UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación Android de realidad aumentada para mostrar imágenes históricas de lugares turísticos de interés

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Francisco de Asís Fuster Andújar

Tutor: Pedro José Valderas Aranda

2014-2015

Resumen

El presente proyecto aborda el desarrollo de una aplicación de realidad aumentada capaz de mostrar imágenes históricas de puntos de interés turístico. Para ello, los usuarios únicamente tienen que enfocar con la cámara de su dispositivo móvil a uno de los puntos de interés y la aplicación le proporcionará imágenes del pasado del mismo. Además la aplicación puede marcar los puntos cercanos en un mapa, mostrar la ruta hasta ellos desde la ubicación del usuario y proporcionar la lista de puntos ordenados por la distancia a este.

La aplicación ha sido desarrollada para ejecutarse en dispositivos Android, aunque se puede adaptar fácilmente a otras plataformas debido a que ha sido implementada mediante el framework PhoneGap. La aplicación hace uso de la geolocalización para obtener la localización tanto del usuario como de los puntos de interés

Palabras clave: realidad, aumentada, android, geolocalización, imágenes, históricas.

Abstract

This project faces the challenge of developing an augmented reality application that shows historical pictures of points of tourist interest. To do so, users just need to aim the camera of its mobile device at a point of interest, and the application will provide them with past pictures of it. In addition, the app can mark other nearby points in a map, show the route to access them from the current location of the user, and provide the list of points ordered by distance.

The application has been developed to be executed in any type of Android device, although it can be easily adapted to other platforms since it has been implemented with the PhoneGap framework. It makes use of the geolocation capabilities of the device in order to obtain the location of both the user and the point of interest.

Keywords: reality, augmented, android, geolocation, pictures, historical.

Tabla de contenidos

1. Introducción

1.1. Motivación

La tecnología de realidad aumentada está, desde la aparición de los smartphones y tablets, al alcance de cualquier usuario. Gracias a estos, la gran mayoría de la población dispone de un dispositivo provisto de un mecanismo por el que poder observar el mundo que les rodea (la cámara digital), así como de conexión a Internet que permite buscar información al instante sobre cualquier recurso en cualquier momento que lo desee. Esto va más allá si consideramos nuevos dispositivos (léase wareables) que empiezan a verse en el mercado tales como las famosas Google Glass. Todo eso hace intuir un futuro muy prometedor para aplicaciones que utilicen de manera acertada la realidad aumentada.

En este contexto, el proyecto que vamos a desarrollar ocupa un espacio dentro del ámbito del turismo y el ocio. Aporta al usuario una mayor interacción con el mundo que le rodea y le ofrece la posibilidad de obtener información de aquello que está observando sin necesidad de buscar qué es, ya que esta información la tendrá de un vistazo.

En este mismo contexto Android es, actualmente, el sistema operativo para dispositivos móviles mas extendido del mercado. Según un informe de Kantar Worldpanel ComTech, Android domina el mercado en España con más del 88% de la cuota.

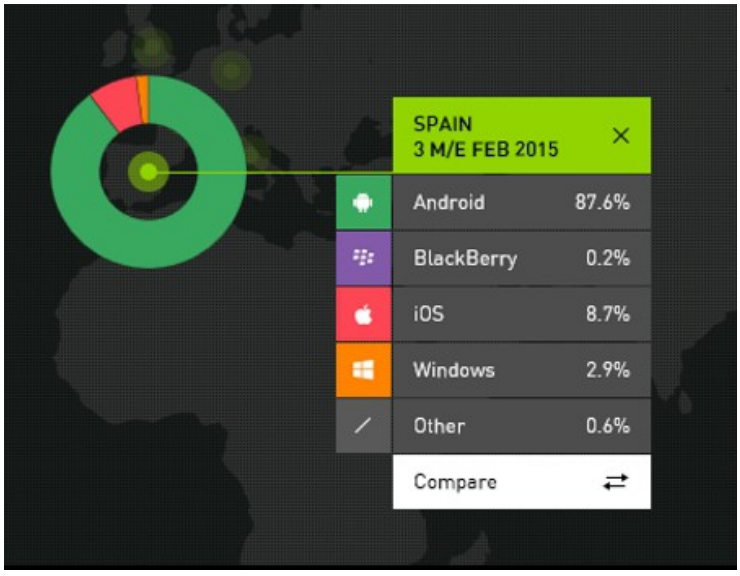


Imagen 1 Estadísticas de Kantar Worldpanel para España

Sin embargo, cualquier aplicación que se precie se encuentra disponible para cualquiera de las actuales plataformas móviles del mercado. Por ello resulta interesante un enfoque en el que nuestra aplicación sea fácilmente desarrollada para todas al mismo tiempo.

Por todo esto, nuestra aplicación es una "guía histórica interactiva" del entorno. Dado que ya existen diversas aplicaciones en el mercado enfocadas a llenar otros espacios de ocio como el gastronómico (TripAdvisor) o de eventos (Fever), nuestro proyecto está enfocado al ocio cultural y más concretamente enfocado a explicar la historia del entorno que nos rodea.

1.2. Objetivos

Los objetivos que nos planteamos en el desarrollo de esta aplicación son:

- Desarrollar una interfaz de usuario que, mediante el acceso a la cámara de nuestro dispositivo móvil y la posición GPS, nos muestre superpuesta a la imagen observada los distintos marcadores de los diferentes puntos de interés o POI's (point of interest en inglés) que nos rodean. Dichos marcadores mostrarán una fotografía identificativa de la localización, así como el nombre y la distancia al mismo.
- Tras seleccionar uno de estos puntos la aplicación mostrará al usuario una tarjeta sobre la misma interfaz de la cámara. Dicha tarjeta contendrá un carrusel de imágenes históricas del punto seleccionado.
- Así mismo la aplicación debe ser capaz de mostrar un mapa de la zona situando dichos POI's en el mismo. Para esto haremos uso de los potentes servicios proporcionados por Google.
- Por otro lado, el usuario podrá observar un listado de los POI's, ordenados por distancia. Tanto desde cada uno de los puntos listados como desde la tarjeta indicada en el primer punto se podrá acceder a una pantalla con más información sobre la localización seleccionada.
- Como se ha comentado, el usuario podrá acceder a más información del POI seleccionado. Esta información comprende un mapa con indicaciones de como llegar, la distancia, un texto sobre el punto, así como, en caso de que existiera un enlace a la página oficial de dicho punto o a la Wikipedia.

1.3. Estructura del documento

El presente documento describe el trabajo y los utilizados en el desarrollo de la aplicación ImageToPast de realidad aumentada para la presentación de imágenes históricas. Para ello se divide el documento en los siguientes apartados:

- **Estudio del arte**: En este punto se da una visión global del estado en el que se encuentra el desarrollo de aplicaciones de realidad aumentada en el entorno de dispositivos móviles. Para ello se presentarán ejemplos de aplicaciones que hagan uso de ésta y cuál es la finalidad que persiguen.
- **Metodología**: Se explican brevemente las diferentes metodologías de desarrollo de software, centrándonos particularmente en la que se ha utilizado para el desarrollo de nuestra aplicación.
- **Contexto tecnológico**: A continuación se presenta el contexto en el que se ha desarrollado la aplicación. Esto incluye un estudio tecnológico tanto de los tipos de aplicaciones (nativas e híbridas), el sistema operativo, los diferentes entornos de desarrollo utilizados o considerados, así como del plugin de realidad aumentada utilizado.
- **Análisis**: Se describen los requisitos de la aplicación por medio del lenguaje de modelado UML (Unified Modeling Language).
- **Diseño**: Se da una visión de la arquitectura de la aplicación, las partes que la conforman y como la plataforma seleccionada hace que interactúen entre sí. También se presenta la aplicación a nivel de interfaz de usuario (IU).
- **Detalles de implementación**: Se explican los detalles de la implementación realizada y la forma de interactuar con los objetos nativos del sistema en el que se despliega la aplicación. Nos centramos sobre todo en aquellos aspectos que han resultado más complejos.
- **Evaluación y pruebas**: Se presenta el ámbito en el que se han desarrollado las diferentes pruebas.
- **Conclusión:** Se da una visión de lo aprendido y cuan provechoso ha sido o puede resultar en el futuro, así como una opinión sobre el entorno y tecnologías utilizadas.
- **Trabajos futuros**: Se proponen nuevas funcionalidades que pueden añadirse a la aplicación en el futuro.

2. Estado del arte

En este capítulo vamos a presentar el estado en el que se encuentra el uso de tecnologías de realidad aumentada para dispositivos móviles sobre la plataforma Android. Para ello vamos presentar un conjunto de aplicaciones que hacen uso de la misma.

2.1. Wikitude

Para comenzar y como no puede ser de otra forma ya que vamos a usar su tecnología para desarrollar nuestro proyecto, presentamos la aplicación Wikitude.

Esta aplicación es, básicamente, un navegador de realidad aumentada. La aplicación viene con una demostración que realiza el reconocimiento de una imagen y superpone elementos en función de a que punto de la misma estemos enfocando. Podemos ver el ejemplo en la imagen 2, parte izquierda.

También viene con una serie de demostraciones de superposición de marcadores sobre la interfaz de cámara para indicar la situación de lugares según la geolocalización. Un ejemplo sería por ejemplo su uso relacionado con el mundo de TripAdvisor que vemos en la imagen 2, parte derecha.

Por último, la aplicación permite a los desarrolladores que estén dados de alta en su web, importar lo que ellos llaman Architect World. Par resumir, se trata de aportar los puntos y poder utilizarlos como en el ejemplo de TripAdvisor.

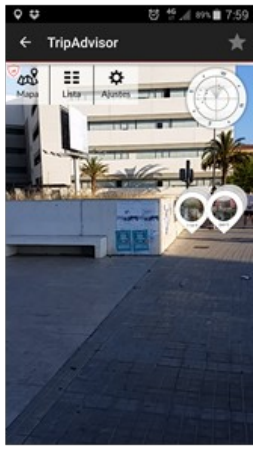


Imagen 2 Aplicación Wikitude (a la izquierda la demostración, a la derecha ejemplo TripAdvisor)

2.2. Layar

Al igual que Wikitude, Layar es un navegador de realidad aumentada.

Del mismo modo permite escanear imágenes, en este caso que contengan el logo de Layar o un código QR. La aplicación nos presentar una imagen 3D superpuesta en la interfaz de la cámara.

Así mismo también permite añadir a esta interfaz una capa con marcadores de lugares mediante geolocalización. Esto se puede observar en el ejemplo de la imagen 3. En esta podemos ver hoteles de la ciudad de Valencia e información de los mismos superpuesta a la interfaz de cámara.

Este ejemplo se acerca mucho al uso que queremos hacer de nuestra aplicación.

Imagen 3 Aplicación Layar con el ejemplo de hoteles

2.3. Car Finder AR

Car Finder AR es una aplicación que nos permite localizar nuestro coche mediante geolocalización. Entre las opciones que nos brinda, existe la opción de utilizar la vista de realidad aumentada para localizar donde hemos estacionado el vehículo.

Claro está que para poder hacer esto, debemos haberle indicado antes a la aplicación las coordenadas en las que se encuentra el mismo. Para ello bastará con almacenarla cuando estemos situados cerca del coche. En la vista de de AR nos permite conocer también la distancia al coche.

En cualquier caso, Car Finder AR nos permite realizar esto con cualquier punto que le indiquemos. Evidentemente no tiene por que ser el coche.

Podemos ver un ejemplo de su funcionamiento en la imagen 4.

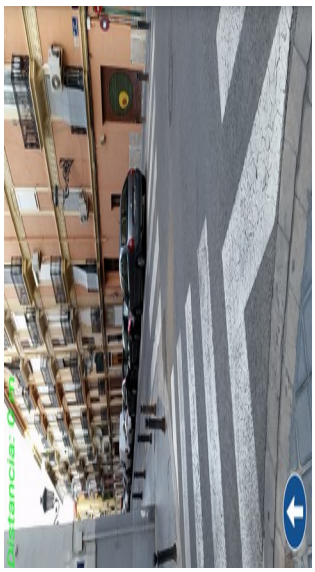


Imagen 4 Aplicación Car Finder AR (al ser una demo, no muestra el marcador de AR)

2.4. Augment

Augment es una aplicación que nos permite superponer objetos 3D sobre la interfaz de la cámara. Esto nos da la posibilidad de, por ejemplo, ver cómo podría quedar un mueble en una habitación antes de comprarlo.

Vemos un ejemplo de su funcionamiento en la página 5.

Imagen 5 Aplicación Augment. Representación virtual de un ordenador portátil en una mesa

3. Metodología

En el siguiente capítulo vamos a realizar un recorrido por algunas de las metodologías del software más utilizadas. Introduciremos las mismas dando una pequeña explicación del flujo de trabajo que se siguen. Finalmente explicaremos con más detalle el modelo utilizado en este proyecto.

3.1. Modelo en cascada

Este modelo, también conocido como Modelo Clásico ordena de forma rigurosa las distintas etapas que se atraviesan en el proceso de desarrollo de software. El inicio de cada una de las etapas del desarrollo ha de esperar a que haya finalizado la inmediatamente anterior, ya que son directamente dependientes.

La versión original fue propuesta por Winston W. Roce en 1970 y posteriormente revisada por Barry Boehm en 1980 e Ian Sommerville en 1985.

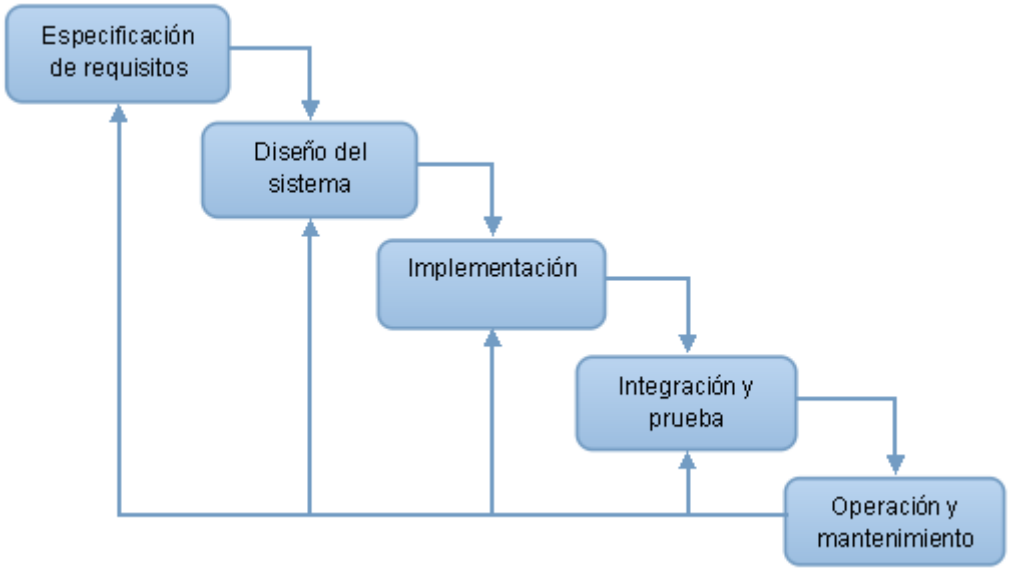


Imagen 6 Flujo de las etapas del Modelo en cascada

1. Especificación de requisito: Se especifican y definen con detalle los servicios, restricciones, necesidades y objetivos del sistema que se va a desarrollar con los usuarios a los que está destinado.

2. Diseño del sistema: Se definen arquitectura, la estructura de datos, la interfaz con el usuario y los procedimientos que comprenderá el sistema objeto de desarrollo.

3. Implementación: Construcción y desarrollo de los distintos módulos y unidades de software que componen el sistema. En este paso se realizan las pruebas unitarias de los distintos módulos.

4. Integración y pruebas: Se realiza la integración de todas las unidades del sistema. Se realizan pruebas funcionales. Finalmente se entrega al cliente.

5. Operación y mantenimiento: Puesta en marcha y mantenimiento del sistema. Corrección de errores detectados por el cliente. Mejora e identificación de nuevos requisitos.

3.2. Modelo en espiral

El ciclo de desarrollo, en este caso, se representa como una espiral en lugar de una sucesión de actividades con vuelta atrás.

En cada bucle o iteración, las actividades u objetivos, que no están fijados a una prioridad concreta, se eligen en función del análisis de riesgos. Este modelo toma en consideración explícitamente el riesgo, a diferencia de otras metodologías.

Este modelo fue definido por Barry Boehm en 1986 y es uno de los más utilizados en el desarrollo de software.

Cada una de las iteraciones (ciclos) se divide en cuatro fases:

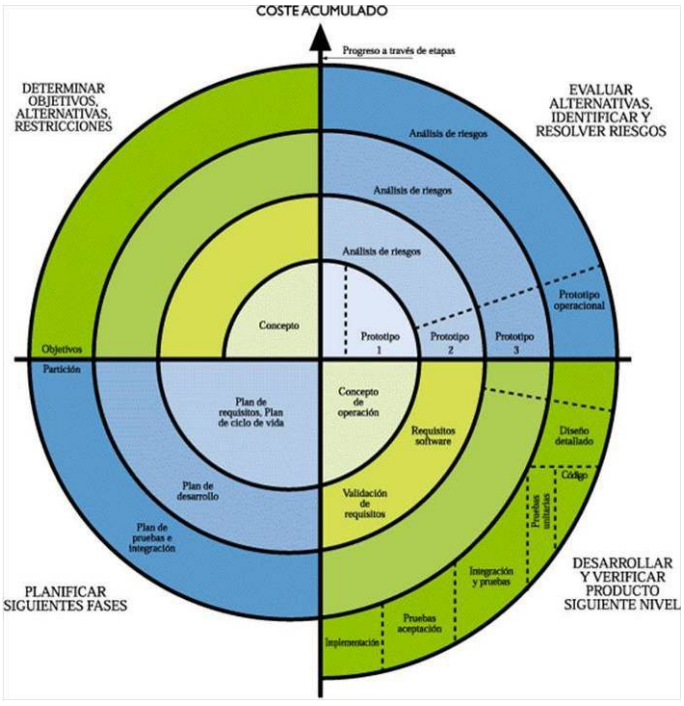


Imagen 7 Representación del ciclo de vida del Modelo en Espiral

1. **Definición de objetivos**: Se definen los objetivos y a partir de éstos se determinan las limitaciones o restricciones del sistema. Se realiza la planificación de manera detallada y se identifican los riesgos.

2. **Evaluación y reducción de riesgos**: Se realiza un análisis detallado de cada uno de los riesgos identificados en el proyecto. Se definen los pasos a seguir para reducir los mismos y se planean estrategias alternativas.

3. **Desarrollo y validación**: Se determina, en función de los riesgos detectados, un modelo para esta fase y se realiza el desarrollo.

4. **Planificación**: Se revisa el estado del proyecto y se determina si continuar con un nuevo ciclo. En caso de ser así, se planifica el siguiente ciclo.

3.3. Modelo en V

El modelo de desarrollo en V o modelo de cuatro niveles se representa de forma gráfica de la siguiente manera:

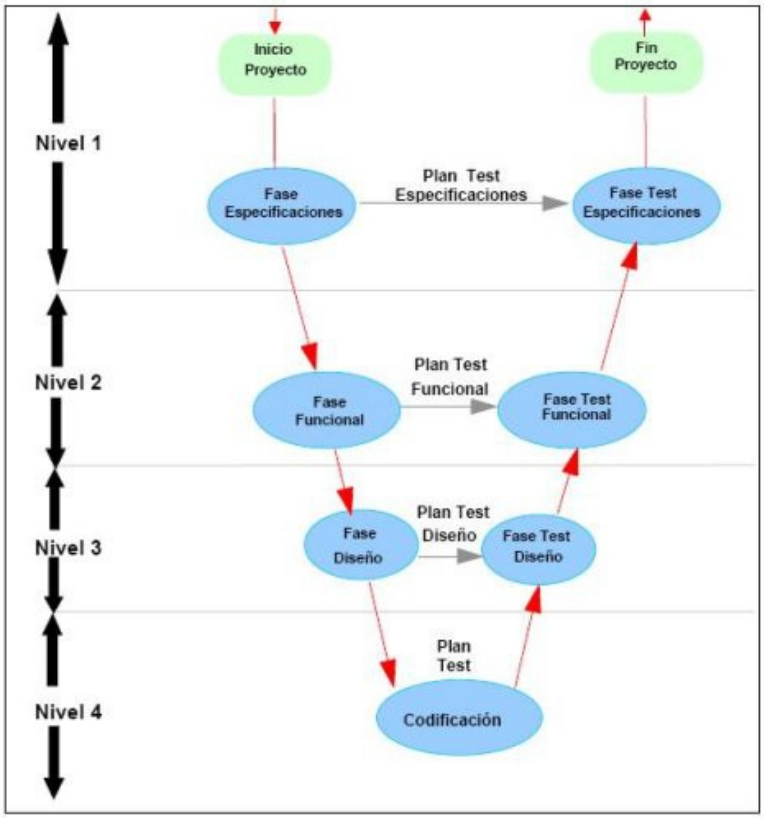


Imagen 8 Representación gráfica del Modelo en V

Representa una secuencia de pasos en el ciclo de vida del desarrollo de un proyecto. En este, para cada fase del desarrollo se define una fase de test o validación. El principio al que obedece el modelo es que, para cada fase del desarrollo debe existir un resultado verificable.

La distancia entre una fase y su correspondiente fase de test, trata de representar de manera proporcional el tiempo que ha de transcurrir entre una y otra.

- El nivel 1 está orientado al cliente y constituyen el inicio y el fin del ciclo y del proyecto. En este nivel se realiza el análisis y especificación de requisitos de sistema a diseñar.
- El nivel 2 está orientado al análisis funcional del sistema. En este se consideran las funciones que son visibles de manera directa o indirecta por el usuario final.
- El nivel 3 es en el que se definirá la arquitectura del sistema. En el se determinan los componentes hardware y software que formarán parte de dicha arquitectura.
- El nivel 4 es el nivel de implementación de la solución. En este se desarrollan los módulos y unidades del sistema.

3.4. Modelo Incremental

Propuesto por Harlan Mills en 1980 este modelo sugiere un enfoque incremental del desarrollo como una forma de evitar la repetición del trabajo en el proceso de desarrollo de software y dar la oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema.

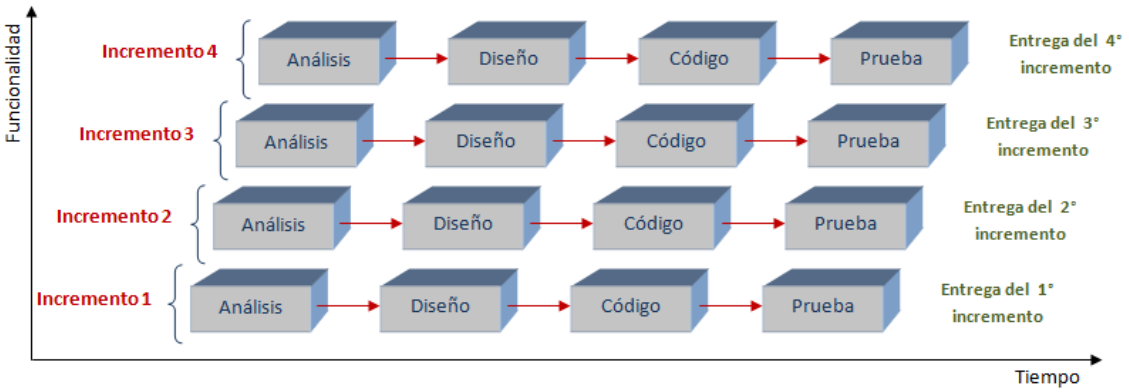


Imagen 9 Representación gráfica del modelo incremental

3.5. Metodología utilizada

La metodología que hemos utilizado en nuestro caso ha sido la del modelo incremental. En este modelo el contacto con el cliente es constante. Es este el que al final de cada iteración descarta o añade nuevos requerimientos de manera que el software se adapte lo mejor posible a sus necesidades. Este proceso se repite hasta conseguir el sistema final.

Este modelo también permite que el usuario no tenga que esperar hasta el final para tener un producto operacional. También permite que el usuario pueda ir definiendo de manera más precisa los requisitos conforme va utilizando las diferentes entregas.

Lo más habitual es realizar las partes con mayor riesgo en los primeros incrementos de modo que estos son sometidos a un mayor número de pruebas y de esta forma evitar riesgos.

Los diferentes incrementos en nuestro proyecto han sido los siguientes:

Incremento	Versión	Interfaces involucradas	Descripción
1	1.0.0	Cámara	Acceso a la cámara mediante integración del plugin de AR y visualización de los POI's
2	1.1.1	Cámara	Presentación de información en la interfaz de cámara de un POI seleccionado
3	1.2.3	Listado de POI's	Listado de los distintos POI's ordenados por distancia.
4	1.3.3	Listado de POI, Cámara y Detalle de POI	Acceso a información más detallada del POI. Localización en un mapa.
5	1.4.3	Mapa	Visualización de la localización conjunto de POI's en un mapa.
6	1.5.4	Detalle POI	Acceso a información externa (web) y funcionalidad "como llegar".

4. Contexto tecnológico

A continuación vamos a introducir el contexto tecnológico en el que hemos desarrollado nuestra aplicación. Veremos las tecnologías de las que se compone como son el sistema operativo, lenguajes y frameworks utilizados. También introduciremos los IDE's y editores de texto con los que hemos desarrollado el proyecto.

4.1. Android

Android es un sistema operativo, concebido y diseñado en sus inicios para ejecutarse sobre dispositivos móviles tales como smartphones y tablets. Sin embargo en la actualidad se está extendiendo a relojes inteligentes, televisiones e incluso automóviles.

Basado en el kernell (núcleo) de Linux, tiene licencias tanto Apache 2.0 como GNU GPL que garantizan a sus usuario finales el poder usar, modificar, estudiar y compartir el software.

El desarrollo de aplicaciones y su distribución (fuera de Google Play, que tiene una cuota de alta) es completamente gratuito.

La arquitectura de Android es la siguiente:

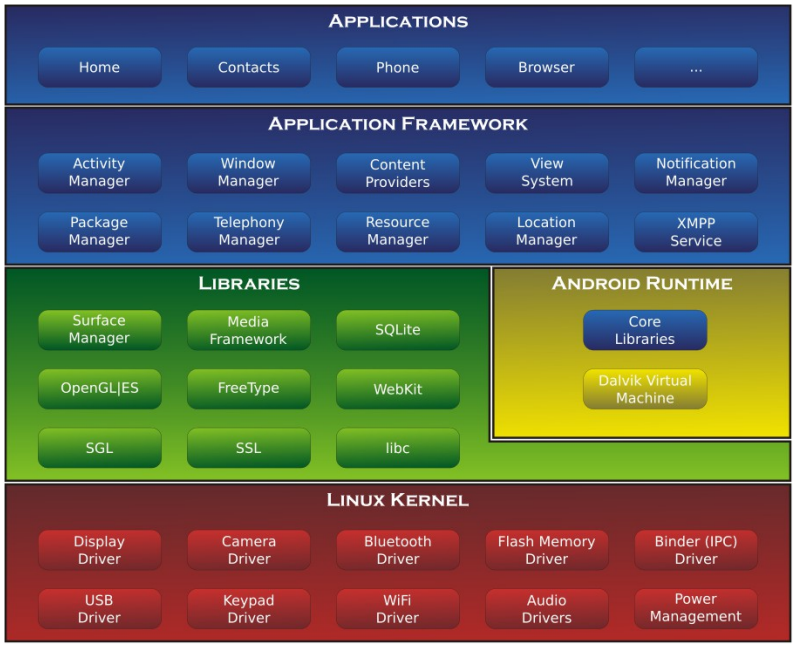


Imagen 10 Arquitectura de aplicaciones Android

Pese a que la mayoría de las aplicaciones nativas de Android están escritas en Java, este no posee una JVM (Java Virtual Machine). El bytecode de Java es compilado en un ejecutable Dalvik y corre en una Maquina Virtual Dalvik, la cual está específicamente diseñada para Android.

Por otro lado, el sistema operativo provee de soporte para conectividad, almacenamiento, streaming, etc.

Android es actualmente el sistema operativo para dispositivos móviles más distribuido en el mundo, funcionando actualmente en 1.300 millones de aparatos.

En la última conferencia I/O de Google se anunció el lanzamiento de su versión M. Siguiendo la tendencia de asociar el nombre de un dulce a las diferentes distribuciones del sistema (Lollipop, KitKat, Jelly Bean,...) se sugiere en las redes que este podría ser Milkshake.

4.2. PhoneGap

Tal y como se describe en su página, PhoneGap es un solución (o plataforma) open source para la creación de aplicaciones móviles multiplataforma con tecnologías basadas en estándares Web como HTML5, CSS3 y JavaScript.

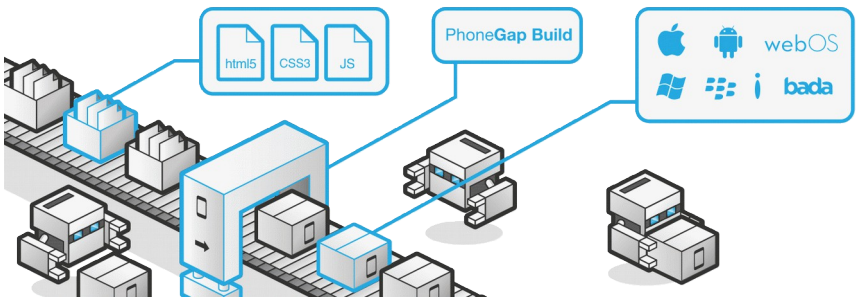


Imagen 11Representación simple del modelo de compilación de PhoneGap

PhoneGap fue desarrollado por la empresa Nitobi bajo licencia de software libre y lanzada en 2008. En 2011 Adobe adquirió Nitobi y por tanto pasó a ser el propietario de la plataforma.

Para preservar el espíritu libre de PhoneGap, Adobe decidió donar el framework a la fundación Apache. Sin embargo, el proyecto pasó a denominarse, primero “Apache Callback”, quedando finalmente el nombre actual como “Apache Cordova”. El nombre original, PhoneGap, es propiedad de Adobe y es el que utiliza para realizar su distribución personalizada de “Cordova”.

Ambas distribuciones son prácticamente iguales, con la diferencia de que Adobe provee de servicios de computación en la nube para su distribución.

Las características de PhoneGap se pueden ver en la siguiente tabla:

	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 6.0+	Blackberry 10	Windows Phone 8	Ubuntu	Firefox OS
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓
Compass	x	✓	✓	x	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✓	✓	✓	✓	x
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	x	✓	✓	✓	x
Network	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓

Imagen 12 Tabla de características extraída de www.phonegap.com

Las aplicaciones que construimos con PhoneGap (o Cordova) son conocidas como aplicaciones híbridas, ya que no se desarrollan utilizando la estructura de un proyecto de aplicación Android y sus recursos (archivos XML para definir interfaces y recursos, y clases Java para definir funcionalidad).

PhoneGap se encarga de realizar el empaquetado de la aplicación que posteriormente desplegaremos en el dispositivo.

Como hemos visto en la tabla de características anterior, PhoneGap nos proporciona acceso al hardware del dispositivo. En nuestro caso, nos interesa el acceso a la cámara, el acelerómetro y el GPS para la geolocalización. Sin embargo, veremos que esto está cubierto por el plugin Wikitude de realidad aumentada que describiremos más adelante. Por otro lado tendremos que hacer uso de otros plugins de PhoneGap fuera del ámbito de la realidad aumentada para poder desarrollar ciertas funcionalidades. Esto se trata en capítulos posteriores.

La instalación de PhoneGap se realiza, en la versión actual, mediante Nodejs. Tras la instalación de este último, basta con lanzar el comando:

```
npm install -g phonegap
```

4.3. Android SDK

Para la realización de aplicaciones para Android, debemos instalar, al igual que para aplicaciones nativas, el SDK (Software Development Kit) de Android. Lo podemos encontrar en:

http://developer.android.com/sdk/installing/index.html?pkg=tools

Una vez descargado y descomprimido siguiendo las instrucciones que nos indica debemos añadirlo al PATH para poder ejecutar el SDK Manager que nos permitirá instalar todos los paquetes y librerías necesarias para el desarrollo con PhoneGap.

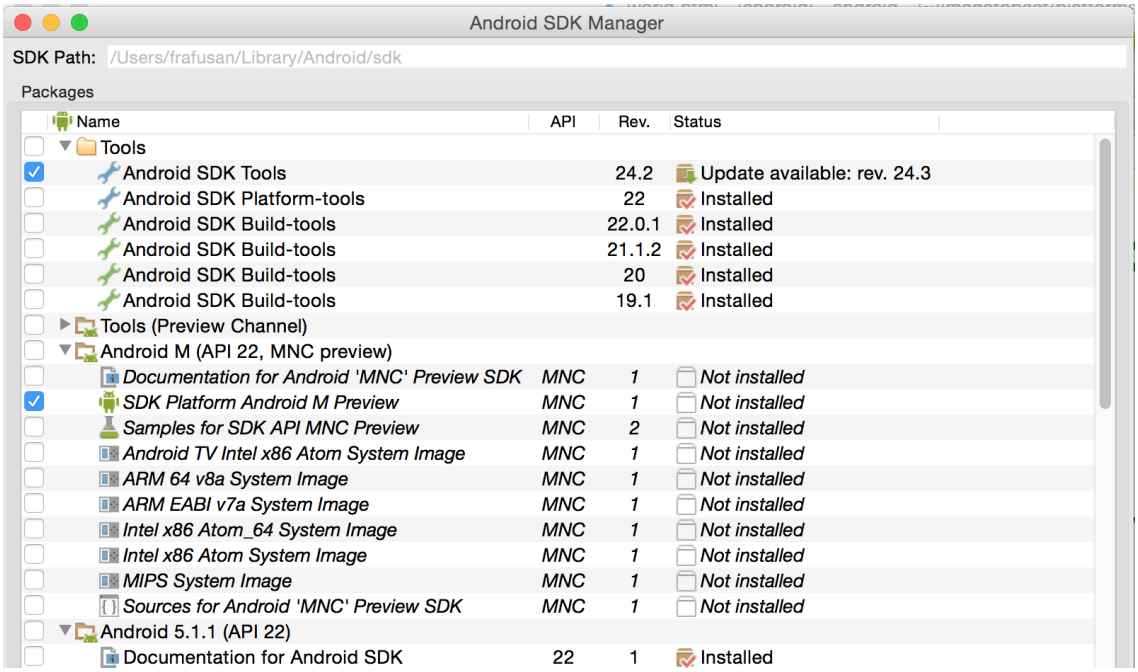


Imagen 13 SDK Manager

Desde aquí también podremos configurar un emulador de dispositivo Android que nos permita, en algunos casos, realizar pruebas de nuestra aplicación. Sin embargo, en nuestro caso esta herramienta no ha sido muy útil ya que el plugin de Wikitude no funciona correctamente en los emuladores. De hecho, en su guía paso a paso para el desarrollo de aplicaciones con el plugin para PhoneGap, nunca se hace mención a estos emuladores (http://www.wikitude.com/developer/documentation/phonegap).

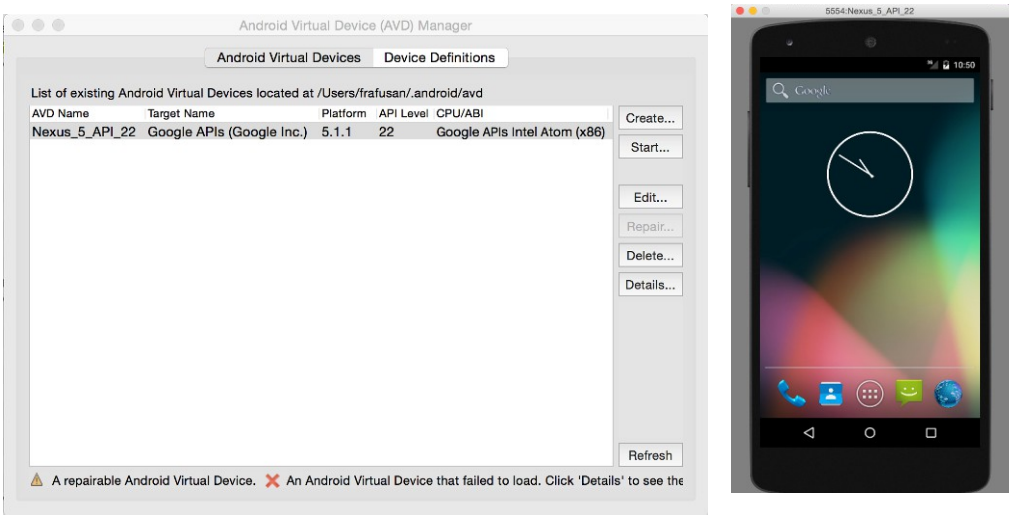


Imagen 14 AVD Manager y emulador de Nexus 5

Hay que puntualizar que el SDK de Android viene también con Android Studio, herramienta que comentaremos más adelante y que por motivos descritos en el apartado correspondiente a la misma, ha venido a sustituir a Eclipse a la hora de desplegar nuestra aplicación en un dispositivo para realizar pruebas.

4.4. Wikitude SDK

Wikitude SDK es un framework "todo en uno" que proporciona servicios de realidad aumentada. Entre estos servicios podemos encontrar:

- Reconocimiento de imágenes
- Renderizado de modelos 3D
- Superposición de video e imágenes.
- Localización basada en realidad aumentada.

Este framework está disponible para el desarrollo de aplicaciones en IO's y Android (smartphones, tablets y smart glasses). Así mismo cuenta con un plugin para desarrollar aplicaciones bajo PhoneGap, que es el que nosotros utilizamos en nuestra aplicación.

Es un plugin de pago, pero nos proporciona una versión de prueba. Además de alguna limitación, el hecho de usar la versión de prueba añade una marca de agua en la vista de cámara y el icono de Wikitude abajo a la izquierda en todo momento.

El SDK de Wikitude se basa en gran medida en tecnologías Web (CSS3, HTML5 y JavaScript) como Cordova y PhoneGap.

Para utilizarlo debemos crear en nuestra aplicación lo que dentro del ámbito del SDK se conoce como ARchitect World (Mundo arquitectónico de realidad aumentada), el cual no es más que un conjunto de páginas HTML que pueden utilizar el API (interfaz de programación de aplicaciones) de Wikitude para crear objetos de AR (en inglés Augmented Reality, realidad aumentada).

El SDK se encarga de integrar los elementos creados (o sus vistas, denominadas ARchitectViews) con la interfaz de usuario, en nuestro caso la vista obtenida por la cámara trasera del dispositivo móvil.

4.5. HTML, CSS y JavaScript

Para el desarrollo de la aplicación, y gracias a la plataforma PhoneGap, como se ha comentado anteriormente, se han utilizado estándares de desarrollo web. En concreto se ha utilizado HTML5, CSS3 y JavaScript.

En esta sección daremos un breve repaso a dichas tecnologías y, en el caso de CSS y JavaScript, los frameworks que hemos utilizado para el desarrollo de nuestra aplicación como han sido JQuery y Materialize.

4.5.1. HTML

Según el W3C (World Wide Web Consortium):

"HTML es el lenguaje para describir la estructura de páginas Web. HTML da a los autores los medios para:

- *Publicar documentos en línea con encabezados, texto, tablas, listas, fotos, etc.*
- *Recuperar información en línea a través de enlaces de hipertexto, con el clic de un botón.*
- *Diseñar formas para la realización de transacciones con servicios remotos, para su uso en la búsqueda de información, hacer reservas, pedidos de productos, etc.*
- *Incluya hojas-distribuidas, videoclips, clips de sonido y otras aplicaciones directamente en sus documentos."*

En el proyecto que nos ocupa hemos utilizado la última revisión de este estándar, conocida como HTML5.

Esta evolución no solo contempla el lenguaje HTML, si no que hace hincapié en su interacción con el resto de tecnologías que permiten realizar aplicaciones Web y sitios más completos y diversos.

Las tecnologías de HTML5 se clasifican según su función en:

- Semántica: Permite describir con mayor precisión cuál es su contenido.
- Conectividad: Permite comunicarse con el servidor de formas nuevas e innovadoras.
- Fuera de línea y almacenamiento: Permite a páginas web almacenar datos, localmente, en el lado del cliente y operar fuera de línea de manera más eficiente.
- Multimedia: Nos otorga un excelente soporte para utilizar contenido multimedia como son audio y video nativamente.
- Gráficos y efectos 2D/3D: Proporcionar una amplia gama de nuevas características que se ocupan de los gráficos en la web como son el lienzo 2D, WebGL, SVG, etc.
- Rendimiento e Integración: Proporcionar una mayor optimización de la velocidad y un mejor uso del hardware.
- Acceso al dispositivo: Proporciona APIs para el uso de varios componentes internos de entrada y salida de nuestro dispositivo.
- CSS3: Nos ofrece una nueva gran variedad de opciones para la sofisticación del diseño.

4.5.2. CSS

Siguiendo con las definiciones que da el W3C (World Wide Web Consortium):

"CSS es el lenguaje para describir la presentación de las páginas Web, incluidos los colores, el diseño, y las fuentes. Permite adaptar la presentación a los diferentes tipos de dispositivos, como grandes pantallas, pantallas pequeñas o impresoras. CSS es independiente de HTML y se puede utilizar con cualquier lenguaje de marcado basado en XML. La separación de HTML de CSS hace que sea más fácil mantener sitios, hojas de estilo a través de la cuota de páginas y páginas a medida para diferentes entornos. Esto se conoce como la separación de la estructura (o contenido) de presentación"

En nuestro proyecto se ha utilizado CSS3, última evolución de este lenguaje y que viene a extender al versión 2.1.

En cuanto al tema de diseño de la interfaz de usuario, la tendencia actual en aplicaciones Android es el lenguaje de diseño (se podría decir que es una guía de estilos)

denominada Material Design. Este aboga por diseñar cada objeto como una pieza material que se puede tocar, pellizcar, deslizar, etc.

Para esta tarea, en el proyecto actual hemos decidido utilizar un framework que nos suministrase este estilo, modificando en el caso de ser necesario algunos componentes.

El framework utilizado es **Materialize** (http://materializecss.com/) que se define como *"un framework web front-end moderno y responsivo"*.

Este framework provee una serie de hojas de estilo (css) y archivos JavaScript que nos ayudan mediante ciertas etiquetas de clase a adaptar nuestro diseño a Material Designó tanto en cuestiones de forma como de comportamiento de los elementos de la interfaz de usuario.

Algunos ejemplos de elementos de **Materialize** son:

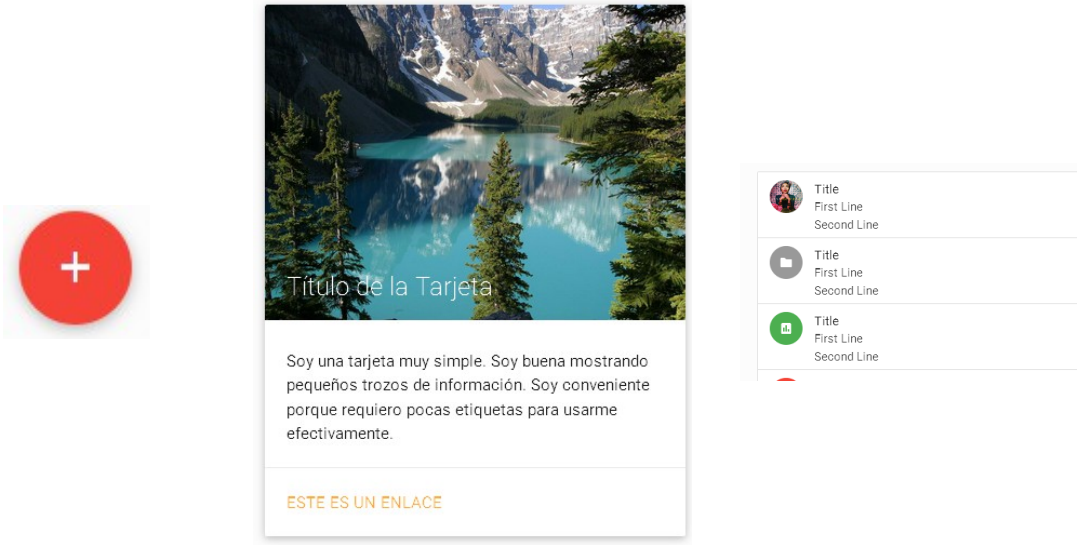


Imagen 15 Algunos de los competentes de Materialize

4.5.3. JavaScript

A diferencia de lo que se puede pensar por su nombre, JavaScript no es "Java interpretado". JavaScript es un lenguaje ligero e interpretado, definido como orientado a objetos, basado en prototipos, débilmente tipado y dinámico. Soporta los paradigmas imperativo, funcional además del OO (Orientado a Objetos). Es utilizado habitualmente del lado del cliente para mejorar la experiencia de usuario en las interfaces.

Sin embargo también es utilizado en entornos de servidor, como por ejemplo con NodeJs, con el que se pueden implementar proxys, sockets y servidores.

Se puede decir que JavaScript es un dialecto de ECMAScript, lenguaje de script estándar. Es un superconjunto del mismo y presenta solo leves diferencias sobre el mismo.

Existen multitud de frameworks JavaScript como Midori**,** AngularJs **y jQuery.** En nuestro caso, dada la dependencia del plugin de realidad aumentada con **jQuery** y **jQuery Mobile** (framework optimizado para dispositivos móviles), hemos decidido utilizar este como base para el desarrollo de nuestra aplicación, como veremos cuando entremos a comentar la implementación con más detalle.

jQuery es una librería JavaScript que provee acceso al DOM (Modelo de Objetos del Documento) para la manipulación de HTML, manejar eventos sobre los elementos del árbol DOM, añadir animaciones, así como agregar interacción mediante AJAX (JavaScript Asíncrono y XML). Es libre y de código abierto.

Está contenido en un solo fichero, añadido en la carpeta destinada a JavaScript de nuestra aplicación. Para su uso, hay que añadirlo al HTML que vaya a usarlo mediante la etiqueta `<script>` como cualquier otro fichero JavaScript. En este proyecto se ha utilizado la versión 2.1.3 de la librería.

jQuery Mobile por su parte, es un framework basado en la librería jQuery, destinado al diseño de sitios Web responsivos, así como aplicaciones accesibles desde cualquier smartphone.

jQuery Mobile provee sus propias hojas de estilo, imprescindibles para que el framework realice correctamente las acciones visuales. Pese a que, como en nuestro caso, el aspecto de la aplicación esta definido por Materialize (framework que también utiliza como base jQuery), es necesario enlazar las hojas de estilo de jQuery Mobile para poder utilizar sus transiciones, el multi-paginado en un solo HTML, etc...

El framework provee los atributos "`data-role`" para los elementos `<div>` que determina la funcionalidad del mismo. Esto se explica más adelante, sirva como ejemplo que un `<div data-role="page">` actuará como una página completa de la aplicación.

La versión utilizada de jQuery Mobile, limitada por el plugin Wikitude (que hace uso del mismo) ha sido la 1.3.2.

4.6. Herramientas de desarrollo

4.6.1. Brackets

Brackets, de Adobe, es un editor de código ligero, desarrollado en HTML, CSS y JavaScript. Está enfocado claramente al desarrollo Web y por ello tiene funciones de autocompletado de código y cierre de etiquetas. También provee de otras ayudas como la edición rápida que, en el caso de estar definiendo colores, nos muestra un colorpicker.

Brackets es capaz de gestionar un árbol de directorios completo, como si fuera un proyecto. Tiene una apertura de vista al vuelo de ficheros. Esto quiere decir que el mismo no quedará abierto si no se hace doble clic o no comenzamos su edición.

Al contrario que Sublime Text (el otro editor liviano que veremos), Brackets no utiliza el sistema clásico de pestañas para la vista de archivos. En cambio, en la parte superior de la barra lateral izquierda muestra una pila con los últimos ficheros abiertos.

Sin embargo, se produjeron diferentes problemas con este editor durante el desarrollo del proyecto. En más de una ocasión nos encontramos con un mensaje de error generado, según la información de Brackets, por problemas de inferencia con algún fichero JavaScript.

Imagen 16 Error de Brackets

Incluso en alguna ocasión el editor dejó de funcionar y quedó bloqueado provocando la perdida de los cambios no guardados.

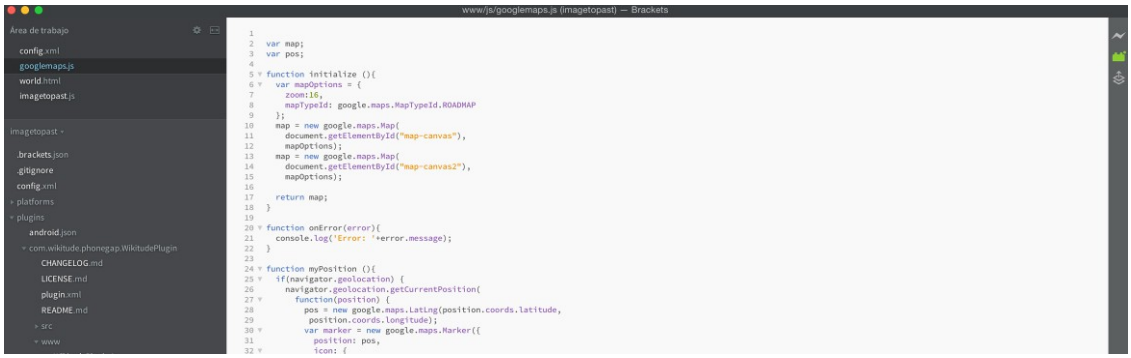


Imagen 17 Editor de código Brackets

4.6.2. Sublime Text

Al igual que Brackets Sublime Text es un editor de código. A diferencia de este, está desarrollado en C++. Sublime Text soporta la sintaxis de multitud de lenguajes de programación, entre ellos HTML, CSS y JavaScript.

También incorpora funciones de autocompletado, multiselección, resaltado de paréntesis, etc. Al igual que Brackets tiene una vista al vuelo de los ficheros, sin embargo y como ya hemos comentado, la gestiona mediante pestañas. Nos permite gestionar carpetas como si fueran proyectos, presentándonos el árbol en la parte izquierda y además incorpora en su parte derecha un mapa (vista previa reducida) del fichero abierto.

Los plugins existentes para Sublime Text permiten añadir funcionalidades como conexión a gestores de versiones (como Subversion o Git), escritura rápida de código o incluso temas para personalizar el entorno.

A diferencia de Brackets, la configuración de Sublime Text es algo más compleja. Sin embargo la experiencia en cuanto a usabilidad una vez configurado ha sido tan positiva como la del primero y en lo que concierne a este proyecto, su fiabilidad ha sido bastante superior.

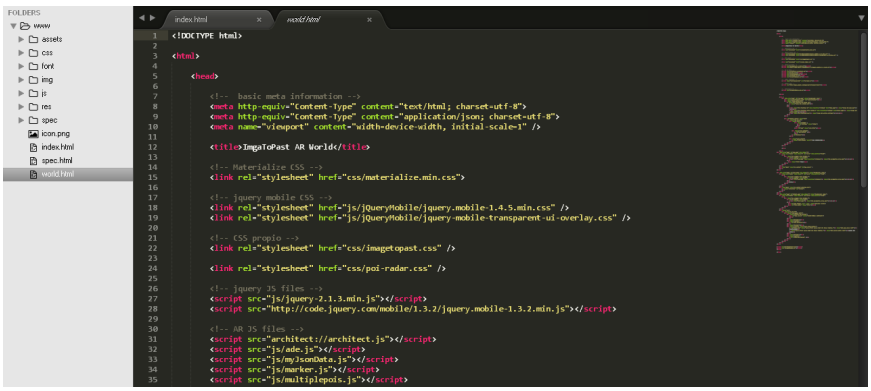


Imagen 18 Editor de código Sublime Text 2

4.6.3. Eclipse

Eclipse es una plataforma de integración de herramientas desarrollada en Java. No está diseñado para ninguna tecnología específica, se considera un **entorno de desarrollo integrado (IDE)** genérico extensible gracias a la gran cantidad de plugins desarrollados para añadirle funcionalidad.

Es cierto que su uso está muy extendido entre desarrolladores de Java y J2EE, sin embargo también contiene extensiones para trabajar con modelos y metamodelos, para utilización de procesos ETL como CloverETL, etc.

Cuenta con características de edición, despliegue, compilación, depuración o ejecución de aplicaciones. Permite gestionar servidores, conectar con sistemas de gestión de versiones como Subversión o Git. Además, provee vistas para cada tipo de proyecto con el fin de hacer más fácil el trabajo en cada uno de ellos. La última versión, que ha sido la utilizada en este proyecto, recibe el nombre de Eclipse Luna.

Sin embargo, para el proyecto que nos ocupa y hasta la última actualización de la plataforma PhoneGap, Eclipse ha sido utilizado únicamente como herramienta de despliegue en dispositivos físicos así como de consulta del log de la aplicación desarrollada. Esto se debe a que el plugin de PhoneGap para Eclipse no ha sido actualizado A partir de la versión 3.0.0 la compilación de proyectos se realiza por medio de la línea de comandos y no tiene reflejo en Eclipse.

Con Eclipse, el proceso seguido fue el de instalar el plugin de herramientas de desarrollo de Android (ADT) y así importar el proyecto desde la carpeta “paltforms/android”. Una vez hecho esto ya podíamos utilizar Eclipse (con algún ajuste para evitar errores por librerías para OS de Apple del plugin de Wikitude) para desplegar la aplicación sobre nuestro dispositivo.

Esto fue posible mientras la compilación de los proyectos en PhoneGap se realizaba mediante ANT, librería para la automatización de la compilación y la construcción de Apache. Como comentaremos en el siguiente punto, no queremos decir que no sea posible seguir realizando las mismas tareas con Eclipse pero la recomendación es ahora pasar a Android Studio.

4.6.4. Android Studio

Android Studio es un IDE exclusivo para la plataforma Android. Basado en el IDE IntelliJ IDEA de la compañía JetBrains. Se distribuye de manera gratuita bajo una licencia Apache 2.0.

Android Studio ha reemplazado a Eclipse (junto con ADT) como IDE principal para el desarrollo de aplicaciones nativas de Android.

En el caso del proyecto desarrollado, este IDE se ha utilizado, al igual que pasaba con Eclipse para el despliegue de aplicaciones en dispositivos físicos y consulta del log.

No es posible crear proyectos híbridos con PhoneGap mediante esta herramienta, a diferencia de lo que sucedía con Eclipse, aunque la situación del plugin de Eclipse haga que sea la misma en ambos casos. Debemos compilar mediante línea de comandos nuestra aplicación PhoneGap, importar el proyecto desde la carpeta "platforms/android" y desplegar en el dispositivo físico.

La diferencia principal radica en la librería para la compilación. Como se ha comentado, Eclipse usaba ANT para realizar la compilación, lo que nos permitía usar este IDE para realizar los despliegues hasta la versión 5.0.0 de PhoneGap. A partir de dicha versión PhoneGap incorpora Gradle para la automatización de la compilación y construcción de proyectos.

Para ser más precisos, PhoneGap incluye Apache Cordova Android 4.0.0. Es esta versión del Cordova la que incorpora Gradle.

También hay que comentar que no es cierto que no se pueda seguir realizando las mismas tareas con Eclipse, sin embargo esto requiere de la instalación de plugins adicionales. Además, revisando los cambios de Apache Cordova encontramos que dice:

Imagen 19 Extracto de la Release 4.0.0 de Apache Cordova Android

"Android Studio está completamente soportado ahora, y se recomienda sobre Eclipse".

5. Análisis

En este capítulo vamos a realizar el análisis de requisitos de la aplicación. Este es uno de los puntos fundamentales en cualquier desarrollo software ya que nos permite determinar el alcance y el contexto del problema que debe resolver nuestro sistema. Para ello vamos a utilizar el lenguaje de modelado UML.

En primer lugar vamos a definir los requisitos funcionales y no funcionales de la aplicación.

A continuación presentaremos un diagrama de clases en el que representaremos los diferentes objetos de los que se compone la aplicación y sus características.

Finalmente mostraremos el diagrama de casos de uso para comprender como el usuario interactúa con el sistema.

5.1. Requisitos

5.1.1. Requisitos funcionales

- **Visualización de POI's mediante la cámara:** La aplicación permite que el usuario pueda ver marcadores que distingan mediante una imagen y el nombre, los lugares de interés que esté enfocando.
- **Selección de POI:** Cuando el usuario seleccione un POI pulsando sobre este, el punto seleccionado debe estar claramente diferenciado.
- **Quitar selección de POI:** Si un POI está seleccionado, pulsar sobre él, sobre otro POI o sobre una zona vacía provoca que este ya no esté seleccionado.
- **Visualización de imágenes historias e información de POI:** Tras la selección del POI el sistema muestra, superpuesto en la interfaz de cámara, imágenes históricas relacionadas con el mismo.
- **Ocultar imágenes e información del POI:** La selección de otro POI o pulsar sobre cualquier otra zona fuera de la presentación de imágenes e información provoca que el sistema oculte dicha presentación.
- **Listar POI:** La aplicación muestra un listado de los POI acompañados de una imagen y ordenados por la distancia al usuario.

- **Visualización de detalles de un POI:** Se muestra un mapa de la localización del punto, así como una descripción más detallada.
- **Situación de los POI en un mapa:** La aplicación deberá mostrar la localización de todos los POI's cercanos en un mapa. Así mismo mostrará con un marcador diferente, la posición del usuario.
- **Cómo llegar:** La aplicación abrirá una aplicación externa como puede ser Google Maps proporcionando la localización del usuario y del punto al que quiere llegar.
- **Abrir web externa:** La aplicación abrirá una página web relacionada con el POI en el navegador predeterminado del dispositivo.

5.1.2. Requisitos no funcionales

En cuanto a los requisitos no funcionales, debe cumplir aquellos comunes a la mayoría de las aplicaciones de este ámbito como son:

- **Rendimiento:** La aplicación debe tener un funcionamiento fluido y proporcionar una experiencia de usuario satisfactoria.
- **Optimización:** El tiempo de ejecución debe ser el menor posible. Tiempos de respuesta cortos.
- **Integración:** La aplicación debe estar integrada en el sistema operativo Android haciendo uso de otras aplicaciones del mismo como puede ser el navegador.
- **Usabilidad:** La aplicación debe ser fácil de usar e intuitiva.
- **Mantenibilidad:** La aplicación será mantenida y actualizando proporcionando mejoras a los usuarios.
- **Disponibilidad:** En este caso nos referimos a la disponibilidad para que el usuario pueda acceder a ella. Para esto, la aplicación se distribuirá mediante Google Play.
- **Fiabilidad:** La aplicación debe informar al usuario de los fallos ocurridos y debe tener en cuenta la recuperación frente a fallos.

5.2. Diagrama de clases

El siguiente diagrama de clases representa los objetos de nuestra aplicación.

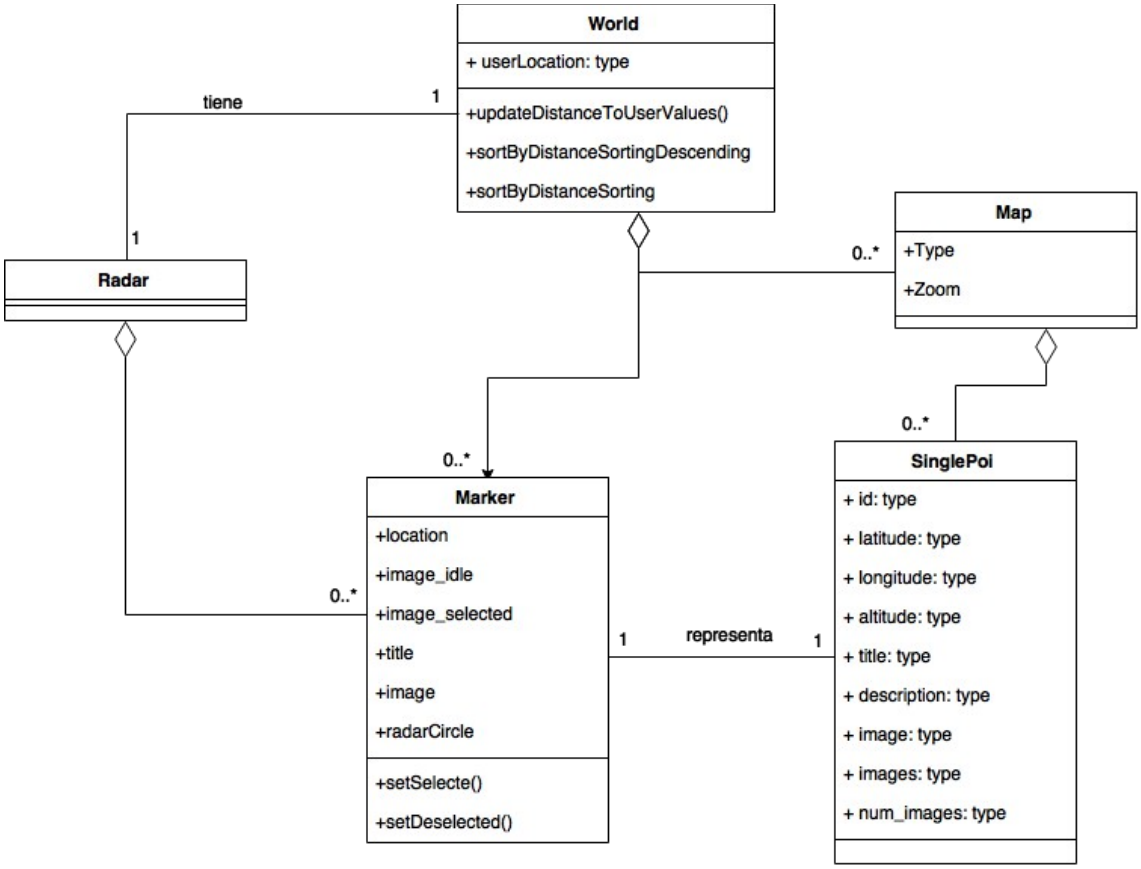


Imagen 20 Diagrama de clases

- **World:** Esta clase representa el conjunto mundo sobre el que se sitúan los diferentes POI's. Contiene la situación del usuario ya que es alrededor de este donde se posicionan los mismos.
- **Radar:** Representación en dos dimensiones del mundo.
- **Map:** Representa el mundo real más allá. de los POI's.
- **Marker:** Representación de un POI dentro del ámbito de la realidad aumentada. Contiene su localización y las imágenes y que representan al mismo dentro de mundo, incluyendo su representación en el radar.

5.3. Casos de uso

Nuestra aplicación está destinada a un actor único. Solo existe un tipo de usuario que interactúa con la misma y por tanto es este el que tiene todas las acciones asociadas. Por tanto solo encontramos un diagrama de casos de uso que presentamos a continuación

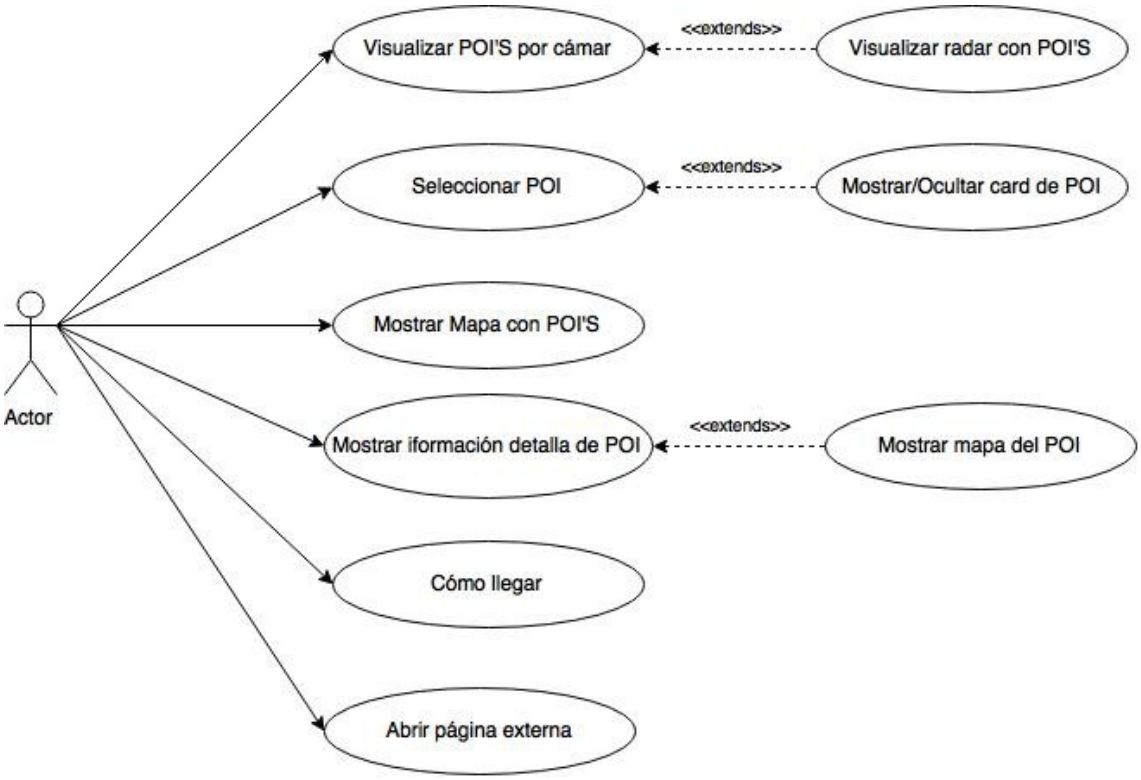


Imagen 21 Diagrama de casos de uso

- **Visualizar POI`S por cámara:** El usuario, enfocando en la dirección de uno de los POI's, puede visualizar los marcadores de dichos puntos.
- **Visualizar radar con POI's:** Siempre que el usuario este visualizando los POI's por la cámara observa un radar donde verá la situación de los mismos.
- **Seleccionar POI:** El usuario realiza la selección pulsando con sobre uno de los POI's. Este cambia su representación dentro con respecto a la situación anterior.

- **Mostrar/Ocultar card de POI:** Siempre que se selecciona un marcador se muestra la tarjeta con la información referente al mismo. Esta se ocultará al seleccionar otro POI o volver a pulsar sobre el actual.
- **Mostrar mapa con POI's:** El usuario accede a un mapa sobre el que puede ver señalizada tanto su posición como la de los POI's que tiene a su alrededor.
- **Mostrar información detallada de POI:** El usuario ve una interfaz en la que se muestra información del POI. Puede ver la distancia al mimos además de botones que activan otras funcionalidades.
- **Mostrar mapa del POI:** En la pantalla de detalles siempre se mostrar un mapa con la situación concreta del POI.
- **Cómo llegar:** El usuario abre un navegador que le dará indicaciones de cómo llegar a dicho POI.
- **Abrir página externa:** El usuario accede a una página externa a la aplicación relacionada con el POI.

6. Diseño

Este capítulo describe la arquitectura de nuestra aplicación así como la relación entre los diferentes componentes utilizados.

Tras esto mostraremos el diseño preliminar de la interfaz gráfica de usuarios utilizada para realizar la aplicación.

6.1. Arquitectura

En nuestro caso, la arquitectura viene determinada por el framework utilizado para el desarrollo de la aplicación. PhoneGap tiene una arquitectura determinada que podemos observar en la siguiente imagen:

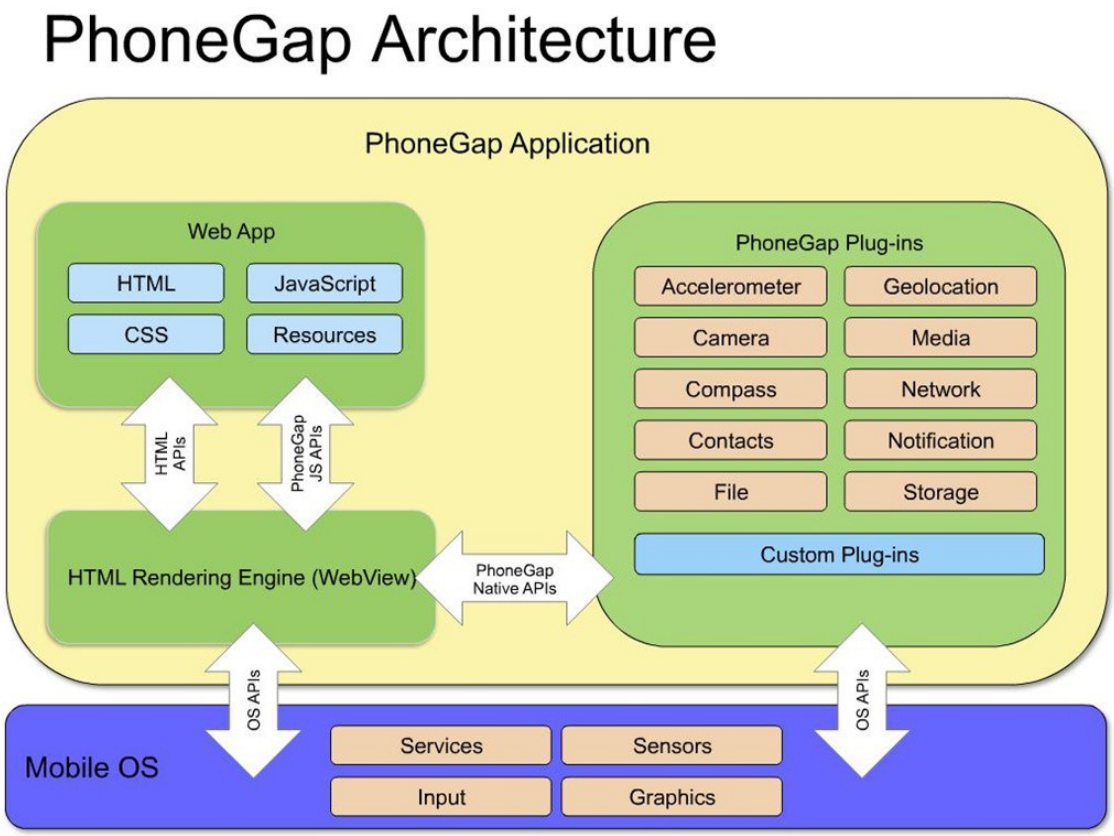


Imagen 22 Representación gráfica de la arquitectura de PhoneGap

Como podemos observar, existen dos zonas claramente diferenciadas. Son la capa de aplicación y la del SO (sistema operativo). En nuestro caso el SO es Android pero hay que recordar que PhoneGap nos permite desarrollar aplicaciones multiplataforma.

Dentro de la capa de aplicación podemos observar que existen nuevos módulos que interactúan entre si.

Por su lado, en la capa concerniente al SO podemos ver los servicios y los distintos componentes hardware con los que podemos interactuar.

Como vemos estas dos capas se comunican mediante las API's del sistema operativo.

Vamos ahora a desglosar la parte concerniente a la capa de aplicación.

6.1.1. Web App

Este es el módulo en la que se encuentra la parte desarrollada en este proyecto. Como vemos está dividida en cuatro partes: HTML, JavaScript, CSS y Resources. Ya se ha descrito en el punto 4 del presente documento cada una de estas tecnologías. Por lo tanto solo hay que mencionar que se refiere a los archivos de cada clase que hemos desarrollado en nuestra aplicación. En cuanto a Resoruces, se refiere a archivos de recursos como puedan ser las imágenes o fuentes para el texto.

Esta parte dentro del proyecto también tiene su propia arquitectura. En nuestro caso sigue una arquitectura típica de las aplicaciones Web, sin embargo no está demás comentar que habría sido posible desarrollar la aplicación mediante una arquitectura MVC (Model View Controller) utilizando AngularJS.

6.1.2. WebView

Es el módulo encargado de renderizar nuestra aplicación y mostrarla en una instancia del navegador del dispositivo. Sin embargo esta instancia tiene la particularidad de no proporcionar acceso a la interfaz de usuario de dicho navegador (favoritos, barra de navegación, etc.). Este componente es nativo del SO, no pertenece a PhoneGap si no que es una aplicación de usuario proporcionada en este caso por Android.

Esta se comunica con la Web App mediante API's proporcionadas para HTML y JavaScript.

6.1.3. Plugins en PhoneGap

Vamos a diferenciar, dentro de este modulo, dos tipos de plugins. Por un lado tendremos los plugins propios de PhoneGap (o Cordova) y los plugins de terceras partes como es Wikitude.

Los plugins se comunican tanto con el SO (mediante sus API's) como con WebView. Para conectar con estos, PhoneGap proporciona API's nativas del framework.

6.1.3.1. Plugins propios del framework

Dentro de los plugins propios de PhoneGap podemos observar que se encuentran aquellos que nos permiten acceder al los distintos elementos hardware del dispositivo y algunos para funciones como la pantalla de inicio de la aplicación, acceso a contactos, etc. En nuestro caso hemos hecho uso del plugin de geolocalización, el de splash screen y el plugin inappbrowser para utilizar aplicaciones nativas del sistema como el navegador.

6.1.3.2. Plugins de terceras partes

PhoneGap permite a los desarrolladores crear plugins que interactúen con su framework. Son los que en la imagen referente a la arquitectura de la aplicación se denominan "Custom plug-ins". Existen muchos y con una gran variedad de finalidades. En nuestro caso hemos usado el plugin de Wikitude para realidad aumentada comentado en el punto 4.

6.2. Interfaces

Los bocetos de interfaz de usuario se realizan con la idea de mostrar al usuario final una vista preliminar de la aplicación que estamos realizando. Estos nos ayuda a definir con más detalle ciertos requisitos.

A continuación mostramos los bocetos utilizados en el desarrollo de nuestra aplicación.

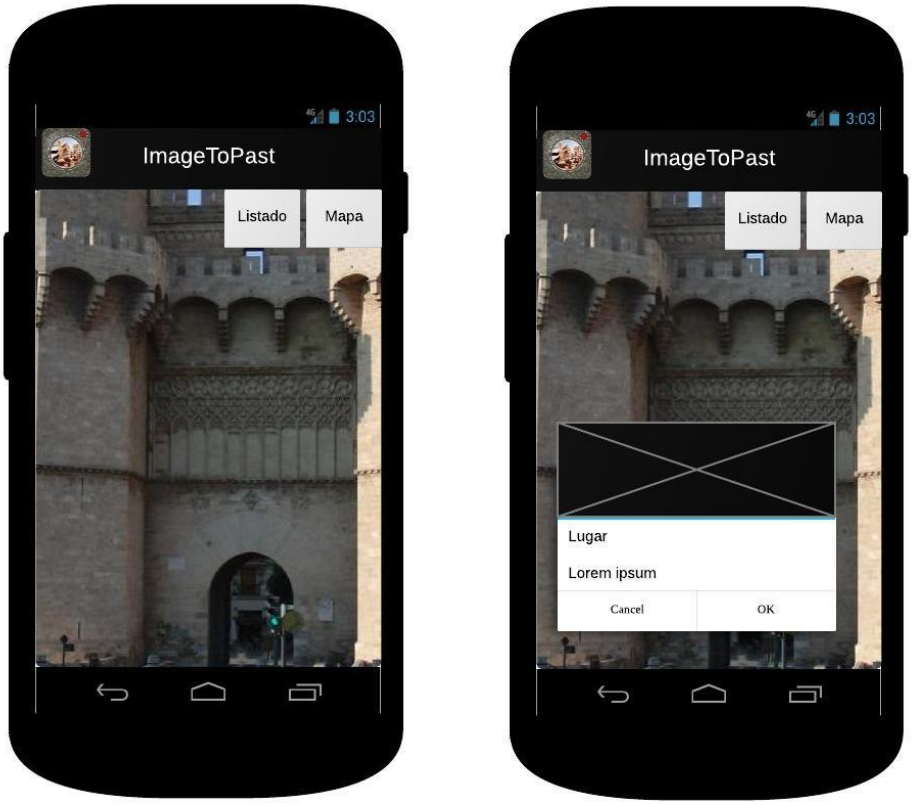


Imagen 23 Interfaz de cámara sin/con tarjeta de información

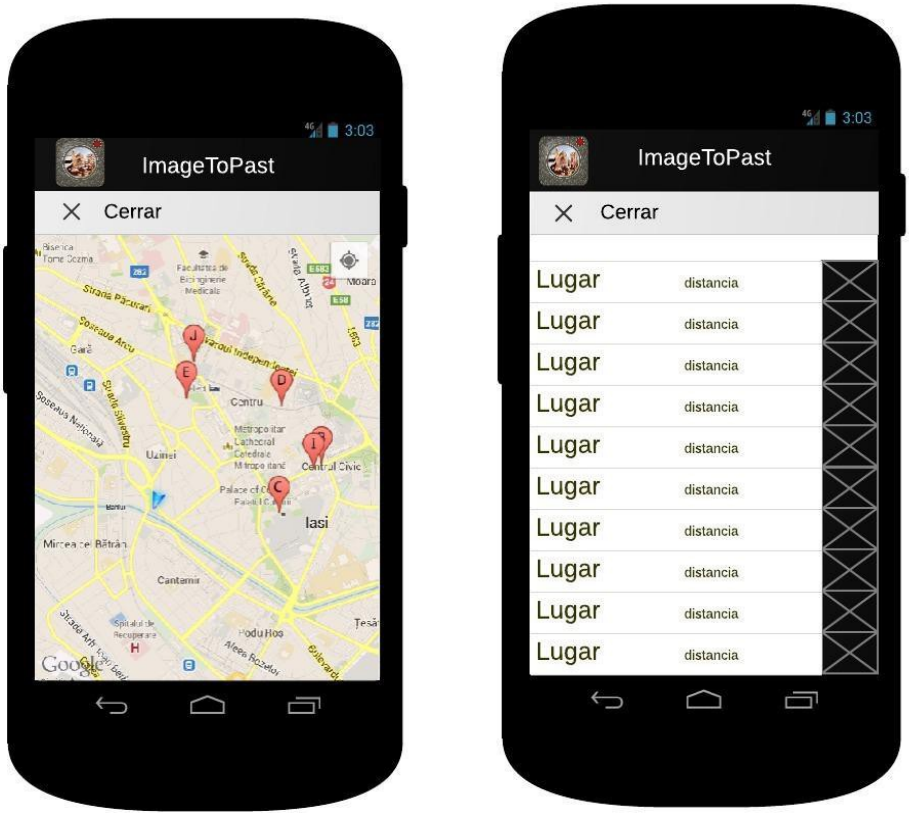


Imagen 24 Interfaz de mapa e interfaz de listado de puntos de interés

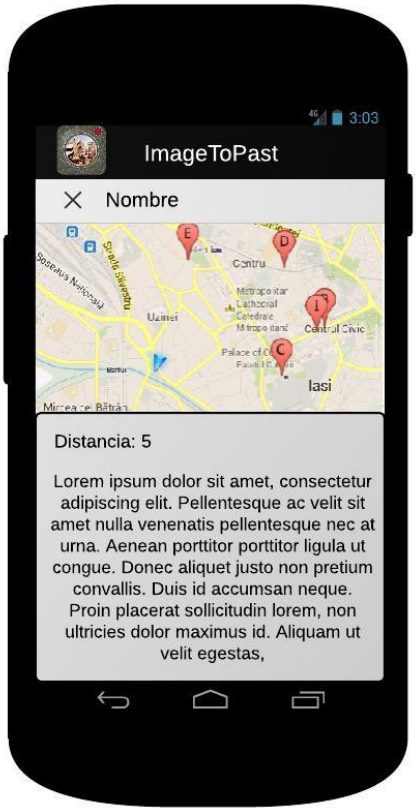


Imagen 25 Interfaz de detalle de un punto concreto

7. Detalles de implementación

En este capítulo describimos la implementación de la aplicación desarrollada. Nos centramos en los aspectos más relevantes del mismo, los problemas encontrados y las soluciones utilizadas. También describimos la interacción entre la aplicación y el plugin de realidad aumentada utilizado (Wikitude).

7.1. Estructura del proyecto

En este apartado describimos brevemente la estructura de directorios del proyecto de modo que al lector le sea más fácil identificar donde se encuentran los elementos a los que se haga referencia en apartados posteriores. La estructura es la siguiente:

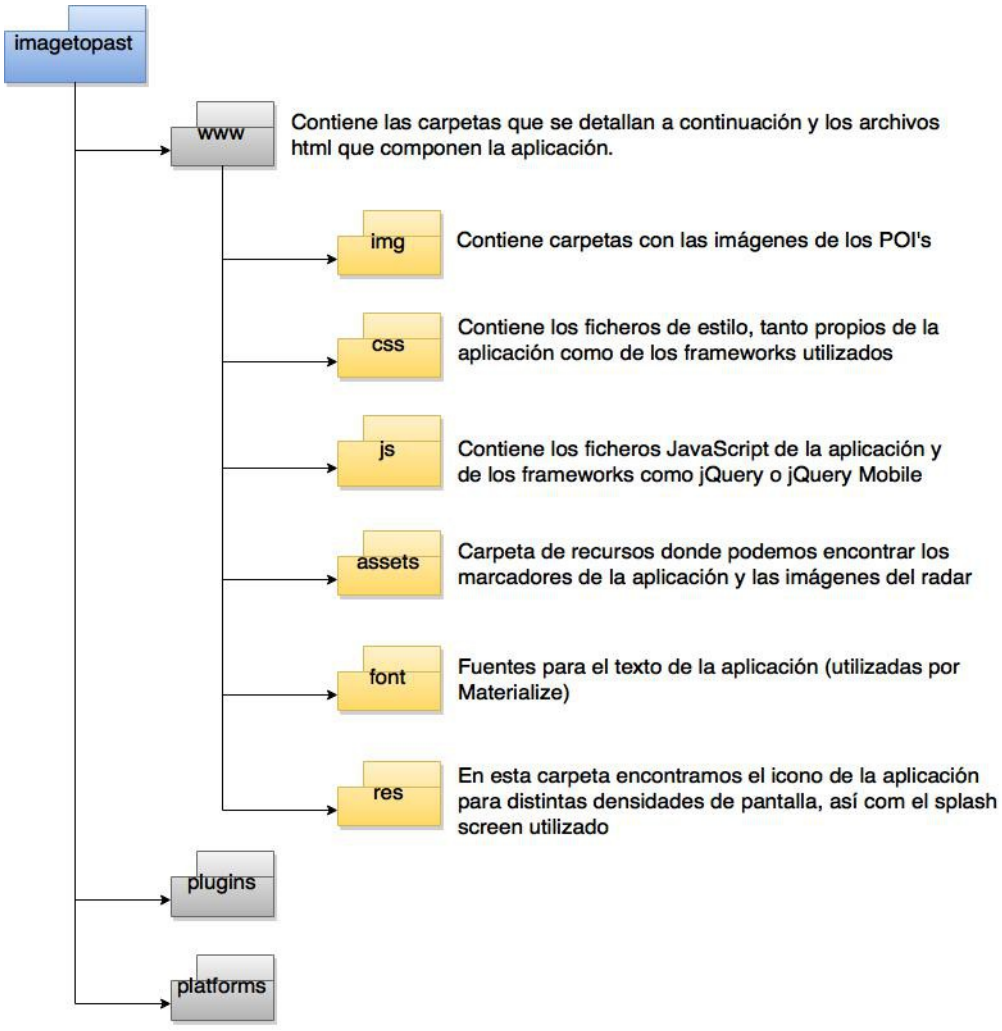


Imagen 26 Estructura de directorios de la aplicación

No vamos a explicar en detalle todas las carpetas, solo indicar que, como parece evidente, en la carpeta del plugins se encuentras los utilizados en el proyecto (Wikitude, geolocalización, splash screen e inappbrowser) y en la carpeta platform encontraremos el resultado de las diferentes compilaciones realizadas para los dispositivos que hayamos elegido (carpeta android en nuestro caso).

A parte de lo descrito existen otros archivos en el proyecto que son de configuración y que, en nuestro caso, PhoneGap se ha encargado de modificar con lo necesario cuando realizábamos algún cambio.

7.2. Inicialización de la AR

El inicio de la aplicación tiene ciertas particularidades que debemos mencionar. Para empezar la aplicación inicia con un fichero HTML que se utiliza, básicamente, para lanzar el script que configura e inicializa el entrono de realidad aumentada. Este fichero será por tanto `index.html`, el cual contiene el siguiente código:

```
43          <script type="text/javascript">
44              app.initialize();
45          </script>
```

Lo único que hace es llamar al método encargado de la inicialización que podemos encontrar en `index.js` y que explicamos a continuación.

`index.js` es el encargado de configurar los eventos que se ejecutan al inicializar la aplicación, comprobar si nuestro dispositivo es compatible con el plugin de AR, enlazar y configurar el mismo. Así mismo, proporciona un entorno para llamar funciones fuera del ámbito del plugin. Esto último lo explicamos en detalle en un apartado posterior, en el que hacemos uso del mismo.

Lo primero que encontramos es una variable llamada `myWorld`. En ella se definen aspectos necesarios para la configuración de Wikitude.

```
1  var myWorld = {
2      "path": "www/world.html",
3      "requiredFeatures": [
4          "geo"
5      ],
6      "startupConfiguration": {
7          "camera_position": "back"
8      }
9  };
```

Los aspectos que se describen son el path, en el que indicamos la URL donde se inicia y ejecuta el plugin de realidad aumentada. El segundo parámetro de configuración indica las características que necesitamos que utilice el plugin que en nuestro caso es la geolocalización. Por último le indicamos la configuración de inicio. En este caso el

único punto que nos interesa a nosotros es qué cámara debe activar. Indicamos que la cámara posterior del dispositivo.

A continuación se define `app`, con sus variables y métodos. Solo mencionar que esto se realizaría con cualquier aplicación creada mediante PhoneGap, así como la definición del método `initialize`, enlazar los diferentes eventos (imprescindible `deviceready`) e implementar los manejadores de dichos eventos. Ahora nos centraremos en la parte específica del proyecto que nos ocupa, la configuración y puesta en marcha de Wikitude.

Para ello, una vez que se ha lanzado el evento que indica que el dispositivo está preparado y realizada la llamada al manejador correspondiente, se ejecuta el siguiente código:

- Creamos una instancia de Wikitude Plugin para poder utilizarla:

```
app.wikitudePlugin =
    cordova.require("com.wikitude.phonegap.WikitudePlugin.WikitudePlugin");
```

Comprobamos si el dispositivo soporta la AR. En caso afirmativo, queda reflejado en la variable `isDeviceSupported` y lanza `loadARchitectWorld` con la variable antes definida. Si la comprobación falla informamos mediante una alerta y también lo indicamos en la variable de igual forma.

```
34      app.wikitudePlugin.isDeviceSupported(function() {
35              app.isDeviceSupported = true;
36              app.loadARchitectWorld(myWorld);
37          }, function(errorMessage) {
38              app.isDeviceSupported = false;
39              alert('Unable to launch ARchitect Worlds on this device: \n' + errorMessage);
40          },
41          [app.wikitudePlugin.FeatureGeo, app.wikitudePlugin.Feature2DTracking]
42      );
```

El método `loadARchitectWorld` es el encargado de cargar el ARchitect World (arquitectura de realidad aumentada) definida en nuestra aplicación en función de los parámetros configurados en la variable `myWorld` definida anteriormente.

```
56  loadARchitectWorld: function(myWorld) {
57      app.wikitudePlugin.setOnUrlInvokeCallback(app.onUrlInvoke);
58
59      if (app.isDeviceSupported) {
60          app.wikitudePlugin.loadARchitectWorld(function successFn(loadedURL) {
61                  /* Respond to successful world loading if you need to */
62              }, function errorFn(error) {
63                  alert('Loading AR web view failed: ' + error);
64              },
65              myWorld.path, myWorld.requiredFeatures, myWorld.startupConfiguration
66          );
67      } else {
68          alert("Device is not supported");
69      }
70  },
```

Como podemos observar, el método local `loadARchitectWorld` llama al método del mismo nombre de la instancia de Wikitude, pasándole como variables los distintos parámetros de configuración.

7.3. Implementación

En las siguientes secciones vamos a explicar los detalles de implementación de la interfaz de usuario y la lógica relacionada con cada uno de los elementos con los que interactúa el usuario. En primer lugar daremos una visión general de la estructura de la interfaz en cuanto a páginas.

Las interfaces están generadas mediante HTML5 y CSS3. En muchos elementos interviene el framework Materialize. No vamos a entrar en la implementación de cada uno de ellos ya que haría este documento extenso y no entra en el ámbito de estudio. Por tanto, nos centraremos en aquellos que interactúen con la lógica de negocio, describiendo el resto.

A continuación nos centramos en cada una de las pantallas de la interfaz para explicar la construcción de los elementos visuales y la lógica de negocio así como la interacción entre ambas.

7.3.1. Estructura de la interfaz

La interfaz de usuario se ha desarrollado por medio de HTML5 y CSS3. Para la estructura de páginas se ha aprovechado la necesidad de Wikitude de utilizar jQuery Mobile. De esta forma se ha desarrollado todo en una sola página HTML, utilizando la potencia de los roles proporcionados por jQuery Mobile para las diferentes estructuras.

La estructura de páginas ha quedado como sigue dentro de archivo `world.html`:

```
50      <body >
51          <div data-role="page" id="cam-page" style="background: none;" >
52           ...
53          </div>
54          <div data-role="page" id="map-page" data-theme="b">
55              <div data-role="header" data-theme="b" id="cerrar" data-position="fixed">
56              ...
57              </div>
58              <div role="main" class="ui-content" id="map-canvas" >
59              </div>
60          </div>
61
62          <div data-role="page" id="list-page" data-theme="b" style="background: none;">
63              <div data-role="header" data-theme="b" id="cerrar" data-position="fixed">
64              ...
65              </div>
66              <div role="main" class="ui-content material_list">
67              ...
68              </div>
69          </div>
70          <div data-role="page" id="detail-page" data-theme="b" style="background: none;">
71              <div data-role="header" data-theme="b" id="cerrar" data-position="fixed">
72                ...
73              </div>
74          </div>
75      </body>
76
```

Podemos observar que la aplicación está divida en cuatro páginas, una por cada `data-role="page"`. Dentro de estas podemos encontrar otras secciones que tienen un rol específico dentro de la propia página como son la cabecera o el contenido principal (`role="main"`).

Estos roles de página y el framework para plataformas móviles de jQuery nos proporcionan la sensación de estar manejando varias páginas cuando en realidad solo utilizamos un archivo. Esto es útil cuando las páginas no contienen una gran cantidad de elementos y la extensión del fichero único no se hace demasiado grande y difícil de manejar. Por otro lado, en el caso que nos ocupa optimiza el tiempo de respuesta ya que no tiene que cargar un nuevo elemento sino transitar entre elementos ya cargados.

7.3.2. Cámara

Los elementos aquí descritos se encuentran dentro del data-role con id "`cam-page`" que se puede ver en la estructura descrita en el punto 7.3.1.

7.3.2.1. Interfaz de usuario

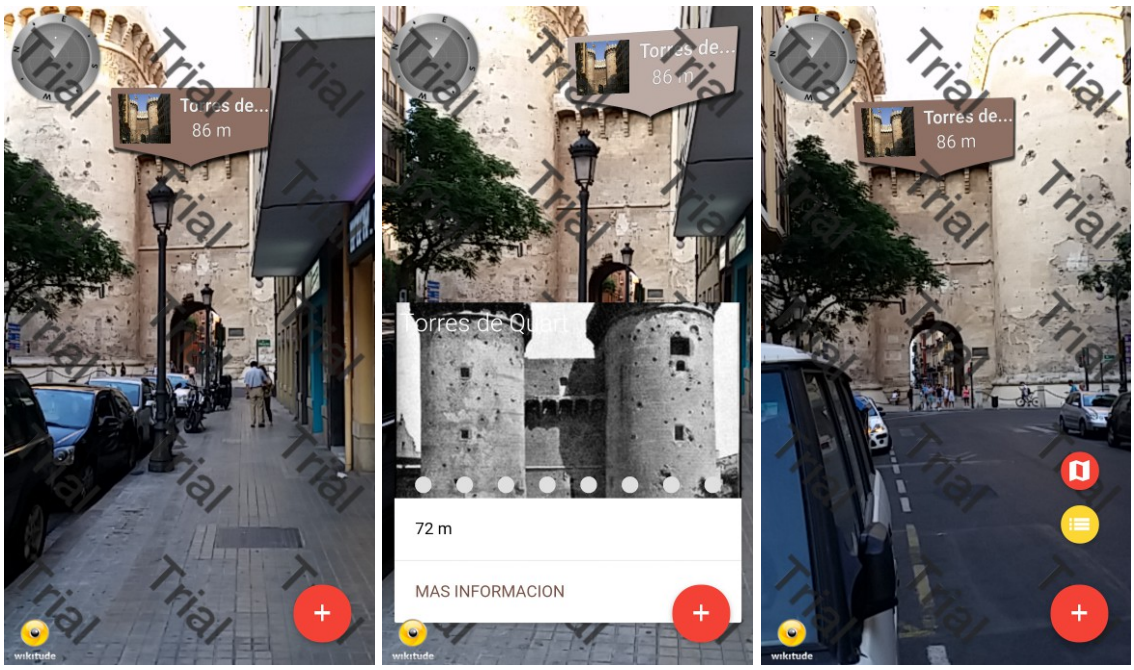


Imagen 27 Interfaz de cámara

El primer elemento que podemos apreciar arriba a la izquierda de todas las imágenes es el radar. Dentro de este elemento veremos que existen una serie de puntos blancos que representan los POI's. El HTML correspondiente a este es:

```
52          <div class="radarContainer_left" id="radarContainer"></div>
```

Está compuesto por dos imágenes, el anillo exterior que indica la orientación del usuario y el círculo interior. Tanto estas imágenes como los puntos son situados por la lógica mediante Wikitude como se explica más adelante.

A continuación podemos ver, abajo a la derecha, una serie de botones que dan acceso a otras interfaces de la aplicación. Inicialmente, solo se muestra el botón con el símbolo +, tal y como se muestra en la primera imagen. Al pulsar sobre este, podemos observar cómo se despliegan el resto de botones. Todos estos elementos están implementados en el framework Materialize.

En las imágenes podemos observar los marcadores de los POI's. Todos los elementos que podemos observar son generados mediante la lógica y se describirán en el siguiente apartado.

Por último, en la segunda imagen vemos la tarjeta encargada de la presentación de imágenes. El elemento en sí también forma parte de Materialize. Sin embargo ha sido necesario modificarlo para ajustarlo a las necesidades del proyecto. El código HTML queda como sigue:

```
62 ▼        <div id="detail-viewer" class="row">
63 ▼            <div class="col s12 m7">
64 ▼                <div class="card">
65 ▼                    <div class="slider" >
66                          <ul id="miSlides" class="slides">
67                          </ul>
68                      </div>
69                      <span id="name" class="card-title"></span>
70                      <div class="card-content">
71                          <p id="distance"></p>
72                      </div>
73                      <div class="card-action">
74                          <a style="color:black;" href="#">MAS INFORMACION</a>
75                      </div>
76                  </div>
77              </div>
78          </div>
```

Como podemos ver, todos los elementos que van a contener información dinámica del POI seleccionado tienen atributo `id`. Además, se ha sustituido el elemento de imagen de la tarjeta normal de Materialize por un slider que nos permite pasar fotos desplazando el dedo.

Esta tarjeta se encuentra oculta y solo se muestra al seleccionar un marcador como el mostrado en las imágenes.

7.3.2.2. Detalles de implementación

A continuación vamos a explicar la creación de los marcadores que Wikitude sitúa sobre la interfaz de la cámara para mostrar dónde se encuentran los mismos. Los atributos de un POI se han definido en la fase de análisis del presente documento. El acceso a estos atributos de hace mediante el paso de un fichero JSON de la siguiente manera.

Obtención y actualización de datos de los POI's

1. Para representar el conjunto de POI's utilizamos la clase `Word`. Cada vez que cambiamos de posición, incluyendo la primera vez que se accede a la aplicación, el plugin ejecuta un método propio llamado `setLocation()`. Esto provoca que se ejecute nuestro método:

```
82      locationChanged: function locationChangedFn(lat, lon, alt, acc) {
83          localizaciones= World.markerList.sort(World.sortByDistanceSorting);
84
85          World.userLocation = {
86              'latitude': lat,
87              'longitude': lon,
88              'altitude': alt,
89              'accuracy': acc
90          };
91
92          /*
93              The custom function World.onLocationChanged checks with the flag World.initiallyLoadedData if the
                function was already called. With the first call of World.onLocationChanged an object that contains geo
                information will be created which will be later used to create a marker using the World.
                loadPoisFromJsonData function.
94          */
95          if (!World.initiallyLoadedData) {
96              /*
97                  requestDataFromLocal with the geo information as parameters (latitude, longitude) creates different
                    poi data to a random location in the user's vicinity.
98              */
99              World.requestDataFromLocal(lat, lon);
100             World.initiallyLoadedData = true;
101         } else if (World.locationUpdateCounter === 0) {
102             // update placemark distance information frequently, you max also update distances only every 10m with
                some more effort
103             World.updateDistanceToUserValues();
104         }
105
106         // helper used to update placemark information every now and then (e.g. every 10 location upadtes fired)
107         World.locationUpdateCounter = (++World.locationUpdateCounter % World.
                updatePlacemarkDistancesEveryXLocationUpdates);
108     },
```

Como podemos observar, primero se almacenan en un vector llamado localizaciones que accesible en todo momento, la lista de marcadores de los POI's.

A continuación guardamos los datos de posición del usuario. Seguidamente se comprueba si los datos ya han sido cargados previamente. Si no es así se llamará al método `requestDataFromLocal` que se encargará de llamar otro método para la carga del JSON en el que se encuentran los datos de nuestros POI's. Después cambia el valor de `initiallyLoadedData` para indicar que los datos están cargados.

En caso contrario (los datos ya están cargados) este método recalcula la distancia al usuario.

En nuestro caso la única función que tiene `initiallyLoadedData` es llamar a

```
138         World.loadPoisFromJsonData(myJsonData);
```

Este es el método encargado de cargar los puntos. Su implementación es la siguiente:

```
20      loadPoisFromJsonData: function loadPoisFromJsonDataFn(poiData) {
21          // empty list of visible markers
22          World.markerList = [];
23
24          // show radar & set click-listener
25          PoiRadar.show();
26          $('#radarContainer').unbind('click');
27
28          // start loading marker assets
29          World.markerDrawable_idle = new AR.ImageResource("assets/newmarker1.png");
30          World.markerDrawable_selected = new AR.ImageResource("assets/newmarker2.png");
31
32          // loop through POI-information and create an AR.GeoObject (=Marker) per POI
33          for (var currentPlaceNr = 0; currentPlaceNr < poiData.length; currentPlaceNr++) {
34              var singlePoi = {
35                  "id": poiData[currentPlaceNr].id,
36                  "latitude": parseFloat(poiData[currentPlaceNr].latitudeOffset),
37                  "longitude": parseFloat(poiData[currentPlaceNr].longitudeOffset),
38                  "altitude": parseFloat(poiData[currentPlaceNr].altitude),
39                  "title": poiData[currentPlaceNr].name,
40                  "image": poiData[currentPlaceNr].image,
41                  "numimages": parseInt(poiData[currentPlaceNr].numimages),
42                  "images": poiData[currentPlaceNr].images,
43                  "description": poiData[currentPlaceNr].description,
44                  "web": poiData[currentPlaceNr].web
45              };
46
47              /*
48                  To be able to deselect a marker while the user taps on the empty screen,
49                  the World object holds an array that contains each marker.
50              */
51              World.markerList.push(new Marker(singlePoi));
52          }
53          World.updateDistanceToUserValues();
54
55          World.updateStatusMessage(currentPlaceNr + ' places loaded');
56      },
```

Como podemos ver, es aquí donde llamamos al método que inicializará el radar. Después de esto define en las líneas 29 y 30 cuáles van a ser las imágenes que se van a mostrar en la interfaz de cámara (`newmarker1.png` y `newmarker2.png`). Una de ellas corresponde al marcador en su estado inicial y el otro cuando uno de ellos está seleccionado.

Posteriormente, entre las líneas 33 y 51, se van inicializando los POI's en base al contenido del fichero JSON. Se cogen los distintos atributos, se añaden a una variable y esta misma es finalmente añadida a un `array`.

Por último, este método llama al encargado de actualizar la distancia con el usuario.

2. Como podemos ver en la línea 50 del método anterior, cuando se añaden los POI's al listado denominado `markerList` se crea al mismo tiempo el objeto `Marker` encargado de representar a dicho POI.

Representación de los POI's tanto en la cámara como en el radar.

De la representación de los puntos se encarga el objeto `Marker`. Para ello lo primero que hace es crear otro objeto, propio de Wikitude al cual le pasamos la latitud y longitud del punto de la siguiente manera:

```
var markerLocation = new AR.GeoLocation(poiData.latitude, poiData.longitude, poiData.altitude);
```

Posteriormente crea para cada `ImageResource` que representa el marcador y que se indicaron en la clase `World` (`markerDrawable_idle`, `markerDrawable_selected`) un objeto `ImageDrawable`. Estos también forman parte de Wikitude. Para definirlos debemos indicar, además del ya mencionado `ImageResource`, el tamaño y una variable de configuración con los siguientes datos:

- `zOrder`: indica el orden en cuanto al eje x de este objeto cuando coincide dentro de una misma representación con otros objetos del tipo `drawable`.
- `opacity`: Similar a CSS indica la opacidad del objeto.
- `onClick`: en este se define la función a ejecutar cuando se realiza clic sobre el objeto.

Un ejemplo sería:

```
this.markerDrawable_selected = new AR.ImageDrawable(World.markerDrawable_selected, 2, {
    zOrder: 1,
    opacity: 0.0,
    onClick: null
});
```

Estos dos componentes son genéricos para todos los puntos que vamos a representar.

Por otro lado se crean dos objetos más. Uno para el título que utiliza un `Label` (objeto de Wikitude) y otro del tipo `ImageDrawable` con información obtenida de cada punto y que se implementan de la siguiente forma:

- Para el título:

```
this.titleLabel = new AR.Label(poiData.title.trunc(10), 0.5, {
    zOrder: 2,
    offsetY: 0.55,
    offsetX: 0.8,
    style: {
        textColor: '#ededed',
        fontStyle: AR.CONST.FONT_STYLE.BOLD
    }
});
```

El objeto `Label` también es de tipo `drawable`. Los atributos que tiene son el texto, el tamaño y una variable de configuración. Esta, como la del `ImageDrawable`, tiene un atributo `zOrder`. Además contiene los parámetro `offsetY` y `offsetX` que indica su posición en los ejes X e Y con respecto a su elemento contenedor. El último parámetro define el estilo del texto.

- Para la imagen:

```
51      this.imageLabel = new AR.ImageDrawable(this.myImage, 1.5, {
52          zOrder: 3,
53          offsetX: -1.5,
54          opacity: 1,
55          onClick: null
56      })
```

Como vemos, para este caso empezamos creando un `ImageResource` que contenga la imagen que queremos que represente a nuestro punto. El resto de parámetros de configuración se han visto en elementos anteriores.

Ahora, vemos como representa los puntos en el radar. La implementación es la siguiente:

```
47      this.radarCircle = new AR.Circle(0.03, {
48          horizontalAnchor: AR.CONST.HORIZONTAL_ANCHOR.CENTER,
49          opacity: 0.8,
50          style: {
51              fillColor: "#ffffff"
52          }
53      });
```

Como vemos en este caso se utiliza un objeto `Circle` de Wikitude. La configuración de este objeto es similar a lo comentado en los anteriores.

Todos los puntos del radar se van añadiendo a un `array` denominado `radardrawables`.

- El objeto que Marker

El objeto `Marker` crea su representación. Para ello utiliza un objeto `GeoObject` de Wikitude. A este solo hay que indicarle la localización (latitud y longitud del punto (`markerLocation`) y cada uno de los objetos "`drawable`" dentro de cada una de las partes de la representación.

```
71      this.markerObject = new AR.GeoObject(markerLocation, {
72          drawables: {
73              cam: [this.markerDrawable_idle, this.markerDrawable_selected, this.titleLabel, this.
                    imageLabel],
74              radar: this.radardrawables
75
76          }
77      });
```

Como vemos, en la configuración del objeto indicamos cada una de las representaciones del mismo, donde se debe pintar (en el radar o en la cámara).

- Para la distancia:

```
81      this.distanceToUser = this.markerObject.locations[0].distanceToUser();
82
83      this.distanceLabel = new AR.Label((this.distanceToUser > 999) ? ((this.distanceToUser / 1000).toFixed(2) + " km")
            : (Math.round(this.distanceToUser) + " m").trunc(10), 0.5, {
84          zOrder: 2,
85          offsetX: 0.5,
86          style: {
87              textColor: '#ededed'
88          }
89      });
90
91      this.markerObject.drawables.addCamDrawable(this.distanceLabel);
```

Por último, para poder obtener la distancia ha sido necesario crear el objeto una vez creado el objeto `GeoObject` para poder utilizar el método `distanceToUser()`. Una vez hecho esto lo añadimos a la cámara mediante el método `addCamDrawable()`.

Obtención de información del POI y presentación en la tarjeta

1. Para que la tarjeta muestre, el elemento `markerDrawable_idle` correspondiente al marcador sin seleccionar provee de una función que responde al evento generado al pulsar sobre el mismo.

```
18              onClick: Marker.prototype.getOnClickTrigger(this)
```

2. Una vez ejecutado el manejador `getOnClickTrigger`, este se encarga de insertar la información del POI en la tarjeta de la interfaz grafica, así como de hacerla visible. Para ello, lo primero que hace es modificar la propiedad `css bottom` de todo el elemento `detail-viewer` (`id` de la tarjeta). Luego, utilizando la propiedad `id` de cada uno de los elementos HTML utilizados para definir la tarjeta el método asigna el valor del atributo correspondiente del punto.

```
133     // NOS DA LA DISTANCIA ENTRE NUESTRA POSICION Y EL MARKER
134     marker.distanceToUser = marker.markerObject.locations[0].distanceToUser();
135     // MOSTRAMOS UN DIV CON MÁS INFORMACION
136     document.getElementById("detail-viewer").style.bottom = "5px";
137     document.getElementById("name").innerHTML = marker.poiData.title;
138     document.getElementById("distance").innerHTML = (marker.distanceToUser > 999) ? ((marker.distanceToUser / 1000).
            toFixed(2) + " km") : (Math.round(marker.distanceToUser) + " m");
139     var numImages = parseInt(marker.poiData.numimages);
140     var carrusel = "";
141     for (i=1; i<=numImages; i++) {
142         carrusel = carrusel+"<li> <img style='height:200px;' src='"+marker.poiData.images+"/"+marker.poiData.id+i+".
                jpg'></li>";
143
144     }
145     document.getElementById("miSlides").innerHTML = carrusel;
146     $('.slider').slider();
147
148     for (i=0; i<localizaciones.length; i++){
149         if (localizaciones[i].poiData.id == marker.poiData.id){
150             id=i;
151             break;
152         }
153     }
```

Vemos que para el caso del slider, se realiza un bucle para ir añadiendo todas las imágenes que se contienen el la carpeta correspondiente.

Este método también se encarga de inicializar el slider encargado de mostrar todas las imágenes de la carpeta indicada.

Por último realizamos un bucle para conocer el índice de este marcador concreto dentro del `array` de localizaciones. Con esto conseguiremos poder acceder a su pantalla de detalle desde la opción “MAS INFORMACIÓN” de la tarjeta.

7.3.3. Mapa

Los elementos aquí descritos se encuentran dentro del data-role con id "`map-page`" que se puede ver en la estructura descrita en el punto 7.3.1.

7.3.3.1. Interfaz de usuario

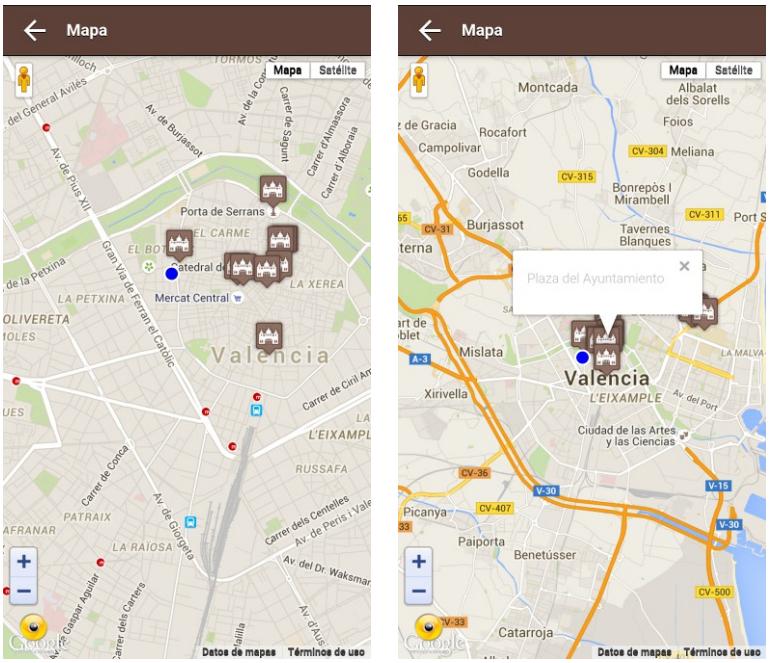


Imagen 28 Interfaz de mapa

La interfaz está dividida en dos zonas. Por una parte tenemos el `header`, el cual contiene el botón de retorno y el título de dicha interfaz como se ve a continuación:

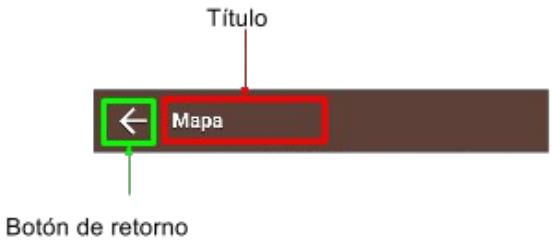


Imagen 29 Header de mapa

El estilo de la misma lo proporciona el framework Materialize. Se construye como una barra de navegación de la siguiente manera:

```
93          <nav>
94              <div class="nav-wrapper brown darken-2">
95                  <ul id="nav-mobile" class="left">
96                      <li><a href="#cam-page" data-transition="slidedown"><i class="mdi-
                        navigation-arrow-back"></i></a></li>
97                  </ul>
98                    <span class='title'>Mapa</span>
99              </div>
100         </nav>
```

Por otro lado tenemos el mapa. Esta parte está generada por medio del API de Google mediante JavaScript. En cuanto a implementación de interfaz gráfica, lo único que tenemos es un `<div>` que será el encargado de contener el mapa:

```
102         <div role="main" class="ui-content" id="map-canvas" >
103         </div>
```

Lo único que debemos hacer es asegurarnos que, dentro del fichero HTML que compone la interfaz se incluye el fichero JavaScript que genera el mapa.

Podemos observar que el objetivo del mapa es identificar en el mismo la situación del usuario (circulo azul) y de los diferentes POI's.

7.3.3.2. Detalles de implementación

Para la implementación, lo primero que debemos tener en cuenta es donde obtener la posición del usuario. Lo normal en estos casos es utilizar el plugin de geolocalización de Apache que forma parte de Cordova (y por ende, de PhoneGap). Sin embargo hay dos cosas a tener en cuenta.

Para empezar, una vez iniciado el plugin de AR de Wikitude, no es posible llamar a la geolocalización de la manera tradicional. Esto es, utilizando el plugin e invocándolo de la siguiente forma:

```
...
if(navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      function(position) {
...
```

Pese a que la aplicación reconocerá que la geolocalización esta activa en el primer `if`, dentro, no ejecutará la función `getCurrentPosition` del plugin.

No se ha encontrado en la literatura del plugin el motivo de la misma. Sin embargo y como veremos cuando comentemos el uso de InAppBrowser, parece que una vez iniciado el navegador de AR Wikitude bloquea el uso de otros plugins.

Lo segundo es, que al hilo de lo comentado, nuestro plugin está constantemente consultando la posición del usuario ya que debe situarlo para poder colocar el entorno de realidad aumentada a su alrededor.

Como hemos visto en el punto 7.3.2.2, existe una función llamada `locationChanged` que se ejecuta cada vez que el usuario cambia de posición. En esta función lo primero que hacemos es almacenar en la variable `userLocation` todos los datos de geolocalización del usuario. Con lo cual no necesitamos más que consultar la misma para obtener los datos necesarios.

Por lo tanto la implementación del mapa con los marcadores del usuario y los diferentes POI's queda de la siguiente forma:

- En primer lugar se define el manejador para que se cree el mapa cuando se muestre el `<div>` con `data-role="map-page"`.

```
48  $(document).on('pageshow','#map-page', function(){
49      generaMapa();
50  });
```

- La función `generarMapa` empieza por definir el objeto `LatLang` del API de Google que contiene la posición del usuario. Con este se definen las opciones del mapa entre las cuales está la de centrar el mapa en la posición del usuario definida en el objeto antes mencionado.

```
10      var LatLang = new google.maps.LatLng(World.userLocation.latitude, World.userLocation.
            longitude);
11      var myOptions = {
12          zoom: 12,
13          center: LatLang,
14          mapTypeId: google.maps.MapTypeId.ROADMAP
15      };
```

- Con esto ya podemos proceder a crear el objeto de mapa. Para ello debemos indicarle el `<div>` donde se va a representar que es el que hemos comentado en el punto de interfaz y que tiene como `id="map-canvas"`.

```
16      var map = new google.maps.Map(document.getElementById("map-canvas"), myOptions);
```

- Para definir el marcador que identifica la posición de usuario en el mapa creamos un objeto `Marker` del API de Google (no confundir con los `Marker` de nuestra aplicación). En él definimos tanto la posición del mismo como la forma. Para este caso hemos optado por un objeto `Circle` (también proporcionado por Google) al que le podemos definir atributos como tamaño, color, etc. Además, le indicamos el mapa en el que se tiene que representar.

```
18        var marker = new google.maps.Marker({
19            position: LatLang,
20            icon: {
21                path: google.maps.SymbolPath.CIRCLE,
22                scale: 7,
23                strokeColor: '#FFFFFF',
24                strokeWeight: 2,
25                fillColor: '#00F',
26                fillOpacity: 1
27            },
28            map: map
29        });
```

- Por último, situamos todos los POI's, almacenados en la variable `localizaciones`, en el mismo mapa. Para este caso, para la representación gráfica del marcado hemos utilizado una imagen. Como vemos, para cada POI creamos un objeto `LatLang` para su localización y es el que se le indica como posición al marcador. También añadimos la ventana de información que se mostrará al pulsar sobre el marcador y el mismo manejador del evento de pulsación.

```
30    localizaciones.forEach(function(element, index, array){
31        var contentString = '<div id="info">'+
32                                '<p>'+element.poiData.title+'</p>'+
33                            '</div>';
34        var infowindow = new google.maps.InfoWindow({
35            content: contentString
36        });
37        var posicionPoi = new google.maps.LatLng(element.poiData.latitude, element.poiData.longitude);
38        var marker = new google.maps.Marker({
39            position: posicionPoi,
40            icon: 'img/marker.png',
41            map: map,
42            title: element.poiData.title,
43            animation: google.maps.Animation.DROP
44        });
45        google.maps.event.addListener(marker, 'click', function() {
46            infowindow.open(map,marker);
47          });
48
49    });
```

7.3.4. Listado de puntos de interés

Los elementos aquí descritos se encuentran dentro del data-role con id "`list-page`" que se puede ver en la estructura descrita en el punto 7.3.1.

7.3.4.1. Interfaz de usuario

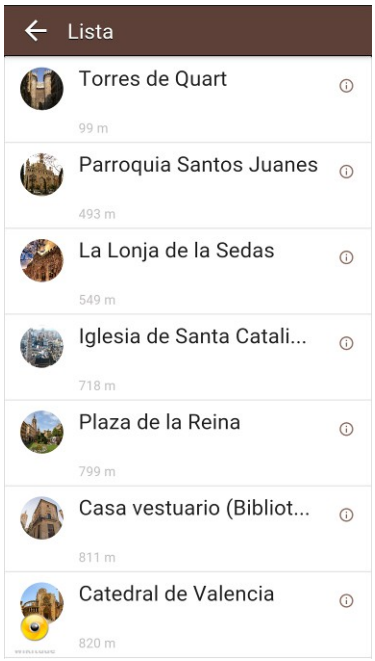


Imagen 30 Interfaz de Listado

En esta interfaz diferenciamos dos zonas. Por un lado `header`, que al igual que en la página de mapa contiene el título de la página y un botón de vuelta atrás representado por un flecha. Por el otro está el listados en sí. Todos los elementos se definen mediante el framework Materialize, encargado de definir el estilo.

Cada uno de los elementos de listado está compuesto por tres elementos que vemos representados en la siguiente figura:

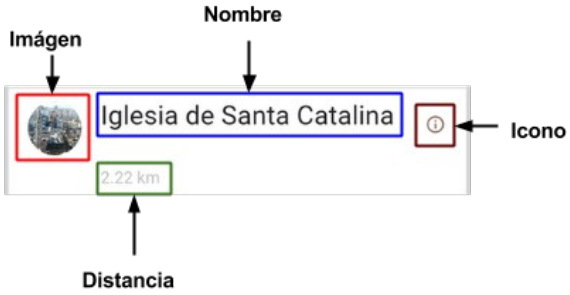


Imagen 31 Elemento del listado

Todos estos elementos se generan dinámicamente y se anidan bajo la estructura de lista siguiente:

```
116         <ul class="collection" id="listado">
117         </ul>
```

7.3.4.2. Detalles de implementación

Para añadir los elementos al listado de POI's con todos sus atributos, incluyendo la distancia al usuario, hemos añadido un manejador que se ejecuta cada vez que se muestra la página. Para ello hemos utilizado los eventos sobre el documento HTML que provee jQuery. La implementación es la siguiente:

```
96  /****************** list-page **********************/
97  $(document).on('pageshow','#list-page', function() {
98      localizaciones = World.markerList.sort(World.sortByDistanceSorting);
99      var lista = "";
100     localizaciones.forEach(function(element, index, array){
101         var distancia = (element.distanceToUser > 999) ? ((element.distanceToUser / 1000).
                toFixed(2) + " km") : (Math.round(element.distanceToUser) + " m")
102         lista = lista + "<li class='collection-item avatar materialize_esp'>\
103           <img src='"+element.poiData.image+"' alt='' class='circle'>\
104           <span class='title'>"+element.poiData.title.trunc(24)+"</span>\
105           <p> <br>\
106              "+distancia+"\
107           </p>\
108           <a href='#detail-page' id='"+index+"' class='secondary-content' data-
                transition='slide'><i class='mdi-action-info-outline itp-list-link'></i></a>\
109         </li>";
110     });
111     $( "#listado").html(lista);
112     $("a").on("click", function(e){
113         id = $(this).attr("id");
114     });
115 });
```

Como podemos ver, lo primero que hacemos es indicar a que `id` asociamos el evento "`pageshow`". Esto se debe a lo que ya comentamos anteriormente, todas nuestras interfaces están en el mismo documento gestionadas por jQuery Mobile gracias al `id`.

Lo primero que realiza la función es extraer del objeto `World` todos los POI's cargados. A este le aplica el método de ordenación por distancia. Así almacenamos en

una variable local denominada `localizaciones` que será la que utilizamos para crear los objetos de la lista. También definimos una variable para almacenar el HTML que incrustamos.

Para ello por cada elemento contenido en el `array localizaciones` hacemos lo siguiente:

- Damos a la distancia un formato más legible almacenándolo en una variable local.
- Creamos un HTML `<li>` que contendrá los datos del POI's
- Dentro del elemento lista, creamos un elemento `<img>` para contener la imagen, un elemento `<span>` para el título y un elemento `<p>` para la distancia. Al título le aplicamos el método `trunc` para acortarlo en caso de superar los 24 caracteres.
- Finalmente se añade el elemento de enlace `<a>` al cual damos como propiedad `id` el índice que ocupa el elemento en el `array`. Esto es imprescindible para realizar la navegación desde este listado a la pantalla de detalles.

A continuación añadimos el elemento listado, que contendrá el código HTML de todos los POI's a elemento con `id #listado`, que es el elemento `<ul>` mostrado anteriormente.

Por último le asociamos a los enlaces el evento "`click`" que obtendrá el `id` del enlace pulsado. Esto lo utilizaremos para cargar los elementos de detalle del punto en la pantalla correspondiente.

7.3.5. Detalles del punto de interés

Los elementos aquí descritos se encuentran dentro del data-role con `id` "`detail-page`" que se puede ver en la estructura descrita en el punto 7.3.1.

7.3.5.1. Interfaz de usuario

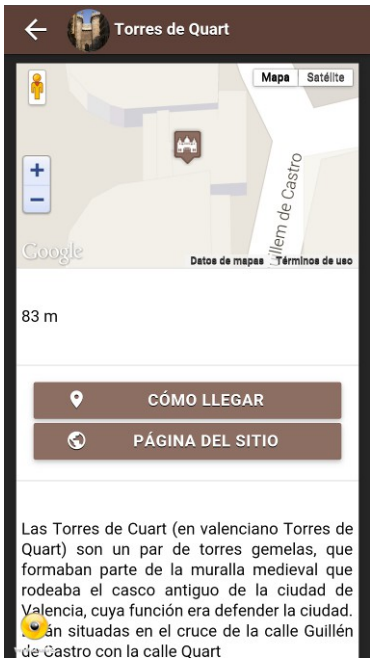


Imagen 32 Interfaz de detalle

Lo primero que vamos a observar es que esta interfaz tiene un `header` específico. Es dinámico y varía en función del POI que hayamos seleccionado. En este observamos, además de la flecha de retorno habitual, una imagen distintiva del punto y en lugar de un título genérico, vemos el nombre del mismo.

Imagen 33 Header de página de detalle

El `header` esta creado con objetos del framework Materialize a los que se ha dado un identificador.

```
<div data-role="header" data-theme="b" id="cerrar" data-position="fixed">
    <nav>
        <div class="nav-wrapper brown darken-2">
            <ul id="nav-mobile" class="left">
                <li><a href="#list-page"><i class="mdi-navigation-arrow-back"></i></a></
                li>
            </ul>
              <img id="info-imagen" src='' alt='' class='miavatar circle'>
              <span id="info-nombre" class='title'></span>
        </div>
    </nav>
```

En el resto de esta interfaz podemos distinguir 4 zonas. La zona de mapa, la de distancia, la que contiene los botones que añaden funcionalidad llamando a aplicaciones externas, y la de descripción. Para este, se ha utilizado de nuevo el objeto "tarjeta" de Materialize y se han añadidos atributos `id` en aquellos sitios en los que se añade la información dinámica del punto.

```
<div class="row">
    <div class="col s12 m5">
        <div class="card-panel white">
            <div class="section map-section">
                <div id="map-canvas2" style="height:200px; padding:0;">
                </div>
            </div>
            <div class="divider"></div>
            <div class="section">
                <p id="info-distancia"></p>
            </div>
            <div class="divider"></div>
            <div class="section" style="text-align:center;">
                <a id="location-link" class="waves-effect waves-light btn brown lighten-1
                "><i class="mdi-maps-place left"></i>Como llegar</a>
                <a id="info-link"class="waves-effect waves-light btn brown lighten-1"><i
                class="mdi-social-public left"></i>pagina del sitio</a>
            </div>
            <div class="divider"></div>
            <div class="section">
                <p id ="info-descripcion" ></p>
            </div>
        </div>
    </div>
```

7.3.5.2. Detalles de implementación

Vamos a dividir por partes. Empezaremos describiendo brevemente como añadimos el mapa. Dado que es una variante del mapa general no nos detendremos demasiado. A continuación describimos como se obtienen los datos de los POI's para representarlos, incluyendo los relacionados con el `header`. Los últimos dos puntos describe las funcionalidades de página externa y como llegar.

- Mapa:

En este caso la creación del mapa en sí es la misma que la realizada en el mapa general. Las diferencias residen en que el único objeto `LatLang` que se crea es el de la localización del POI. Este es el que centra el mapa una vez cargado. El zoom que damos al mapa también es diferente en las opciones de mapa.

```
52      var LatLang = new google.maps.LatLng(World.userLocation.latitude, World.userLocation.
            longitude);
53      var myOptions = {
54          zoom: 20,
55          center: LatLang,
56          mapTypeId: google.maps.MapTypeId.ROADMAP
57      };
```

La construcción del mapa es el mismo con la única diferencia del `<div>` contenedor que se indicia en la llamada. En este caso es el de la página de detalles con `id="map-canvas2"`.

Por último, para crear el marcador hacemos lo mismo que para el marcador de cada punto en el mapa general. La diferencia es que no hay que crear un nuevo `LatLang` ya que es el mismo que el que centra el mapa.

- Obtención de datos:

Como hemos visto en la implementación de la interfaz, se ha dado un id a todos los elementos que tienen que contener información específica del POI.

Para que esta información se represente hemos creado un manejador que se activa cada vez que se muestra la página de detalles. Este es el encargado de añadir la información tanto a los elementos del `header` como al resto de elementos contenidos en la tarjeta de detalles. Esta es su implementación

```
77  $(document).on('pageshow','#detail-page', function(){
78      generaMapaMini();
79      $('#info-imagen').attr("src", localizaciones[id].poiData.image);
80      $('#info-nombre').html(localizaciones[id].poiData.title);
81      $('#info-distancia').html((localizaciones[id].distanceToUser > 999) ? ((localizaciones[id].
            distanceToUser / 1000).toFixed(2) + " km") : (Math.round(localizaciones[id].distanceToUser) + " m
            "));
82      $('#info-descripcion').html(localizaciones[id].poiData.description);
83      $('#location-link').attr("onClick","enviarPagina('http://maps.google.com/maps?saddr="+World.
            userLocation.latitude+","+World.userLocation.longitude+"&daddr="+localizaciones[id].poiData.
            latitude+","+localizaciones[id].poiData.longitude+"&directionsmode=walking&zoom=17');");
84      var web = localizaciones[id].poiData.web;
85      $('#info-link').attr("onClick","enviarPagina('"+web+"');");
86  });
```

- Abrir página externa:

Para implementar la funcionalidad que nos permite abrir una página externa se ha optado por que esta se abra en una instancia de un navegador instalado en Android (Chrome, Firefox, navegador nativo,…). Se ha tomado esta decisión ya que esto mejora la experiencia de usuario permitiéndole acceder a todas las funcionalidades del navegador (barra de navegador, favoritos, etc.) que no tendría si abrimos el navegador del sistema dentro del modulo WebView.

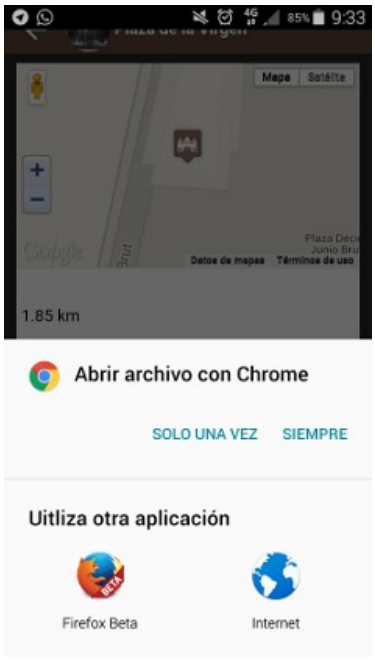


Imagen 34 Selección de navegador de al pulsar Página del sitio

Para ello ha sido necesario el uso del plugin InAppBrowser que permite que una vez que el usuario a pulsado el botón para abrir la página, este pueda seleccionar el navegador que desea utilizar.

El problema que nos encontramos es que, al igual que con la geolocalización, Wikitude no nos permite utilizar la funcionalidad de InAppBrowser mientras nos encontremos dentro del navegador de AR. Esto ya lo comentamos cuando hablamos de la implementación del la funcionalidad de Mapa.

Debido a esto hemos tenido que realizar una pequeña modificación al uso tradicional del plugin. Para empezar y como se puede ver en el código del punto inmediatamente anterior (línea 85), hemos añadido al enlace del que nos lleva a dicha página el atributo `onClick` con un método llamado `enviarPagina` que tiene como argumento la URL de la web.

Esta función lo único que hace es lo siguiente:

```
65  function enviarPagina(web){
66      document.location = 'architectsdk://action=openPage?p='+web;
67  }
```

Al retornar la URL como string mediante `document.location` con "`architectsdk://`", el plugin Wikitude automáticamente lanza `onUrlInvoke` situada en `index.js`. Esta se ejecuta fuera del ámbito del navegador AR, lo que permite realizar llamadas a las funciones de los diferentes plugins. Es conveniente no abusar de esta función ya que ralentiza el funcionamiento de la aplicación.

Como vemos nosotros le hemos dado una estructura con una variable `action` que utilizaremos para definir para qué se está llamando a `onUrlInvoke` y otra llamada `p` para indicar la página.

La implementación de `onUrlInvoke` es la siguiente:

```
77      onUrlInvoke: function (url) {
78          if ( 'openPage' == url.substring(22,30) ) {
79              app.openPage(url.substring(33));
80
81          } else
82            alert(url + "not handled");
83      },
84
85      locationChanged: function locationChangedFn(lat, lon, alt, acc) {
86          alert("lat: "+lat+", long: "+lon );
87      }
88      // --- End Wikitude Plugin ---
89  };
```

La ejecución transcurre de la forma que vemos. En función del contenido de la URL que le hemos indicado en `document.location`. Para ello solo tenemos que compararla con esta (o con la parte que nos interesa como es nuestro caso) y en función de si contiene el valor esperado realizar la operación que se desea.

En nuestro caso estamos usando la url para indicar también la página que tiene que abrir en el navegador externo.

A continuación se llama a una función llamada `openPage` con la subcadena de caracteres que contiene la url.

```
52      openPage: function(web){
53          var ref = window.open(web, '_system', 'location=yes');
54          ref.addEventListener('loadstart', function(event) { alert('start: ' + event.url); });
55          ref.addEventListener('loadstop', function(event) { alert('stop: ' + event.url); });
56          ref.addEventListener('loaderror', function(event) { alert('error: ' + event.message);
57          ref.addEventListener('exit', function(event) { alert(event.type); });
58      },
```

Esta llama a `window.open` con la variable `'_system'` para que utilice los navegadores instalados en el sistema. Además se añaden diferentes manejadores para los distintos eventos que se producen al interactuar con el navegador.

- Cómo llegar:

Como podemos ver la implementación de esta funcionalidad es igual que la anterior. La única diferencia y la que marcará como el sistema interpreta la tarea a realizar es la formación de la URL. En la línea 84 podemos ver que esta se forma llamando a la web de Google Maps (http://maps.google.com/maps) y se le pasan cuatro parámetros. Estos parámetros son los siguientes:

- Saddr: Punto de origen. En nuestro caso la longitud y latitud del usuario.
- Daddr: Punto de destino. Latitud y longitud del POI.

- Directionsmode: Modo para la ruta (a pie, en coche, en transporte). Hemos seleccionado a pie por defecto.
- Zoom: zoom para el mapa.

Al pulsar el botón del a interfaz, el sistema nos muestra una pantalla en la que nos indica con que aplicación queremos abrirlo. Si seleccionamos Google Maps nos situará los puntos origen y destino en sus respectivos lugar y nos indicará la ruta a pie. Podemos ver el funcionamiento en la siguientes imágenes.

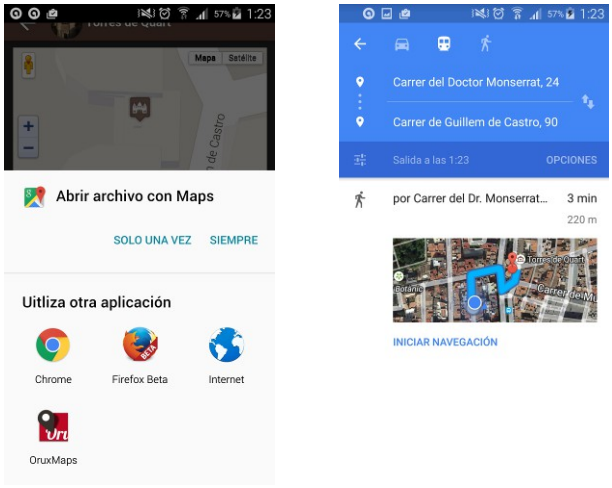


Imagen 35 Selección de aplicación para función Como llegar y apertura en Google Maps

8. Evaluación y pruebas

Para la realización de pruebas de usuario se decidió utilizar un grupo en Google Groups para mantener contacto con los testers. De este modo, además de poder utilizarlo como herramienta de feedback, nos ha permitido hacerles llegar la aplicación así como un formulario para que valoren la misma.

8.1. Pruebas de usuario

Lo primero que hemos podido hacer gracias a consola para desarrolladores de Android y el grupo de Google es distribuir la aplicación para al descarga por parte de los diferentes usuarios.

Para ello hemos utilizado la distribución en beta-testing que nos proporciona Developer Console.

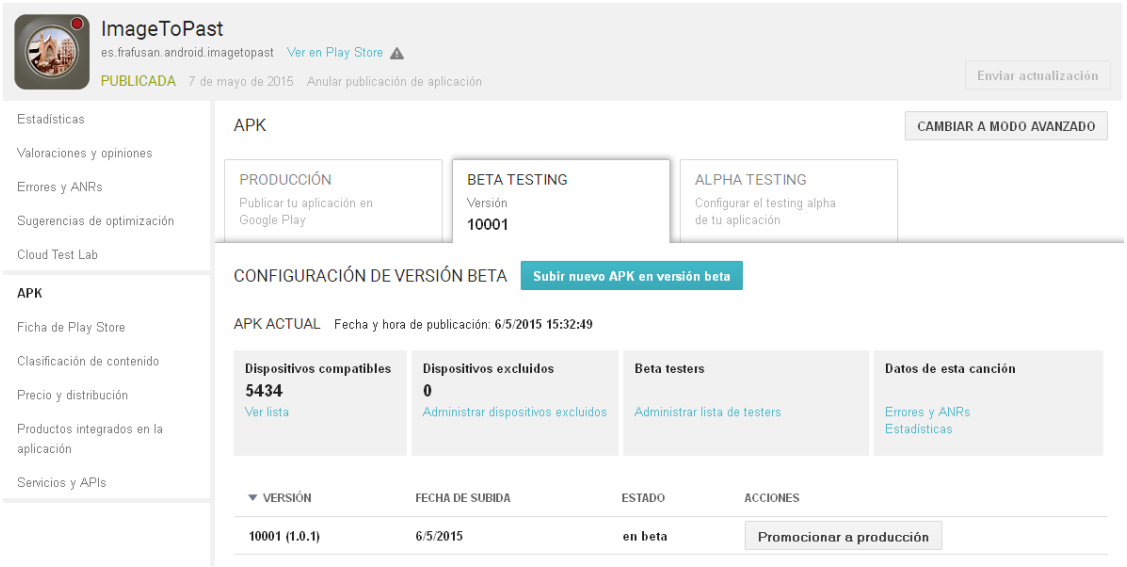


Imagen 36 Consola de Google Play para desarrolladores

En ella podemos ir viendo las diferentes APK's subidos, además nos da la opción de promocionar una a producción cuando consideremos que está listo.

Además nos permite indicarle un grupo de Google Groups como lista de testers que podrán descargar la aplicación en este momento.

Imagen 37 Formulario para añadir gurpos de testers

8.1.1. Formulario de satisfacción

La segunda ventaja ha sido la de poder distribuir un formulario de satisfacción a todos lo miembros del grupo de testers desde Google Groups.

Imagen 38 Ejemplo de formulario de aceptación

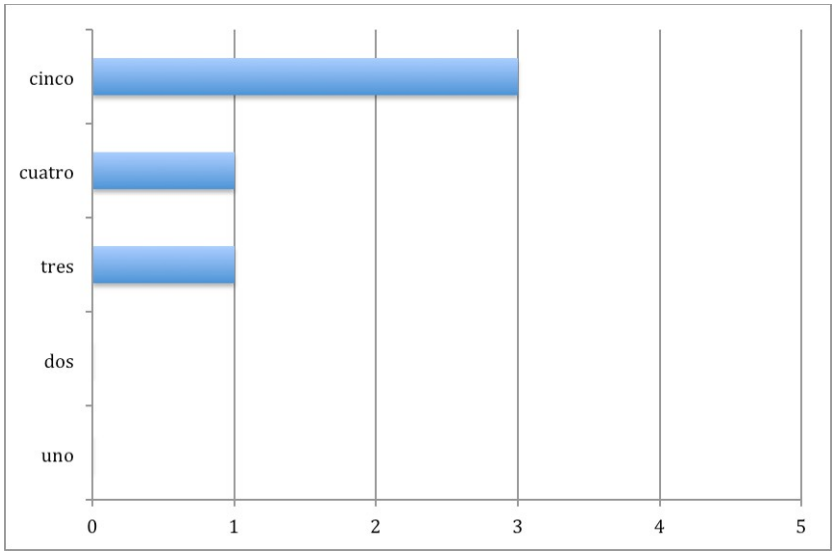
8.1.2. Resultados de las pruebas de satisfacción

Las pruebas han sido realizadas por medio de 5 personas. Las características demográficas de los usuarios que han realizado las pruebas responde a los siguientes conjuntos:

- Entre 25-35 años:
  - 1 hombre con conocimientos avanzados de informática y manejo de dispositivos móviles.
  - 1 mujer con conocimientos avanzados de informática y manejo de dispositivos móviles.
- Entre 35-50 años:
  - 1 mujer con conocimientos medios de informática y nivel de usuario en el manejo de dispositivos móviles.
  - 1 hombre con conocimientos básicos de informática y nivel de usuario en el manejo de dispositivos móviles.
- Mas de 50 años
  - 1 hombre con conocimientos básicos de informática y nivel de usuario en el manejo de dispositivos móviles

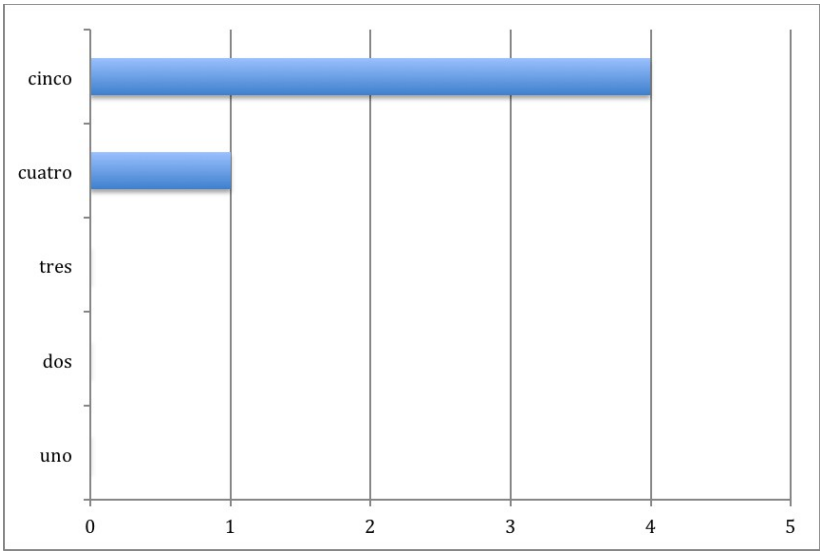
El resultado de las diferentes cuestiones planteadas es el siguiente:

- Valore de 1 a 5 la facilidad de aprendizaje de la aplicación



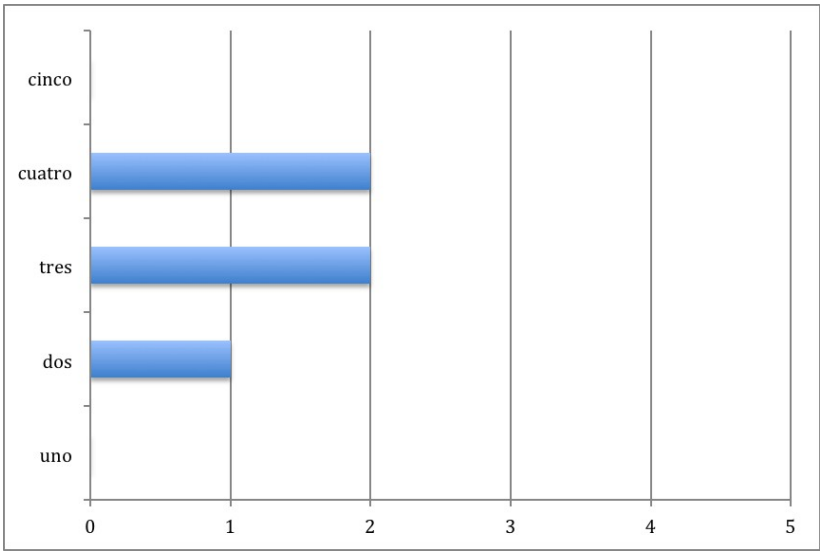
Se puede apreciar que en general los usuarios han considerado que la aplicación tiene un periodo y una facilidad de aprendizaje bastante apreciable.

- Valore de 1 a 5 el aspecto de la aplicación



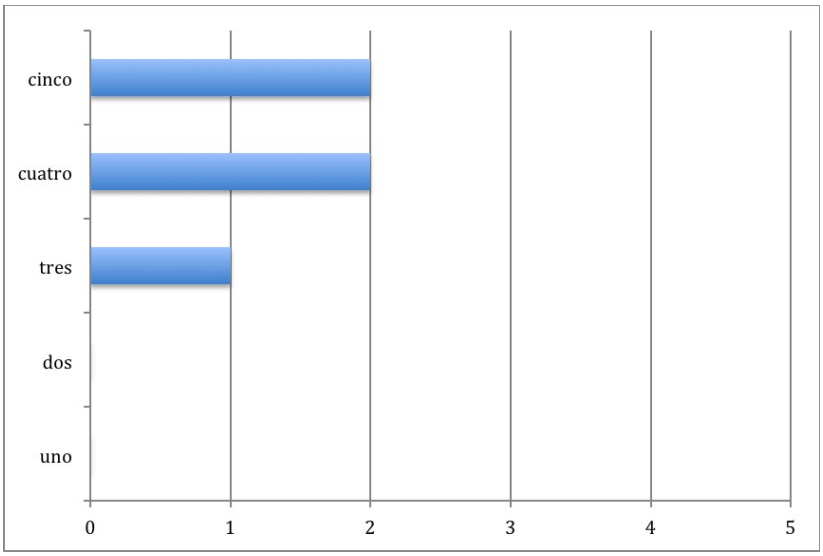
El aspecto dado a la aplicación mediante los elementos gráficos del framework Materialize han sido muy bien acogidos por los usuarios. El hecho de que la aplicación tenga un aspecto similar a la tendencia actual, en contraposición con las primeras versiones en las que se utilizaba el aspecto básico de las aplicaciones de jQuery Mobile ha sido bien recibido.

- Valore de 1 a 5 la fluidez de la aplicación



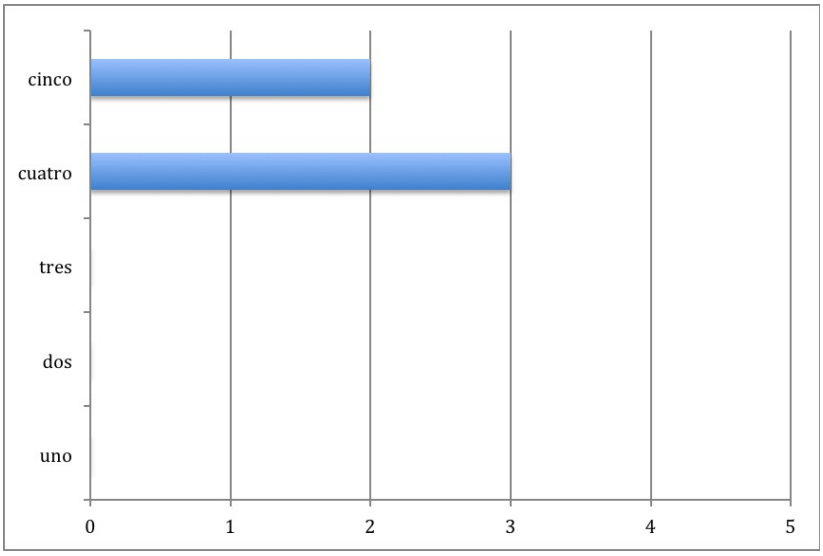
Este aspecto ha sido sin duda el peor valorado entre los usuarios. Los problemas presentados por el plugin, que en muchos casos repintaba o bloqueba la interfaz de cámara daban la sensación a los usuarios de que la aplicación se atascaba.

- Valore de 1 a 5 la utilidad de la aplicación dentro de su contexto



La aplicación tiene buena acogida en el contexto de guía turistica interactiva dentro del ámbito planteado.

- Valoración global de la aplicación



Por último mostramos una estadística de valoración global de la aplicación cuya tendencia indica que la misma ha tenido buena acogida entre los usuarios

9. Conclusiones

El proyecto desarrollado en el presente trabajo de fin de grado nos ha permitido profundizar tanto en el desarrollo de aplicaciones para dispositivos móviles como en el mundo de la realidad aumentada. El primero de estos dos campos está en pleno esplendor gracias a la gran difusión de smartphones en el mundo. El segundo se encuentra aun en una fase temprana tanto en el uso como en el desarrollo del mismo.

En cuanto al desarrollo para Android, gracias al proyecto hemos podido aprender las diferencias entre las aplicaciones híbridas y nativas. Hemos podido observar que son dos desarrollos de características muy diferentes. Habernos decantado por el primero se basó en la experiencia y conocimientos previos en tecnologías Web y el tiempo disponible para el mismo. Sin embargo, lo visto de las aplicaciones nativas nos motiva para seguir indagando en el mundo del desarrollo para plataformas móviles.

Por otro lado, la tecnología de realidad aumentada nos ha parecido realmente prometedora. Si en el mercado proliferan y se abaratan dispositivos fácilmente portables como son las Google Glass, pensamos que este tipo de tecnología sufriría un auge. Es evidente en cualquier caso que aún le queda camino por recorrer.

Otro aspecto que hemos podido apreciar en primera persona gracias a este proyecto es la velocidad a la que avanza la tecnología relacionada con los móviles. Durante el desarrollo de la aplicación hemos sufrido la actualización de Android a su versión Lollipop y posteriores actualizaciones de la misa. La actualización del Framework de PhoneGap, que incluso modificó el sistema de compilación y nos obligó a cambiar el IDE con el que desplegar las aplicaciones. A nivel personal, esto ha sido toda una experiencia ya que el campo de desarrollo profesional que desempeño no incorpora los cambios con tanta velocidad.

10. Trabajos futuros

En este capítulo se presentan una serie de ideas que podrían implementarse con la idea de mejorar la experiencia del usuario al interactuar con la aplicación. Así mismo se presenta una mejora necesaria en el caso de un crecimiento exponencial de los POI's.

10.1. Limitador de distancia

Actualmente para evitar la aglomeración de POI's en la interfaz de cámara la distancia máxima a la que debe estar el mimo para poder ver su representación es de 150 metros. Puede que en algún momento el usuario desee bien expandir este radio o bien reducirlo con el fin de ver más o menos marcadores. Para ello en una pantalla de configuración el usuario tendría la opción de definir el radio de acción de la aplicación. Esto quiere decir que solo se mostrarían los POI's que estuvieran en una distancia determinada. El método de entrada podría ser mediante un botón deslizante.

10.2. Diferenciar entre distintos puntos

Crear marcadores distintos en función del tipo de POI que estamos viendo (palacios, museos, catedrales, mercados, plazas,...).

Del mismo modo, en la pantalla de configuración mencionada en el punto anterior, el usuario podría definir qué tipos de POI quiere visualizar haciendo que el resto se ocultara en el resto de interfaces.

10.3. Indicaciones en la vista de realidad aumentada

Cuando el usuario estuviera dentro a una distancia máxima determinada del POI objetivo, pulsando un botón sobre la interfaz de cámara, aparecerían sobre la misma, flechas indicativas para llegar al punto. Estas cambiarían conforme fuera avanzando por la calle.

10.4. Capa de persistencia

Actualmente los datos de los diferentes POI se almacenan en un fichero con estructura JSON. El hecho de haber elegido esto en contraposición a un base de datos viene determinada por el volumen de datos que se maneja en la aplicación actual y porque el rendimiento de la misma es mayor de esta forma.

En el caso de extender la aplicación (varias ciudades, países,…) el manejo de la información sería inviable tal y como se realiza actualmente. En este caso sería interesante utilizar una capa de persistencia más potente y manejable mediante una base de datos.

11. Bibliografía

[1] Letelier Torres, P. *Introducción al proceso de desarrollo software*. Consultado el 1 de junio de 2015, en https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CCEQFjAA&url=http%3A%2F%2Fwww.dsic.upv.es%2Fasignaturas%2Ffacultad%2Flsi%2Fdoc%2FIntroduccionProcesoSW.doc&ei=Hz2SVYjZMsfaU4LvrJAJ&usg=AFQjCNGAyOzvxjChM6izsIaI_eLoQhtk2Q

[2] Almunia, P. (2011). ***Ciclo de vida en el Desarrollo de Software, segunda parte***. Consultado el 1 de junio de 2015, en http://blog.iedge.eu/tecnologia-sistemas-informacion/desarrollo/pablo-almunia-ciclo-de-vida-en-el-desarrollo-de-software-segunda-parte/.

[3] Galo Fariño, R. (2011). ***Modelo Espiral de un proyecto de desarrollo de software***. Consultado el 1 de junio de 2015, en http://www.ojovisual.net/galofarino/modeloespiral.pdf

[4] Rodríguez, J. (2008). ***Metodología del desarrollo de software. El modelo en V o de cuatro niveles***. Consultado el 1 de junio de 2015, en http://www.iiia.csic.es/udt/es/blog/jrodriguez/2008/metodologia-desarrollo-sotware-modelo-en-v-o-cuatro-niveles

[5] W3C (2014). ***HTML. A vocabulary and associated APIs for HTML and XHML***. Consultado el 2 de junio de 2015, en http://www.w3.org/TR/html5/

[6] W3C (s.f). ***HTML & CSS***. Consultado el 2 de junio de 2015, en http://www.w3.org/standards/webdesign/htmlcss

[7] Mozilla Developer Network (2015). ***HTML5***. Consultado el 2 de junio de 2015, en https://developer.mozilla.org/es/docs/HTML/HTML5

[8] GENBETA:DEV (2015). ***Eclipse IDE***. Consultado el 4 de junio de 2015, en http://www.genbetadev.com/herramientas/eclipse-ide

[9] Wikipedia (2015). ***Eclipse (software)***. Consultado el 4 de junio de 2015, en http://es.wikipedia.org/wiki/Eclipse_%28software%29#Caracter.C3.ADsticas

[10] Stackoverflow (2015). ***Phonegap- basic architecture***. Consultado el 6 de junio de 2015, en http://stackoverflow.com/questions/26648378/phonegap-basic-architecture

[11] García Echegaray, B. (2014). ***Introducción a Phonegap: webview, plugins y herramientas***. Consultado el 7 de junio de 2015, en http://blog.garciaechegaray.com/2014/10/18/phonegap-workshop.html

[12] Jiménez, X. (2013) ***19 aplicaciones de realidad aumentada***. Consultado el 8 de junio de 2015, en http://blogs.elperiodico.com/masdigital/afondo/19-aplicaciones-de-realidad-aumentada

[13] Santero, J (2013). ***Las mejores aplicaciones de realidad aumentada para Android***. Consultado el 8 de junio de 2015, en http://sevilla.abc.es/mobility/las_mejores_app/android/las-mejores-app-android/las-mejores-aplicaciones-de-realidad-aumentada-para-android/

Glosario de acrónimos

- **POI**: (del ingles *Ponit Of interest*) Punto de interés.
- **UML:** (por sus siglas en inglés, *Unified Modeling Language*) Lenguaje de Modelado Unificado
- **IU:** Interfaz de Usuario
- **IDE:** (en inglés *Integrated development environment*) Entorno de Desarrollo Integrado.
- **GNU/GPL:** (en inglés *GNU/General Public Lincense*) Un tipo de licencia de uso público. El acrónimo GNU es recursivo y deriva de *GNU's Not Unix*.
- **JVM:** (en ingles *Java Virtual Machine*) Máquina Virtual Java.
- **HTML:** (en ingles *HyperText Markup Language*) Lenguaje de marcas de hipertexto.
- **CSS:** (en inglés de C*ascading Style Sheets*) Hoja de estilo en cascada.
- **GPS:** (en inglés *Global Positioning System*) Sistema de posicionamiento global.
- **SDK:** (en inglés *Software Development Kit*) Kit de desarrollo de software.
- **AR:** (en inglés *Augmented Reality*) Realidad aumentada.
- **SVG:** (en inglés *Scalable Vector Graphics*) Gráficos vectoriales redimensionables.
- **API:** (en inglés *Application Programming Interface*) Interfaz de programación de aplicaciones.
- **W3C:** (en inglés *World Wide Web Consortium*) consorcio internacional que realiza recomendaciones para la World Wide Web.
- **OO:** Orientado a Objetos.
- **DOM:** (en inglés *Document Object Model*) Modelo de objetos del documento.
- **AJAX:** (en inglés *Asynchronous JavaScript And XML*) JavaScript asíncrono y XML
- **ETL:** (en inglés *Extract, Transform and Load*) Extracción, transformación y carga.

- **ADT:** (en inglés *Android Developer Tools*) Herramientas de desarrollo de Android.
- **ANT:** (en inglés *Another Neat Tool*) Otra ingeniosa herramienta. Herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción.
- **MVC:** (en inglés *Model View Controller*) Modelo Vista Controlador.
- **SO:** Sistema operativo.
- **JSON:** (en inglés *JavaScript Object Notation*) Es un formato ligero para el intercambio de datos.
- **URL:** (en inglés *Uniform Resource Locator*) Localizador de recursos uniforme.

imagetopast

Manual de usuario de ImageToPast

Aplicación Android realizada en el desarrollo del Trabajo Final de Grado en la Escuela Técnica Superior de Informática (Universidad Politécnica de Valencia).

1. Descripción de la aplicación

La aplicación ImageToPast es una aplicación para smartphones y tablets Android enfocada a la versión 5 o superiores de este sistema operativo. Es una aplicación híbrida construida mediante PhoneGap. Hace uso de la tecnología de realidad aumentada proporcionada por Wikitude para mostrar información al usuario por medio de la cámara del dispositivo.

Es una guía turística interactiva centrada en la información histórica de puntos de interés. Mediante la geolocalización y la cámara proporciona al usuario información en tiempo real de lo que le rodea.

Además permite conocer la situación del los puntos de interés mediante un mapa y obtener más información de dichos puntos así como indicaciones para llegar a estos.

2. Descarga de la aplicación

Dado que la aplicación se encuentra en proceso de testeo solo los miembros del grupo ImageToPast de Google Groups pueden acceder a la descarga de la misma. Para ello se les hace llegar un enlace con una URL para participar como tester. Una vez aceptada puede acceder a la descarga de la aplicación en Goole Play.

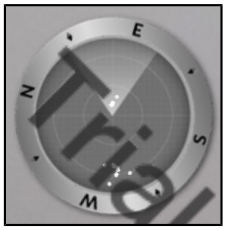
La instalación de la aplicación es automática. En determinado momento se pide al usuario la aceptación de permisos para que la aplicación pueda acceder a ciertos servicios como la geolocalización. Debemos pulsar aceptar para continuar con la instalación.

Cámara y puntos de interés

Nada más iniciar la aplicación esta accede a la vista de la cámara del dispositivo. En esta podemos observar diferentes objetos. Estos varían en función de la interacción del usuario con la aplicación.

La primera vista tiene los componentes del radar, los marcadores en caso de estar enfocando algún punto de interés y un botón en la esquina inferior derecha.

3.1 Radar



Dentro del componente de radar, que a su vez actúa como brújula indicándonos los puntos cardinales, podemos ver la posición de los distintos lugares marcados como puntos blancos. Esto nos ayuda a localizar los puntos para enfocar con la cámara.

3.2 Botones

Si pulsamos sobre el botón con el símbolo “+” se despliegan los botones que nos dan acceso a las pantallas de mapas (botón naranja) y listado (botón amarillo).

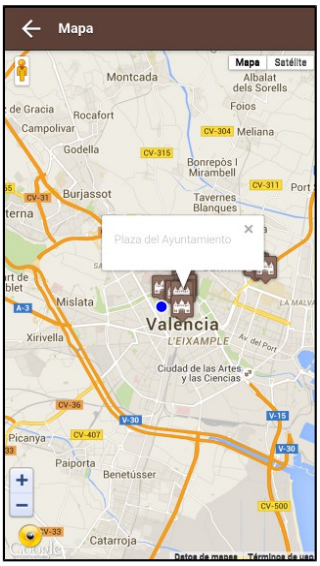
3.3 Tarjeta

Al pulsar sobre un marcador la aplicación nos muestra una tarjeta en la que observamos diferentes imágenes del punto de interés. Las imágenes se van sucediendo una detrás de otra. Sin embargo el usuario podrá pasar de una a otra deslizando el dedo por las mismas de izquierda a derecha o viceversa.

Al pulsar sobre el botón “MAS INFORMACIÓN” se accede a la pantalla de detalles del punto de interés.

Por último para que la tarjeta desaparezca basta con pulsar sobre cualquier parte de la pantalla que no corresponda a la tarjeta, sobre otro marcador o sobre el mismo.

3. Mapa



Para acceder al mapa basta con pulsar el botón naranja de los dos desplegados por el botón “+” de la página de la cámara.

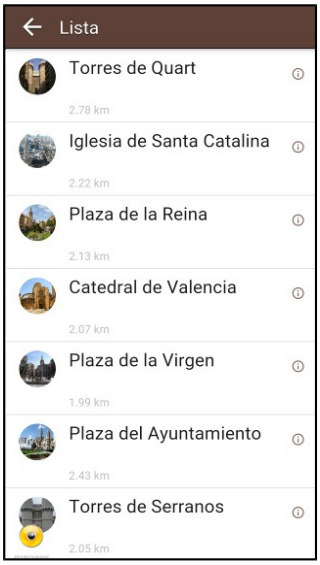
En esta pantalla podemos observar un mapa centrado en nuestra posición la cual está resaltada mediante un punto azul.

A nuestro alrededor vemos los marcadores de los diferentes puntos de interés que nos rodean. El mapa es una extensión de Google Maps y por lo tanto nos permite utilizar sus opciones como la vista de satélite y demás. Podemos hacer zoom sobre el mapa.

Si pulsamos en alguno de los marcadores la aplicación nos muestra un cartel indicándonos el nombre del punto de interés seleccionado.

Para volver a la página de cámara solo hay que pulsar el botón con el icono de flecha que ha en la cabecera (arriba izquierda).

4. Listado

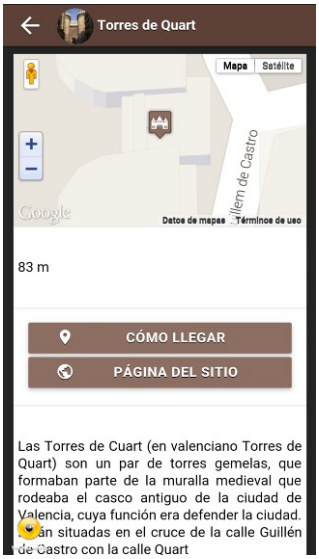


Para acceder al mapa basta con pulsar el botón amarillo de los dos desplegados por el botón “+” de la página de la cámara.

La pantalla presenta un listado con todos los puntos de interés ordenados por distancia al usuario. Cada elemento del listado consta del nombre, la distancia y una imagen identificativa. A la derecha veremos el icono que al pulsar nos permite acceder a la pantalla de detalles del punto de interés seleccionado.

Al igual que en la pantalla de mapas, si pulsamos el botón con el icono de flecha de la cabecera, volveremos a la cámara.

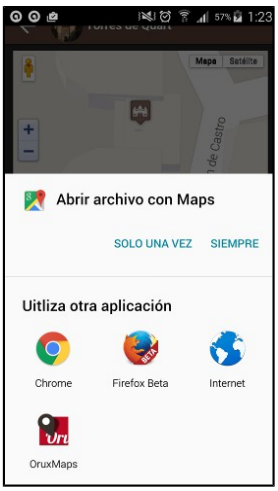
5. Detalle de punto de interés

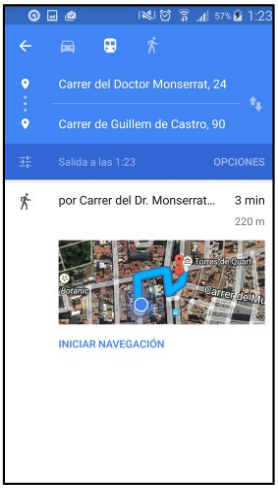


El acceso a esta pantalla se realiza de dos formas. La primera es desde el botón “MAS INFORMACIÓN” de la tarjeta mostrada en la cámara al seleccionar un punto de interés. La segunda desde el listado, pulsando en el icono de la derecha de uno de los elementos del listado.

La pantalla nos muestra un pequeño mapa con la localización del punto de interés seguido por la distancia al mismo. A continuación nos muestra dos botones con las opciones “COMO LLEGAR” y “PAGINA DEL SITIO”. Por último vemos un texto con información del lugar. Si deslizamos el dedo de arriba hacia abajo, en caso de que la información se extienda, la tarjeta se desplaza en la misma dirección.

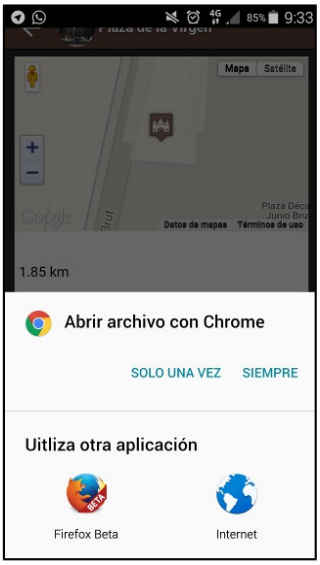
6. 1 Como llegar





Al pulsar sobre “COMO LLEGAR” se nos muestra, en función de las aplicaciones instaladas en nuestro dispositivo, una pantalla para seleccionar cual de estas queremos que nos de la indicaciones para llegar al lugar. Se muestra un ejemplo con Google Maps.

6.2 Pagina del sitio



Si pulsamos “PAGINA DEL SITIO” sucederá algo parecido, con la salvedad que las aplicaciones solo corresponderán a navegadores web. Si seleccionamos un de ellos, nos abrirá la página relacionada.

Como en el resto de pantallas el botón con icono de flecha nos devuelve a la página de la que procedemos.