presionando el botón *ENTRAR* en la pantalla de inicio de sesión.

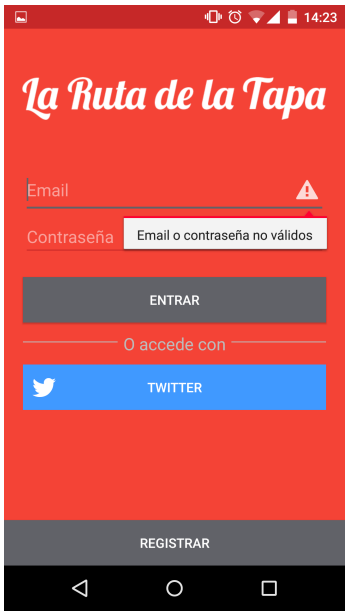


Figura 38. Pantalla inicio de sesión con datos inválidos

A través de cualquiera de estas acciones, si han sido válidas, se muestra la pantalla principal del sistema. A su vez recordar que, si ya se hubiera iniciado sesión anteriormente, esta pantalla sería la que se habría mostrado una vez acabada la carga por parte del *splash*.

Figura 39. Pantalla principal: mapa

La vista principal de la aplicación es un mapa con todos los bares posicionados sobre él. Si no se hubiera seleccionado una ruta anteriormente, antes de poder ver el mapa, aparece

un dialogo para elegir la ruta entre las disponibles. Éste muestra las más cercanas a la posición actual si se ha podido obtener la situación del dispositivo.

Figura 40. Diálogo selección de ruta

Una vez en el mapa, si se selecciona un bar en concreto podemos observar que se muestran los botones relacionados con la aplicación de Google Maps. Si presionamos alguno, esta aplicación se abre con la localización del bar seleccionado y así podemos obtener el trayecto hacia él.

Figura 41. Mapa con bar seleccionado

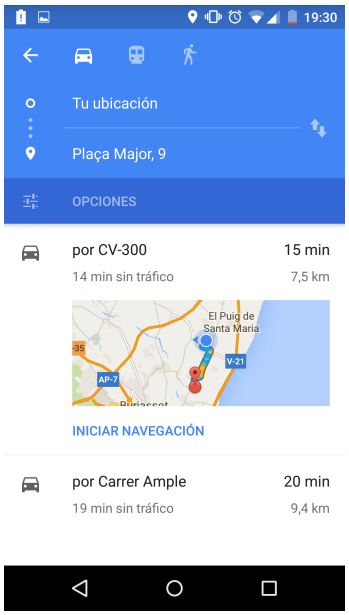


Figura 42. Aplicación Google Maps a través de la aplicación *La ruta de la tapa*

Por otra parte, todas las interacciones una vez estemos dentro de la aplicación se realizan mediante el menú. También se puede volver siempre al mapa presionando el botón de atrás, a excepción de las vistas de los bares que volverán a la pantalla de la lista a la que corresponden. Desde el menú podemos acceder a las opciones de *Bares y tapas, Clasificación, Logros, Salir de la ruta* y *Cerrar sesión*. Además podemos observar, que se muestran los datos de la cuenta actual con la que se ha iniciado sesión así como la foto de perfil de Twitter si el registro se hubiera realizado a través de esta opción.

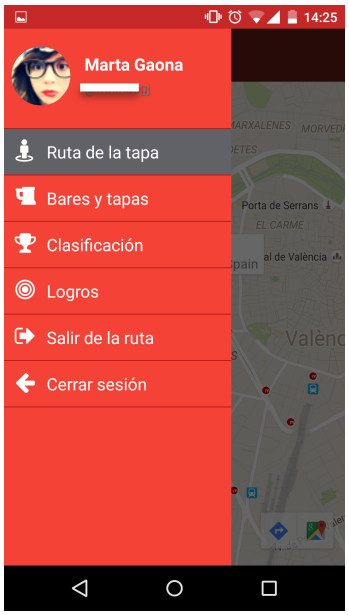


Figura 43. Menú de la aplicación

Salir de la ruta y cerrar sesión no incorporan ninguna actividad adicional. Si se presiona salir de la ruta volveremos a la vista de mapa con el dialogo de selección de nueva ruta. Si cerramos sesión, volveremos a la pantalla de inicio de sesión.

En cuanto al resto del menú comenzaremos por la acción más sencilla: los logros. Al acceder a esta pantalla presionando la opción del menú correspondiente se puede observar un listado de logros donde se muestra su icono, su nombre, una pequeña descripción, el número de veces que hay que realizar la acción y la cuenta actual. Si el logro se ha conseguido se encuentra enmarcado en rojo, sino en gris. Por ahora hay tres acciones que actualizan el estado: acceder a una nueva ruta, tomar una tapa, es decir, leer un QR y votar una tapa tras haber leído el QR.

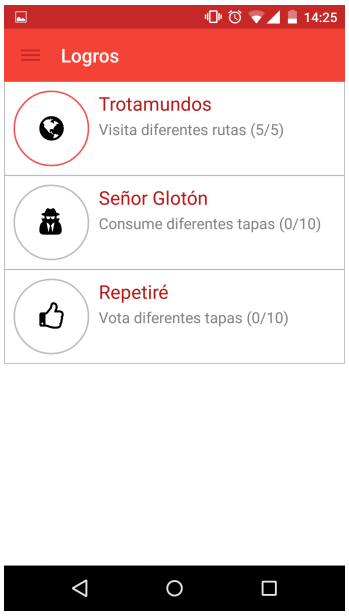


Figura 44. Pantalla *Logros*

Si en cambio presionamos la opción *Bares y tapas* se mostrará el listado de los bares participantes.

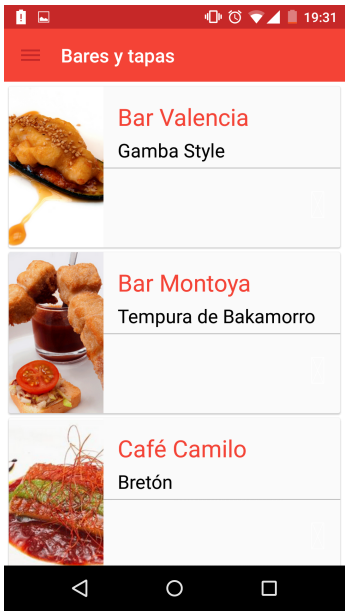


Figura 45. Pantalla *Bares y tapas*

Si presionamos sobre alguno, se accederá a la vista del participante. En ella podemos ver los datos más relevantes tanto del bar como de la tapa así como una foto que la identifique.

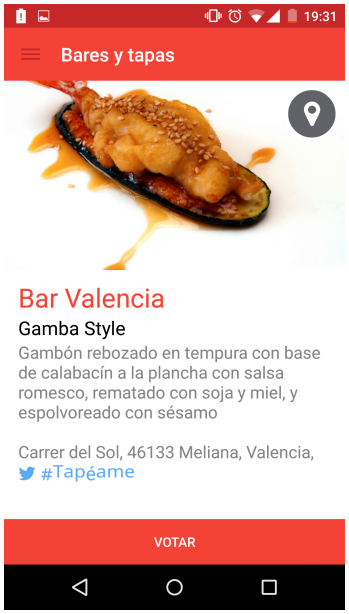


Figura 46. Pantalla *Vista bar participante*

Si presionamos el botón situado sobre la foto, el de localización, la aplicación nos mostrará el mapa de la actividad principal con el bar deseado ya marcado. Por otra parte, hay que tener en cuenta que, si la votación ya ha finalizado, se mostrará la misma pantalla de vista de bar pero sin el botón de *VOTAR*.

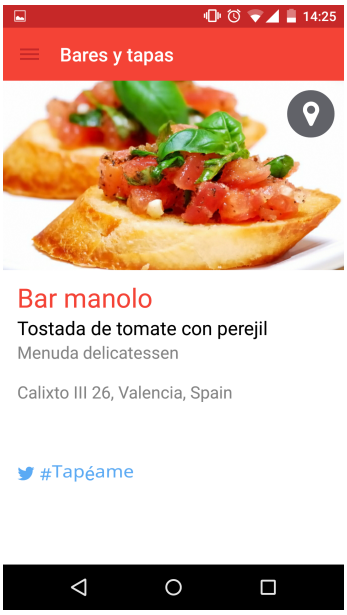


Figura 47. Pantalla *Vista bar participante* con votación finalizada

Si se ha consumido una tapa, el propietario del bar nos tiene que proporcionar un código QR. Tras presionar el botón, la aplicación abre automáticamente la cámara del dispositivo y al leer el código, se muestra la opción de votar mediante un código de tenedores. Si el voto es válido se muestra un mensaje de ¡Gracias por participar! Y se colorean los tenedores en rojo según la votación otorgada, en caso contrario, se muestra un mensaje indicando que se ha producido un error.

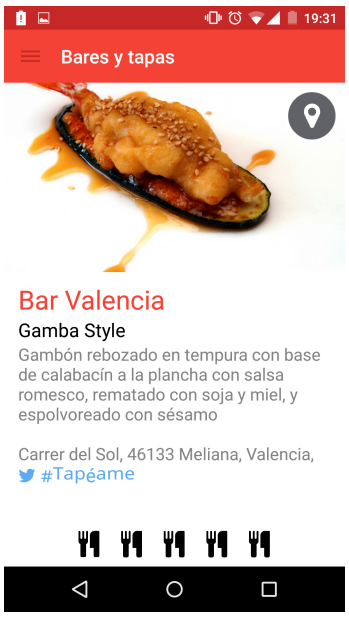


Figura 48. Pantalla *Vista bar previa a la votación*

Además también se encuentra el botón *#Tapéame*. Presionando en él, podemos crear un *tweet* desde la propia aplicación que además, por defecto, incluirá los *hashtag* #RutaDeLaTapa y el nombre del bar.

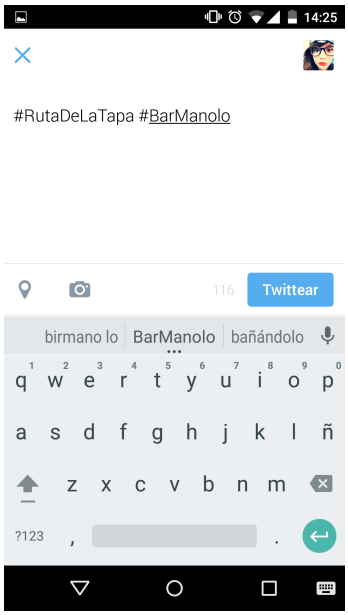


Figura 48. Creación de un *tweet*

Por último nos queda la opción *Clasificación*. Puede encontrarse en dos estados: en curso o finalizada. Si la votación para elegir la mejor tapa de la ruta sigue en curso, se muestra una pantalla que muestra un mensaje indicando dicho suceso.

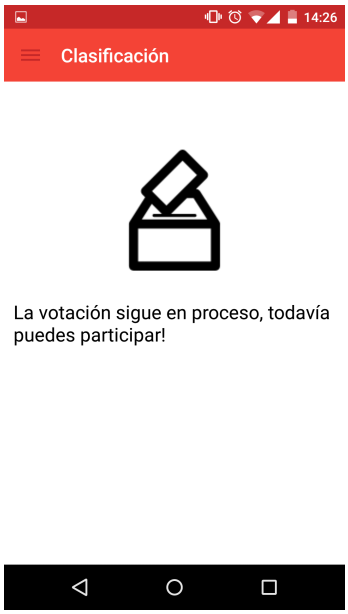


Figura 49. Pantalla clasificación en curso

Sin embargo, si ya ha finalizado, se muestra un listado con los bares ganadores — con un máximo de tres — y un icono a modo de trofeo que indica su posición: dorado si es el ganador, plateado si ha quedado en segundo puesto y bronce si es el tercero. En el ejemplo, podemos ver que existían únicamente dos ganadores.

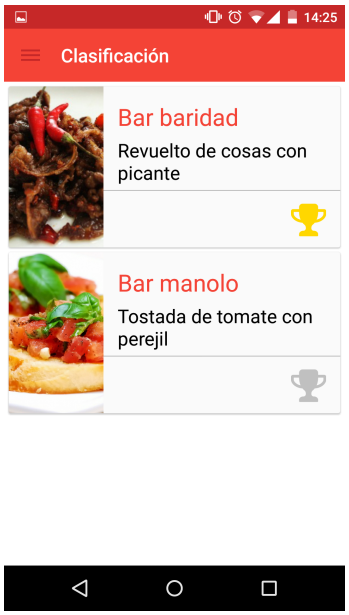


Figura 50. Pantalla clasificación finalizada

Si accedemos a un bar desde esta pantalla, la vista mostrada nunca podrá tener la acción *VOTAR* pero será la misma que ya se ha visualizado anteriormente.

En conclusión, podemos ver que toda la funcionalidad deseada se ha añadido a la aplicación con un diseño agradable y fácil de entender. Aún siendo una primera implementación, podría ser ya una aplicación usable y que cumpliera la funcionalidad mínima esperada de una *app* relacionada con las rutas de la tapa. Para finalizar el documento, se expondrán las conclusiones obtenidas tras su realización.

10. Conclusión

El tiempo es el mejor actor: siempre encuentra el final perfecto.

– Charles Chaplin –

A lo largo de este proyecto se ha llevado a cabo el desarrollo de la aplicación *La ruta de la tapa* para dispositivos móviles. Para ello se hizo un estudio de mercado por tal de justificar la necesidad de dicha aplicación y hacia que dispositivos y público se quería orientar, cuestión que resultó ser un factor determinante a la hora de su diseño. Las conclusiones obtenidas fueron que se implementaría para dispositivos basados en el sistema operativo Android, ya que el porcentaje de usuarios bajo este SO es mayor frente a otros tales como iPhone o Windows Phone. Del mismo modo se observó que realizar esta aplicación cubría una carencia existente en el conjunto de aplicaciones disponibles en la tienda *Play Store*.

Tras ello, se estableció el propósito de lograr un entorno donde consultar diversas rutas de la tapa para que posibles consumidores pudieran recibir información acerca de éstas. Además al alcance propuesto abarcaba necesidades tales como obtener la localización de los participantes, consultar las tapas que ofertaban, poder realizar su votación a través de la aplicación y dotar a la *app* de un carácter más social e interactivo por medio de la inclusión de Twitter y el sistema de logros.

El paso siguiente fue traducir dichas necesidades en funcionalidades y requisitos propios de la aplicación para su uso como fueron el idioma de la aplicación, el tema visual elegido o los requisitos funcionales en concordancia con el alcance especificado.

A partir de esta definición técnica, mediante el análisis de todos los datos recopilados, se elaboraron los casos de uso. Gracias a dicho análisis, se pudo realizar un diseño completo de la aplicación basado en un modelo MVC que respondiera a la funcionalidad requerida. En esta fase se decidió tanto el modelo de datos a utilizar, como las acciones que podrían realizar los clientes y una primera versión de cómo resultarían las vistas de la

aplicación. Asimismo, el diseño de la interfaz se hizo de forma que resultara intuitivo para un usuario inexperto de modo que la experiencia resultara más *user-friendly*.

Seguidamente se llevo a cabo la implementación del sistema. Por una parte se desarrolló la aplicación cliente para *smartphones* bajo el entorno Android. Dicha *app*, se comunica tanto con el servidor node.js implementado en este proyecto como con otro externo desarrollado por el alumno David Valero García, con el que se ha trabajo de forma colaborativa. Todo el intercambio de mensajes se realiza mediante el uso de las APIs públicas de éstos sistemas.

Por último, para concluir la elaboración del producto definido a lo largo de este proyecto, se realiza el despliegue tanto en la nube, para el caso del servidor, como en un dispositivo móvil compatible en el caso de la aplicación cliente. Con ello, se pudo probar el funcionamiento adecuado de ambos, desvelando fallos que se han corregido para obtener un producto robusto. Las pruebas finales muestran que se han cumplido los objetivos propuestos al inicio.

Personalmente, la producción de este proyecto ha resultado gratificante y una gran fuente de aprendizaje. Aunque los lenguajes eran ya conocidos, como son Java y JavaScript, el hecho de aplicarlos a un sistema real y contextualizado, que además incorporaba aspectos propios de Android, ha necesitado de una investigación previa que ha resultado muy formativa de cara al futuro en el ámbito empresarial. Además, este nueva situación que se me ha planteado para trabajar contenidos teóricos aplicados a un producto real, me ha ayudado a desarrollar capacidades propias del sentido de la iniciativa y el espíritu emprendedor.

Para finalizar, aunque el resultado del producto, personalmente, resulta satisfactorio la aplicación podría adquirir nuevas características en un futuro. Las líneas a seguir podrían ser , como ideas generales, la inclusión de un histórico de rutas visitadas, un seguimiento de los bares visitados durante la visita a una ruta concreta, etc.; dando posibilidad a un crecimiento y mejora posterior de la aplicación, de modo que *La ruta de la tapa* siempre esté en continuo proceso de evolución y adaptación ante las nuevas necesidades de los usuarios.

11. Bibliografía

Libros

About Android

- Clifton, I. (2013). *Android User Interface Design: Turning Ideas and Sketches into Beautifully Designed Apps (Usability)*. Ed: Perason. New Jersey.
- Friesen, J. Smith, D. (2011). *Android Recipes: A Problem-Solution Approach*. Ed: Apress. New York.
- Nudelman, G. (2013). *Android Design Patterns: Interaction Design Solutions for Developers*. Ed: Wiley. San Francisco.
- Phillips, B. (2013). *Android Programming: The Big Nerd Ranch Guide (Big Nerd Ranch Guides)*. Paperback. Georgia.
- Tomás, J. (2013). *El gran libro de Android avanzado*. Ed: Marcombo. Barcelona.

About coffeescript

- Jiménez Villar, J. (2013). *Un pequeño gran libro de coffeescript*. Ed: Leanpub. Bilbao.

About JavaScript

- Crockford, D. (2013). *JavaScript: The Good Parts*. Ed: O'Reilly. Minnesota.
- Haverbeke, M. (2014). *Eloquent JavaScript: A Modern Introduction to Programming*. Paperback. Berlin.

About node.js

- Hughes-Croucher T. Wilson M. (2012). *Node: Up and Running: Scalable Server-Side Code with JavaScript*. Ed: O'Reilly. San Francisco.
- Kiessling, M. (2012). *The Node Beginner Book: A Comprehensive Node.js Tutorial*. Ed: lulu.com. Colonia.
- Mardanov, A. Chaakkaev, A. (2013). *Express.js Guide: The Comprehensive Book on Express.js*. Ed: CreateSpace. San Francisco.

About TDD

- Blé, C. (2010). *Diseño ágil con TDD*. Ed: lulu.com. Tenerife.

Sitios web

About Android

- Android Team. *Android – History*. Recuperado febrero 2015, desde https://www.android.com/history/
- Android Team. *Android Developers Blog*. Recuperado febrero 2015, desde http://android-developers.blogspot.com.es/
- Android Team. *Develop Apps | Android developers*. Recuperado febrero 2015, desde http://developer.android.com/develop/index.html
- Android Team. *Storage Options*. Recuperado abril 2015, desde http://developer.android.com/guide/topics/data/data-storage.html
- Cuello, J. Vittone, J. *Diseñando apps para móviles*. Recuperado enero 2015, desde http://www.appdesignbook.com/es/contenidos/presentacion/
- Levia, A. *Material Design Everywhere. Using APPCompat 21*. Recuperado febrero 2015, desde http://antonioleiva.com/material-design-everywhere/
- Martinez, S. *Buenas prácticas usando Fragments en Android*. Recuperado febrero 2015, desde http://gpmess.com/blog/2014/04/16/buenas-practicas-usando-fragments-en-android/

About Google Maps

- Google Team. *Google Maps Android API*. Recuperado marzo 2015, desde https://developers.google.com/maps/documentation/android/

About JavaScript

- V8 Team. *V8 JavaScript Engine*. Recuperado enero 2015, desde https://code.google.com/p/v8/

About node.js

- Node.js Foundation. *Node.js*. Recuperado enero 2015, desde https://nodejs.org/

About Twitter

- Twitter, Inc. *Fabric, the easy way to build the best apps*. Recuperado abril 2015, desde https://get.fabric.io/
- Twitter, Inc. *Twitter Kit for Android*. Recuperado abril 2015, desde https://dev.twitter.com/twitter-kit/android

About technologies

- Fiedling, R. Reschke, J. Hypertext Transfer Protocol (HTTP/1.1). Recuperado marzo 2015, desde http://tools.ietf.org/html/rfc7230
- Marqués, Asier. *Conceptos sobre APIs REST*. Recuperado febrero 2015, desde http://asiermarques.com/2013/conceptos-sobre-apis-rest/
- MongoDB, Inc. *MongoDB*. Recuperado enero 2015, desde https://www.mongodb.org/
- SQLite Team. *SQLite*. Recuperado mayo 2015, desde http://www.sqlite.org/

Others

- IDC Corporate USA. *Smartphone OS Market Share, Q1 2015*. Recuperado mayo 2013, desde http://www.idc.com/prodserv/smartphone-os-market-share.jsp
- Mediapost Group. *4 estrategias para mejorar la fidelización de clientes*. Recuperado mayo 2015, desde http://www.mediapostgroup.es/marketing-relacional/4-estrategias-mejorar-fidelizacion-clientes/
- The Blueroom Project. *Ya está disponible Food Tourism 2014, nuevo estudio de The Blueroom Project*. Recuperado enero 2015, desde http://www.blueroom.es/noticia/ya-esta-disponible-food-tourism-2014-nuevo-estudio-de-blueroom-project/