Universidad de Jaén

Escuela Politécnica Superior de Jaén

Guía Turístico Virtual

Trabajo Fin de Grado presentado por

Jesús Cuenca López

Supervisado por Arturo Montejo Ráez

Septiembre, 2014

Departamento de Informática

Grado en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Dr. D. Arturo Montejo Ráez tutor del Trabajo Fin de Grado titlulado Guía Turístico Virtual que presenta D. Jesús Cuenca López, autoriza su presentación, defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2014.

Jesús Cuenca López Arturo Montejo Ráez

Agradecimientos

Este proyecto no habría sido posible de no ser por la ayuda y el apoyo de muchas personas; tantas que es imposible nombrarlas a todas. Sin embargo, haré lo que pueda:

Gracias a mis profesores por darme a conocer un mundo nuevo, y hacer nacer en mí ese afán de querer seguir aprendiendo.

Gracias a mis compañeros, que han hecho este viaje algo más entretenido.

Gracias a mis amigos por su apoyo, por ayudarme siempre que lo he necesitado y, sobre todo, por aguantarme.

Y por último, gracias a mis padres y a mi hermano, no sólo por facilitarme todo aquello que he necesitado durante todo este tiempo, sino por todas las otras cosas que hacen los padres y los hermanos, y que han conseguido hacerme llegar donde estoy: gracias por las enseñanzas, por los consejos y, por qué no, por las regañinas; gracias por las sonrisas en momentos difíciles, por tener las palabras exactas en cada momento; y gracias por todas esas cosas intangibles que ayudan mucho más de lo que parecen en un principio.

Índice general

Índice de figuras

Índice de cuadros

CAPÍTULO 1

Introducción

1.1. Introducción al proyecto

Es evidente que hoy en día, el afán por viajar y hacer turismo ha cobrado gran auge: a las personas nos gusta visitar lugares nuevos, recorrer sitios diferentes y, en definitiva, conocer mundo.

Sin embargo, siendo ésta la generación de las nuevas tecnologías, atrás quedaron cosas como los mapas de carreteras gigantes, el lápiz y el papel a la hora de trazar rutas, y el preguntar a los pueblerinos si vamos por el buen camino.

La tecnología lo ha cambiado todo y, en este caso, a mejor: hoy en día un mismo dispositivo puede decirte por qué ruta debes ir, qué restaurantes visitar y reservarte alojamiento en el mejor hotel de la ciudad. Todo sin salir de casa.

Existen aplicaciones móviles que pueden detectar ciertos lugares cerca de ti, como cafeterías o tiendas de alimentos. También existen otras que te ofrecen información sobre un determinado lugar, de forma que puedas saber de antemano dónde vas a ir o qué vas a visitar.

Sin embargo, son muy pocas las aplicaciones orientadas de verdad al turismo: aplicaciones que te muestren los puntos de interés turístico o lugares de importancia a la hora de viajar, y que, además, te den cierta información sobre el mismo. O que permitan valorar un lugar que has visitado de forma que otros usuarios puedan saber a qué se van a enfrentar antes de llegar a él.

La aplicación **Guía Turístico Virtual** pretende exactamente eso: ser una guía de información enfocada al turismo.

1.2. Propósito

Esta propuesta surge con la finalidad de mostrar al usuario lugares de interés turístico cercanos a su entorno, de forma que le sea más sencillo la elección y planificación de su visita a un lugar concreto.

Esta labor se realizará a través de una aplicación Android, diseñada tanto para Smartphones como para Tablets, de forma que el usuario pueda consultarla en cualquier momento.

Dicha aplicación será alimentada a través de un servidor REST, de forma que la consulta de información sea eficiente y veloz.

A su vez, este servidor REST recuperará y guardará la información en una base de datos no relacional, o NoSQL, que permitirá una rápida lectura/escritura.

1.3. Motivación

Han sido varios los motivos que han llevado a la realización de este proyecto:

1. La constatación de que en el mercado, aunque existen diversas aplicaciones capaces de encontrar puntos de interés cercanos al usuario, la mayoría de estas aplicaciones solo se centran en comercios, centros de estética o restaurantes, no hay ninguna centrada en el ámbito turístico, o que verdaderamente ofrezcan información sobre el punto de interés además de su localización.

2. El aumento y popularidad actual de Smartphones y Tablets, hasta el punto que prácticamente todo el mundo posee un dispositivo de este tipo.

3. La facilidad de transporte y utilización de este tipo de dispositivos, de forma que la aplicación esté disponible a los usuario siempre que la necesiten.

4. El hecho de que el 87 % de estos dispositivos utilicen el sistema operativo Android.

Teniendo esto en cuenta, se ha diseñado y desarrollado **una aplicación** para la plataforma **Android** que permitirá al usuario conocer los puntos de interés que existen a su alrededor, dando una breve descripción del mismo basada en información extraída de automáticamente de la web. Además, la aplicación dará al usuairo la opción de valorar y comentar cada punto de interés.

A su vez, se ha diseñado un **servidor REST** que será el encargado de nutrir de información a la aplicación Android, además de actualizar los puntos de interés en función de las valoraciones y comentarios de los usuarios.

Por último, el **servidor REST** se comunica con una base de datos NoSQL, particularmente MongoDB, que almacenará la información sobre los puntos de

interés de forma no-relacional. Esta base de datos permitirá realizar búsquedas basadas en geoposicionamiento de forma eficiente, lo que es ideal para los objetivos de nuestra aplicación.

1.4. Objetivos

Los objetivos de este proyecto son los siguientes:

1. Búsqueda bibliográfica de la documentación necesaria para el desarrollo del proyecto.
2. Estudio en profundidad de la plataforma Android.
3. Estudio detallado de la tecnología REST.
4. Estudio, comprensión y aprendizaje de qué son y cómo funcionan las bases de datos no relacionales (NoSQL). Particularmente, el estudio se centrará en **MongoDB** que será la base de datos utilizada.
5. Implementación de una aplicación para la plataforma Android que muestre al usuario final información sobre los puntos de interés cercanos al mismo, permitiéndole además valorar, comentar y compartir cada uno de ellos.
6. Diseño e implementación de un servidor REST con el que se comunicará la aplicación Android, y que permita realizar de forma eficaz y segura las distintas funcionalidades que la aplicación ofrece.
7. Diseño de la base de datos no relacional que permita almacenar la distinta información que se necesitará de los puntos de interés.
8. Recogida de datos sobre puntos de interés para poblar la base de datos a partir de Google Places API y DBpedia.
9. Estudio de la efectividad, seguridad y facilidad de uso del sistema mediante pruebas.
10. Generación de toda la documentación necesaria.
11. Memoria detallada del proyecto realizado.

CAPÍTULO 2

Estado Del Arte

2.1. Introducción

Como se ha comentado anteriormente, a pesar de que actualmente existen diversas aplicaciones que proveen al usuario de los lugares que están próximos a él, son pocas las que ofrecen unos servicios orientados al turismo: la mayoría de las aplicaciones actuales tan sólo permiten localizar **cualquier lugar** que esté dentro de las inmediaciones del usuario, mostrando, en ocasiones, tan solo el nombre de dicho lugar y su posición geográfica.

El usuario recibe así una cantidad abrumadora de lugares, de los que la mayoría no le son de interés.

También existen excelentes aplicaciones turísticas que ofrecen justo lo que un usuario podría buscar: localización de puntos de interés turístico cercanos, información sobre ellos, posibilidad de ver valoraciones de otros usuarios y poder valorar ellos mismos el lugar, etc. Sin embargo, por lo general estas aplicaciones son diseñadas para una zona geográfica concreta, como una ciudad o una provincia concretas, pero no a nivel general.

2.2. Google Places API

La API de Google Places es un servicio que devuelve información sobre sitios, definidos en la API como establecimientos, ubicaciones geográficas o sitios de interés importantes, mediantes solicitudes HTTP. En las solicitudes de sitios, las ubicaciones se especifican en forma de coordenadas de latitud/longitud. [Google]

De esta forma, utilizando la API de Google Places, y mediante una simple solicitud HTTP, podemos obtener información sobre todos los sitios cercanos a una determinada ubicación geográfica. Además, la API permite restringir la búsqueda de forma que solo se recupere la información de aquellos sitios que coinciden con al menos uno de los tipos específicados en la consulta. Así, por ejemplo podemos preguntar por aquellos lugares que sean del tipo 'museo'.

La información que la API de Google Places proporciona de cada lugar es limitada: para cada uno de los sitios de interés, Google Places devuelve, entre

otros, los siguientes atributos: el nombre del lugar, su ubicación geográfica (latitud/longintud), el tipo o tipos a los que el lugar pertenece (museo, restaurante, sinagoga, etc.) y la valoración de ese lugar por los usuarios de Google.

Muchas de las aplicaciones actuales que ofrecen servicios de búsqueda de lugares basadas en proximidad al usuario están fundamentadas en la API de Google Places: Google Maps sin ir más lejos. Sin embargo, los datos que proporciona la API de Google Places son pocos a la hora de mostrar al usuario información de puntos de interés turístico.

2.3. Plataformas móviles actuales

El mundo de los smartphones ha explotado en popularidad. Tanto es el deseo e interés por uno (y el desdén de no querer volver a un 'teléfono tonto' después de probar uno), que a desaparición de los teléfonos convencionales es inminente.

El propósito de esta sección es descubrir tanto los orígenes de los smartphones como ver las ventajas y desventajas que ofrecen los sistemas operativos dominantes.

2.3.1. ¿Qué es un smartphone?

La palabra 'smartphone' significa, literalmente, "teléfono inteligente". Sin embargo, un smartphone no piensa, ni nos hace más inteligentes.

La principal diferencia entre un smartphone y un teléfono convencional es que nos abre las puertas hacia un mundo de posibilidades, de acciones previamente reservadas para un dispositivo más completo o complejo como pudiera ser un PC. Si un teléfono convencional puede hacer llamadas y, quizás, dejarnos escuchar radio FM, MP3 o tomar fotos, un smartphone nos permitirá navegar por la web, enviar y recibir documentos y editarlos, jugar los más recientes títulos móviles, y no sólo escuchar música o ver vídeos, sino crearlos.

Así pues, una buena definición de smartphone es que se trata de una computadora de bolsillo.

2.3.2. Breve historia

Tradicionalmente, se conoce a un smartphone como un teléfono móvil construído en torno a un sistema operativo móvil de mayor complejidad y posibilidades que un teléfono tradicional. Otro aspecto clave es la posibilidad de instalar aplicaciones y brindarle la oportunidad a desarrolladores, de que creen aplicaciones para esta plataforma.

Estos dispositivos se originaron como una derivación de los PDAs, o Personal Data Assistants. Antes del auge de los smartphones, eran muchos los que solían utilizar un dispositivo adicional al teléfono que les permitía hacer todo lo mencionado (navegar por la web, editar y revisar documentos, etcétera). Las plataformas eran aún algo primitivas si las comparamos a las actuales, pero ese "feeling" de tener una computadora en la mano, ya existía. El paso lógico era fusionar el teléfono con la PDA. Así pues, nacieron los primeros smartphones.

Hubo una gran cantidad de plataformas en aquel entonces. Sistemas Operativos como Symbian, Windows Mobile, Palm OS, competían por la atención de los usuarios.

Pero el alcance en audiencia de los dispositivos era limitada. Debido a la complejidad y lo poco amigable que resultaba el sistema operativo, sólo los más los amantes de la tecnología terminaban utilizándolos. Incluso la salida del Blackberry de RIM, que logró popularizar los smartphones entre empresarios por su genial cliente de correo y su sistema operativo más sencillo de utilizar, no llegó a tocar al público en general, siendo utilizado más por empresarios.

Hasta que llegó el iPhone.

El iPhone de Apple marcó una nueva era. Gracias a la simplicidad de uso (toda interacción con el teléfono lo hacemos a través de pantallas táctiles, con gestos naturales), el iPhone quitó la barrera más grande que la mayoría percibía al interactuar con un teléfono inteligente: la complejidad. Esto, junto a un dispositivo que cubría más las necesidades más comunes que centrarse en completar complejas tareas (el primer iPhone se lanzó sin un App Store y aplicaciones de terceros, centrándose más en el acceso a la web, correo y música), abrió las puertas para que millones de personas se dieran cuenta de que, en efecto, un dispositivo que nos permita estar conectados a la web constantemente, que actúe como una

suerte de mini computadora, era algo que todos deseábamos. El iPhone fue, pues, el teléfono que democratizó y popularizó el uso de los smartphones.

Tras la salida del iPhone, el mercado de teléfonos sufrió un cambio dramático. Las plataformas que antes se consideraban como las más innovadoras pasaron de la noche a la mañana a considerarse como reliquias. Symbian, Windows Mobile, Palm OS; una a una, las plataformas que marcaron los primeros años de los smartphones, fueron desapareciendo.

En paralelo, sin embargo, Google estaba desarrollando una plataforma móvil para poder competir contra iOS (el sistema operativo del iPhone, bajo el cual corren todas las aplicaciones). Ideado inicialmente como un sistema similar al Blackberry OS (es decir, estaba creado para funcionar con un teclado físico), el rumbo de Android la plataforma de Google rápidamente tuvo que cambiar de dirección, pues era claro que el futuro estaba en la interacción con pantallas táctiles (sin necesidad de utilizar un stylus, o lápiz óptico, tan habituales en las PDAs).

Tomó tiempo, pero Android por fin se convirtió en una plataforma rival, ofreciendo similares (y, en algunos casos, mejores) características que el teléfono de Apple. La gran ventaja, no obstante, estuvo en el plan inicial de Google: al permitirle a los fabricantes utilizar su sistema operativo de manera gratuita (a diferencia de otras plataformas como Windows Mobile en aquel entonces, donde por cada terminal vendido, la empresa tenía que pagarle a Microsoft cierta cantidad de los ingresos, el costo de la licencia de Windows Mobile), logró llenar el mercado con dispositivos con Android. Muchos fabricantes que antes apostaron por otras plataformas, como Samsung, Sony(en aquel entonces Sony Ericsson), LG o Motorola; saltaron al sistema operativo de Google para poder competir directamente con Apple.

Con esto llegamos al día de hoy, donde el mercado está prácticamente dominado por ambas plataformas. Que sean las dos principales no significa que sean las únicas. De momento, hay una fuerte lucha por el tercer lugar, donde Microsoft, con su renovado Windows Phone 8.1 OS (su sistema operativo moderno, como reemplazo al antiguo Windows Mobile), y Blackberry (antes RIM) con su Blackberry 10 OS, están compitiendo por conseguir una mayor cuota de mercado.

2.3.3. Android

A pesar de tener un duro comienzo, con el tiempo, ha ido mejorando considerablemente, hasta el punto de convertirse en un referente.

En septiembre de 2013 se superaron los mil millones de dispositivos Android en todo el mundo desde que se pusiera a la venta el primer dispositivo, una cifra que parece que durante este 2014 va a ser doblado.

Se estima que sólo durante este 2014 se van a vender más de 1.000 millones de dispositivos Android. En otras palabras, lo que Android tardó en vender en cinco años lo podrá vender este año en tan solo 12 meses. Con esta previsión, en 2014 se superarían los dos mil millones de dispositivos Android en todo el mundo. [Android, 2014]

Sistema Operativo	**2012**	**2013**	**2014**	**2015**
Android	503.690	877.885	1.102.572	1.254.367
Windows	364.272	327.956	159.855	422.726
iOS/Mac OS	213.690	266.769	344.206	397.234
RIM	34.581	24.019	15.416	10.597
Chrome	185	1.841	4.793	8.000
Otros	1.117.905	801.932	647.572	528.755
Total	**2.216.322**	**2.300.402**	**2.474.414**	**2.621.678**

Cuadro 2.1: Ventas de dispositivos por Sistema Operativo

A lo largo del tiempo, Android ha ido evolucionando y mejorando tanto su funcionalidad como su eficienca, con lo que diversas versiones del sistema operativo han sido publicadas desde su primer dispositivo. El reparto de versiones actualmente es el siguiente: [Google, 2014]

A la vista de estos datos, está claro que si se quiere desarrollar una aplicación que llegue al mayor número de usuarios posible, y que sea accesible a la mayor parte de la población, la mejor plataforma es Android, no sólo por el estado actual, sino porque en el futuro las previsiones son que seguirá creciendo. Android es una plataforma muy extendida, madura, y con mucho futuro por delante.

Versión	Nombre	API	Distribución
2.2	Froyo	8	0.8 %
2.3.3 2.3.7	Gingerbread	10	14.9 %
4.0.3 4.0.4	Ice Cream Sandwich	15	12.3 %
4.1.x	Jelly Bean	16	29.0 %
4.2.x		17	19.1 %
4.3		18	10.3 %
4.4	KitKat	19	13.6 %

Cuadro 2.2: Distribución de versiones de dispositivos Android

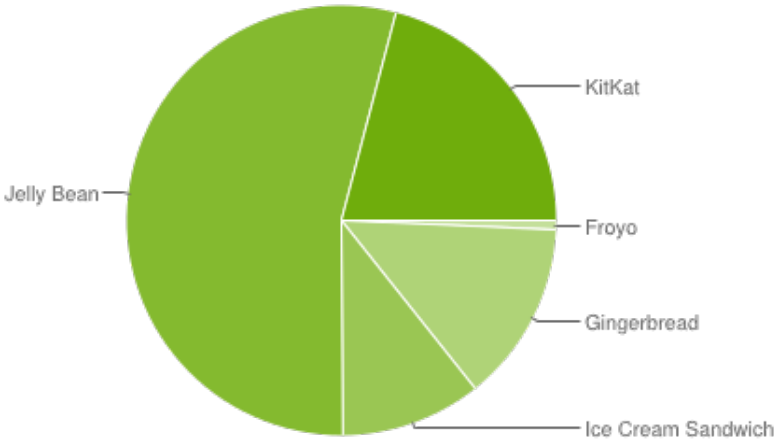


Figura 2.1: Distribución de versiones de Android 2014

2.4. Aplicaciones reales

Como se ha mencionado anteriormente, ya existen aplicaciones que ofrecen al usuario la posibilidad de obtener los lugares de interés geográficamente próximos a él. A su vez, también existen diversas aplicaciones orientadas a facilitar al usuario su visita turística a un lugar determinado.

En esta sección veremos algunas de esas aplicaciones, qué ventajas ofrecen y qué cosas que podrían mejorarse.

2.4.1. Buscar lugar cerca de mí

Buscar lugar cerca de mí [Plt] es un completo recurso para encontrar lugares de cualquier tipo próximos al usuario. Disponible para Android de forma gratuita.

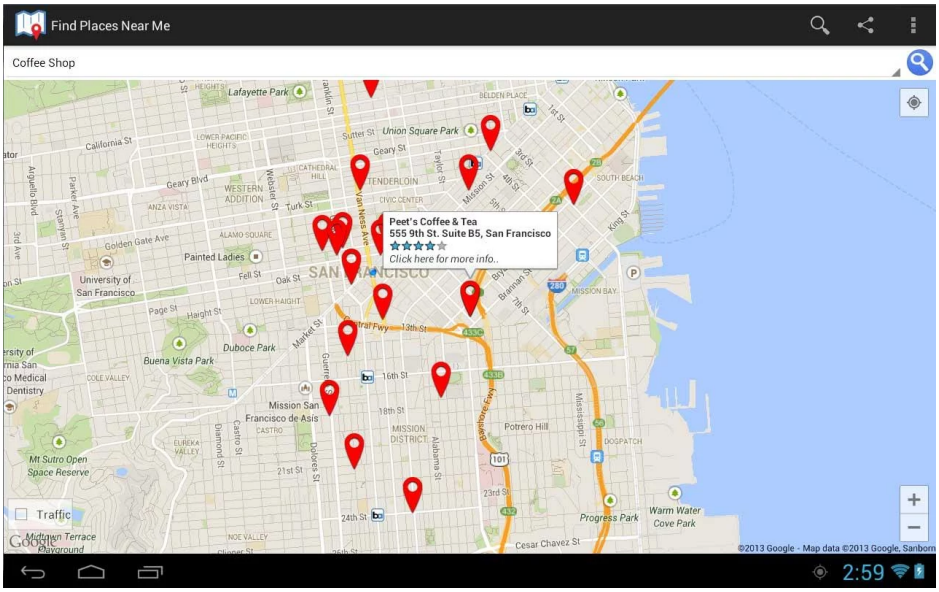


Figura 2.2: Buscar lugar cerca de mí

Algunas ventajas que aporta la aplicación son:

- Realiza búsquedas de lugares cercanos de cualquier categoría.
- Permite filtrar resultados en función del tipo de establecimiento o punto de interés.
- Adaptado tanto a móviles como a tablets.

Además de estas ventajas, la aplicación presenta los siguientes inconvenientes:

- Ofrece muy poca información sobre los lugares cercanos.
- Ofrece búsquedas por nombre que no funcionan muy bien.
- Tan solo permite filtrar resultados por tipo de uno en uno.
- No permite comentar ni valorar los lugares.
- Se basa en consultar la API de Google Places y parsear la información.

2.4.2. Be Your Guide - Toledo

Entrando más en el ámbito turístico, **Be Your Guide - Toledo** [S.L.] es una completa guía de Toledo donde el usuario puede acceder a toda la información sobre los puntos de mayor interés de la cuidad de Toledo, combinando monumentos, cultura, gastronomía, hostelería y ocio.

Está desarrollada para la plataforma Android y es completamente gratuita.

Figura 2.3: Be Your Guide - Toledo

Como se puede apreciar en la pantalla principal de la aplicación, la información ha sido dividida en varias secciones o apartados (Museos y monumentos, Alojamiento, Vida nocturna, Audioguías, Bares y restaurantes, y compras) facilitando al usuario la navegación por la misma.

Algunas de las principales ventajas o características que presenta la aplicación son:

- Interfaz de usuario agradable y fácil de utilizar.
- Audioguías para profundizar más en muchos de los puntos de interés disponibles.

- Disponibilidad de mapas integrados con funcionalidad de "cómo llegar".
- Información abundante y actualizada de cada punto de interés, establecimiento o audioguía.
- Disponible en 6 idiomas.
- Gratuita.

Como principales inconvenientes podemos destacar los siguientes:

- La aplicación está centrada únicamente en Toledo: no permite buscar información de ningún otro lugar.
- Aplicación pesada, sobre todo en el modo offline.

Otras aplicaciones similares

Existen muchas otras aplicaciones que ofrecen servicios muy similares, si no los mismos, que las anteriormente mencionadas. Por ejemplo:

- Cerca de Mi, Lugares cercanos [Creativo]
- Lugares cerca de mí [Court]
- Places Near Me [Infosolutions]

2.5. Conclusiones

Teniendo en mente las aplicaciones anteriores se hace evidente que la mayor parte de las aplicaciones actuales basadas en localización cercana se pueden dividir en dos grandes grupos:

1. Aplicaciones que se limitan a ofrecer lugares próximos al usuario haciendo uso de alguna API externa a la propia aplicación, como la API de Google Places, sin aportar al usuario más información que la ubicación geográfica del punto de interés.

2. Excelentes aplicaciones turísticas que ofrecen una información detallada e interesante de cada lugar, pero que limitan su información a un ámbito o región muy concreto, de forma que si deseas hacer turismo fuera de la zona abarcada, la aplicación no te servirá de mucha ayuda.

Las aplicaciones incluidas en la primera categoría suelen ser menos descargadas por los usuarios, o desinstaladas del dispositivo al poco tiempo de su adquisición, debido a la poca funcionalidad que ofrecen. Una simple búsqueda en Google Maps puede darte la misma o mejor información que este tipo de aplicaciones, que suelen ser pesadas y poco intuitivas.

Por otro lado, son sin duda las aplicaciones del segundo tipo las más utilizadas y mejor valoradas por el usuario ya que, aunque están restringidas a una zona concreta, ofrecen una mayor y más veraz información.

Es por todo esto que el principal objetivo de este proyecto es desarrollar una aplicación sencilla que facilite al usuario, no solo los lugares de interés turístico cercanos a él, si no también cierta información sobre ellos, además de la posibilidad de realizar valoraciones y comentarios. Para ello es necesario una base de datos propia que almacene esta información, y un API que permita al cliente Android comunicarse con ella.

CAPÍTULO 3

Android

3.1. Android

Android es un sistema operativo basado en Linux para dispositivos móviles creado por Andy Rubin. En 2005 fue adquirido por Google Inc. y Andy Rubin pasó a formar parte de la compañía como director de plataformas móviles para Google.

En el año 2007, cerca de 100 grandes compañías de varios sectores dentro de las telecomunicaciones, dispositivos móviles, semiconductores, desarrollo software y comercialización, formaron la Open Handset Alliance con el objetivo de desarrollar estándares abiertos para móviles. El sistema operativo Android es una pieza fundamental de esta alianza, y parte de su código se encuentra liberado bajo la licencia Apache.

El primer lanzamiento de Android se llevó a cabo en noviembre de 2007, y en octubre del año siguiente se abrió Android Market para la descarga de aplicaciones. A partir de entonces, y debido a las características de código abierto, portabilidad y adaptación a cualquier tipo de hardware, optimización en el consumo de memoria o alta definición para gráficos y sonido, Android ha experimentado un gran crecimiento, hasta ser la plataforma dominante en cuanto a cuota de mercado.

Todas las características descritas anteriormente, así como la consolidación de Android como la plataforma líder para dispositivos móviles y la relativa facilidad a la hora desarrollar aplicaciones en cuanto a información disponible desde webs oficiales para desarrolladores han hecho que Android haya sido elegida como la plataforma idónea para la realización de este proyecto.

3.2. Evolución de Android

En el Cuadro 3.1 se puede observar, desde su lanzamiento en 2007, la evolución en cuanto a versiones y nivel de API de la plataforma Android. Cada revisión tiene un nombre comercial que hace referencia a un postre, siguiendo un orden alfabético.

Versión	Nombre	API	Distribución
1.0	Apple Pie	1	<0.1 %
1.1	Banana Bread	2	<0.1 %
1.5	Cupcake	3	<0.1 %
1.6	Donut	4	<0.1 %
2.0/2.1	Éclair	5	<0.1 %
2.2	Froyo	8	0.8 %
2.3.3 2.3.7	Gingerbread	10	14.9 %
4.0.3 4.0.4	Ice Cream Sandwich	15	12.3 %
4.1.x		16	29.0 %
4.2.x	Jelly Bean	17	19.1 %
4.3		18	10.3 %
4.4	KitKat	19	13.6 %

Cuadro 3.1: Evolución de Android

3.3. Arquitectura

Como podemos ver en la Figura 3.1, Android tiene una arquitectura dividida en cuatro capas: [Profeta, 2011]

- Núcleo: está formado por el sistema operativo Linux v2.6. Actúa, por tanto, como interfaz entre la aplicación y el dispositivo hardware, y proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de drivers para dispositivos. Es de código abierto bajo licencia GPL v2.

- Runtime: basado en el concepto del máquina virtual de Java (lenguaje utilizado para el desarrollo de aplicaciones para Android), Google desarrolló una nueva máquina virtual acorde a las limitaciones de los dispositivos móviles, llamada Dalvik.

- Librerías nativas: son un conjunto de librerías en C/C++ compiladas en el código nativo del procesador. Estas librerías proporcionan acceso a las funcionalidades del sistema a través de la capa que explicaremos a continuación.

- Entorno de la aplicación: pensada como una capa de acceso a las distintas APIs que permita reutilizar los componentes que la forman fácilmente, lo que simplifica el desarrollo de aplicaciones.

- Aplicaciones: conjunto de aplicaciones instaladas en un dispositivo Android. Comprende tanto las instaladas en el dispositivo por defecto (tales como un cliente de correo electrónico, el calendario o la agenda de contactos) como las realizadas por otros desarrolladores disponibles que pueden ser descargadas a través de la plataforma Google Play.

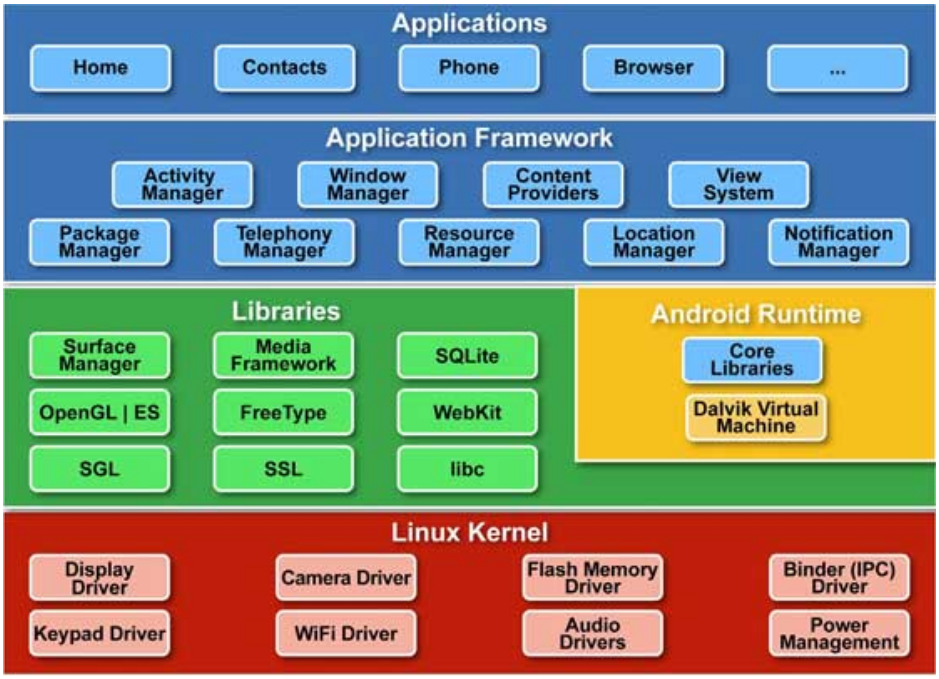


Figura 3.1: Arquitectura de Android

3.4. Componentes

Los componentes son elementos clave en el desarrollo de aplicaciones para Android. Cada componente representa un punto a través del cual el sistema puede acceder a la aplicación, y algunos de ellos también implementan la lógica de la aplicación tras la interacción del usuario con la interfaz. Existen cuatro tipos de componentes en Android:

- Actividades: una actividad implementa la lógica para una pantalla de la interfaz de usuario. Una aplicación, por tanto, estará formada por un conjunto de actividades independientes, aunque pueden interactuar entre ellas e intercambiar información.

- Servicios: un servicio es un componente que se ejecuta en segundo plano y que permite realizar operaciones internas de la aplicación no asociadas a la interfaz de usuario.

- Proveedores de contenido: permiten almacenar y compartir el conjunto de la información que manejan las aplicaciones.

- Receptor de anuncios: permiten responder a anuncios de tipo broadcast. Estos anuncios pueden ser generados por el sistema o bien por una aplicación. No constan de interfaz, se ejecutan en segundo plano y suelen mostrar una notificación para anunciar la ocurrencia del evento.

Los componentes utilizados para el desarrollo de una aplicación han de ser declarados en el archivo AndroidManifest.xml. Cada aplicación tiene el suyo propio. El archivo AndroidManifest.xml permite al sistema operativo conocer toda la información esencial acerca de una aplicación antes de ejecutarla. Algunas de las que se incluye, además de los componentes, son los permisos con los que cuenta la aplicación, librerías, versiones soportadas, etc.

3.5. Seguridad

El sistema operativo Android está basado en privilegios, y cada aplicación que se ejecuta en el sistema, está identificada bajo un identificador de usuario y un identificador de grupo. Además, todas las aplicaciones para Android han de estar firmadas a través de un certificado que permite identificar a su autor.

En cuanto a las aplicaciones en concreto, Android define un nivel de seguridad más que les permite el acceso a las distintas funcionalidades del sistema. Para que una aplicación pueda hacer uso de ciertas características protegidas del dispositivo, es necesario declarar los permisos pertinentes dentro del archivo AndroidManifest.xml.

3.6. Almacenamiento de información

Android proporciona distintas opciones para el almacenamiento de información. En función de las necesidades de nuestra aplicación hemos de considerar una u otra opción. Las distintas opciones son las siguientes:

- Preferencias compartidas: es una clase que permite el almacenamiento de las preferencias del usuario en nuestra aplicación, permitiendo almacenar los valores en pares de tipo clave/identificador y valor asociado a cada clave.
- Almacenamiento interno: almacenamiento de datos en la memoria del dispositivo. Estos datos son de carácter privado, lo que significa que sólo nuestra aplicación puede acceder a ellos.
- Almacenamiento externo: los dispositivos compatibles con el sistema operativo Android soportan almacenamiento externo. Los archivos guardados en este tipo de almacenamiento son accesibles desde fuera de nuestra aplicación y pueden ser modificados.
- Bases de datos: Android proporciona soporte para bases de datos SQLite. Todas las bases de datos creadas desde nuestra aplicación son accesibles desde cualquier clase de la misma, pero no por otras aplicaciones

3.7. Conclusión

Con todo esto, y junto con lo mencionado en la Sección 2.3.3, es evidente que Android es la plataforma idónea a la hora de realizar un proyecto de este tipo: un sistema operativo seguro, bien estructurado, con una excelente documentación y, sobre todo, con una curva de aprendizaje aceptable.

Es por ello que se ha elegido Android, además de por el crecimiento esperado de los usuarios que utilizarán este sistema operavito en España.

CAPÍTULO 4

Tecnología REST

4.1. REST

La **Transferencia de Estado Representacional** (Representational State Transfer) o **REST** es una técnica de arquitectura software para sistemas hipermedia distribuidos. [Fernández, 2013; Tomas, 2014; Wikipedia, c]

Aunque el término *REST* se refería originalmente a un conjunto de principios de arquitectura, en la actualidad se usa en el sentido más amplio para describir cualquier interfaz web simple que utilice HTTP, sin las abstracciones de los protocolos basados en patrones de intercambio de mensajes.

Dicho de otro modo, REST no es más que un conjunto de principios que define la interacción entre distintos componentes. El protocolo más usado que cumple esta definición es el protocolo HTTP.

Los servicios que siguen los principios REST son llamados con frecuencia *RESTful*.

4.2. Reglas definidas por REST

REST define una serie de reglas que toda aplicación que quiera denominarse RESTful debe cumplir. Estos principios fueron diseñados especialmente para construir un sistema escalable y eficiente.

1. Arquitectura Cliente-Servidor
2. Sin estado (stateless)
3. Conjunto de operaciones bien definidas
4. Identificación universal de recursos
5. Uso de hipermedios

4.2.1. Arquitectura Cliente-Servidor

Es necesaria una separación clara y concisa entre los dos agentes básicos en el intercambio de información: el cliente y el servidor. Ambos agentes deben ser independientes entre sí, de forma que aumente la flexibilidad y escalabilidad del sistema en todos los sentidos: los clientes no tienen por qué saber nada acerca del servidor, y éste podrá ofrecer el servicio independientemente del cliente que lo solicite.

4.2.2. Sin estado (stateless)

Cada mensaje o intercambio de información contiene todos los datos necesarios para comprender la petición, sin necesidad de que ni el cliente ni el servidor almacenen ninguna información adicional.

Una petición completa e independiente hace que el servidor no tenga que recuperar ninguna información de contexto o estado al procesar la petición. De esta manera, mejora el rendimiento de los servicios y simplifica el diseño e implementación tanto del servidor y como de sus componentes.

4.2.3. Conjunto de operaciones bien definidas

Debe existir un conjunto de operaciones bien definidas que se aplican a todos los recursos: HTTP en sí define un conjunto pequeño de operaciones. Las más importantes o más utilizadas son GET, POST, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD (Create, Read, Update, Delete) en bases de datos.

- POST se utilizaría para crear un recurso en el servidor,
- GET se usaría para obtener un recurso,
- PUT cambiaría el estado o actualizaría un recurso, y
- DELETE elimiaría un recurso.

Sin embargo, aunque REST define de forma explícita el significado de cada una de las operaciones HTTP, normalmente su uso es ambiguo y son utilizadas de forma incorrecta. Por ejemplo, una petición POST puede crear un recurso si todavía no existe en el servidor, o actualizarlo en caso contrario. Una petición GET puede conllevar a la destrucción de un recurso cuando esta operación se considera sólo de lectura.

4.2.4. Identifiación universal de recursos

Cada recurso debe poder ser identificado únicamente a través de un identificador de recursos uniforme o *URI* (Uniform Resource Identifier).

Las URI de los servicios RESTful deben ser intuitivas, como si de una interfaz auto-documentada se tratara.

4.2.5. Uso de hipermedios

Es necesario el uso de hipermedios tanto para la información de la aplicación como para las transmisiones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML, aunque últimamente la codificación de información en formato JSON está ganando auge, debido a la mejor lectura, comprensión e interpretación de los datos tanto por parte del cliente como del servidor.

4.3. Recursos

Un concepto importante de REST es la existencia de *recursos* que pueden ser accedidos utilizando un identificador global (la URI citada anteriormente). Para manipular estos recursos, los componentes o agentes de la red (clientes y servidores) se comunican a través de una interfaz estándar (como, por ejemplo, HTTP) e intercambian *representaciones* de estos recursos.

La petición puede ser transmitida por cualquier número de conectores (por ejemplo clientes, servidores, túneles, cachés, etc.) pero cada uno de ellos lo hace

sin ver más allá de su propia petición: **stateless**. Así, una aplicación puede interactuar con un recurso conociendo el identificador del mismo y la acción requerida, y obviando si existen otros agentes intermediarios. La aplicación, sin embargo, debe comprender el formato de la información devuelta.

4.4. Conclusión

Como se puede comprobar, atrás quedaron las antiguas y lentas interfaces SOAP para la comunicación con el servidor: REST provee un servicio mucho más sencillo, fácil de escalar y que permite diferenciar cliente de servidor.

Es por ello que se ha elegido utilizar REST como tecnología subyacente en el servidor, que permitirá comunicar los clientes con los recursos almacenados en la base de datos.

CAPÍTULO 5

Bases de Datos NoSQL

5.1. Un poco de historia

Desde la aparición de los primeros ordenadores, junto con el nacimiento de las aplicaciones ofimáticas y las primeras páginas webs, ha existido la necesidad de almacenar información: opciones de configuración de un programa, datos de usuarios, características de productos, etc. Pronto surgió la urgente de necesidad de disponer de un lugar donde guardar de forma segura y permanente toda esta información: nacieron así las primeras bases de datos. [midepa, 2011; Wikipedia, a]

En la década de los 60 los ordenadores bajaron los precios y las empresas privadas empezaron a adquirirlos: la necesidad de guardar la información aumentó, y se dio paso al uso de discos para almacenar datos relacionados que permitía consultarlos sin saber la ubicación exacta de los mismos.

En esta misma época nacieron los primeros modelos de gestión de bases de datos: el Modelo Jerárquico y el Modelo en Red, que almacenaban la información de forma estructurada y ya permitían asociaciones entre registros.

Y en la década de los 70 nacen las Bases de datos Relacionales, de la mano de Edgar Frank Codd, quien además propone una serie de reglas que seguir en estos sistemas. Las Bases de Datos Relacionales, debido a su sencillez y su estructuración de los datos, empezaron a ganar terreno a los modelos jerárquicos y en red.

Casi de la mano de las Bases de Datos Relacionales, en la época de los 80 nace el lenguaje de consulta SQL (Structured Query Lenguage), que rápidamente se convirtió casi en un estándar de la industria, debido a su fácil programación. Entre los 80 y los 90 empezaron a surgir nuevos modelos de bases de datos que estaban basados en el modelo relacional de bases de datos: Oracle, Microsoft Access o SQL Server como ejemplos de sistemas propietarios, y MySQL, PostgreSQL o Firebird como ejemplos de sistemas con licencia libre.

5.2. Bases de datos en la actualidad

En la actualidad, el modelo Relacional es el más extendido y utilizado para cualquier tipo aplicaciones y, con él, el lenguaje SQL es el más usado para realizar consultas.

Sin embargo, con el abaratamiento de los sistemas de almacenamiento y el desarrollo de Internet han nacido nuevos paradigmas de programación como la Computación en la Nube (Cloud Computing), Big Data o Big Users, y con ellos ha surgido la necesidad de que las bases de datos sean capaces de almacenar y procesar grandes cantidades de datos eficientemente, y que ofrezcan un alto rendimiento a la hora de escribir y leer datos, por lo que las bases de datos relacionales están haciendo frente a muchos nuevos retos. Especialmente en aplicaciones con gran concurrencia y escalado, como motores de búsqueda o servicios de redes sociales, parece inadecuado utilizar bases de datos relacionales para almacenar y recuperar información tan dinámica.

Una de las respuestas a estos desafíos han sido las Bases de Datos NoSQL.

5.3. Qué es NoSQL

NoSQL agrupa una gran variedad de diferentes tecnologías que nacieron como respuesta a un gran aumento en el volumen de datos que se almacenaban sobre usuarios, objetos y productos, la frecuencia con la que se accede a esos datos, y las necesidades de potencia y procesamiento. Las bases de datos relacionales no fueron diseñadas para hacer frente a los problemas de escalado y agilidad con los que se enfrentan las aplicaciones modernas, ni fueron construidas para utilizar los avances en almacenamiento barato y procesamiento disponible hoy día.

Por tanto, el término NoSQL es, en realidad, una pobre traducción de lo que realmente la palabra implica. Fue acuñado así debido a que SQL se ha convertido en casi el lenguaje estándar a la hora de tratar con bases de datos relacionales. Sin embargo, NoSQL viene a significar **Not Only SQL**, y no propone ningún nuevo lenguaje, sino una nueva forma de almacenar los datos de forma más eficiente.

5.4. Propiedades de NoSQL

Comparado con las bases de datos relacionales, NoSQL ofrece más escalabilidad y rendimiento. Además, las bases de datos relacionales no fueron diseñadas para ofrecer ciertas facilidades que NoSQL sí:

- Grandes volúmenes de datos
- Sprint ágiles, iteraciones rápidas,...
- Programación orientada a objetos fácil y flexible
- Arquitectura eficiente y fácilmente escalable horizontalmente.
- ...

A continuación podremos ver algunas de las principales cualidades de las bases de datos NoSQL.

5.4.1. Esquema dinámico

Las bases de datos relacionales necesitan saber el esquema de la información que almacenan antes incluso de poder guardarlos. Esto es, antes de poder introducir una tupla en una base de datos relacional debes indicar cuáles son los campos que va a almacenar y de qué tipo.

De la misma forma, si en algún momento se requiere modificar el esquema de una base de datos relacional es necesario redefinir el esquema por completo. Por ejemplo, si se desea añadir una nueva columna en una tabla relacional, es necesario modificar el esquema entero de la tabla para insertar la columna y, después, migrar la base de datos antigua al nuevo esquema.

Las BD NoSQL fueron construidas para permitir **insertar información sin tener un esquema predefinido**. Esto permite realizar cambios significativos en la base de datos en 'tiempo de ejecución', sin preocuparte de interrupciones.

5.4.2. Auto Sharding (particionado)

Debido a la forma en que están estructuradas, las BD Relacionales son difíciles de escalar horizontalmente: es necesario crear réplicas completas de la base de datos con sistemas espejo, y tener complejos balanceadores de carga que distribuyan las peticiones a los diferentes servidores.

NoSQL soporta Auto Sharding nativamente. Esto significa que por defecto y automáticamente, la base de datos repartirá la información por un número arbitrario de servidores, sin siquiera ser consciente de cuántos hay en el sistema ni cuál es su topología.

5.4.3. Replicado

NoSQL también **soporta replicado automático**, proporcionando gran disponibilidad y fácil recuperación ante un desastre.

5.4.4. Caché integrada

Muchas BD NoSQL tienen excelentes técnicas de cacheado, manteniendo en caché la información utilizada frecuentemente, tanto leída como escrita.

5.5. Inconvenientes de NoSQL

Sin embargo, las bases de datos NoSQL no son la panacea.

NoSQL también presenta ciertos inconvenientes. NoSQL no sustituirá nunca a las bases de datos relacionales, pero tampoco fue diseñado para ello: proporciona ciertas ventajas que las bases de datos Relacionales no puede dar, pero es a costa de ciertas garantías que éstas aseguran.

5.5.1. NoSQL no garantiza ACID

El motivo principal por el que NoSQL no podrá sustituir a las bases de datos relacionales es que no garantiza las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Las bases de datos NoSQL ofrecen más eficiencia y rapidez a costa de la integridad de los datos: se supone que no es la base de datos la que debe realizar esta tarea, sino el desarrollador de la aplicación.

En su lugar, NoSQL está basado en el modelo BASE:

- **Basic Availability** Disponibilidad ante todo: replicado, escalabilidad,...
- **Soft State** La consistencia es problema del desarrollador, y no de la base de datos
- **Eventual Consistency** En algún momento del futuro los datos convergerán en un estado de consistencia (aunque sin garantías).

5.5.2. Falta de consistencia

Muy relacionado con el anterior, este es otro gran inconveniente de las BD NoSQL. El escalado de la base de datos y la agilidad de las peticiones se basan en un sistema de cacheado que mantiene en memoria un gran número de operaciones antes de volcarlas a disco. Por ello, un fallo de energía puede hacer que los datos se pierdan de forma permanente.

5.5.3. Tecnología muy específica

NoSQL ha sido diseñado para cubrir unas necesidades muy concretas, con lo que el rango de aplicaciones que pueden estar basadas en NoSQL es bastante limitado: existen aplicaciones o servicios que, por su naturaleza, no podrán encajar nunca en un ambiente NoSQL.

5.5.4. Tecnología poco madura

Aunque no es algo que acabe de nacer, NoSQL sigue siendo una tecnología poco madura, y aún quedan ciertos aspectos que pulir. Es por ello que existen ciertas personas a las que les sigue siendo difícil inclinar la balanza ante este tipo de tecnología frente a las bien asentadas bases de datos relacionales.

5.6. Tipos de bases de datos NoSQL

Como se mencionó anteriormente, NoSQL es un conjunto de técnicas que proporcionan mejor rendimiento que las BD Relacionales para algunas aplicaciones. Sin embargo, NoSQL tan solo esboza los rasgos generales de dichas bases de datos.

Existen diferentes tipos de bases de datos NoSQL, y de cada tipo hay diversas implementaciones; cada una añade, modifica o mejora cierta característica, o especializan otras bases de datos para su aplicación particular, definiendo así una nueva base de datos NoSQL.

A continuación se expondrán los tipos de bases de datos NoSQL más conocidos y utilizados.

5.6.1. Document Store (Basadas en documentos)

El concepto central de una base de datos documental es el término Documento. Mientras que cada implementación de una BD Documental difiere del resto, en general, todas asumen que los documentos encapsulan y codifican la información siguiendo un formato estándar (XML, YAML, JSON, BSON y hasta PDF, Word, Excel,...).

De esta forma, los documentos en una base de datos orientada a documentos hacen las veces de tuplas o filas de las bases de datos relacionales. Sin embargo, los documentos son menos rígidos: no necesitan un esquema predefinido, ni tienen que tener los mismos atributos o claves.

Cada documento se identifica únicamente mediante una clave. Esta clave puede ser una simple cadena, o una ruta hacia el archivo en la base de datos. Normalmente estas claves se mantienen indexadas para que la recuperación sea lo más rápida posible.

5.6.2. Graph Database (Basadas en grafos)

Una Graph Database es una base de datos que utiliza grafos como estructura básica para almacenar la información. Están basadas en la teoría de grafos: en ellas se emplean nodos, propiedades y aristas para representar la información. Los nodos representan elementos de información, como personas, cuentas, productos,...

Las propiedades son información importante que relaciona los nodos. Por ejemplo, el parentesco dos familiares, la relación laboral entre dos empleados, o las palabras que empiezan por la letra F en un diccionario. Cualquier cosa que sea pertinente para una base de datos en concreto.

Las aristas son las líneas que conectan nodos con nodos, o nodos con propiedades, y representan la relación entre ellos. La información más importante de estas bases de datos está almacenada en las aristas, y es de ellas de donde se puede sacar verdadera información.

Comparadas con las BD Relacionales, las bases de datos NoSQL basadas en grafos son más rápidas para conjuntos de datos asociativos, y mapean más directamente a la estructura de aplicaciones orientadas a objetos. Escalan más naturalmente a grandes conjuntos de datos y normalmente no necesitan operaciones JOIN muy costosas.

5.6.3. Key-Value Stores (Basadas en clave-valor)

Las bases de datos basadas en Clave-Valor, también llamadas Tablas Hash Distribuidas, son enormes diccionarios en los que cada pedacito de información (valores) está identificado por un único id (claves), así que para recuperar un valor simplemente tienes que preguntar por su clave, y el sistema te responderá,

independientemente de dónde se encuentre almacenado dicho valor en el sistema distribuido.

Además, en ciertas implementaciones, cada clave puede tener asociada más de un valor (multidimensionales), e incluso, que un valor sea otra clave. De esta forma, se podrían relacionar pares de entradas entre sí formando árboles o jerarquías entre las mismas.

Está optimizado para operaciones atómicas (como sumas o restas) y ofrece lecturas y escrituras muy rápidas, además de un buen balanceo de carga.

5.7. Conclusión

Aunque las bases de datos NoSQL no son muy conocidas todavía, cada día es mayor el número de usuarios que lo conocen y descubren sus grandes cualidades. De hecho, las redes sociales más famosas utilizan de algún tipo esta tecnología para garantizar la rapidez y escalabilidad de sus servicios.

Se ha elegido pues utilizar una base de datos no relacional para almacenar la información de nuestro sistema, de forma que sea fácilmente escalable en un futuro, proporcionando la velocidad y garantías que nuestra aplicación necesita.

CAPÍTULO 6

MongoDB

6.1. MongoDB

MongoDB es una Base de Datos Basada en Documentos disponible para las principales plataformas (Linux, Windows y OSX) que proporciona un alto rendimiento, alta disponibilidad y fácil escalado. MongoDB transforma las tablas de las Bases de Datos Relacionales en documentos codificados en JSON con esquemas dinámicos, haciendo que la integración de ciertos tipos de aplicaciones más fáciles y rápidas (sobre todo las orientadas a objetos). [MongoDB, 2013; Wikipedia, b]

Una distribución de MongoDB puede hospedar varias bases de datos. Cada una de estas bases de datos almacena un conjunto de colecciones (Collections) que, a su vez, guardan un conjunto de documentos. Cada documento es una agregación de pares Clave-Valor con un esquema dinámico. Normalmente, cada documento posee un identificador único que lo diferencia del resto de documentos de la base de datos.

De esta forma, una colección sería un conjunto de documentos que estén relacionados entre sí por algún motivo, de forma que sea más fácil encontrarlos en nuestra base de datos

Algunas de las características que destacan de MongoDB son las siguientes:

6.2. Replicado

MongoDB ofrece una alta disponibilidad y aumenta la potencia con conjuntos replicados: dos o más copias de los datos. Cada copia puede actuar como réplica principal o secundaria en cada momento: la réplica principal realizará todas las lecturas y escrituras por defecto mientras que las secundarias mantendrán una copia de los datos primarios usando la replicación por defecto. De esta forma, cuando un conjunto falla, un algoritmo decidirá cuál de los secundarios debe convertirse en primario. Además, también se pueden realizar lecturas de los conjuntos secundarios (porque estén más cerca, o porque el primario esté bloqueado escribiendo), pero teniendo en cuenta que se mantiene el principio de Eventual Consistency.

6.3. Balanceo de carga

Heredado de las Bases de Datos NoSQL, MongoDB implementa el escalado horizontal utilizando Sharding. Los datos serán divididos en función de una clave y se repartirán en varios shards. De esta forma no se sobrecarga a un solo servidor realizando preguntas sobre ventas de diferentes años.

Así, si MongoDB está siendo ejecutado en varios servidores, el balanceo de carga, junto con el replicado, pueden hacer el sistema resistente a fallos en el hardware. Además, el sistema es capaz de auto gestionarse de forma que si, en algún momento, se añaden más servidores o equipos, la carga se distribuirá por ellos automáticamente.

6.4. Indexado

Cualquier campo de un documento de MongoDB puede ser indexado, agilizando la búsqueda y recuperación de datos. También se pueden crear índices secundarios.

Entre los distintos tipos que ofrece, MongoDB permite crear **índices geoespaciales** de forma que, en lugar de indexar un identificador o un nombre, una posición geográfica (latitud/logintud) es elegida como candidato para realizar búsquedas de forma eficiente.

6.5. Recuperación de datos

MongoDB suporta búsquedas por campo, rango e, incluso, mediante expresiones regulares.

Los índices geoespaciales pueden ser utilizados para realizar búsquedas basadas en ubicaciones geográficas como, por ejemplo, escoger ciertos documentos cuyas coordenadas se encuentren en una región específica, o búsquedas basadas en distancias con respecto a un punto determinado.

6.6. Principales problemas de MongoDB

A pesar de todas sus ventajas, MongoDB también tiene algunos contras que hay que tener en cuenta antes de elegirlo como modelo de base de datos de nuestra aplicación.

Las principales críticas se centran en que MongoDB usa un testigo de lectura-escritura: se permite varias lecturas concurrentes a la misma base de datos, pero tan sólo puede haber una operación de escritura en el sistema (incluyendo las operaciones de lectura). Esto quiere decir que, mientras se esté escribiendo en disco, no se pueden realizar lecturas de la información. En recientes versiones de MongoDB, este testigo se ha mejorado, de forma que tan solo bloquean peticiones de lectura si la información que necesitan se está escribiendo o no está en caché.

6.7. Conclusión

Las bases de datos no relacionales o NoSQL ofrecen muchas ventajas sobre las tradicionales bases de datos relacionales, pero a cambio de ciertas garantías.

Sin embargo, para este proyecto, una base de datos NoSQL es idónea: permite lecturas rápidas y eficientes, reduciendo el tiempo de espera del usuario; permite replicado y distribución de carga si en un futuro es necesario escalar el sistema; y ofrece la posibilidad de construir índices basados en geoposicionamiento, lo que agilizará las búsquedas por proximidad de puntos de interés turístico.

Por otro lado, MongoDB parece ser la distribución indicada para esta aplicación ya que es de código abierto y posee una gran comunidad de desarrolladores trabajando a sus espaldas: existe mucha documentación y casos de uso, su instalación es sencilla y se puede empezar a trabajar con ella inmediatamente.

6.8. Conclusión

Dentro de las bases de datos no relacionales, las bases de datos basadas en documentos era la que mas se ajustaba al sistema implementado. Son diversas

las implementaciones de este tipo de bases de datos que existen hoy día en el mercado.

Sin embargo, finalmente se ha seleccioando MongoDB por ser un implementación de código abierto, con una gran comunidad de desarrolladores activos trabajando en ella, y por la facilidad de integración con los lenguajes de programación utilizados en el servidor, además de todas las ventajas mencionadas anteriormente.

CAPÍTULO 7

Ingeniería del Software

7.1. Introducción

Una vez que hemos presentado el propósito y los objetivos que se pretenden alcanzar con la realización de este proyecto, así como el sistema operativo en el que se basará la aplicación, se va a detallar el proceso seguido para el desarrollo y posterior implementación de la aplicación a través de técnicas de ingeniería del software. [Ing]

La ingeniería del software significa la construcción de software de calidad con un presupuesto limitado y una fecha de entrega en contextos de cambio continuo. Otra definición complementaria define el concepto como *el establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales.*

El proceso de ingeniería del software comprende varias fases y en este proyecto se ha estructurado de la siguiente forma:

- **Análisis:** Consiste en realizar un análisis acerca de las tareas específicas que el software ha de poder realizar, así como el modo de funcionamiento o respuesta de cada una de ellas durante la interacción con el usuario. Este análisis se llevará a cabo a través de:
  - Definición de requisitos
  - Planificación
  - Casos de uso
- **Diseño:** Consiste en realizar un diseño tanto estructural, es decir, desde el punto de vista de la implementación y codificación, como del desarrollo de un prototipo visual de la aplicación, es decir, en cuanto a la interfaz y el modo en que el usuario interactúa con ella. Podemos subdividir esta fase en:
  - Diagrama de clases del diseño
  - Diseño de la interfaz
- **Implementación:** Consiste en traducir el modelo que hemos obtenido en las fases de análisis y diseño a una aplicación real ejecutable sobre una

máquina. Este proceso queda descrito a través de:

- Arquitectura
- Tecnologías utilizadas

7.2. Definición de Requisitos

El propósito de esta fase es asegurar que el proyecto cumple con las expectativas de sus clientes y de sus interesados, tanto externos como internos, siendo el proceso que garantiza el vínculo entre lo que esperan los clientes y usuarios, y lo que se va a desarrollar.

Si bien muchos de sus principios pueden ser adaptados a todo tipo de proyectos, es en los proyectos de desarrollo de software donde adquieren todo su sentido, garantizando el proceso y sirviendo de referencia para asegurar y controlar los cambios que en el proyecto puedan surgir.

Un requerimiento es la condición o capacidad que debe tener un sistema, producto, servicio o componente para satisfacer un contrato, estándar,especificación, u otros documentos formalmente establecidos.

Son todas aquellas características observables que cualquier interesado desea que estén contenidas en el sistema. Como requisitos se incluyen las necesidades, deseos y expectativas del patrocinador, cliente, usuarios, y otros interesados.

Como requerimiento se podría establecer:

- Una capacidad necesaria para un cliente o usuario que soluciona un problema o consigue un objetivo.
- Una capacidad que debe incluirse en un sistema para satisfacer los objetivos del proyecto.
- Una restricción impuesta por algún interesado.

Dado que en este proyecto no hay un cliente real, se interpretará que los requerimientos han sido adquiridos gracias a un conjunto de entrevistas con un cliente virtual, a través de las cuales se han definido los requisitos del proyecto.

7.2.1. Requisitos funcionales

Estos requerimientos se utilizan para determinar que hará el software, definiendo las relaciones de su operación y su implementación, sin olvidar que deben ser explícitos también en lo que el sistema no debe hacer y que validaciones se deben realizar, teniendo en cuenta cuál será el comportamiento del sistema.

Los requerimientos funcionales se pueden dividir en dos puntos de vista:

- El primero tiene relación con el usuario, donde se identifica la relación del usuario con el sistema desde el punto de vista del mismo.
- El segundo tiene relación con el sistema dando respuesta al usuario, es decir desde el punto de vista de lo que realiza el sistema.

Los requisitos funcionales son los siguientes:

1. La aplicación debe ofrecer al usuario una lista de puntos de interés turístico cercanos a su ubicación geográfica.
2. La aplicación ofrecerá al usuario información más detallada de cada punto de interés, compuesta por: nombre del lugar, imagen descriptiva del mismo, ubicación, distancia a la que se encuentra del usuario y breve descripción del lugar.
3. Se podrá ampliar la información sobre el lugar viendo su página desde Wikipedia.
4. El usuario podrá visualizar el lugar en la aplicación Google Maps.
5. La aplicación permitirá al usuario valorar un determinado lugar.
6. La aplicación permitirá al usuario realizar un comentario de un determinado lugar, así como ver los comentarios de otros usuarios.
7. El usuario podrá almacenar en la propia aplicación aquellos puntos de interés que desee, de forma que pueda consultarlos de manera offline.
8. El usuario podrá recargar la información de los lugares cercanos en cualquier momento.

7.2.2. Requisitos no funcionales

Estos requerimientos se basan en las restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo, estándares, usabilidad, portabilidad, entre otros.

Los requerimientos no funcionales son los requerimientos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento.

Los requerimientos no funcionales surgen de la necesidad del usuario, debido a las restricciones en el presupuesto, a las herramientas utilizadas, a las políticas de la organización, a la necesidad de interoperabilidad con otros sistemas de software o hardware o a factores externos como los reglamentos de seguridad, las políticas de privacidad, etcétera.

Requerimientos de apariencia

- La aplicación deberá tener una interfaz gráfica uniforme.
- Se debe respetar lo máximo posible la guía de estilo de Android para no confundir al usuario.
- Debe ser una aplicación nativa para la plataforma Android.
- Su uso debe ser intuitivo para reducir el tiempo de entrenamiento y soporte.
- Tanto la interfaz como los mensajes deben estar en castellano y, opcionalmente, en inglés. Los mensajes de error deben ser lo suficientemente informativos para ayudar al usuario a actuar en consecuencia.

Requerimientos de rendimiento

- Cuando la aplicación Android haga uso de la red, éste deberá ser mínimo.
- Se deben aprovechar los recursos disponibles.

- Los tiempos de respuesta deben ser aceptables, tanto en el servidor REST como en la aplicación.

Requisitos de fiabilidad

- La base de datos debe almacenar la información de forma consistente.

Requisitos de seguridad

- Tanto la aplicación Android como el servidor deben proporcionar un entorno serguro, sin pérdida de datos, y con el menor número de fallos posible.

Requerimientos de portabilidad

- La aplicación debe funcionar tanto en dispositivos tipo smartphone como en tablet.
- Deberá tener una interfaz adaptada en cada situación o tipo de dispositivo.

Requerimientos de software

- Servidor:
  - Sistema operativo: Ubuntu 14.04 LTS
  - MongoDB v2.6.4.
  - MongoDB Driver para PHP v1.5.4.
  - Servidor web Apache configurado para Ubuntu:
    - Apache 2.4.7 (Ubuntu)
    - PHP 5.5.9
  - CodeIgniter v2.2.0 como framework sobre el que desarrollar el servidor REST.

- Aplicación móvil:
  - Entorno de desarrollo integrado (IDE): entorno de programación que ha sido empaquetado como un programa de aplicación. Se compone de una serie de herramientas entre las que se pueden encontrar un compilador, un depurador, un editor de código y un constructor de interfaz gráfica. El IDE utilizado ha sido Android Studio v0.8.6
  - Android SDK Tools
  - Java 1.8.
- Software adicional:
  - Herramienta de diseño: para crear el conjunto de diagramas es conveniente utilizar alguna herramienta que permita la creación de diagramas UML. El software utilizado ha sido Astah 6.8.

Requerimientos hardware

- Smartphone o tablet con las siguientes características:
  - Procesador de un núcleo (1 GHz).
  - 512 Mb de RAM como mínimo.
  - 10 Mb de almacenamiento disponible para almacenar la aplicación.
  - Sistema operativo Android 4.0 o superior.
  - Aplicación Google Play Store instalada.
  - Google Play Services disponibles (instalados y activos).
  - El dispositivo debe tener activadas las opciones que permitan la utilización de redes y satélites GPS para determinar la ubicación geográfica del dispositivo.
- Servidor con las siguientes características mínimas:
  - Procesador DualCore o superior.

- 4 Gb de memoria RAM como mínimo.
- 120 Gb de disco duro aproximadamente.

7.3. Planificación

7.3.1. Estimación de tiempos

La estimación de tiempos del proyecto será la siguiente:

Actividad	**Duración**
Búsqueda bibliográfica	1 semana
Estudio de las tecnologías	2 semanas
Análisis del proyecto	2 semanas
Diagramas de diseño	3 semanas
Diseño de la interfaz Android	2 semanas
Diseño de la base de datos	2 semanas
Diseño del servidor	2 semanas
Corrección, errores, pruebas	1 semana
Completar memoria del proyecto	3 semanas
Tiempo total	**18 semanas**

Cuadro 7.1: Estimación de tiempos

Las tareas de diseño se realizarán en paralelo, junto con las actividades de documentación.

La duración de la realización del proyecto se ha estimado en 18 semanas aproximadamente.

7.3.2. Diagrama de Gantt

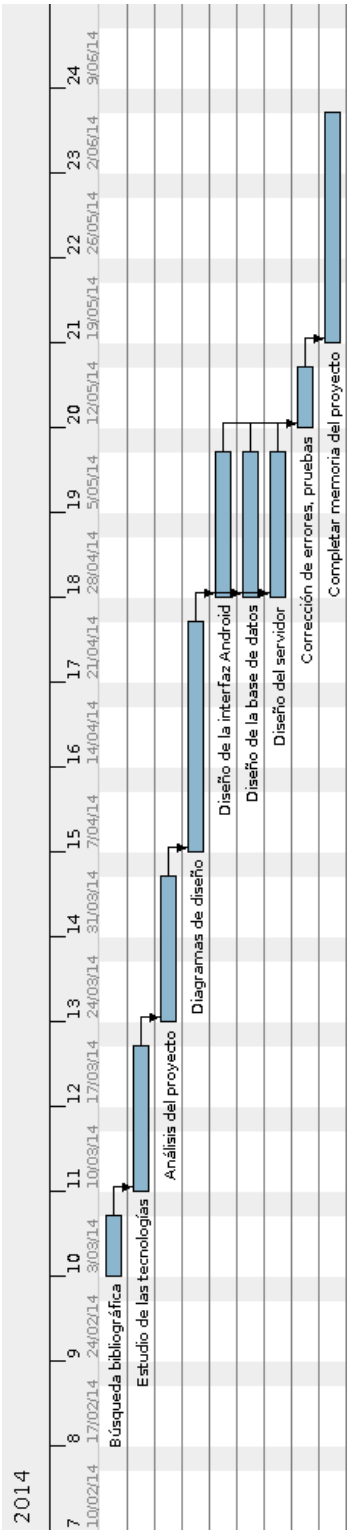


Figura 7.1: Diagrama Gantt del proyecto

7.3.3. Estimación de costes

El equipo de desarrollo que va a llevar a cabo el proyecto está compuesto por los siguientes componentes:

- Jefe de proyecto.
- Analista.
- Diseñador.
- Programador.
- *Tester*, encargado de realizar pruebas.

Jornada laboral: lunes, martes, miércoles, jueves y viernes. 8 horas/día.

Recurso	€/h	Resumen de actividades	Horas		Coste €
Jefe proyecto	60	Supervisión de las actividades.		20	3.000
Analista	45	Análisis del proyecto.		60	2.700
Diseñador	42	Diagrama otras tareas de diseño. Diseño de interfaz de la aplicación. Diseño de base de datos MongoDB. Diseño del servidor REST.	50 20 20 30	120	5.040
Programador	36	Implementación de la interfaz. Codificación de la interfaz. Inserción de la base de datos. Implementación del servidor REST. Integración de MongoDB. Despliegue del servidor.	30 30 20 40 10 20	150	5.400
Tester	20	Simulación y pruebas del sistema		40	800
Total					**16.940**

Cuadro 7.2: Estimación de costes

De esta forma, el coste del personal del proyecto será de un total de 16.940€.

A esto, se deberá sumar el coste de hardware y licencias software utilizadas, que se detalla en el Cuadro 7.3:

El total del coste de hardware y software es de 91,25€

De esta forma, el presupuesto estimado total del proyecto es de 15.511,25€.

Hardware	€/día	Días utilizado	Coste €
5 Equipos informáticos	0,70	75	52,5
Nexus 5	0,40	40	16
Nexus 10	0,55	30	16,5
Software			
Astah Professional	0,25	25	6,25
Total			91,25

Cuadro 7.3: Coste hardware y licencias software

7.4. Análisis

7.4.1. Casos de uso

En este apartado se analizarán los requerimientos previamente definidos mediante una serie de diagramas de casos de uso. Los casos de uso representan una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Los casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.

Cada requerimiento funcional debe estar cubierto, al menos, por un caso de uso. Sin embargo, un caso de uso puede contener varios requerimientos. Para describir cada uno de ellos se va a utilizar el lenguaje natural para que sea fácilmente entendible por el cliente.

La estructura de un caso de uso es la siguiente:

- **Nombre:** identifica al caso de uso de forma única.
- **Descripción:** información sobre el caso de uso que se va a describir.
- **Actores:** entidades externas que se comunican con el sistema. Describen un tipo de usuario y tienen un nombre que los identifica. En nuestro caso existen dos actores: el usuario y el servidor.
- **Precondiciones:** condiciones que han de cumplirse para poder iniciar el caso de uso.

- **Flujo de eventos:** eventos principales y orden en que se suceden a lo largo de la interacción entre el actor y el sistema para un caso de uso.
- **Postcondiciones:** estado en que queda el sistema cuando la interacción del actor ha terminado.
- **Excepciones:** situaciones de error que se puedan llegar a producir.

Un caso de uso puede presentar escenarios alternativos, éstos son distintos flujos de eventos secundarios que pueden darse lugar durante la interacción del actor y el sistema.

Previo a la descripción de cada uno de los casos de uso, vamos a representar una serie de diagramas. En primer lugar representaremos el diagrama frontera, que describe completamente la funcionalidad del sistema y su entorno.

Por ser el diagrama frontera una representación general del sistema, para cada caso de uso vamos a realizar un diagrama que nos permita profundizar y conseguir un mayor nivel de detalle acerca de su comportamiento. Para ello se pueden emplear dos tipos de relaciones:

- «**extend**»**:** es una relación cuya dirección va hacia el caso de uso a detallar, y representa comportamientos excepcionales del caso de uso, tales como errores, etc.
- «**include**»**:** es una relación cuya dirección es contraria a la relación «extend» y representa un comportamiento común del caso de uso.

Diagrama frontera

El diagrama frontera del sistema es el siguiente:

Analizaremos ahora los casos de uso definidos en el diagrama frontera del sistema, que cubren los objetivos especificados.

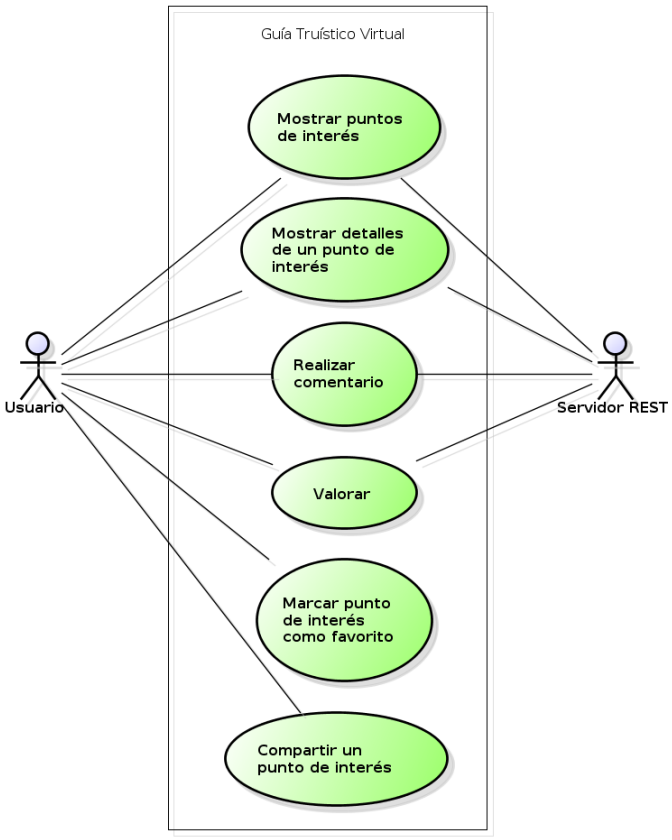


Figura 7.2: Diagrama frontera de la aplicación

Listado de lugares cercanos

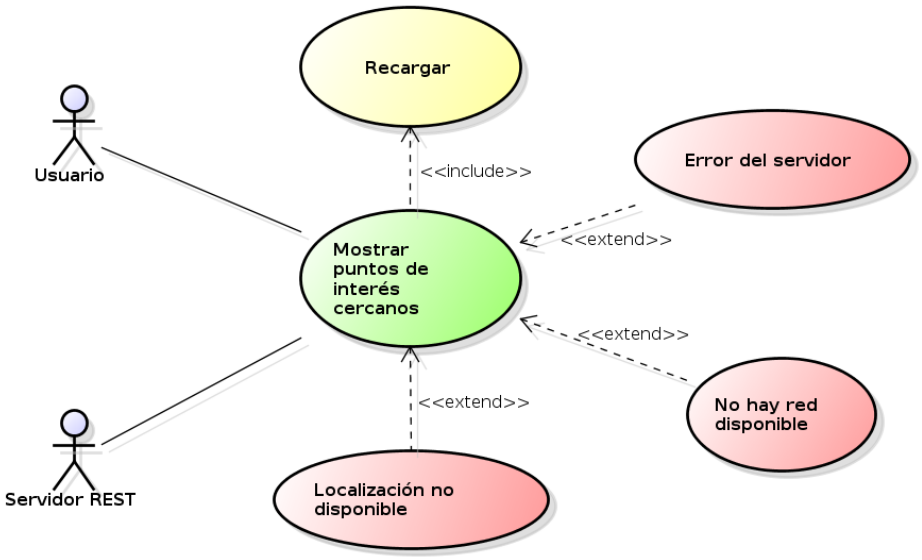


Figura 7.3: Listado de puntos de interés cercanos

Caso de Uso 1	
Nombre	Mostrar puntos de interés cercanos
Descripción	Define la funcionalidad de mostrar aquellos puntos de interés cercanos al usuario.
Actores	Usuario, Servidor REST
Precondiciones	Ninguna
Flujo de eventos	1. Se muestra la pantalla principal de la aplicación. 2. El sistema contacta con el servidor para recuperar puntos de interés cercanos 3. Se refresca la vista para mostrar los resultados
Postcondiciones	Ninguna
Excepciones	Localización no disponible: si los servicios de ubicación no están activados. No hay red disponible: si no hay disponibilidad de conectar a Internet Error del servidor: si se produce un error al contactar con el servidor REST.

Cuadro 7.4: Listado de puntos de interés cercanos

Detalles de un punto de interés

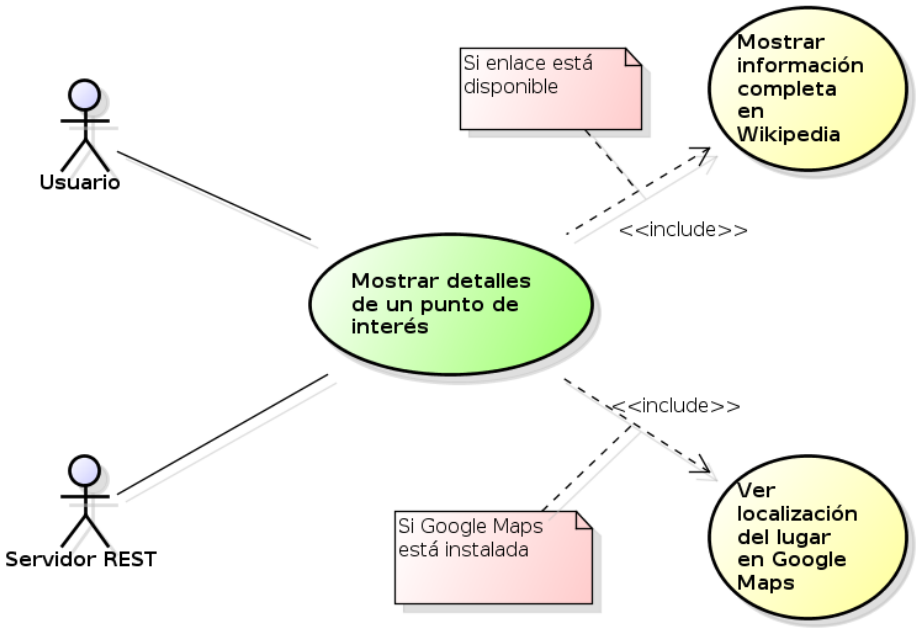


Figura 7.4: Detalles de un punto de interés

Caso de Uso 2	
Nombre	Mostrar los detalles de un punto de interés.
Descripción	Representa la funcionalidad de mostrar de forma detallada la información de un punto de interés concreto.
Actores	Usuario
Precondiciones	Ninguna
Flujo de eventos	1. Se muestra la pantalla de detalles de apliación 2. La aplicación muestra los detalles del punto de interés que tiene alamacenados en caché.
Postcondiciones	Ninguna

Cuadro 7.5: Detalles de un punto de interés

Marcar un punto de interés como favorito

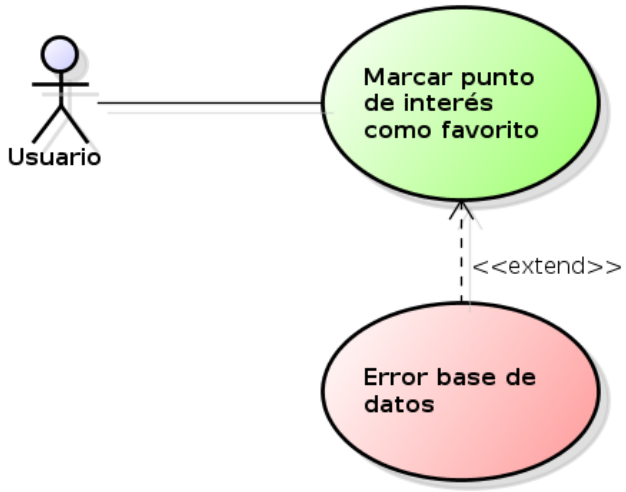


Figura 7.5: Detalles de un punto de interés

Caso de Uso 3	
Nombre	Marcar un punto de interés como favorito.
Descripción	Define la funcionalidad de marcar como favorito un punto de interés, para que pueda visualizarse de forma offline
Actores	Usuario
Precondiciones	El lugar no ha sido previamente marcado como favorito.
Flujo de eventos	1. El usuario pulsa el botón 'Marcar como favorito'. 2. La aplicación almacena la información del punto de interés en la base de datos de Android. 3. Se refresca la vista para mostrar el icono de 'Marcar como favorito'sombreado.
Postcondiciones	Se almacena la información del lugar en la base de datos.

Cuadro 7.6: Marcar un punto de interés como favorito

Caso de Uso 3.1	
Nombre	Desmarcar un punto de interés como favorito.
Descripción	Define la funcionalidad de desmarcar como favorito un punto de interés, para que pueda visualizarse de forma offline
Actores	Usuario
Precondiciones	El lugar ha sido previamente marcado como favorito.
Flujo de eventos	1. El usuario pulsa el botón 'Marcar como favorito'. 2. La aplicación elimina la información del punto de interés en la base de datos de Android. 3. Se refresca la vista para mostrar el icono de 'Marcar como favorito'sin sonbrear.
Postcondiciones	Se elimina la información del lugar en la base de datos.

Cuadro 7.7: Desmarcar un punto de interés como favorito

Comentar un lugar

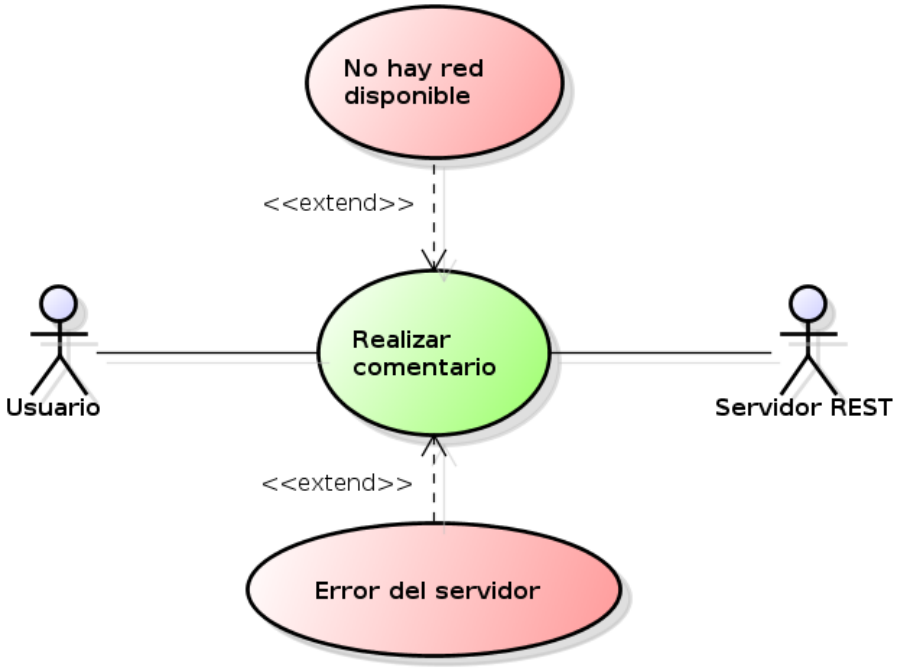


Figura 7.6: Comentar un punto de interés

Caso de Uso 4	
Nombre	Realizar un comentario en un punto de interés.
Descripción	Define la funcionalidad realizar un comentario en un punto de interés.
Actores	Usuario, Servidor REST
Precondiciones	Ninguna
Flujo de eventos	1. Se muestra la pantalla de detalles del lugar. 2. El usuario introduce su nombre y el comentario que desee realizar sobre el punto de interés. 3. La aplicación se pone en contacto con la API para actualizar la base de datos en el servidor.
Postcondiciones	La información del lugar queda actualizada en el servidor.
Excepciones	No hay red disponible: si no hay disponibilidad de conectar a Internet Error del servidor: si se produce un error al contactar con el servidor REST.

Cuadro 7.8: Comentar un punto de interés

Valorar un punto de interés

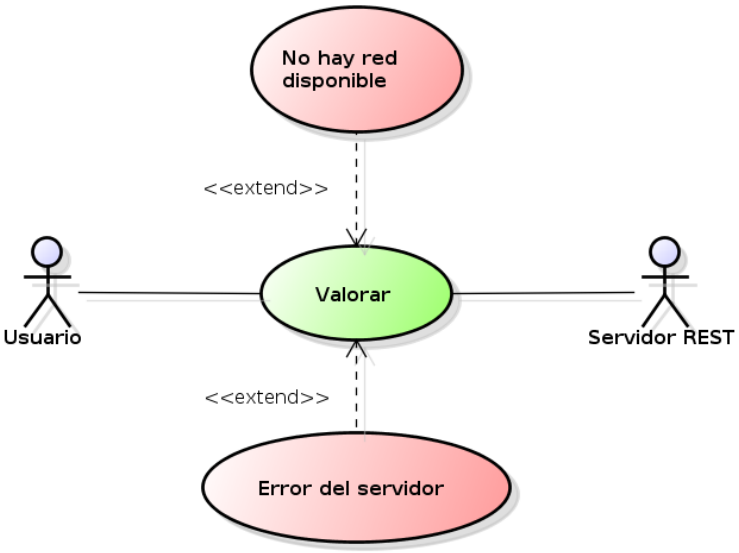


Figura 7.7: Valorar un punto de interés

Caso de Uso 5	
Nombre	Realizar una valoración en un punto de interés.
Descripción	Define la funcionalidad realizar una valoración en un punto de interés.
Actores	Usuario, Servidor REST
Precondiciones	Ninguna
Flujo de eventos	1. Se muestra la pantalla de detalles del lugar. 2. El usuario introduce su valoración del punto de interés. 3. La aplicación se pone en contacto con la API para actualizar la base de datos en el servidor.
Postcondiciones	La información del lugar queda actualizada en el servidor.
Excepciones	No hay red disponible: si no hay disponibilidad de conectar a Internet Error del servidor: si se produce un error al contactar con el servidor REST.

Cuadro 7.9: Valorar un punto de interés

Compartir un punto de interés

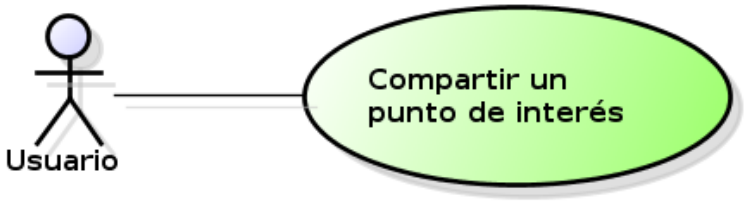


Figura 7.8: Compartir un punto de interés

Caso de Uso 6	
Nombre	Compartir un punto de interés con otras aplicaciones.
Descripción	Define la funcionalidad compartir un determinado punto de interés con otras aplicaciones.
Actores	Usuario
Precondiciones	Ninguna
Flujo de eventos	1. Se muestra la pantalla de detalles del lugar. 2. El usuario pulsa sobre el botón compartir. 3. El usuario elige la aplicción donde desea compartir el punto de interés.
Postcondiciones	Ninguna

Cuadro 7.10: Compartir un punto de interés

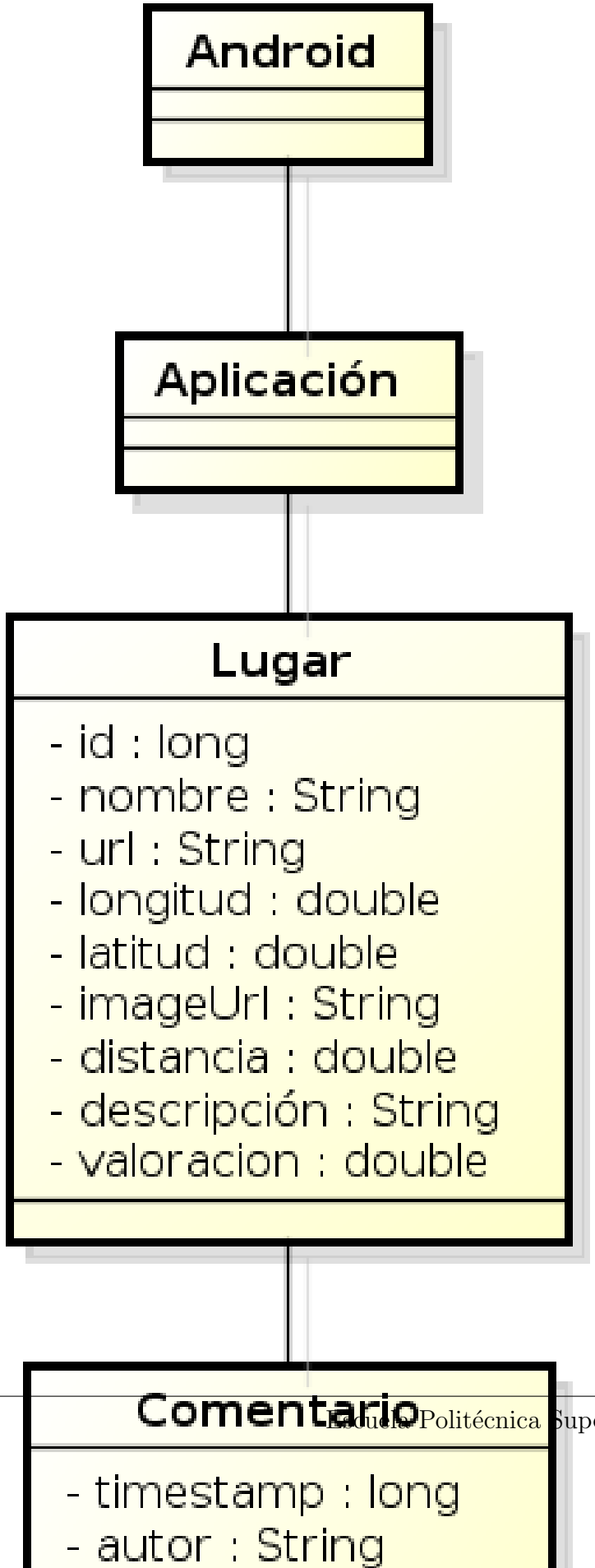
7.4.2. Diagrama de dominio

Una vez que hemos definido los requisitos que nuestra aplicación ha de cumplir y el comportamiento que ha de tener, vamos a identificar las clases candidatas cuyos objetos son necesarios para realizar los casos de uso.

El diagrama de dominio expresa el conocimiento ganado del problema que estamos analizando.

El diagrama de dominio para la aplicación Android es:

En el servidor REST, el diagrama de dominio es idéntico, exceptuando que el modelo 'Place' no tendrá el atributo 'saved', ya que esta es una funcionalidad propia del cliente Android en la que el servidor no interviene.


Android
Aplicación
Lugar
- id : long
- nombre : String
- url : String
- longitud : double
- latitud : double
- imageUrl : String
- distancia : double
- descripción : String
- valoracion : double
Comentario
- timestamp : long
- autor : String

7.5. Diseño

7.5.1. Diagramas de secuencia

Los diagramas de secuencia describen la interacción entre los objetos de una aplicación y los mensajes recibidos y enviados por los objetos.

Son una forma de diagrama de interacción que muestra los objetos como líneas de vida a lo largo de la página y con sus interacciones en el tiempo representadas como mensajes dibujados como flechas desde la línea de vida origen hasta la línea de vida destino.

En cada diagrama el tiempo avanza hacia abajo, y el ordenamiento de los eventos debería seguir el orden indicado en el caso de uso asociado.

Los componentes básicos que conforman un diagrama de secuencia son:

- Líneas de Vida: representa un participante individual en un diagrama de secuencia.
- Mensajes: se muestran como flechas. Los mensajes pueden ser de distintos tipos (síncronos, asíncronos, llamadas, señales, etc.).
- Ocurrencia de ejecución: un rectángulo fino a lo largo de la línea de vida denota la ocurrencia de ejecución o activación de un foco de control.

En estos diagramas se representan los eventos generados por el actor, su orden y los eventos del sistema. Mediante estos diagramas se consigue una visión más concreta de la aplicación y su funcionamiento interno.

El objetivo de estos diagramas es profundizar más en la implementación de cada una de las funcionalidades especificadas en la fase de análisis anterior.

Diagrama de secuencia del proceso de recuperar lugares cercanos

El flujo de eventos asociados al caso de uso que incluye esta funcionalidad:

1. El usuario abre la aplicación o pulsa sobre el botón 'Refrescar'.
2. La aplicación comprueba que es posible obtener la ubicación geográfica del dispositivo. En caso contrario muestra un error al usuario indicándole el problema y sugiriéndole una solución.

3. La aplicación recupera la ubicación del dispositivo.

4. El sistema comprobará entonces si es posible comunicarse a través de internet. En caso contrario, avisará al usuario mostrando un mensaje informativo.

5. La aplicación creará una petición HTTP para contactar con el servidor y recuperar los lugares cercanos al mismo. Esta petición ha de ser asíncrona, de forma que la interfaz de usuario no se paralice mientras ocurre, ya que puede tardar cierto tiempo. De esta forma, la aplicación comunica a Android que quiere lanzar un proceso en un nuevo hilo de ejecución y éste lo lanza.

6. El servidor recibe la petición, se comunica con la base de datos para extraer los lugares cercanos a la posición del dispositivo, y los devuelve en formato JSON.

7. La aplicación recibe estos datos, los parsea, y los añade a la interfaz.

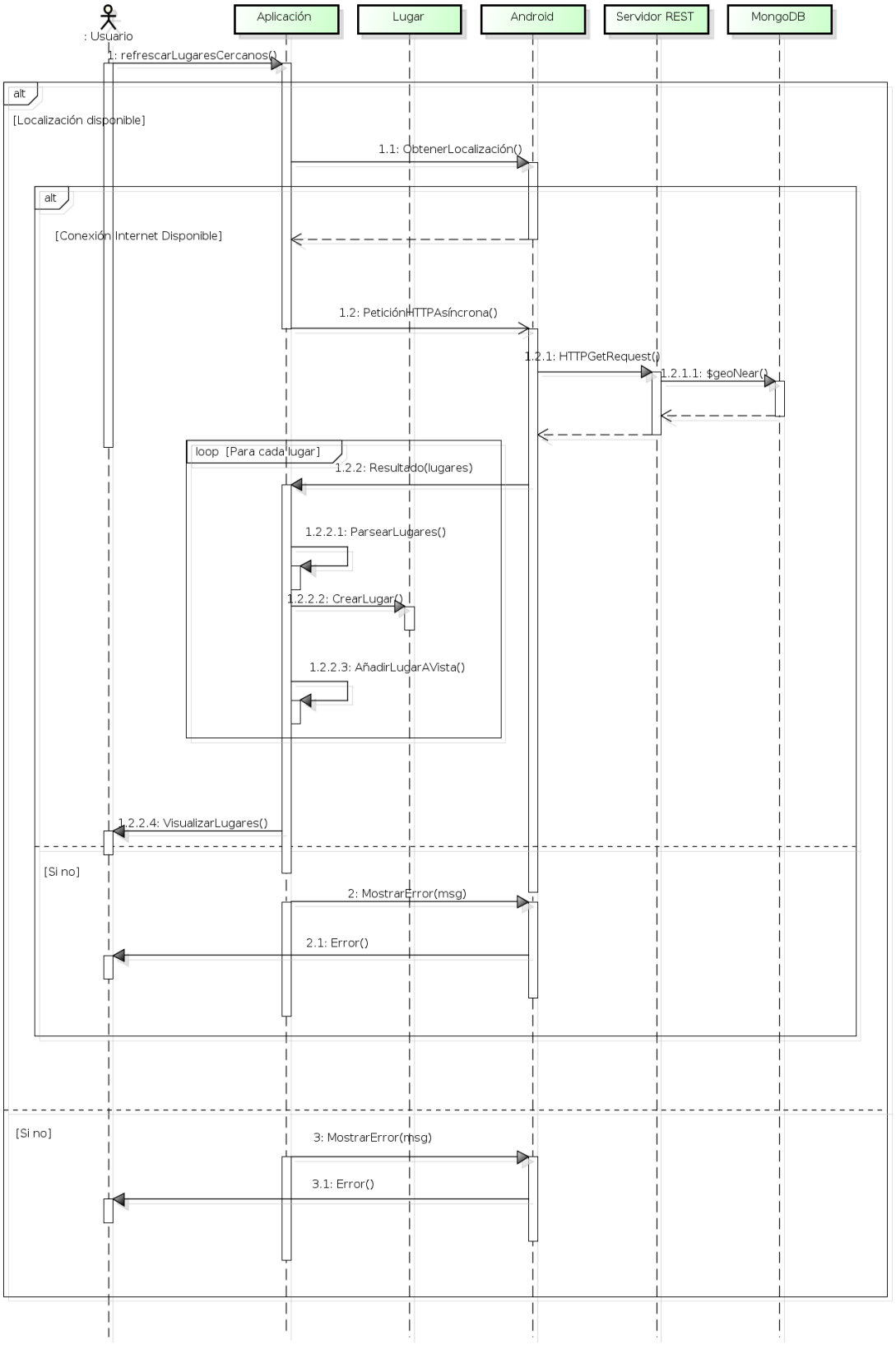


Figura 7.10: Proceso de recuperación y visualización de lugares cercanos

Diagrama de secuencia del proceso de realizar un comentario

El flujo de eventos asociados al caso de uso que incluye esta funcionalidad:

1. El usuario selecciona la opción 'Realizar un comentario'.
2. La aplicación muestra un formulario con los datos que se deben rellenar.
3. El usuario completa los campos y envía el comentario.
4. La aplicación comprueba que existe conexión a internet. En caso contrario advierte al usuario mostrando un mensaje informativo.
5. La aplicación notifica a Android que comienza una tarea asíncrona para realizar la conexión con el servidor.
6. El servidor recibe la petición, la procesa actualizando la base de datos, e informa de si la operación ha sido satisfactoria o ha surgido algún error.
7. La aplicación recibe esa información, la parsea y la interpreta. En caso de que un error haya ocurrido, informa al usuario mostrando un mensaje de error.
8. Si la operación se ha completado correctamente, advierte al usuario con un mensaje informativo, y actualiza el lugar añadiendo el nuevo comentario.

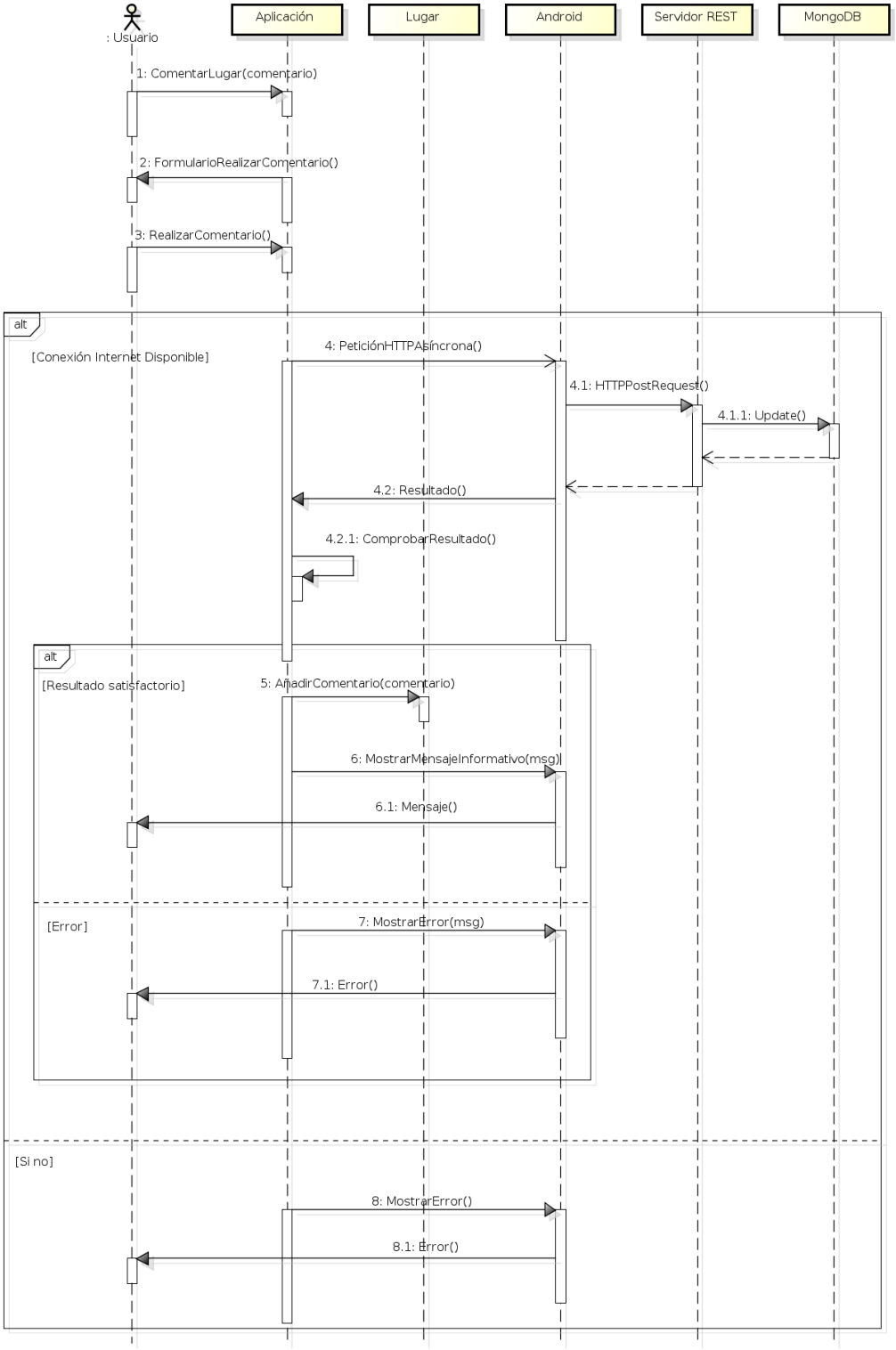


Figura 7.11: Proceso de realizar un comentario

Diagrama de secuencia del proceso de valorar un lugar

El flujo de eventos asociados al caso de uso que incluye esta funcionalidad:

1. El usuario selecciona la opción 'Valorar lugar'.
2. La aplicación muestra un formulario con los datos que se deben rellenar.
3. El usuario completa los campos y envía la valoración.
4. La aplicación comprueba que existe conexión a internet. En caso contrario advierte al usuario mostrando un mensaje informativo.
5. La aplicación notifica a Android que comienza una tarea asíncrona para realizar la conexión con el servidor.
6. El servidor recibe la petición, la procesa actualizando la base de datos, e informa de si la operación ha sido satisfactoria o ha surgido algún error.
7. La aplicación recibe esa información, la parsea y la interpreta. En caso de que un error haya ocurrido, informa al usuario mostrando un mensaje de error.
8. Si la operación se ha completado correctamente, advierte al usuario con un mensaje informativo, y actualiza el lugar añadiendo la nueva valoración.

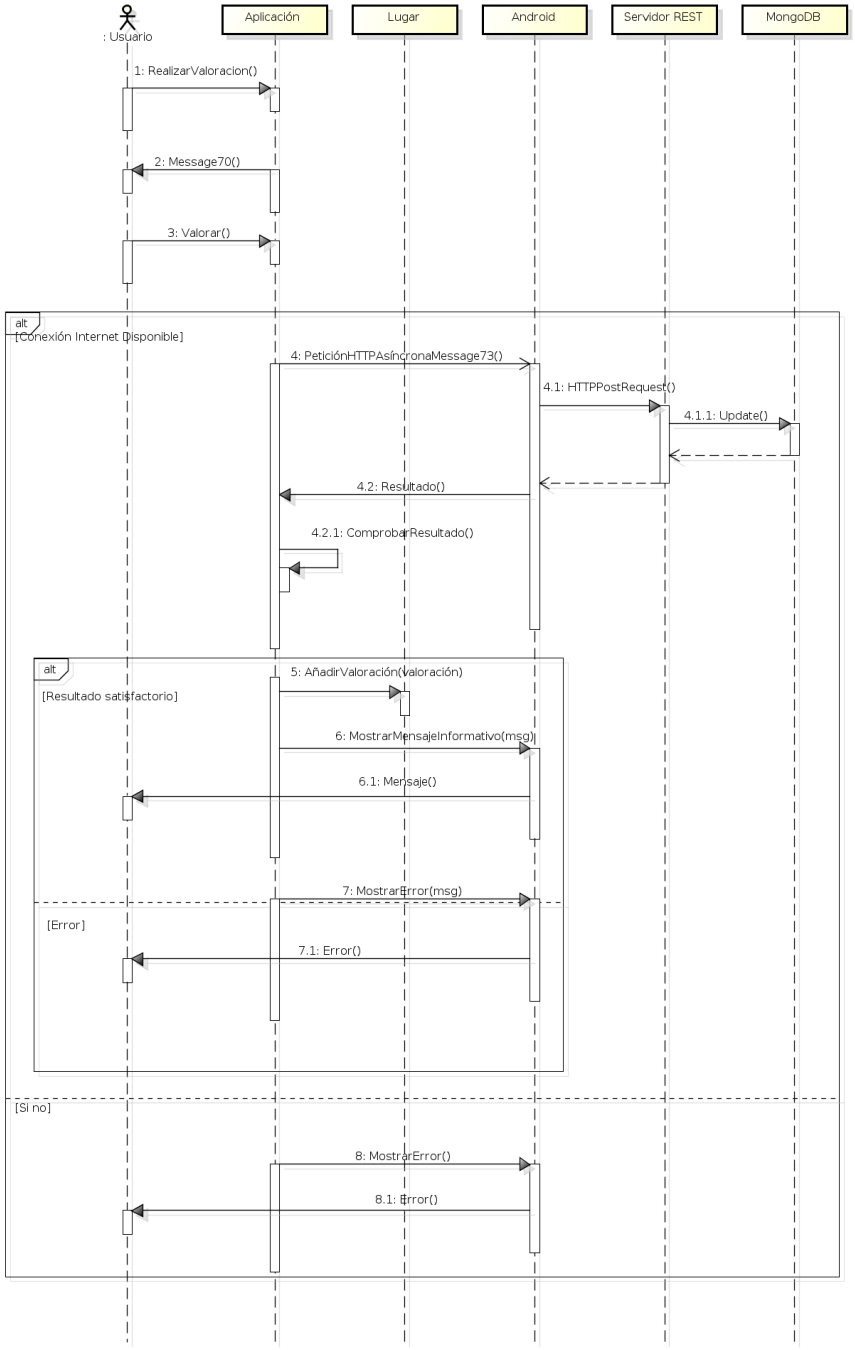


Figura 7.12: Proceso de realizar una valoración

Diagrama de secuencia del proceso de marcar un lugar como favorito

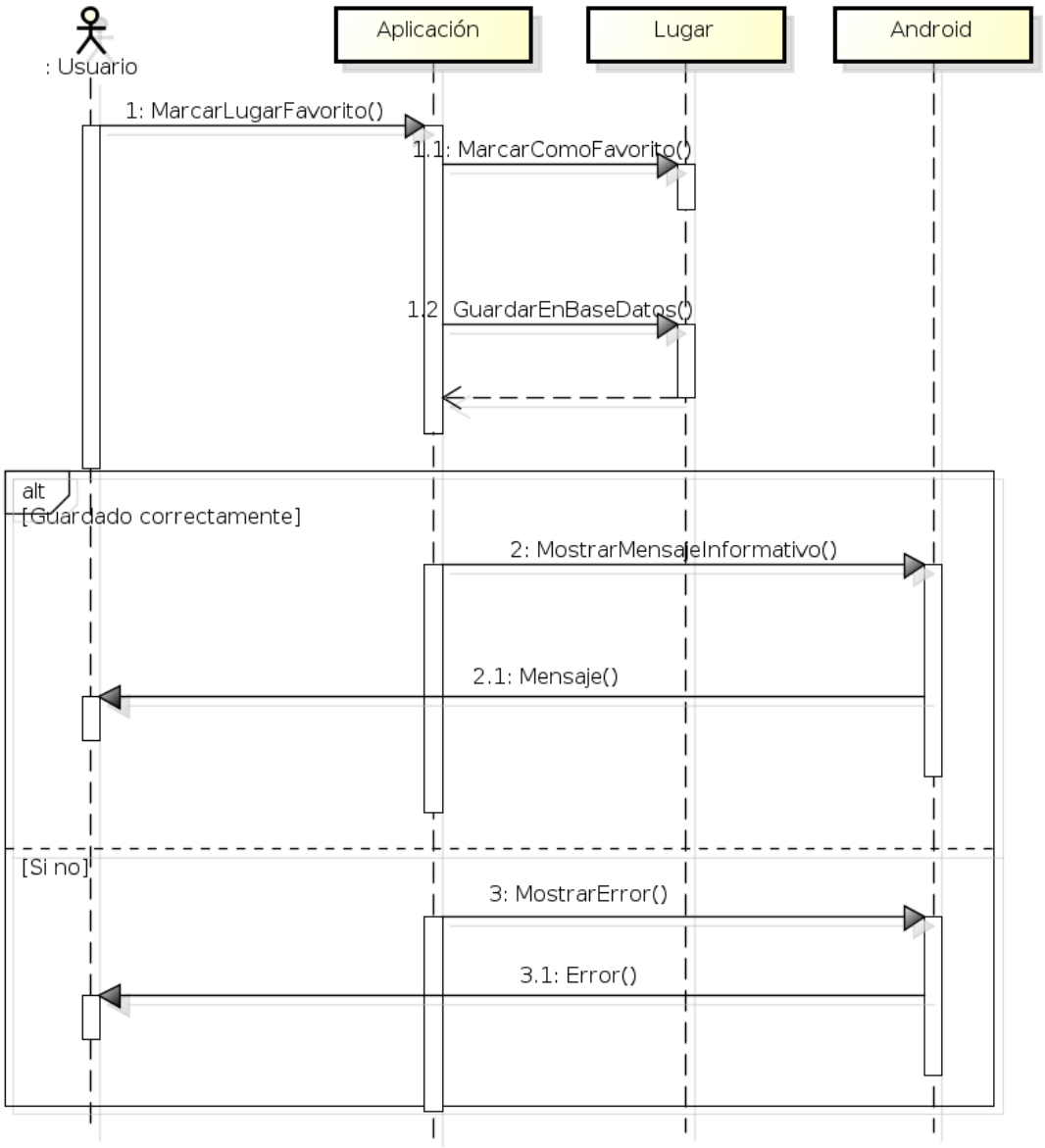


Figura 7.13: Proceso de marcar un lugar como favorito

El flujo de eventos asociados al caso de uso que incluye esta funcionalidad:

1. El usuario selecciona la opción 'Marcar como favorito'.
2. La aplicación actualiza el objeto Lugar para marcarlo como favorito.
3. La aplicación guarda en la base de datos de Android el lugar, para poder visualizarlo de manera offline.

4. Si el objeto se almacena correctamente, la aplicación actualiza la vista para indicar al usuario que el lugar ya está guardado.

5. En caso contrario, la aplicación muestra un mensaje informativo advirtiendo al usuario de que ocurrió un problema.

Diagrama de secuencia del proceso de compartir un lugar

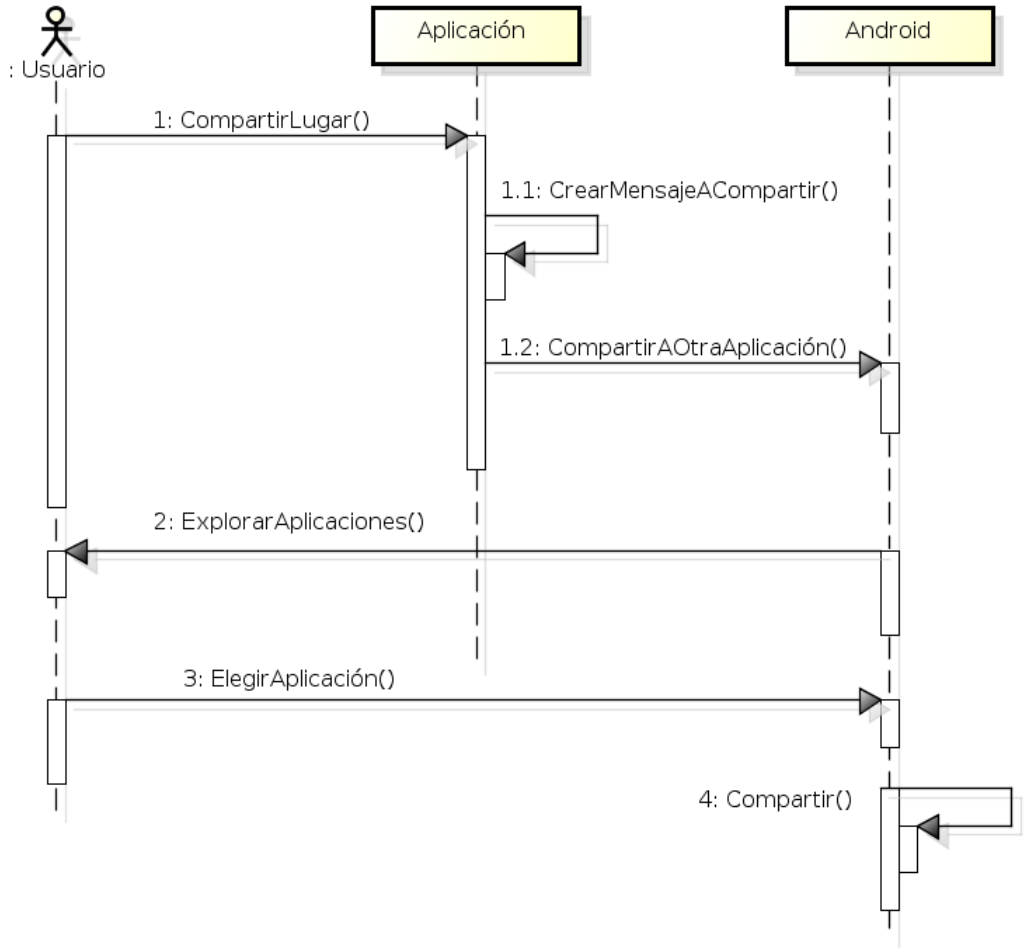


Figura 7.14: Proceso de compartir un lugar con otras aplicaciones

El flujo de eventos asociados al caso de uso que incluye esta funcionalidad:

1. El usuario selecciona la opción 'Compartir'de un lugar.
2. La aplicación genera un texto descriptivo del lugar para poder ser compartido con otras aplicaciones.
3. La aplicación advierte a Android que desea compartir información con otras aplicaciones instaladas en el dispositivo.
4. Android muestra al usuario un listado de las aplicaciones con las que podría compartir el lugar.

5. El usuario elige un aplicación de entre las disponibles.

6. Android se encarga de transmitir nuestro mensaje a la aplicación elegida por el usuario.

7.5.2. Diseño de datos

El diseño de datos consiste en descubrir y la definir completamente de los procesos y características de los datos de la aplicación. El diseño de datos es un proceso de perfeccionamiento gradual que abarca desde la cuestión más elemental, "¿Qué datos requiere la aplicación?", hasta los procesos y estructuras de datos precisos que proporcionan dichos datos. Si el diseño de datos es bueno, el acceso a los datos de la aplicación será rápido y fácil de mantener, y podrá aceptar sin problemas las futuras mejoras de los datos.

Aunque no se debería hacer suposiciones sobre la tecnología de almacenamiento de datos utilizada para almacenar y recuperar los datos de la aplicación, al ser SQLite el gestor de base de datos que utiliza Android nativamente, la elección es fácil desde el principio.

Modelo Entidad-Relación

Cuando se utiliza una base de datos para gestionar información, se está plasmando una parte del mundo real en una serie de tablas, registros y campos ubicados en un dispositivo; creándose un modelo parcial de la realidad. Antes de crear físicamente estas tablas en el ordenador se debe realizar un modelo de datos. [Miles y Hamilton, 2006]

Se suele cometer el error de ir creando nuevas tablas a medida que se van necesitando, haciendo así el modelo de datos y la construcción física de las tablas simultáneamente. El resultado de esto acaba siendo un sistema de información parcheado, con datos dispersos que terminan por no cumplir adecuadamente los requisitos necesarios.

Las bases de datos son un gran pilar de la programación actual, ya que nos permiten almacenar y usar de forma rápida y eficiente cantidades ingentes de

datos con cierta facilidad. En la actualidad se usa de forma mayoritaria las bases de datos relacionales (dominadas por distintos gestores a través del lenguaje SQL, en gran medida).

El Modelo de Entidad-Relación es un modelo de datos basado en una percepción del mundo real que consiste en un conjunto de objetos básicos llamados entidades y relaciones entre estos objetos, implementándose en forma gráfica a través del Diagrama Entidad Relación.

- **Entidad:** Las entidades representan *objetos* o *cosas* que se diferencian claramente entre sí.

  Los elementos de cada una de las entidades deben estar diferenciados perfectamente unos de otros. Una entidad puede ser un objeto con existencia física, como un lugar, o un objeto con existencia conceptual, como un comentario.

  En nuestro caso, podemos definir:

  - Lugar
  - Comentario

- **Atributos:** Los atributos son características o propiedades asociadas a la entidad que toman valor en una instancia particular; de esta forma, es posible su identificación unívoca.

  Cada entidad contiene distintos atributos que dan información sobre esta entidad.

- **Relación:** Es un vínculo que permite definir una dependencia entre varias entidades, es decir, permite exigir que varias entidades compartan ciertos atributos de forma indispensable.

- **Relaciones de cardinalidad:** Se pueden encontrar distintos tipos de relaciones según como participen en ellas las entidades. Por ejemplo, un lugar puede tener varios comentarios, pero un comentario solo pertenece a un lugar concreto.

  Existen varios tipos de relaciones: **uno a uno**, **uno a varios** o **varios a uno**, y **varios a varios**.

- **Claves:** Es un subconjunto del conjunto de atributos comunes en una colección de entidades, que permite identificar inequívocamente cada una de las entidades perteneciente a dicha colección. Existen distintos tipos de claves:

  - **Superclave:** aplica una clave a varios atributos de la entidad, para así asegurarse que en su conjunto no se repitan varias veces y así no poder entrar en dudas al querer identificar un registro.

  - **Clave primaria:** identifica de forma inequívoca un solo atributo, no permitiendo que se repita en la misma entidad. Se suelen utilizar campos numéricos autoincrementados como clave primaria, de forma que cada vez que añadamos un registro a la base de datos, la clave primaria aumenta en una unidad.

  - **Clave externa:** este campo tiene que estar estrictamente relacionado con la clave primaria de otra entidad, para así exigir que exista previamente esa clave.

Normalización

La normalización es el proceso de organizar los datos de una base de datos. Se incluye la creación de tablas y el establecimiento de relaciones entre ellas según reglas diseñadas tanto para proteger los datos como para hacer que la base de datos sea más flexible al eliminar la redundancia y las dependencias incoherentes.

Los datos redundantes desperdician el espacio de disco y crean problemas de mantenimiento. Si hay que cambiar datos que existen en más de un lugar, se deben cambiar de la misma forma exactamente en todas sus ubicaciones. Un cambio en la dirección de un alumno es mucho más fácil de implementar si los datos sólo se almacenan en la tabla Alumnos y no en algún otro lugar de la base de datos.

Hay algunas reglas en la normalización de una base de datos. Cada regla se denomina una "forma normal". Si se cumple la primera regla, se dice que la base de datos está en la "primera forma normal". Si se cumplen las tres primeras reglas, la base de datos se considera que está en la "tercera forma normal". Aunque son posibles otros niveles de normalización, la tercera forma normal se considera el

máximo nivel necesario para la mayor parte de las aplicaciones.

Los pasos a dar para pasar el esquema conceptual a la tercera forma normal son los siguientes:

- Primera forma normal (FN1)
  - Eliminar los grupos repetidos de las tablas individuales.
  - Crear una tabla independiente para cada conjunto de datos relacionados.
  - Identificar cada conjunto de datos relacionados con una clave principal.
- Segunda forma normal (FN2)
  - Crear tablas independientes para conjuntos de valores que se apliquen a varios registros.
  - Relacionar estas tablas con una clave externa.
- Tercera forma normal (FN3)
  - Eliminar los campos que no dependan de la clave.

Diseño en MongoDB

Como se ha mencionado anteriormente, MongoDB es una base de datos sin esquema. Esto implica que no es necesario hacer un diseño de datos para poder almacenar información.

Sin embargo, es muy deseable realizar este diseño, de forma que quede especificado desde un principio, y sea consistente con el resto de la aplicación, teniendo en cuenta que será utilizado por dos agentes, en dos lenguajes de programación, dos tecnologías subyacentes diferentes.

El esquema será el siguiente:

- **_id:** Identificador del recurso. En este caso, identificador del lugar o punto de interés. MongoDB se encargará de asignar un identificador único para cada documento, de forma que se pueda localizar inequívocamente.

- **nombre:** Nombre del punto de interés.
- **url:** URL a la página externa del punto de interés, de donde se ha extraido la información del mismo.
- **desc:** Breve descripción del punto de interés.
- **localización:** Ubicación geográfica del punto de interés, guardada como un array en la forma *[latitud, longitud]*.
- **imageURL:** Dirección de la imagen destacada del punto de interés.
- **comentarios:** Comentarios realizados al lugar. Será guardado como un array de objetos de la forma *[{autor, contenido, timestamp}]*
  - **autor:** Nombre del usuario que realizó el comentario.
  - **contenido:** Comentario realizado.
  - **timestamp:** Instante temporal en el que se realizó el comentario.
- **valoraciones:** Valoraciones realizadas por los usuarios al lugar. Será almacenado como un array de números reales.

Tablas de la Aplicación

A pesar de que MongoDB es una base relacional sin esquema, Android ofrece una base de datos SQL nativamente para poder almacenar información, como se comentó anteriormente.

Es por ello por lo que, en este caso, sí es necesario realizar un diseño de datos detallado.

Lugares

- **id:** *String* Identificador del lugar o punto de interés. Será clave primaria de la tabla.
- **nombre:** *String* Nombre del punto de interés.
- **url:** *String* URL a la página externa del punto de interés, de donde se ha extraido la información del mismo.

- **desc:** *String* Breve descripción del punto de interés.
- **latitud:** *Double* Latitud del punto de interés.
- **longitud:** *Double* Longitud del punto de interés.
- **imageURL:** *String* Dirección de la imagen destacada del punto de interés
- **valoracion:** *Double* Valoración media del lugar.

Comentarios

- **autor:** *String* Nombre usuario que realió el comentario.
- **contenido:** *String* Comentario realizado.
- **timestamp:** *Double* Instante de tiempo en el que se realizó el comentario. Usado como clave primaria de la tabla.
- **lugar_id:** *String* Identificador del lugar al que pertenece el comentario. Clave foránea de la tabla.

7.5.3. Diseño de la interfaz

El diseño de la interfaz de usuario es la parte que más hay que cuidar de la aplicación, ya que la calidad de la interfaz de usuario puede ser uno de los motivos que conduzca a un sistema al éxito o al fracaso.

Usabilidad en aplicaciones móviles

El éxito de los smartphones y tablets es debido a su facilidad de uso. Este alto nivel de usabilidad se ha conseguido gracias a su interfaz táctil y a un muy buen diseño de las aplicaciones móviles. La interfaz táctil es la que permite que el usuario se comunique con el dispositivo sin necesidad de ningún elemento especial. No hacen falta teclados, ratones ni punteros. Con un sólo dedo el usuario puede comunicarse de manera rápida, cómoda y natural.

Pero esta interfaz no serviría de mucho si los smartphones y las tablets no tuvieran aplicaciones pensadas para ser usadas con los dedos y en movimiento.

En el proceso de diseño de la interfaz de la aplicación no sólo se han de tener en cuenta aspectos estéticos. Lógicamente, es deseable que la app sea atractiva visualmente, pero es más importante tener en cuenta cómo y dónde se va a usar la aplicación.

Como ya se ha indicado, el usuario no va a tener ningún elemento adicional (ratón, puntero, etc.) para interactuar con la aplicación. Y, además, hay que tener en cuenta que la aplicación va utilizarse en muchas ocasiones en movimiento. El usuario estará utilizando la app mientras anda, habla y/o está concentrado en otra cosa.

Estos contextos han de ser tenidos en cuenta a la hora de diseñar la interacción del usuario con la aplicación, ya que una mala decisión puede hacer que la aplicación no sea usable en determinadas situaciones.

Contexto de uso

En el caso del móvil, el entorno es cambiante, dinámico. El usuario puede estar distraído o tener prisa, por lo que la estructura de navegación tiene que ser muy simple y hay que evitar (más que nunca) los pasos innecesarios. Por otro lado, la tarea que está realizando el usuario puede interrumpirse por pérdida de cobertura, por una llamada entrante o por una simple distracción.

Hetereogenidad de dispositivos Android

Los dispositivos en sí tienen unas características físicas que afectan negativamente a la usabilidad de una aplicación, como por ejemplo:

- Tamaño de la pantalla.
- Tamaño de las teclas.
- Dificultad para escribir texto.
- Ancho de banda e inestabilidad de la conexión (esto, en realidad, atañe más al servicio que al dispositivo).

De todos los problemas, el más grave es el de la heterogeneidad de los dispositivos. A diferencia de un ordenador personal, que prácticamentese manejan de forma similar todos ellos, aquí las limitaciones inherentes al dispositivo, junto con la heterogeneidad de los mismos, son una fuerte barrera a la usabilidad:

- Algunos móviles tienen más líneas y caracteres por línea.
- Cada uno tiene puestas de una manera las hardkeys (controles hardware) y softkeys (controles programables que aparecen típicamente en la parte inferior de la pantalla). Esto hace que algunas teclas, como la de ir atrás, por ejemplo, puedan estar situadas en lugares distintos en cada móvil.
- Algunos usan fuentes proporcionales, otros no; algunos tienen negritas y otros estilos, otros no.
- Las paletas de colores pueden ser diferentes.
- Algunos tienen acceso a menús vía teclado numérico, otros no.
- El formato de los enlaces y de las barras de scroll puede diferir en función del móvil.

Criterios a seguir

- **Escribir es difícil**

  Siempre es preferible la selección de opciones a la escritura, aunque lleve más pasos. Si no hay más remedio, es importante tener controlado el formato de entrada.

- **Menos es más**

  No hay que dar datos simplemente porque se tienen; sólo hay que dar información relevante. Es recomendable usar abreviaturas y un estilo de escritura conciso.

  La información más importante debe aparecer en la parte de arriba de la pantalla, y las acciones por defecto (mapeadas en las softkeys) deben de ser las más importantes.

Hay que vigilar con el uso del espacio en blanco, porque las líneas vacías pueden despistar e inducir al usuario a pensar que no hay nada más bajo ellas.

- **Navegación**

  La estructura de la aplicación debe ser sencilla, ancha y baja. Es decir, que es preferible tener muchas categorías con poca profundidad.

  La gestión de la navegación hacia atrás es crucial y no debería inhabilitarse. Es importante conservar la información introducida anteriormente para no obligar al usuario a teclearla de nuevo.

- **Cada pulsación de tecla empeora la usabilidad**

  Es aconsejable reducir el número de pulsaciones de tecla entre el principio y el final de cada tarea.

Logotipo

El logotipo de la aplicación se ha diseñado teniendo en cuenta la finalidad del programa. Como se pretende ofrecer al usuario un conjunto de puntos de interés cercanos, se ha decidido que la mejor imagen que representa a la aplicación es un "pin" de los usados en mapas para posicionar lugares, sobre dos círculos concéntricos:

Figura 7.15: Logotipo de la aplicación

Estilo

Todo el estilo de la aplicación se ha realizado siguiendo la guía de estilo que Google propone para diseño de aplicaciones para Android.

Tema de Android

Los temas son un mecanismo para mantener una consistencia en el estilo de la aplicación. El estilo define unas propiedades visuales de los elementos (color, fuente, tamaños) que diferencian las distintas aplicaciones, y pueden convertirse en la marca de tu compañía.

Android ofrece tres temas distintos:

- Holo Light
- Holo Dark
- Holo Light con barra de acciones oscura.

Será esta última la elegida por nuestra aplicación.

Storyboard

Un storyboard es un conjunto de ilustraciones mostradas en secuencia con el objeto de servír de guía para seguir la navegación y estructura de una aplicación.

Suelen ser en blanco y negro, sin mucho lujo de detalles, indicando la funcionalidad básica de la aplicación.

El storyboard de esta aplicación sería el siguiente:

Al iniciar la aplicación, se cargarán los lugares cercanos a la posición del usuario. stas se mostrarán en una lista, indicando en nombre del punto de interés, una pequeña imagen del mismo, y la distancia a la que se encuentra del usuario.

Cuando el usuario pulse sobre cualquier elemento de la lista, una nueva pantalla se iniciará mostrando en más profundidad los detalles del lugar seleccionado.

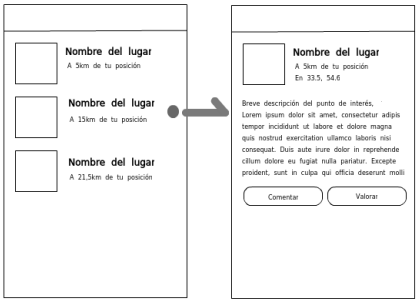


Figura 7.16: Storyboard para mostrar detalles

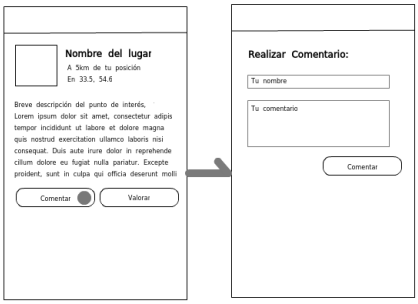


Figura 7.17: Storyboard para realizar un comentario

En la pantalla de detalles, el usuario podrá pulsar en el botón "Comentar", que lo llevará a una nueva actividad donde podrá introducir su nombre, su comentario.

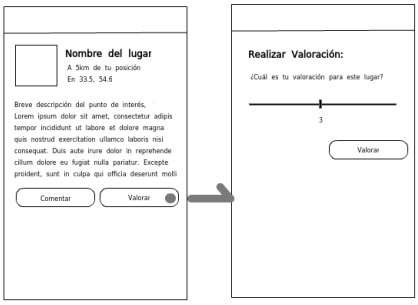


Figura 7.18: Storyboard para realizar una valoración

De nuevo en la pantalla detalles, si el usuario pulsa en el botón "Valorar" una nueva actividad es mostrada. En ésta, el usuario podrá realizar su valoración y pulsar un botón para enviarla al servidor.

7.5.4. Mensajes informativos y de error

Una vez diseñadas las pantallas, se tiene una idea muy clara de cómo se va a desarrollar físicamente la interacción entre usuario y la aplicación, y por tanto de las posibles situaciones de error que pueden darse en esa interacción. Por ello, ahora es el momento de diseñar los mensajes de error y las notificaciones del sistema.

La usabilidad y el diseño centrado en el usuario han planteado una metodología de trabajo basada en la prevención de los errores. De hecho, el proceso de diseño cíclico planteado por los expertos en usabilidad se basa en la prevención constante de los errores a través de la observación directa de los usuarios con la aplicación. Sin embargo, los errores ocurren, más veces de lo que nos gustaría. Por lo tanto, desde la usabilidad se aconseja el diseño de los mensajes de error como una parte más del proceso de diseño de cualquier programa.

Al diseñar estos mensajes deberían seguirse estas reglas:

- Utilización de un lenguaje sencillo y fácil de entender para todos los usuarios (evitar el lenguaje técnico).
- Permitir comprender a los usuarios la causa que ha provocado el error. Para ello, la redacción debe ser concisa y precisa para no dejar margen de duda.
- Intentar establecer una solución al problema expuesto. Una vez que el usuario sabe que es lo que ha provocado el error, necesita saber cómo solucionarlo.
- Informar al usuario de lo que ha ocurrido si su acción no ha provocado ningún cambio visual apreciable.

En la aplicación se han tenido que diseñar algunos mensajes, tanto informativos como de error, entre los que destacan:

- *Cargando lugares cercanos:* Como ya dijimos, la conexión con el servidor se debe realizar de forma asíncrona para que no se paralice la interfaz. Cuando esta acción comienza, un mensaje informativo es mostrado al usuario para que sepa que existe una tarea en curso en segundo plano. Además, se muestra una animación en la interfaz para enfatizar este proceso.

- *Error al conectar con el servidor*
- *No se puede acceder a internet*
- *No se puede obtener la localización del dispositivo*

7.5.5. Implementación

En la fase de implementación se lleva a cabo todo el proceso de codificación del software necesario para que el sistema finalmente implementado cumpla con los requisitos establecidos en la fase de análisis y se corresponda con el diseño establecido en la fase de diseño.

Es muy importante haber realizado correctamente las fases anteriores para que esta fase, que suele ser la que más recursos necesita, sea lo menos costosa posible.

Para la implementación de la aplicación se ha utilizado un ordenador de sobremesa y dos dispositivos Android, un smartphone y una tablet.

Lenguajes utilizados

Para el desarrollo del sistema se han utilizado los siguientes lenguajes de programación:

- **Java:** es un lenguaje orientado a objetos cuyo potencial reside en la compilación a código intermedio o bytecode de las aplicaciones desarrolladas en este lenguaje. Esto permite que una aplicación pueda ejecutarse en múltiples plataformas y de manera independiente al hardware a través de una máquina virtual java.

  Java es el lenguaje utilizado para desarrollar la lógica de las aplicaciones para Android. Utiliza la máquina virtual Dalvik para la ejecución en sistemas operativos Android.

- **XML:** es un lenguaje de marcas para documentos que permite almacenar información de forma estructurada. Android lo utiliza como estándar para el desarrollo de las interfaces de usuario.

- **SQL:** es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar de forma sencilla información de interés de bases de datos, así como hacer cambios en ellas.

- **PHP:** es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Se considera uno de los lenguajes más flexibles, potentes y de alto rendimiento conocidos hasta el día de hoy.

  PHP se ha utilizado para implementar el servidor REST con el que se comunica la aplicación Android.

7.5.6. Pruebas

Las pruebas de caja negra se centran principalmente en lo que "se quiere" de un módulo o sección específica de un software, es decir, es una manera de encontrar casos específicos en ese modulo que atiendan a su especificación.

Las pruebas de caja negra se limitan a que el tester pruebe con "datos" de entrada y estudie como salen, sin preocuparse de lo que ocurre en el interior.

Estas pruebas, principalmente, se centran en módulos de interfaz de usuario (pantalla, ficheros, canales de comunicación...) pero suelen ser útiles en cualquier módulo ya que todos o la mayoría tienen datos de entrada y salida que se pueden comprobar y verificar. Como cualquier otra prueba, las de caja negra se apoyan y basan en la especificación de requisitos y documentación funcional, estos requisitos suelen ser más complejos que los internos, para ello realizaremos una "cobertura de especificación" que será muy recomendable para conseguir probar el mayor campo posible.

Lo más deseable a la hora de realizar pruebas de caja negra es realizar una cobertura completa, pero, en la mayoría de los casos no es suficiente, siempre hay que combinarlas con pruebas de integración, ya que por mucho que funcionen los datos de entrada/salida, por dentro o en terceros sistemas, pueden existir defectos que no se están teniendo en cuenta. Estos defectos pueden no acarrear problemas

a corto plazo, pero a lo largo del tiempo pueden aparecer.

Estas pruebas permiten encontrar:

1. Funciones incorrectas o ausentes.
2. Errores de interfaz.
3. Errores en estructuras de datos o en accesos a las Bases de Datos externas.
4. Errores de rendimiento.
5. Errores de inicialización y terminación.

Las pruebas de caja negra son las siguientes:

Requisito funcional 1: La aplicación debe ofrecer una lista de puntos de interés turístico cercanos a su ubicación geográfica.

- *Se abre la aplicación.* El sistema muestra una lista con los lugares cercanos a la posición del usuario.

Requisito funcional 2: La información ofrecerá al usuario una información más detallada de cada punto de interés.

- *El usuario pulsa sobre un punto de interés en la lista inicial.* Se abre una nueva vista con los detalles del lugar.
- *La información no cabe en pantalla.* Se muestra la información que se pueda, permitiendo al usuario haciendo scroll para seguir leyendo la descripción.

Requisito funcional 3: Se podrá ampliar la información sobre el lugar viendo su página desde Wikipedia. **Requisito funcional 4:** El usuario podrá visualizar el lugar en la aplicació con Google Maps.

- *El usuario pulsa sobre un lugar en la lista principal.* Se muestran los detalles del lugar. Se visualiza un menú con las opciones “Ver en web” y “Ver en Google Maps”.
- *Se pulsa sobre el botón “Ver en web”.* Se abre el navegador por defecto con la web de Wikipedia del lugar seleccionado.
- *Se pulsa sobre el botón “Ver en Google Maps”.* Se abre la aplciación Google

Maps posicionando el mapa en la ubicación del lugar, con una etiqueta para indicar al usuario la localización del mismo.

Requisito funcional 5: La aplicación permitirá al usuario valorar un determinado lugar

- *El usuario pulsa el botón de "Valorar" desde la vista de detalles.* Se muestra la vista para realizar un comentario.
- *El usuario introduce su valoración y envia el formulario.* La valoración se recibe en el servidor, la pantalla vuelve de nuevo a la vista de detalles.

Requisito funcional 6: La aplicación permitirá al usuario realizar un comentario de un determinado lugar y ver los de otros ususarios.

- *El usuario entra en la vista de detalles de un lugar.* Se muestran los detalles, con los comentarios de otros usuarios.
- *Los comentarios no caben en la pantalla.* Se permite al usuario hacer scroll para ver todos los mensajes.
- *El usuario intenta enviar el comentario sin introducir los datos.* Se muestra un mensaje informativo pidiendo que introduzca los campos faltantes.
- *El usuario introduce los campos y envía el formulario.* El comentario se envía al servidor, la pantalla vuelve hacia atrás, los comentarios se actualizan.

Requisito funcional 7: El usuario podrá almacenar en la propia aplicación aquellos puntos de interés que desee, de froma que pueda consultarlos de manera offline.

- *El usuario pulsa sobre el botón "Añadir a favoritos" en la vista de detalles.* El punto de intrés se guarda en la base de datos SQL de la aplicación.
- *El usuario pulsa el botón "Atrás".* La aplicación vuelve a la vista principal. En la parte superior de la lista aparecen los lugares marcados como favoritos por el usuario.

Requisito funcional 8: El usuario podrá recargar la información de los lugares en cualquier momento.

- *El usuario entra en la aplicación.* Se muestra la lista con lugares cercanos, junto con un menú con la opción "Recargar".
- *El usuario pulsa sobre el botón "Recargar".* La aplicación contacta con el servidor, con la nueva ubicación del dispositivo. La aplicación obtiene la información, la transforma y actualiza la vista principal.

CAPÍTULO 8

Conclusión

8.1. Conclusiones

A pesar de existir diversas aplicaciones pensadas para ser de ayuda al ocio y al turismo, los actuales servicios ofrecen unas funcionalidades muy limitadas:

- Existen aplicaciones que ofertan una genial ayuda para realizar turismo, pero que están muy restringidas en cuanto a su región de uso.
- Existen otras que simplemente se basan en utilizar la API de Google Places para mostrar al usuario una mínima información de todo lo que tiene alrededor.

Es por eso que surge la idea de la realización de este proyecto, con un claro objetivo docente, y como prioridad principal el crear una aplicación móvil que conecte sus servicios con una API. A su vez, el proyecto servirá para dar a conocer otro tipo de bases de datos, las no relacionales, para el almacenamiento de datos masivo, no estructuradas y escalables.

Se ha optado por la plataforma Android como base de la aplicación por ser la plataforma móvil más extendida hoy día. Además, hay terminales en un amplio rango de precios, por lo que las barreras de entrada a este tipo de usuarios son menores.

Por otro lado, se ha escogido una base de datos NoSQL como MongoDB, ya que este tipo de bases de datos son más rápidas y eficientes a la hora de leer y escribir registros, debido a su condición de no estructuradas. Además, las bases de datos NoSQL son potencialmente muy escalables, característica que será de utilidad en un futuro, si la aplicación crece.

Inicialmente se ha realizado una definición del problema para mostrar sus objetivos y ventajas principales respecto a otras soluciones el mercado. También se han revisado las posibles necesidades de un cliente real en el análisis de requisitos.

Se ha hecho un estudio en profundidad de las distintas tecnologías empleadas en el proyecto, como Android, Java, SQLite, XML,... De igual forma, se han estudiado las tecnologías REST y las bases de datos NoSQL, para tener una mayor comprensión de su utilización, ventajas e inconvenientes. Seguidamente se ha llevado a cabo un diseño de la aplicación.

8.2. Desarrollo Futuro

El trabajo realizado en este Trabajo Fin de Grado se considera acorde con las limitaciones de tiempo y medios empleados.

Si se dispusiera de más tiempo, se podrían añadir diversas mejoras y funcionalidades al sistema para mejorar las capacidades del mismo.

Una de las mejoras podría ser dar al usuario la capacidad de registrarse en el sistema: ya que tenemos una base de datos, podríamos utilizarla para incluir los usuarios que utilizan nuestra aplicación, de forma que podamos ofrecerles mejores servicios.

Uno de estos servicios podría ser el de sincronizar los lugares favoritos de cada usuario en la nube, de forma que estén disponibles en todos sus dispositivos en los que esté registrado.

Otra mejora sería diseñar un cliente web: debido a que en la tecnología REST, cliente y servidor son dos entidades completamente independientes, la construcción de una plataforma web como cliente se simplifica mucho ahora que el servidor ya está diseñado. El cliente solo debería conectar con el servidor de la forma en que éste ha sido definido, quedando únicamente el diseño de la interfaz web como tarea pendiente.

APÉNDICE A

Contenidos del CD

Todo lo necesario para la instalación de la aplicación se encuentra en el CD-ROm adjunto a esta documentación.

Su contenido es el siguiente:

- Carpeta **Memoria**: contiene la memoria en formato PDF.
- Carpeta **Aplicación**: contiene la aplicación y el código fuente
  - Carpeta **Código fuente**: código fuente de la aplicación
  - **Guía Turístico Virtual**: contiene el archivo .apk de instalación de la aplicación en el dispositivo Android.

Apéndice B

Guía de Instalación

A continuación vamos a describir cada uno de los pasos que hemos de seguir para instalar nuestra aplicación en un dispositivo. Para ello, en primer lugar, hemos de comprobar que la versión del sistema operativo Android del dispositivo es igual o superior a la 4.0.

Obviamente, antes de comenzar con el procedimiento, deberemos obtener el archivo APK de la aplicación que deseamos instalar en nuestro dispositivo Android. Como se ha comentado anteriormente, este archivo se encuentra en la carpeta "Aplicación/Guía Turístico Virtual" del CD-ROM.

Conectamos el smartphone o tablet al PC y en el momento en que el sistema lo solicite, activamos el modo de almacenamiento masivo. También podremos montar la tarjeta SD como lo hacemos con un pendrive.

Figura B.1: Conexión mediante MTP

El siguiente paso es transferir el mencionado archivo APK a la tarjeta, para

ello lo arrastramos a la misma. En este punto es aconsejable crear una carpeta dentro de la SD con el objeto de hacer más fácil recordar su ubicación.

Ahora sólo nos queda indicarle a Android que permita la instalación de programas fuera de Google Play, para ello nos desplazamos hasta "Ajustes¿Seguridad¿Orígenes desconocidos" y activamos la opción.

Figura B.2: Activación orígenes desconocidos

Una vez hecho esto, sólo falta buscar el archivo con cualquier explorador de archivos e instalarlo.

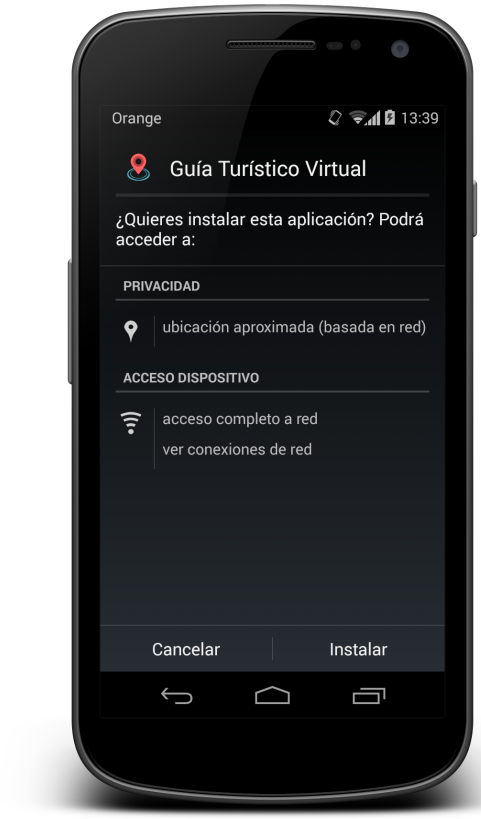


Figura B.3: Instalando aplicación

Figura B.4: Aplicación instalada

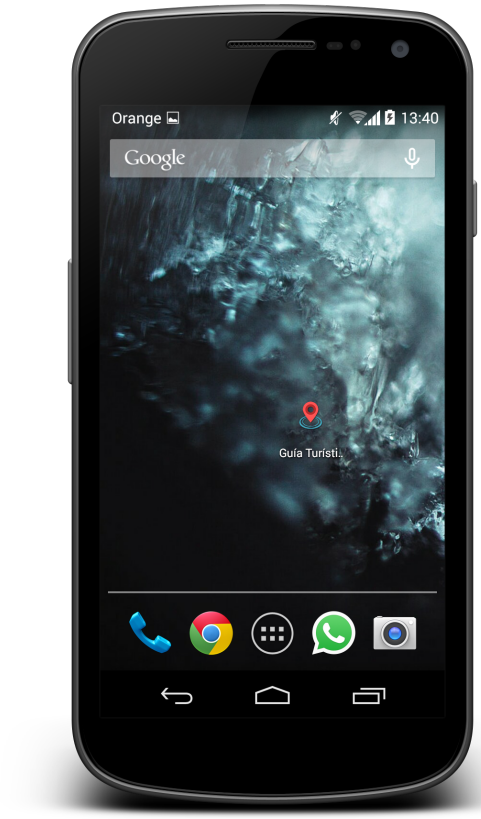


Figura B.5: Aplicación disponible tras ser instalada

APÉNDICE C

Manual de Usuario

A lo largo de este manual vamos a describir cómo acceder a toda la funcionalidad de la aplicación para permitir realizar un uso correcto de la misma.

La aplicación es muy sencilla, y ofrece un flujo unidireccional en casi todas las funcionalidades, de forma que resulte fácil para el usuario.

C.0.1. Pantalla principal

La pantalla principal muestra la funcionalidad primordial de la aplicación: una lista de lugares cercanos al usuario que pueden ser de interés turístico.

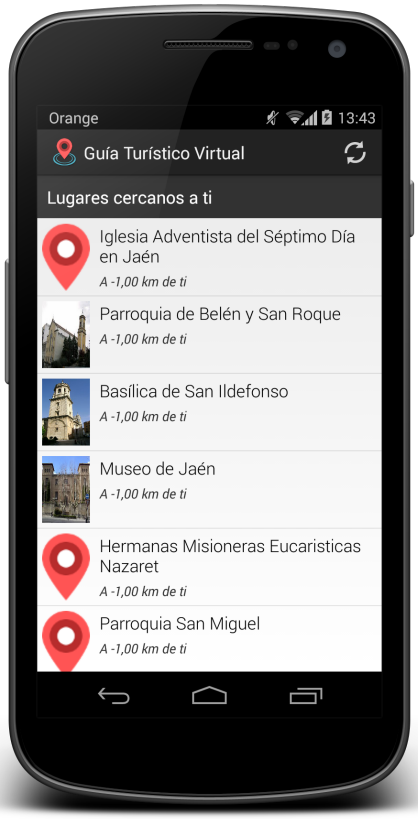


Figura C.1: Vista principal

C.0.2. Detalles de un lugar

Al pulsar sobre cualquier lugar de la lista, la aplicación uestra la información básica del punto de interés: imagen destacada, breve descripción, localización, distancia a la que se encuentra, y comentarios de los usuarios.

Figura C.2: Detalles de un lugar

En esta misma pantalla, el usuario podrá añadir el punto de interés como favorito, verlo en el mapa, o ver su descripción completa en Wikipedia. También podrá valorar y comentar el punto de interés a través de los botones destinados a ello.

C.0.3. Añadir como favorito

Si el usuario pulsa el botón "Añadir como favorito", la aplicación guarda el lugar en la base de datos para poder verlo de forma offline.

Figura C.3: Punto de interés guardado como favoritos

C.0.4. Ver en Wikiepedia

Si el usuario pulsa el botón "Ver en web", la aplicación lo redirigirá a el navegador web para poder ver la infromación del lugar en Wikipedia. Si el usuario tiene instalada la app de Wikipedia, podrá verlo en ahí en lugar de en el navegador.

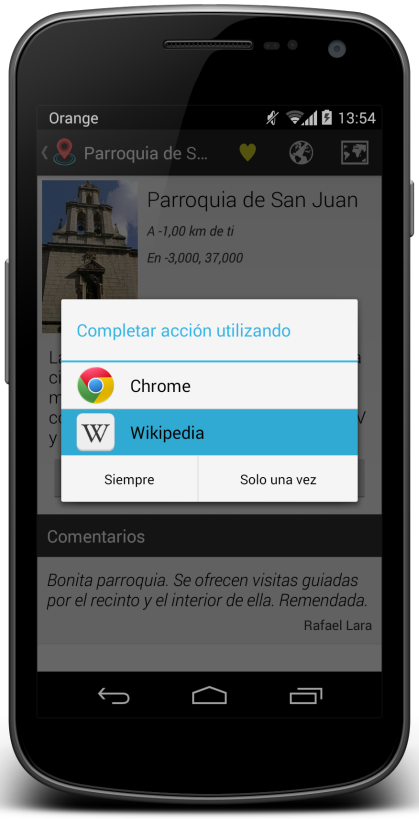


Figura C.4: Seleccionar ver en Wikipedia o en Navegador

C.0.5. Ver en Maps

Cuando el usuario pulsa el botón "Ver en Maps", la aplicación lo redirige a Google Maps, ubicando el lugar en el punto exacto del mapa donde se encuentra, con una señal descriptiva.

C.0.6. Comentar

Pulsando sobre el botón "Comentar", una nueva vista se abre permitiendo al usuario insertar su nombre y su comentario sobre el lugar.

Figura C.5: Vista del lugar en Wikipedia

C.0.7. Valorar

De igual forma, pulsando sobre el botón "Valorar" permitirá al usuario valorar el punto de interés.

C.0.8. Vista en modo offline

Aquellos puntos de interés marcados por el usuario como favoritos aparecerán siempre en la pantalla principal, incluso aunque no se encuentre conexión a internet.

Figura C.6: Vista del lugar en Google Maps

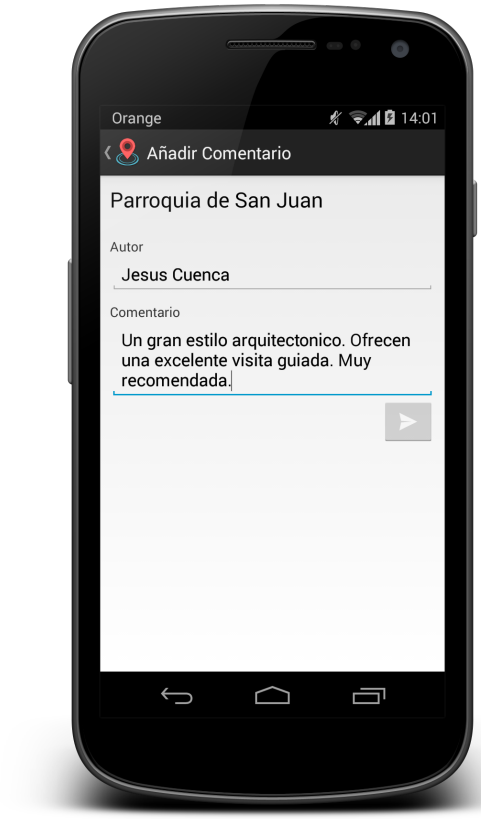


Figura C.7: Vista añadir comentario

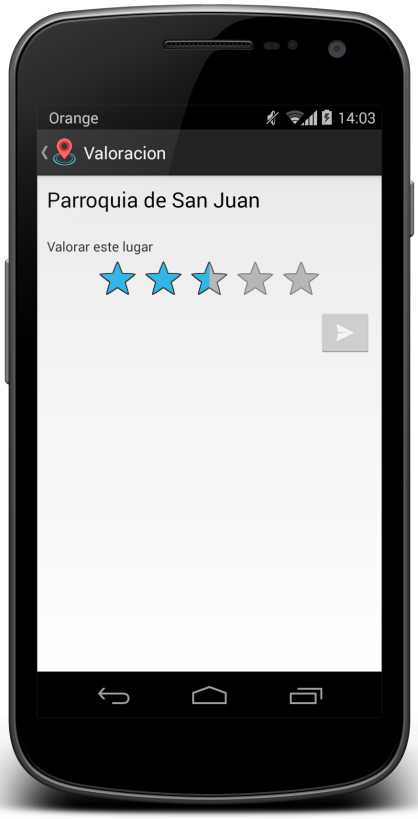


Figura C.8: Vista valorar punto de interés

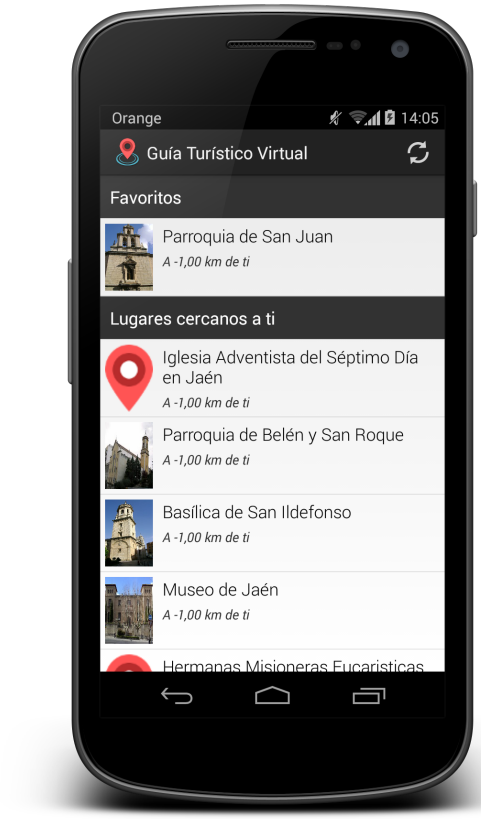


Figura C.9: Listado de lugares, incluyendo los favoritos

APÉNDICE D

Otras tecnologías utilizadas

D.1. Android Studio

El 16 de mayo de 2013 Google lanzó una nueva plataforma de desarrollo de aplicaciones para dispositivos smartphones y tablets con Android llamada Android Studio. Este nuevo IDE podrá sustituir al entorno de desarrollo Eclipse con su correspondiente SDK para desarrollo de aplicaciones Android.

Android Studio está basado en IntelliJ IDEA un IDE para Java de Jetbrains. Google, por lo tanto, no ha desarrollado un IDE desde cero, pero sí ha adaptado este IDE para el desarrollo de aplicaciones para Android.

Android Studio incluye multitud de características que lo hacen muy interesante para usarlo en vez de Eclipse, alguna de ellas:

- Editor WYSIWYG, Live Coding, diseño en tiempo real Rendering App.
- Consola Developer - consejos de optimización, asistencia para la traducción, seguimiento de referencia, campañas y promociones, métricas de uso.
- Provisión para versiones beta y escalonados Rollout.
- Build Gradle.
- Soluciones rápidas y refactorización específica de Android.
- Herramientas para capturar el rendimiento, facilidad de uso, compatibilidad

de versiones.

- Capacidades ProGuard y aplicación de firma.
- Asistentes basados en plantilla para crear diseños y componentes comunes Android.
- Un editor de diseño avanzado, que le permite arrastrar y soltar componentes a la interfaz de usuario y previsualizar diseños para múltiples configuraciones de pantalla a la vez.
- Integración con Google Cloud Messaging.

Android Studio está en continua revisión y aún está en versión beta (Early Access Preview) pero está teniendo un avance muy rápido. Se publican nuevas versiones prácticamente cada semana, siendo la última publicada la versión 0.6.1.

Tras comparar Eclipse y Android Studio, se decidió utilizar este último debido a que, a pesar de ser una versión beta, en mi opinión está más avanzado que Eclipse y además ofrece muchas más funcionalidades fundamentales para desarrollar en Android.

Además Android Studio incluye una utilidad para importar proyectos ya realizados en Eclipse por lo que la transición de un proyecto en desarrollo de un IDE a otro, es mucho más fácil.

D.2. LaTeX

LaTeX es un sistema de composición de textos, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas.

LaTeX es software libre bajo licencia LPPL.

LaTeX es un sistema de composición de textos que está formado mayoritariamente por órdenes construidas a partir de comandos de TeX, pero con la ventaja añadida de poder aumentar las capacidades de LaTeX utilizando comandos propios del TeX. Esto es lo que convierte a LaTeX en una herramienta práctica y útil pues, a su facilidad de uso, se une toda la potencia de TeX. Estas características

hicieron que LaTeX se extendiese rápidamente entre un amplio sector científico y técnico, hasta el punto de convertirse en uso obligado en comunicaciones y congresos, y requerido por determinadas revistas a la hora de entregar artículos académicos.

LaTeX ha sido utilizado para generar la memoria de este proyecto.

Bibliografía

Xataka Android. Xataka android. 2014. URL http://www.xatakandroid.com/mercado/gartner-preve-que-se-superaran-los-2-mil-millones-de-dispositivos-android-en

Apps Court. Lugares cerca de mí. URL https://play.google.com/store/apps/details?id=com.appscourt.maps.places.near.me.

EnBlanco Creativo. Cerca de mi, lugares cercanos. URL https://play.google.com/store/apps/details?id=com.cercademi.

Alberto Fernández. Servicios restful. 2013. URL http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas.

Google. Google places api. URL https://developers.google.com/places/?hl=es.

Google. Versiones android. 2014. URL https://developer.android.com/about/dashboards/index.html#Platform.

Netwin Infosolutions. Places near me. URL https://play.google.com/store/apps/details?id=com.placesnearme.

midepa. Servicios restful. 2011. URL http://histinf.blogs.upv.es/2011/01/04/historia-de-las-bases-de-datos/.

Russ Miles y Kim Hamilton. *Learning UML 2.0*. Ed. OŔeilly Media, 2006.

MongoDB. Introducción a mongodb. 2013. URL http://www.mongodb.org/about/introduction/.

Heliumix Solution Plt. Buscar lugar cerca de mí. URL https://play.google.com/store/apps/details?id=com.heliumix.findnearbyplace.

Profeta. Arquitectura de android. 2011. URL http://labibliadelprogramador.blogspot.com.es/2012/09/estructura-de-android.html.

Nexora Solutions S.L.

Eduard Tomas. Qué es rest. 2014. URL http://www.desarrolloweb.com/articulos/que-es-rest-caracteristicas-sistemas.html.

Wikipedia. Bases de datos. a. URL http://es.wikipedia.org/wiki/Base_de_datos.

Wikipedia. Mongodb. b. URL http://en.wikipedia.org/wiki/MongoDB.

Wikipedia. Representational state transfer. c. URL http://es.wikipedia.org/wiki/Representational_State_Transfer.