Valencia4Foreginers Mobile

Francisco Rodríguez Navarro
Máster en Ingeniería Informática
Desarrollo de aplicaciones web

Ignasi Lorente Puchades
César Pablo Córcoles Briongos

Enero 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Valencia4Foreginers Mobile*
Nombre del autor:	*Francisco Rodríguez Navarro*
Nombre del consultor/a:	*Ignasi Lorente Puchades*
Nombre del PRA:	*César Pablo Córcoles Briongos*
Fecha de entrega (mm/aaaa):	01/2018
Titulación::	*Máster en Ingeniería Informática*
Área del Trabajo Final:	*Desarrollo de aplicaciones web*
Idioma del trabajo:	*Castellano*
Palabras clave	*Información, lugares,audioguia, usuarios*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El objetivo del proyecto es crear una aplicación web híbrida donde el usuario pueda consultar información detallada de los lugares de interés de la ciudad de València, la aplicación irá dirigida principalmente a gente extranjera por lo que la lengua utilizada es el inglés.

Los lugares de interés se mostrarán de dos formas, listado y mapa, este segundo dará al usuario algunas funcionalidades extras, como geolocalización, radio de cercanía configurable de los lugares más cercanos y cálculo de la ruta hasta el lugar seleccionado. La información detallada de cada lugar constará de imágenes acompañadas por texto o audio a elección del usuario.

La idea de desarrollar esta aplicación es debido a diferentes factores, un factor es el casi inexistente número de aplicaciones de temática similar que funcionen de manera eficiente y estable, otro factor es las deficiente interfaz de las aplicaciones, todo esto lleva a pensar en el desarrollo de una aplicación que mejore notablemente estos puntos.

Para conseguir un funcionamiento eficiente y que la aplicación se adapte perfectamente a todos los tipos de dispositivos móviles se utiliza el *framework* Ionic 3 ([1]), una API desarrollada en NodeJS ([3]), y la API de *Google Maps*.

Después de haber realizado un análisis exhaustivo de las aplicaciones que tratan una temática similar, se llega a la conclusión de que sería realmente útil contar con una aplicación fluida, dinámica y que sea más atractiva para el turismo que quiere visitar València.

Abstract (in English, 250 words or less):

The goal of the project is to create a hybrid web application, where the user can consult detailed information of the sights of València city, the application is thought to foreign people so the used language is the English. The sights is going to show in two ways, list and map, the second one will give to the user some extra functionalities like geolocation, configurable proximity radio of the closest sights and route calculation to the selected sight. The detailed information of each sight will consist of images accompanied by text or audio at the user's choice.

The idea to develop this application is due to different factors, one of this factors is the fact that there are not a lot of applications of the same topic that work in a stable and efficient way, another factor is the poor interface of applications, all this leads to think to develop an application which improves remarkably this points.

To achieve an efficiently performance and that the interface of application fits all mobile devices is going to use Ionic 3 framework ([1]), NodeJs ([3]) to develop an API and use of Google Maps API.

After have carried out and exhaustive analysis of applications which have similar topic, we reach the conclusion that would be really useful have a fluent, dynamic and attractive application to the tourist that want to visit València.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Hoy en día y sobre todo en nuestro país, el turismo es el motor principal de nuestra economía y año tras año va alcanzando más importancia dentro de este sector. Entidades y ayuntamientos están creando aplicaciones que pueden hacer más fácil y agradable la visita de los turistas a las ciudades.

En la ciudad de València existen un variado número de aplicaciones, las cuales ofrecen al turista diferentes funcionalidades como puede ser información sobre los lugares, mapa de los lugares, fotos, etc. Pero el funcionamiento y la calidad que tienen estas aplicaciones hacen que el uso de estas no sea del todo satisfactorio, ya que tienen *bugs*, falta de fluidez, interfaz anticuada y obsoleta, haciéndola muy poco atractiva al usuario. ¿De qué sirve tener una aplicación si nadie la utiliza? A partir de esta pregunta empieza a ganar fuerza, la necesidad de crear una aplicación que solvente todos estos aspectos. Se buscan tecnologías más punteras y que mejor rendimiento dan en dispositivos móviles para el desarrollo de esta aplicación.

Lo que aporta la aplicación respecto a sus competidoras, es una interfaz más intuitiva y atractiva, con un contenido más dinámico y mucho más fluido, ya que si se actualiza cierta información está se descargará al móvil en segundo plano sin que el usuario se vea afectado. También se utilizará de forma mucho más eficiente *Google Maps* ahorrando tiempos de carga.

Respecto al contenido, la aplicación mostrará al usuario los lugares de interés (museos, monumentos, catedrales, iglesias…). Estos lugares podremos visualizarlos mediante un mapa donde gracias a la geolocalización nos da la posibilidad de acotar el radio de búsqueda para que sólo nos muestre los lugares más cercanos del lugar donde nos encontramos. Por otra parte también nos dará la posibilidad de buscar la ruta más cercana para llegar. Otra forma de visualizar los lugares de interés sería mediante un listado.

Una vez tengamos el lugar de interés seleccionado, podremos acceder a una ficha que contará con audioguia, texto y fotos sobre el lugar. Gracias a todas estas funcionalidades se consigue una aplicación mucho más atractiva, ágil y estable comparada con sus competidoras.

1.2. Objetivos del Trabajo

Los objetivos que se intentan conseguir son los siguientes:

- Dar la posibilidad al usuario de visualizar los lugares más cercanos a su posición actual, pudiendo delimitar el radio de acción.

- A través de una interfaz atractiva y renovada, atraer a un número mayor de usuarios potenciales.

- Mostrar la información de cada lugar de manera interactiva, dando la opción al usuario de escuchar un audioguía o leer un texto, dicha información hará referencia a las fotos numeradas incluidas en la ficha.

1.3. Enfoque y método seguido

Después de haber analizado las tecnologías existentes y sabiendo los puntos a destacar en la aplicación, se ha decido utilizar las siguientes tecnologías:

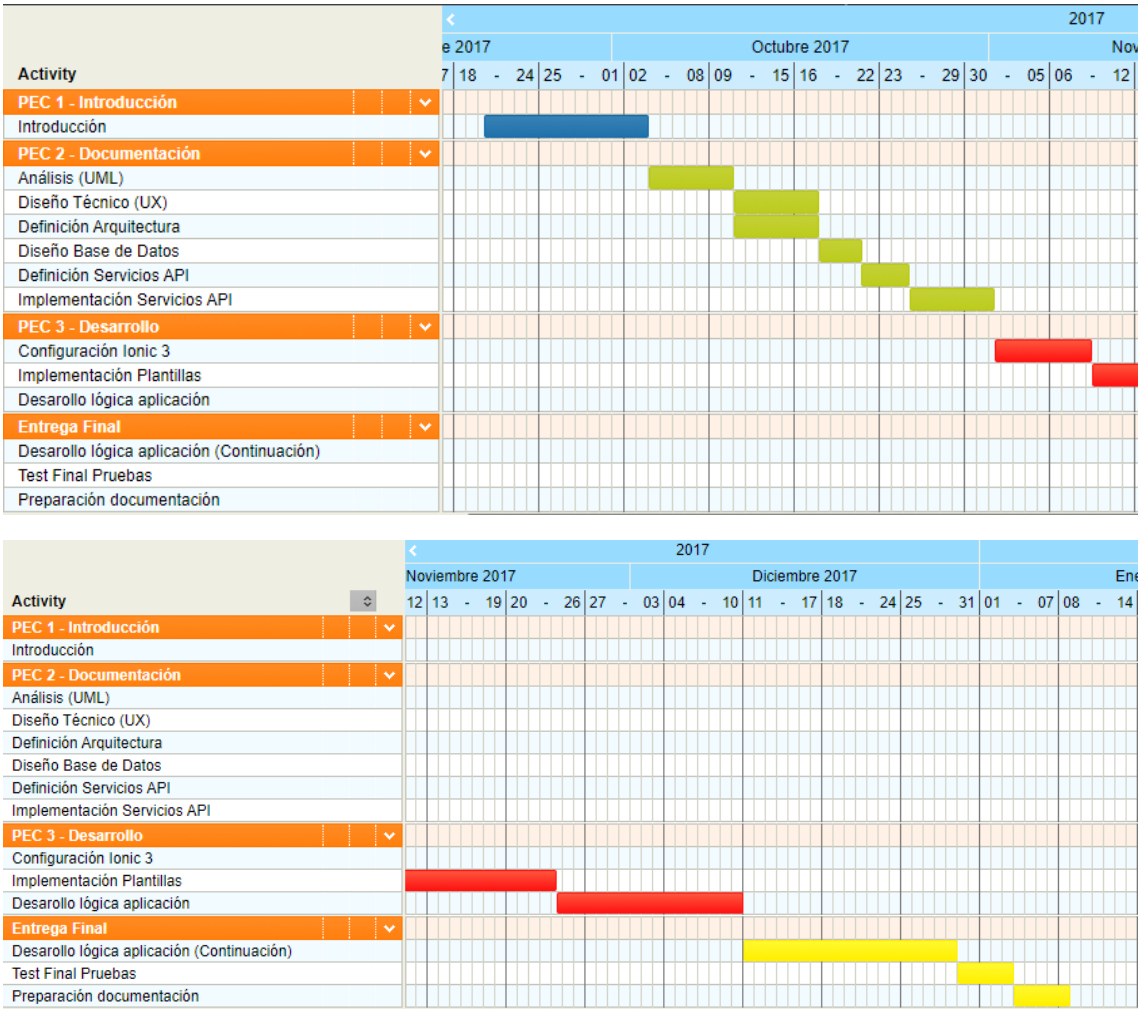
La parte frontal de la aplicación se va a desarrollar mediante un *framework* llamado Ionic 3 ([1]), el cual es un *framework* específico para aplicaciones híbridas, ya que mediante Bootstrap ([2]) se adapta perfectamente la aplicación a los dispositivos móviles, aparte hace uso de Angular 4 con el cual se consigue un rendimiento elevado gracias a la potencia que nos brinda la programación en javaScript y el uso de *SPA (Single Page Application)*, todo esto nos dará una estabilidad y una sensación de fluidez que hará que nuestra aplicación destaque por encima de la competencia. En la parte *back* tendremos corriendo una base de datos mySql, y para poder servir toda esta información al *front* se va a desarrollar una API basada en *nodeJS* ([3]), gracias al cual conseguiremos un muy buen rendimiento a la hora de realizar peticiones a la base de datos. También se creará una base de datos local (SQLite ([4])), la cual utilizaremos en caso que el usuario no disponga de conexión a internet, ya que esto es una situación muy común cuando un turista viaja a otro país.

Hay que destacar que mediante el uso de Ionic 3, la aplicación la podremos compilar para diferentes sistemas operativos móviles, ya sea Android, Ios o Windows Phone, por lo que nos ahorraremos mucho tiempo de desarrollo y aparte podremos acceder a un número mucho mayor de usuarios potenciales.

1.4. Planificación del Trabajo

- Diagrama de Gantt

Ilustración 1 Diagrama Gantt



- Tabla resumen

Tarea	Fecha Inicio	Fecha Fin	Descripción
PEC1 - Introducción	20/09/2017	03/10/2017	
Definición proyecto	20/09/2017	03/10/2017	Definición objetivos, tecnologías y planificación del trabajo
PEC2 - Documentación	04/10/2017	01/11/2017	
Análisis (UML)	04/10/2017	10/10/2017	Definición de los requisitos, diagrama de clases y casos de uso de la aplicación

Diseño Técnico (UX)	11/10/2017	17/10/2017	Creación de borradores de las plantillas a utilizar en la aplicación
Def. Arquitectura	11/10/2017	17/10/2017	Diseño de la arquitectura web de la aplicación (MVW)
Diseño Base de Datos	17/10/2017	21/10/2017	Creación y diseño de la base de datos y
Definición Servicios API	21/10/2017	25/10/2017	Se van a definir los servicios a utilizar en la aplicación
Creación Servicios API	25/10/2017	01/11/2017	Se van a programar la lógica de los servicios
PEC3 - Desarrollo	02/11/2017	10/12/2017	
Configuración Ionic3	02/11/2017	09/11/2017	Configuración del *framework* Ionic3 (librerías,repositorios…)
Implementación plantillas	09/11/2017	24/12/2017	Se integrarán las plantillas en la aplicación
Desarrollo lógica aplicación	24/12/2017	10/12/2017	Se iniciará la programación de la lógica de la aplicación
Entrega Final	11/12/2017	08/01/2018	
Desarrollo lógica aplicación (Continuación)	11/12/2017	29/12/2017	Se programará toda la lógica de la aplicación
Test Final Pruebas	29/12/2017	03/01/2018	Realización de la batería final de pruebas
Preparación documentación	03/01/2018	08/01/2018	Preparación de los entregables para la presentación

2. Diseño de la aplicación

2.1. Diagramas UML

En este apartado se detalla la estructura, funcionalidades y el aspecto de la aplicación, para ello se han creado borradores (mock up), diagramas UML en los cuales se incluye casos de uso y diagramas de clase.

En esta aplicación el **perfil del usuario** va a ser único ya que contaremos con un solo perfil de usuario, ya que durante el proceso de diseño se desestimó (por ahora) la posibilidad de tener varios perfiles de usuario con el objetivo de darle más sencillez de uso a la aplicación y eliminar barreras como puede ser la obligación de registrarse para utilizar ciertas funcionalidades, aunque no se descarta en un futuro según cómo evolucione la aplicación aplicar más perfilados al usuario.

El perfil de usuario el cual no necesita registro le permite hacer de uso de las siguientes **funcionalidades**.

Código	Descripción
CU_1	Ver las diferentes opciones que la aplicación le ofrece al usuario.
CU_2	Listar todos los lugares y monumentos disponibles en la aplicación
CU_3	Modificar el radio de lejanía del listado de los lugares disponibles.
CU_4	Visualizar los lugares ubicados en un mapa interactivo
CU_5	Modificar el radio de lejanía en el mapa interactivo
CU_6	Reproducir la audioguía de un monumento en cuestión
CU_7	Visualizar slider con fotos del monumento
CU_8	Visualizar la información del lugar en formato texto
CU_9	Acceder a google maps para mostrar la ruta más rápida al monumento.
CU_10	Visualizar el listado de las rutas predefinidas disponibles
CU_11	Ver la ruta predefinida en el mapa
CU_12	Visualizar galería de imágenes con los lugares que incluye la ruta predefinida

2.1.1. Diagramas y fichas casos de uso

2.1.1.1. Diagramas actividad

2.1.1.1.1. Diagrama actividad listado rutas predefinidas

Este diagrama representa el flujo que el usuario sigue para mostrar el listado de rutas predefinidas, este flujo cubre los casos de uso CU_11, CU_12

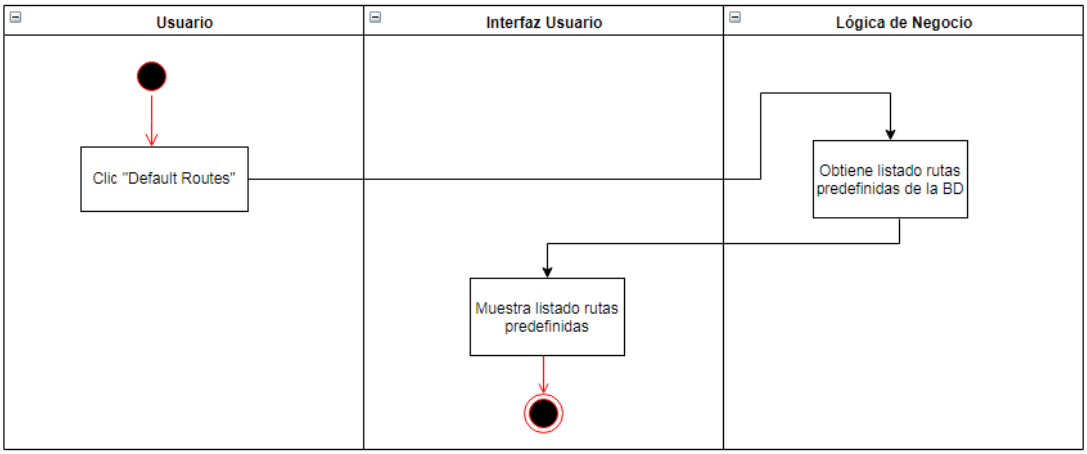


Ilustración 2 Diagrama flujo actividad listado rutas predefinidas

2.1.1.1.2. Diagrama actividad listado monumentos

Este diagrama representa el flujo que el usuario sigue para mostrar el listado de monumentos, este flujo cubre los casos de uso CU_1, CU_2

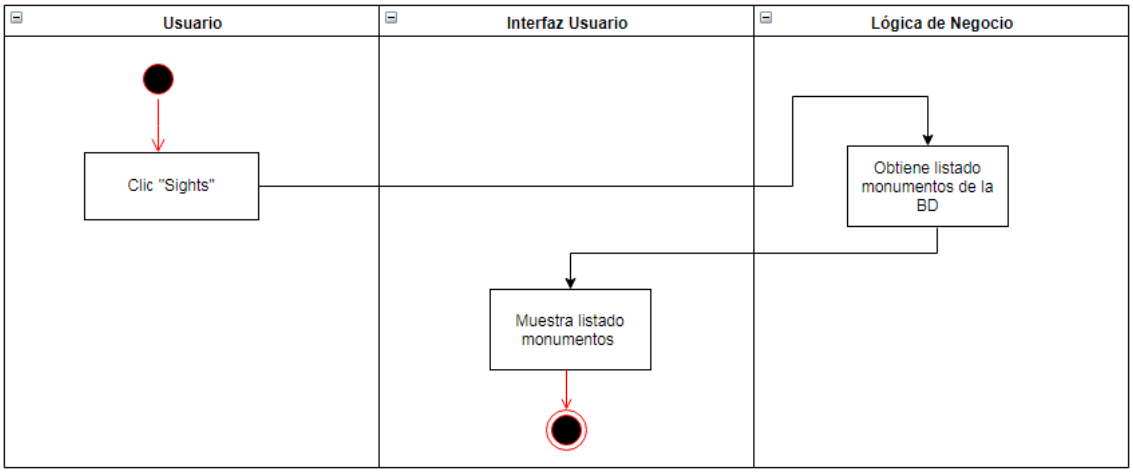


Ilustración 3 Diagrama actividad listado monumentos

2.1.1.1.3. Diagrama actividad mapa interactivo

Este diagrama representa el flujo que el usuario sigue para mostrar el mapa con monumentos, este flujo cubre los casos de uso CU_4, CU_5

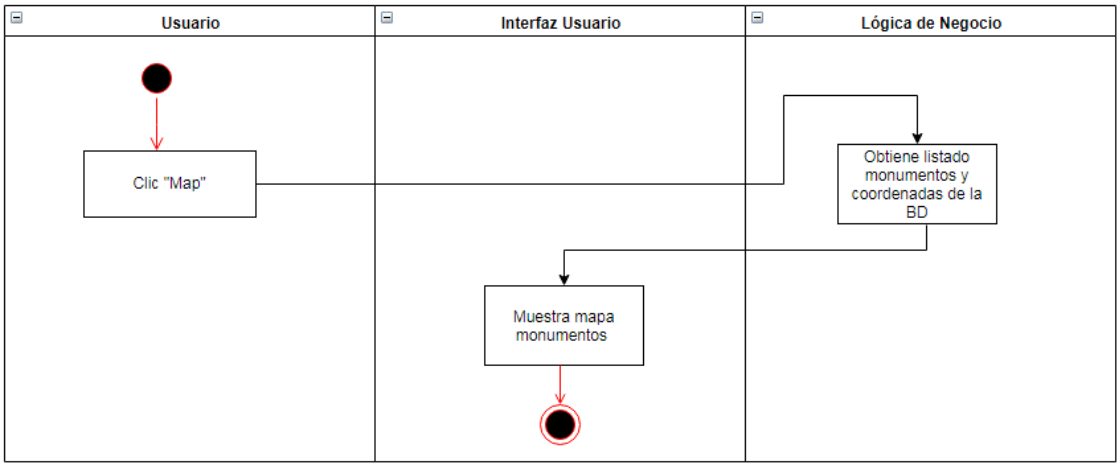


Ilustración 4 Diagrama actividad mapa interactivo monumentos

2.1.1.1.4. Diagrama actividad ficha monumento

Este diagrama representa el flujo que el usuario sigue para mostrar el detalle de un monumento, este flujo cubre los casos de uso CU_6, CU_7, CU_8, CU_9

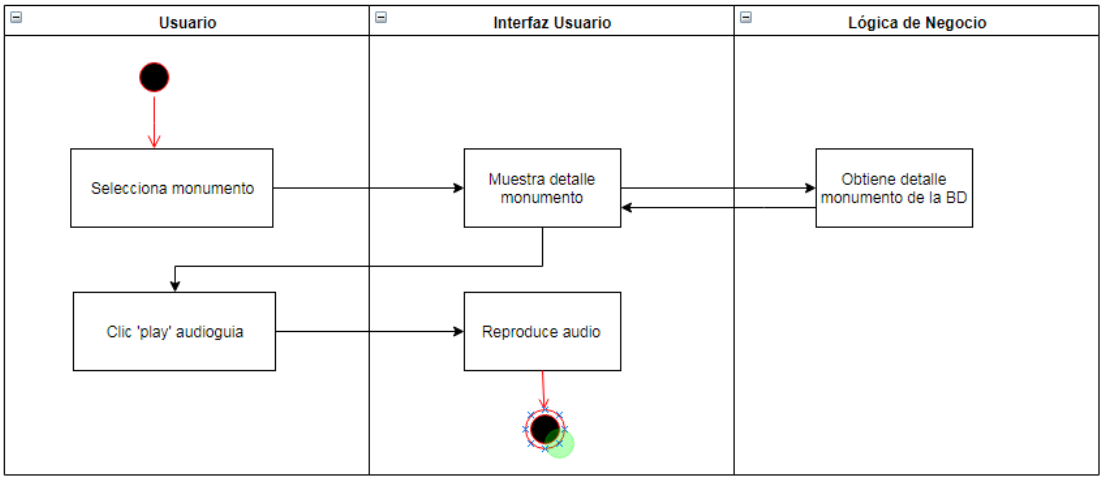


Ilustración 5 Diagrama actividad ficha monumento

2.1.1.2. Fichas casos de uso

2.1.1.2.1. Caso de uso general

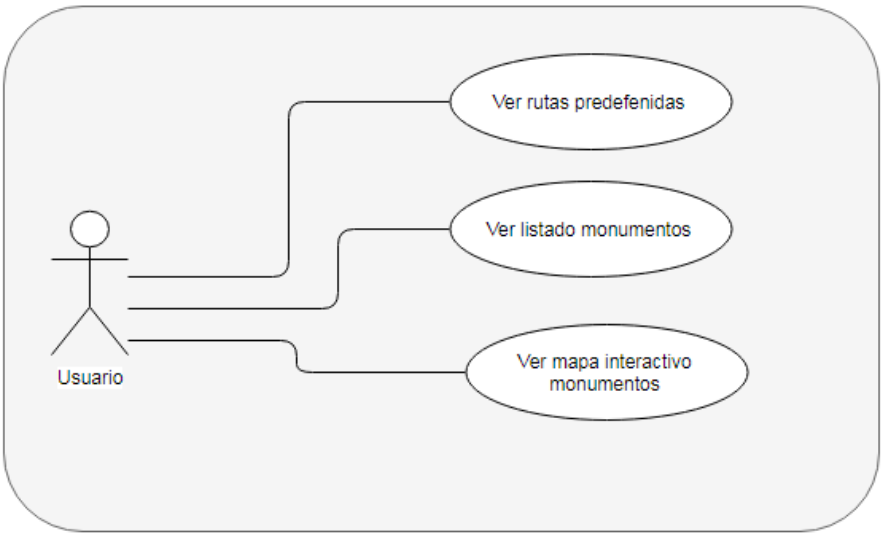


Ilustración 6 Caso de uso general

Identificador	CU_1
Caso de uso	General
Actores	Usuario
Propósito	Mostrar al usuario las diferentes opciones disponibles
Resumen	El usuario va a poder visualizar las diferentes funcionalidades que la aplicación le ofrece.
Precondiciones	
-	
Flujo principal	
1. Acceder a la aplicación 2. El sistema le mostrará un listado con las opciones disponibles	

2.1.1.2.2. Caso de uso listado monumentos

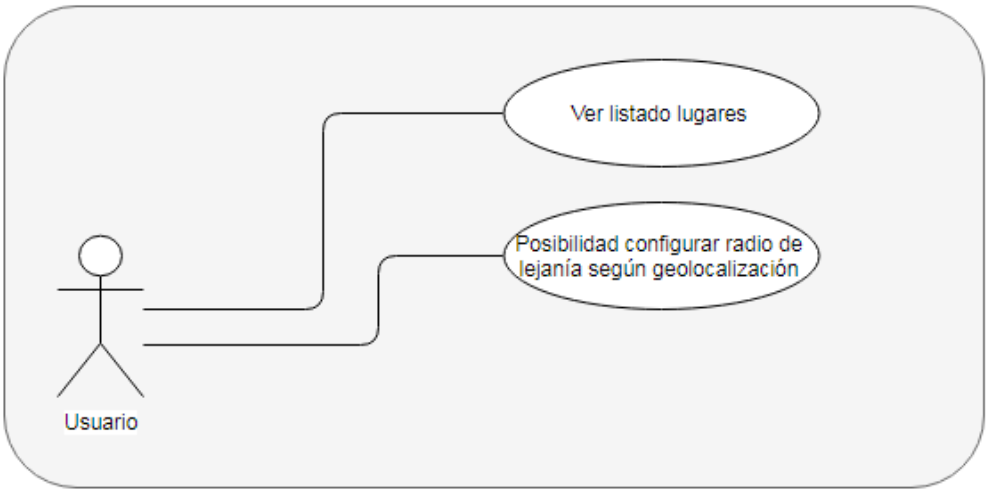


Ilustración 7 Caso de uso listado monumentos

Identificador	CU_2, CU_3
Caso de uso	Listado monumentos
Actores	Usuario
Propósito	Mostrar monumentos/lugares
Resumen	El usuario podrá ver un listado de monumentos y ajustar el radio de lejanía a partir de su ubicación
Precondiciones	
Haber entrado en la sección *sights* previamente	
Flujo principal	
1. Acceder a la aplicación 2. Elegir la sección *sights (list)*. 3. El sistema le mostrará un listado con todos los monumentos	

2.1.1.2.3. Caso de uso Mapa Interactivo Monumentos

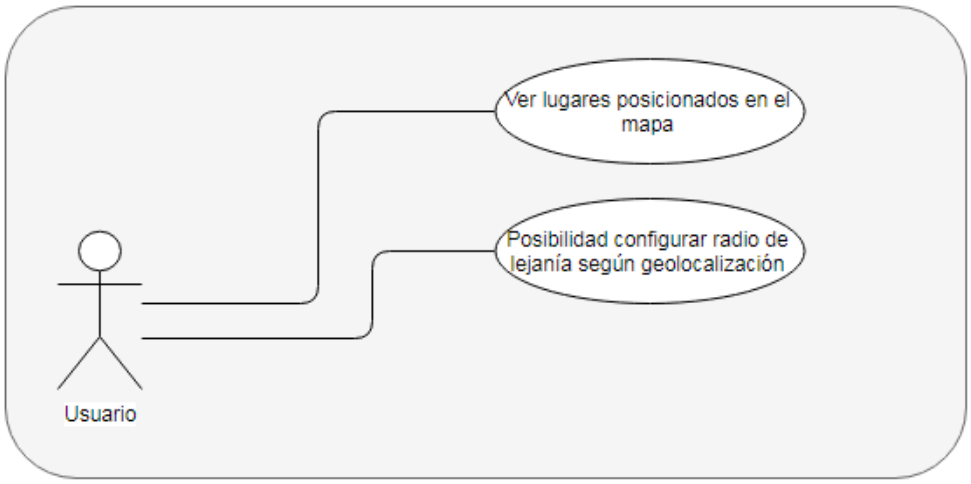


Ilustración 8 Caso de uso Mapa Interactivo Monumentos

Identificador	CU_4, CU_5
Caso de uso	Mapa interactivo Monumentos
Actores	Usuario
Propósito	Mostrar mapa con los lugares ubicados en él
Resumen	Mapa interactivo en el cual veremos los monumentos en un radio de lejania configurable a partir de nuestra posición actual.
Precondiciones	
Haber accedido a la sección *sights*	
Flujo principal	
1. Acceder a la aplicación 2. Elegir la sección *sights (map)* 3. El sistema le mostrará en un mapa todos los monumentos	

2.1.1.2.4. Caso de uso Ficha Monumentos

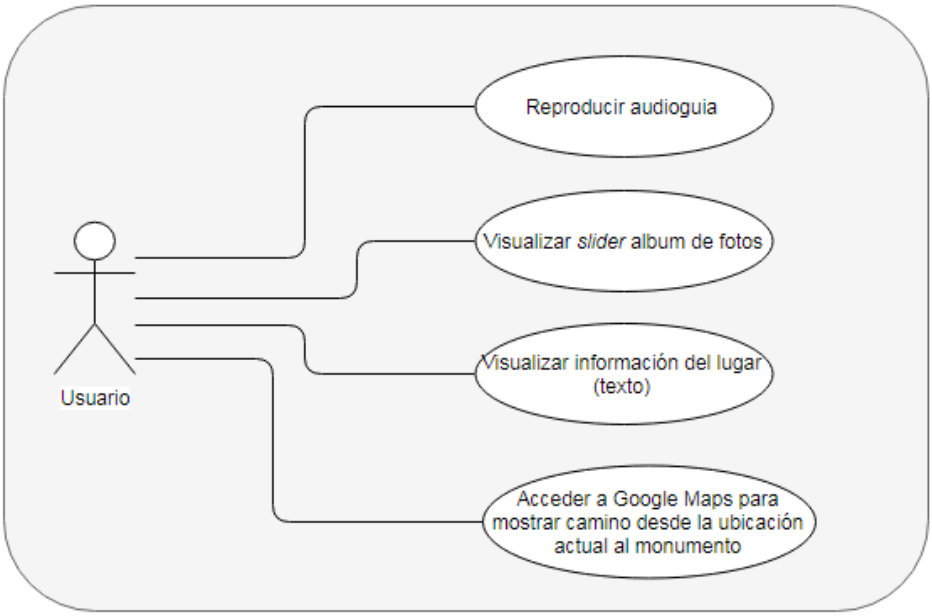


Ilustración 9 Caso de uso Ficha Monumentos

Identificador	CU_6, CU_7, CU_8, CU_9
Caso de uso	Ficha Monumentos
Actores	Usuario
Propósito	Mostrar toda la información de un monumento
Resumen	El usuario podrá acceder a la ficha donde podrá reproducir un audioguia, visualizar una galería de fotos, leer información en formato texto y trazar una ruta al monumento desde su posición actual.
Precondiciones	
Haber accedido al listado o mapa de los lugares y haber hecho clic sobre alguno de ellos.	
Flujo principal	
1. Acceder a la aplicación 2. Elegir la sección *sights (map)/ sights (list)* 3. Hacer clic sobre el botón 'See details' de algún monumento 4. El sistema mostrará toda la información del monumentos	

2.1.1.2.5. Caso de uso listado rutas predefinidas

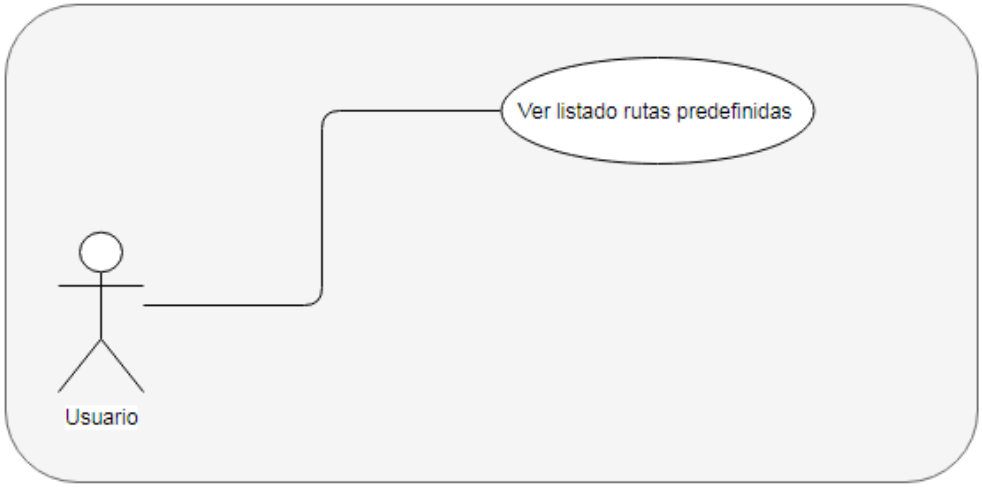


Ilustración 10 Caso de uso Listado Rutas Predefinidas

Identificador	CU_10
Caso de uso	Listado rutas predefinidas
Actores	Usuario
Propósito	Mostrar listado de rutas predefinidas
Resumen	Mostrar información al usuario de las rutas predefinidas existentes
Precondiciones	
Haber entrado a la sección *default routes*	
Flujo principal	
1. Acceder a la aplicación 2. Elegir la sección *default routes* 3. El sistema le mostrará las rutas disponibles	

2.1.1.2.6. Casos de uso ficha ruta predefinida

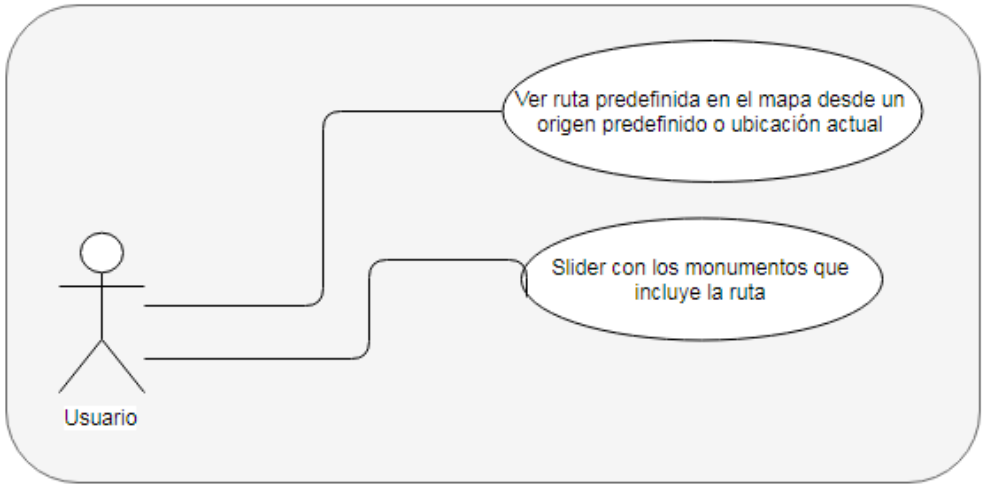


Ilustración 11 Caso de uso Ficha Ruta Predefinida

Identificador	CU_11, CU_12
Caso de uso	Ficha ruta predefinida
Actores	Usuario
Propósito	Visualizar los monumentos que contiene la ruta
Resumen	Se mostrará al usuario la ruta predefinida de manera interactiva en un mapa, viendo el recorrido y que monumentos incluye, también contiene una galería de imágenes donde se muestra al usuario fotos de los monumentos.
Precondiciones	
Haber hecho clic sobre alguna ruta predefinida de la sección '*default routes*'	
Flujo principal	
1. Acceder a la aplicación 2. Elegir la sección *default routes* 3. Hacer clic sobre el botón 'See details' de una ruta. 4. El sistema le mostrará la información detallada de la ruta	

2.1.2. Diagrama de clases

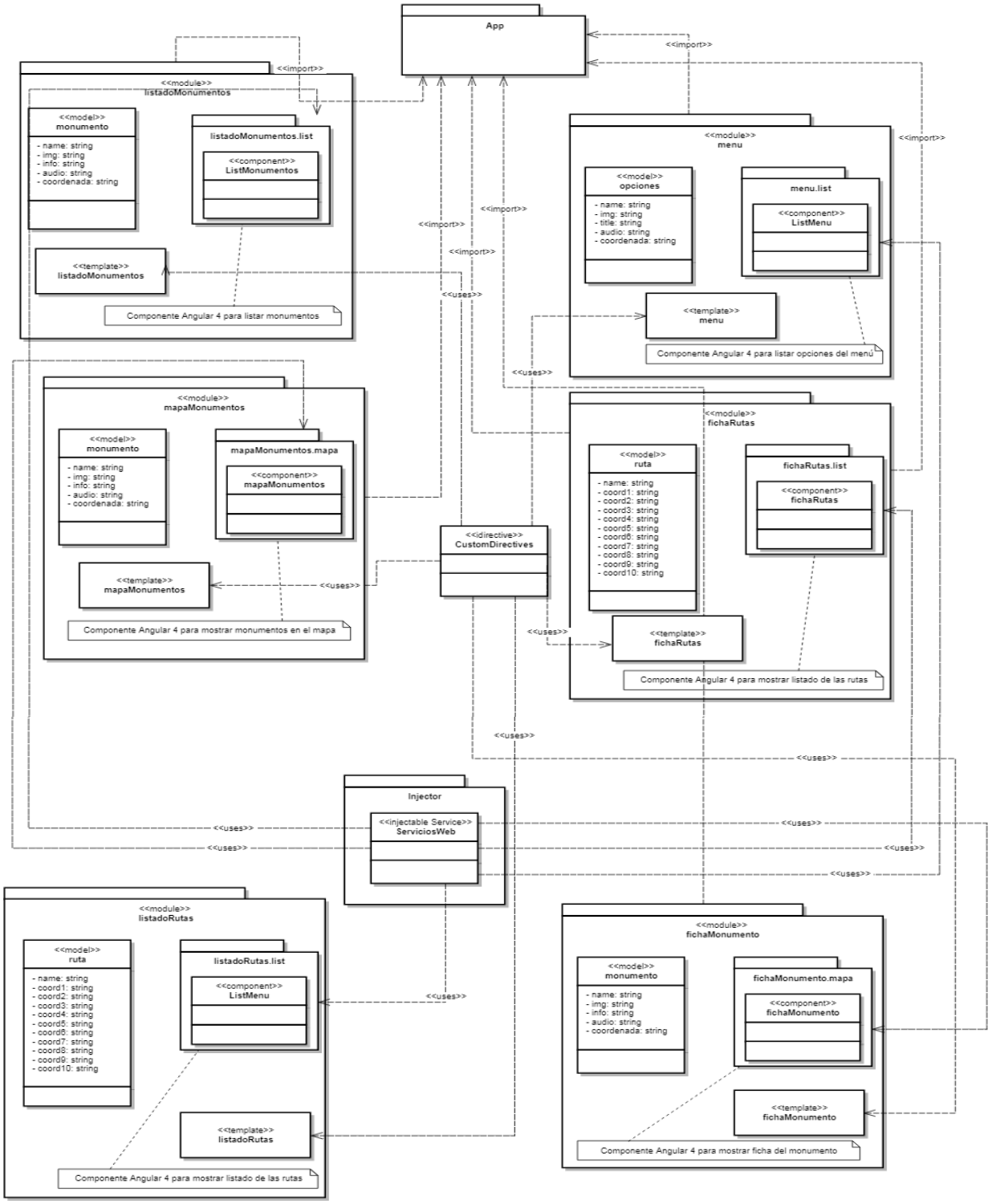


Ilustración 12 Diagrama de clases

2.1.3. Usabilidad/UX (DCU)

2.1.3.1. Menú principal

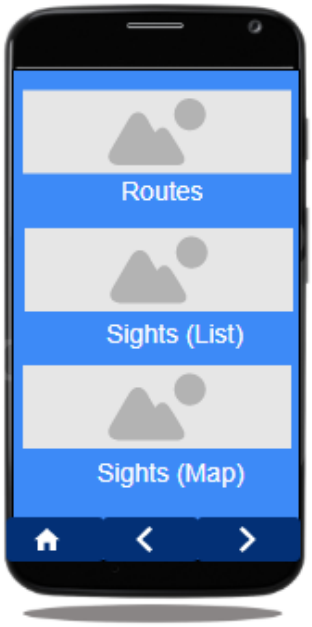


Ilustración 13 Wireframe menú principal

Menú principal

2.1.3.2. Listado monumentos

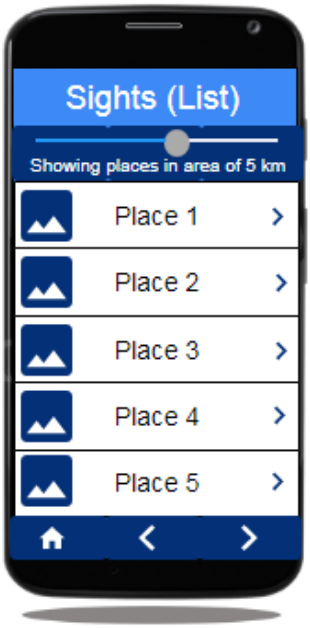


Ilustración 14 Wireframe listado monumentos

Listado monumentos

2.1.3.3. Mapa interactivo monumentos

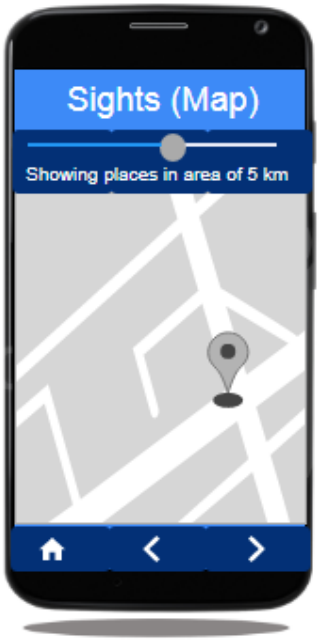


Ilustración 15 Wireframe mapa interactivo

Mapa interactivo monumentos

2.1.3.4. Ficha monumento

Ilustración 16 Wireframe ficha monumentos

Ficha monumento

2.1.3.5. Listado rutas predefinidas

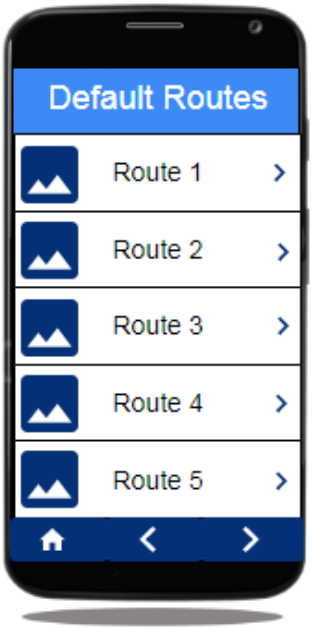


Ilustración 17 Listado rutas predefinidas

Listado rutas predefinidas

2.1.3.6. Ficha ruta predefinida

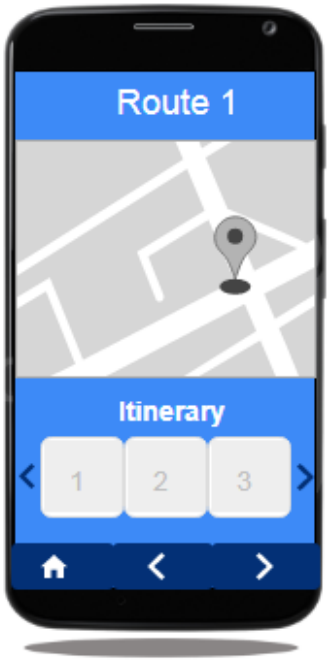


Ilustración 18 Ficha ruta predefinida

Ficha ruta predefinida

2.1.4. Arquitectura

La aplicación a desarrollar tiene la siguiente arquitectura global, la cual a la hora de profundizar en ella, se va a dividir en tres grupos principales, front, back y base de datos.

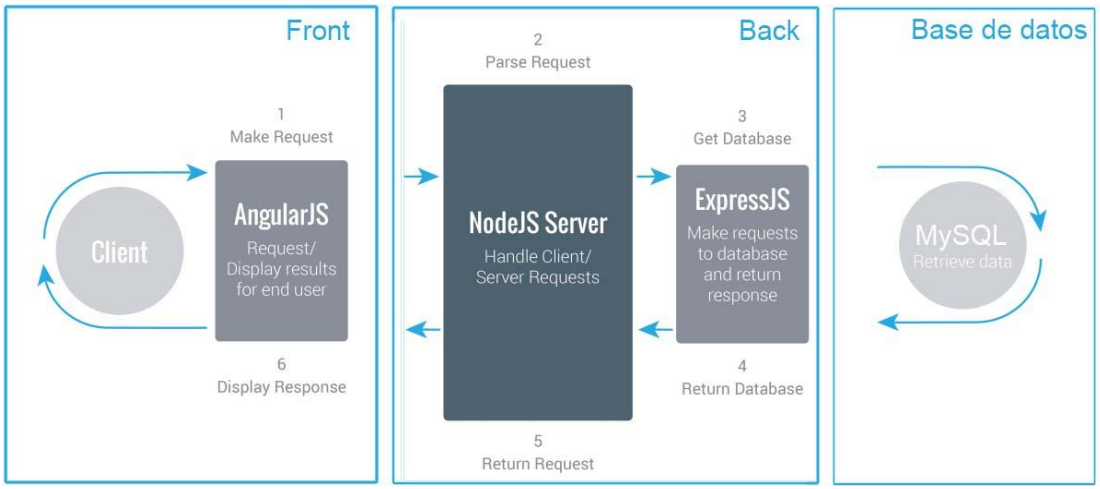


Ilustración 19 Arquitectura Global

En la parte front se va a centrar en la arquitectura de Angular 4, el cual es el *framework* encargado de la parte lógica, en el grupo back se detallará la arquitectura de nodeJS usando las librerías Express para realizar peticiones a la base de datos y por último en el grupo de base de datos hablaremos básicamente de la base de datos.

2.1.4.1. Front

En apartados anteriores se ha comentado que el framework Ionic 3 usa por debajo otro famework que es Angular 4 el cual es el que soporta toda la arquitectura, ésta sigue el patrón de MVW (*Model View Whatever*), el último término difiere del clásico MVC(*Model View Controller*) ya que se va a poder utilizar lo que más se adapte a nuestras necesidades ya puede ser *Model View Controller, Model View Adapter, Model View ViewModel*, etc.

La siguiente imagen nos muestra una visión de conjunto de la arquitectura de angular 4.

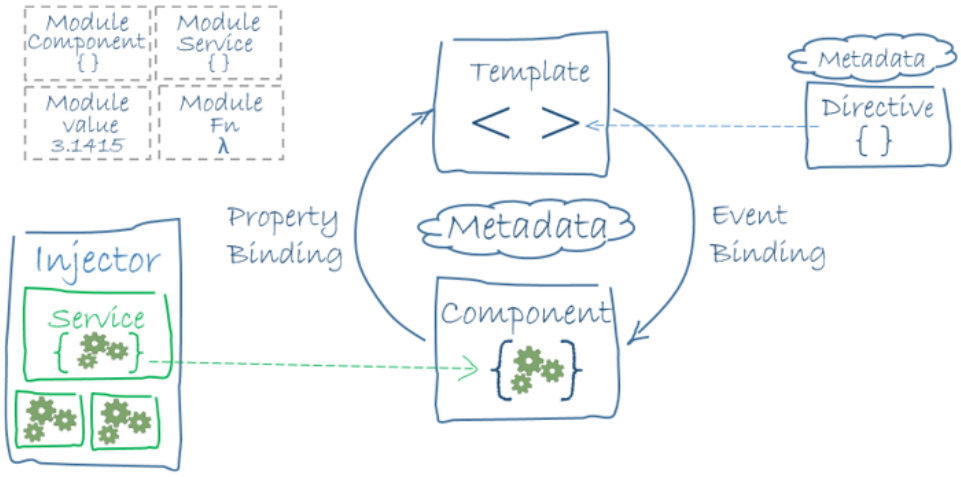
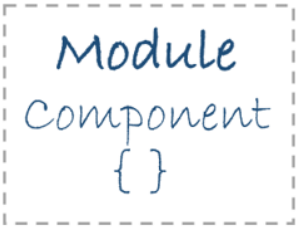


Ilustración 20 Arquitectura Front

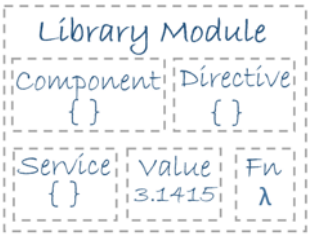
A continuación vamos a explicar con más detalle cómo funciona cada parte de la que consta la arquitectura:

- Módulos



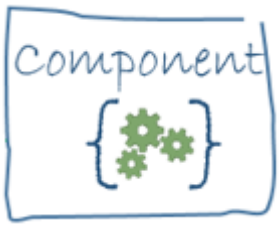
Las aplicaciones desarrolladas con angular, son aplicaciones modulares, por lo que podremos tener diferentes módulos.

- Librerías de Angular



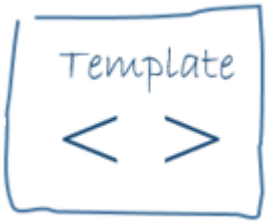
Angular consta de librerías que nos proporcionaran funcionalidades muy útiles a la hora de desarrollar la lógica de la aplicación.

- Componente



El componente es el elemento que conecta los servicios con la plantilla y el cual contiene la mayor parte de la lógica de la aplicación ya que aquí va a ser donde traten los dates antes de enviarlos a la plantilla.

- Plantillas

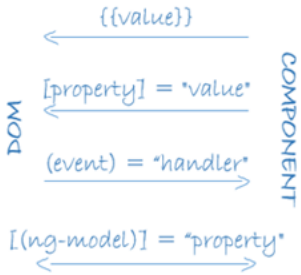


Cada componente constará de una plantilla asociada, normalmente tienen formato html y mediante el uso de la sintaxis de angular inyectaremos dinámicamente nuestra información en el DOM.

- Metadatos

Los metadatos indican a angular de qué manera hay que procesar una clase.

- Enlaces de datos



Esta es la parte en la que destaca angular ya que nos ofrece un mecanismo para coordinar partes de una plantilla con las partes de un componente. Solo hay que añadir anotaciones en la plantilla para conectar ambos lados.

- Directivas

Las directivas se encuentran en las plantillas y cuando angular las renderiza transforma el DOM según las instrucciones de las directivas insertadas.

- Servicios

Los servicios se refiere a cualquier dato, función que nuestra aplicación necesita, en nuestro caso contendrá las funciones encargadas de recoger la información de nuestra base de datos a través de la API

- Inyección depen.

La inyección de dependencias es la manera de proveer al componente de los servicios que se necesitan en ese momento.

2.1.4.2. Back

En la parte *back* de la aplicación vamos a contar con una API desarrollada mediante nodeJS que se utiliza como manejador para gestionar las peticiones por parte del cliente, por otra parte también vamos a utilizar *ExpressJS* el cual nos servirá para conectar nuestra API con la base de datos y poder realizar peticiones a ésta. La arquitectura quedaría tal que así:

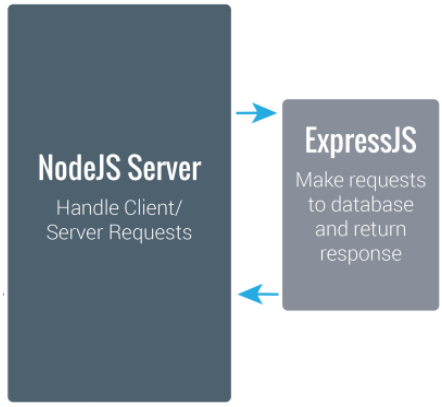


Ilustración 21 Arquitectura Back

2.1.4.3. Base de datos

El diseño de la base de datos se ha llevado a cabo utilizando el *software* PowerDesigner, esta herramienta permite transformar modelo conceptual de una base de datos en uno físico, con el físico ya podremos generar un script para posteriormente importar a la base de datos en cuestión. En estos apartados veremos los modelos conceptual y físico que se han diseñado para la aplicación.

2.1.4.4. Modelo Conceptual

Este modelo va a ser en el cual se van a representar los conceptos del mundo real, sin tener en cuenta la plataforma y paradigma de modelado.

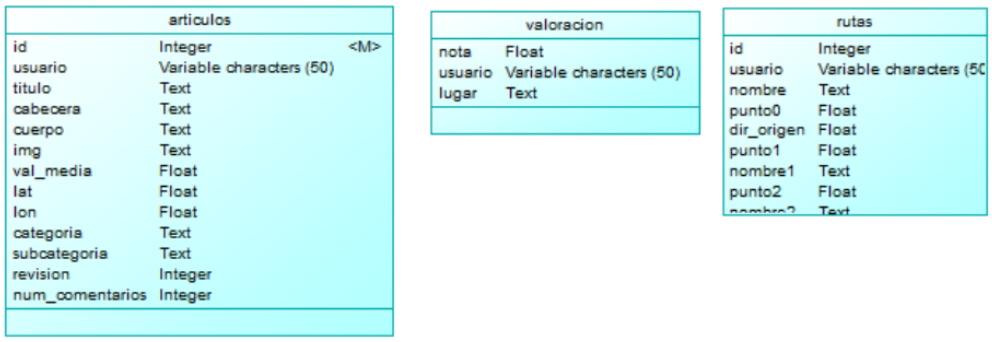


Ilustración 22 Modelo Conceptual

2.1.4.5. Modelo Físico

Es un modelo el cual depende del motor que vayamos a estar corriendo en nuestra base de datos, por lo que habrá que diseñarlo de acuerdo a los estándares del motor que utilicemos. En la siguiente imagen podemos ver el modelo físico.

Habiendo finalizado nuestro diseño, vamos a generar el *script* sql para importar a nuestra base de datos mediante la herramienta myPHPAdmin.

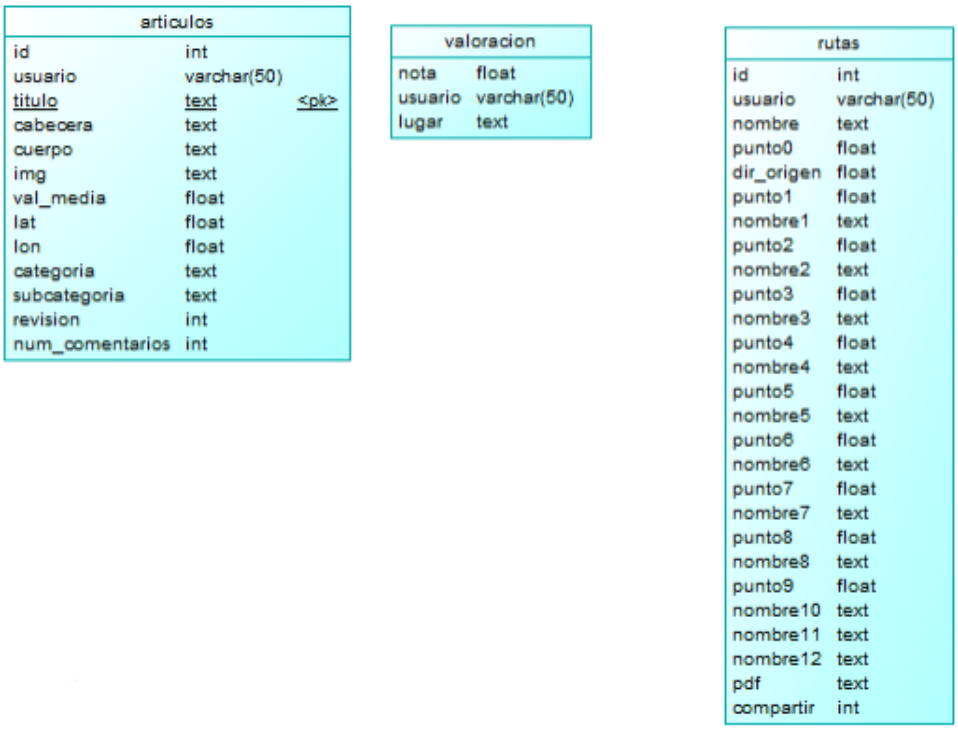


Ilustración 23 Modelo Físico

3. Desarrollo

3.1. Extractos de código

3.1.1. Base de datos & Api

En este apartado se detallará la construcción de la parte backend, tanto la base de datos como la API, todo alojado en heroku.com[6], la cual es una plataforma que ofrece servicio de computación en la Nube la cual soporta distintos lenguajes de programación en nuestro caso NodeJS.

En un principio se diseñó la base de datos con un motor mySql que corría en local, pero a la hora de promover la base de datos a heroku, se ha tenido que convertir a postgresql mediante un *script* ya que mySql no entraba dentro del plan gratuito.

A continuación expongo un fragmento de la API donde se muestra un servicio el cual proporciona la información de las rutas.

```
restapi.get('/rutas', function(req, res){
  con.query("Select a.cabecera,a.cuerpo,a.img, REPLACE(a.titulo,'+',' ') as titulo_format,r.* "+
    "FROM articulos a left join rutas r ON a.nombre_ruta = r.nombre  where a.subcategoria "+
    "= 'ruta' and a.idioma = 'en'", function (err, result) {
   if (err) throw err;
      res.header('Access-Control-Allow-Origin', "*");
      res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
      res.header('Access-Control-Allow-Headers', 'Content-Type');
      res.json({ "data" : result });
    });
});
```

Ilustración 24 Extracto API nodeJS

3.1.2. Ionic

3.1.2.1. Servicios

En la aplicación tendremos un archivo llamado ‘app.service.ts’ donde almacenaremos todas las llamadas que vamos a realizar a la API aparte de llamadas a la API de Google Maps. Estas funciones serán llamadas desde los controladores.

A continuación muestro un extracto de una función que solicita a la API información de las rutas , la cual la procesa y la devuelve en forma de JSON.

```
loadRoutes(){
  return this.http.get(this.host+'/rutas').map(response => response.json());
}
```

Ilustración 25 Ejemplo llamada servicio web

3.1.2.2. Controladores

Aquí es donde se almacenará la lógica de la aplicación, aquí se procesará la información para posteriormente enviar a la vista, dentro de los controladores se pueden importar todo tipo de librerías, en nuestro caso para el desarrollo de esta aplicación las librerías importadas son:

Librería	Descripción
@ionic-native/text-to-speech	Librería nativa Android que nos da la funcionalidad de convertir texto en voz
@ionic-native/geolocation	Librería nativa Android que nos proporciona información de la localización del dispositvo
@ionic-native/status-bar	Gestiona la apariencia de la barra de estado de Android
@agm/core	Librería que implementa la API de Google Maps en Angular 4
@agm/snazzy-info-window	Facilita la modificación de las open window de google maps.
@angular/http	Nos proporciona las librerías para establecer conexiones http con la API
@angular/core	Librería la cual incluye el núcleo de Angular 4
ionic-angular	Librería que integra angular 4 con ionic y las librerías cordova
ng2-order-pipe	Nos da la funcionalidad de ordenar arrays en Angular 4

Los controladores que se han desarrollado en la aplicación son:

- Menu.ts
- ListadoMonumentos.ts
- FichaMonumentos.ts
- DefaultRoutes.ts
- FichaRuta.ts
- Map.ts

A continuación se va a mostrar extractos de código de las funcionalidades más importantes de cada controlador.

<u>Menu</u>

Estas funciones se encargan de redireccionar de un controlador a otro.

```
goToSights(){

  this.navCtrl.push(ListadoMonumentos);

}

goToDefaultRoutes(){

  this.navCtrl.push(DefaultRoutes);

}

goToSightsMap(){
  this.navCtrl.push(SightMap);
}
```

Ilustración 26 Funciones Navegación

ListadoMonumentos

La función ngOnInit la cual se ejecuta al inicio, lo que hace es recoger todos los lugares y calcular la distancia y el tiempo que se tardaría andando entre el lugar en cuestión y tu posición actual.

```
ngOnInit(){

  this.service.loadArticles().subscribe( response => {
    this.sights = response.data;
    let contador:number = 0;
    this.geolocation.getCurrentPosition().then((position) => {
      response.data.forEach(function (item) {
        this.service.getDistanceTwoPoints(position.coords.latitude+','+position.coords.longitude,item.lat
          +','+item.lon).subscribe( distance => {
          if (distance['status'] == 'OK' && distance['rows'][0]['elements']['0']['status'] == 'OK' ){
            this.origin = position.coords.latitude+','+position.coords.longitude;
            this.sights[contador].distance = distance['rows'][0]['elements']['0']['distance']['text'];
            this.sights[contador].time = distance['rows'][0]['elements']['0']['duration']['text'];
            this.sights[contador].showDetailsToggle = false;
            this.sights[contador].icon = 'ios-arrow-down';

          }

          if (response.data.length - 1 == contador){
            this.cargado = true;
            console.log(this.sights);
          }
          contador++;
      });
    }, this);
  });
});
```

Ilustración 27 Extracto código ListadoMonumentos

FichaMonumentos

Estas dos funciones se encargarán de utilizar la librería *text-to-speech* para convertir el contenido de texto en voz, la función startReadText se encargará de iniciar la reproducción y la función stopReadText se encargará de para la reproducción.

```
startReadText(){

    this.tts.speak(this.sight.speech)
     .then(() =>
       console.log('Success')
     )
     .catch((reason: any) => console.log(reason));
     this.playing = true;
}



stopReadText(){

    this.tts.stop()
     .then(() =>
       console.log('Success')
     )
     .catch((reason: any) => console.log(reason));
     this.playing = false;
}
```

Ilustración 28 Extracto código FichaMonumentos

DefaultRoutes

Esta función se ejecuta al inicio y lo que hace es llamar al servicio loadRoutes y traerse el listado de rutas predefinidas disponibles.

```
ngOnInit(){

  this.service.loadRoutes().subscribe( response => {

    this.routes = response.data;
    let contador:number = 0;
    response.data.forEach(function (item) {
      this.routes[contador].showDetailsToggle = false;
      this.routes[contador].icon = 'ios-arrow-down';

      if (response.data.length - 1 == contador){
        this.cargado = true;
        console.log(this.routes);
      }
      contador++;
    },this);
  });
}
```

Ilustración 29 Extracto código DefaultRoutes

FichaRuta

La función ngOnInit debido a su extensión se va a mostrar solo las partes más importantes. En esta primera parte se realiza dos cargas de servicios, una los detalles de la ruta y otra detalles por cada monumento que contenga la ruta

```
ngOnInit(){

    this.service.loadRoute(this.id).subscribe( response => {

        this.route = response.data[0];
        let markers:any = [];
        let points:any = [];
        this.geolocation.getCurrentPosition().then((position) => {
            Object.keys(this.array).forEach(function(keyArray,indexArray) {

                let name = 'nombre'+keyArray;
                let point = 'punto'+keyArray;

                Object.keys(response.data[0]).forEach(function(key,index) {
                    if(key == name && this.route[name] != ''){
                        this.service.getImageRouteItems(this.route[name]).subscribe( responseImg => {
                            if (response.data[0] !== undefined){
                                if (Number(keyArray) != 0){
                                    this.imgs.push({
                                        idImg: name,
                                        img: responseImg.data[0].img,
                                        nombre: name, punto: point,
                                        id: responseImg.data[0].id
                                    });

                                }

                            }
                        });
                    }
                },this);
```

Ilustración 30 Extracto código FichaRuta (I)

En esta segunda parte se construye los *markers* que irán en el mapa, también sacaremos la distancia en línea recta desde el punto donde nos encontremos hasta cada lugar de la ruta para poder después definir un área y filtrarlo por el radio de acción.

```
//Calculamos la distancia entre dos puntos y lo almacenamos en un array
this.service.getDistanceTwoPoints(position.coords.latitude+','+
    position.coords.longitude,this.route[point].split(',')[0]+','+
    this.route[point].split(',')[1]).subscribe( distance => {
    let distancia:any;
    distancia = this.getDistanceLine(position.coords.latitude,position.coords.longitude,
        this.route[point].split(',')[0],this.route[point].split(',')[1]);

    if (distance['status'] == 'OK' && distance['rows'][0]['elements']['0']['status'] == 'OK' ){
        if(Number(keyArray) != 0 && this.route[name] != '' && this.route[point] != ''){

            if(Number(keyArray) == 2){
                this.origin = this.route[point];
            }

            points.push({
                location:this.route[point],
                stopover: true
            });


            markers.push({
                location:this.route[point],
                url:'http://www.myiconfinder.com/uploads/iconsets/256-256-a5485b563efc4511e0cd8b
                scaledSize: { height: 40,width: 40},
                visited: false,
                distance: distance['rows'][0]['elements']['0']['distance']['text'],
                time: distance['rows'][0]['elements']['0']['duration']['text'],
                distanceLine: distancia
            });

            this.destination = this.route[point];
        }
    }
```

Ilustración 31 Extracto código FichaRuta (II)

En esta tercera parte almacenamos en *localStorage* la ruta, ya que el usuario va a tener la funcionalidad de marcar como vistos los lugares y que se guarde de forma permanente.

```
        if (this.array.length == (Number(keyArray) + 1)){

            if(localStorage.getItem(this.id+' - route') == null){
                this.markers = markers;
                localStorage.setItem(this.id+' - route', JSON.stringify(markers));
            }
            else{
                this.markers = JSON.parse(localStorage.getItem(this.id+' - route'));
            }

            this.points = points;
            console.log(this.markers);
            this.cargado = true;
        }
    });
},this);

this.currentLocation = {
        location: position.coords.latitude +','+ position.coords.longitude,
        url: 'https://d30y9cdsu7xlg0.cloudfront.net/png/25718-200.png',
        scaledSize: { height:40, width:40}
};
```

Ilustración 32 Extracto código FichaRuta (III)

Y en esta última parte se ejecutará el *listener* de la geolocalización para cuando el usuario se mueva con su dispositivo vaya actualizándole la posición de manera continua.

```
let options:any = {enableHighAccuracy: true,desiredAccuracy: 0, frequency: 1 };
let watch = this.geolocation.watchPosition(options);
watch.subscribe((position) => {
    this.currentLocation = {
        location: position.coords.latitude +','+ position.coords.longitude,
        url: 'https://d30y9cdsu7xlg0.cloudfront.net/png/25718-200.png',
        scaledSize: { height:40, width:40}
    };
});

this.styles = this.service.getMapStyle();
```

Ilustración 33 Extracto código FichaRuta (IV)

Map

Por no poner código repetidos, la función principal en este controlador es exactamente igual a la del controlador FichaRuta, la diferencia es que en esta cargamos todos los monumentos de la base de datos mientras que en el otro solo se cargan los lugares que pertenecen a esa ruta.

3.1.2.3. Vistas

En este apartado se van a mostrar tanto los componentes de ionic utilizados como las directivas propias.

Los componentes nos permitirán diseñar el aspecto gráfico de nuestra aplicación utilizando únicamente etiquetas HTML y algunos atributos para opciones de configuración.

Las directivas según la definición que nos da Angular son marcadores sobre los elementos DOM que indica al compilador HTML que aplique un determinado comportamiento, más a delante veremos un ejemplo de ello.

La aplicación consta de las siguientes vistas:

- Menu.html
- ListadoMonumentos.html
- FichaMonumentos.html
- DefaultRoutes.html
- FichaRuta.html
- Map.html

A continuación se muestra algunos componentes propios de Ionic que podemos encontrar.

```
<ion-content class="card-background-page">
  <ion-grid>
    <ion-row>
      <ion-col col-12 col-sm-4 [style.width.px]="width/3">
      <ion-card class="card-margin" (click)="goToDefaultRoutes()">
        <img  [style.height.px]="height/3" src="./assets/img/default-routes.png"/>
        <ion-fab left top>
        <button ion-fab>
          <ion-icon name="ios-map-outline"></ion-icon>  Default Routes
        </button>
      </ion-fab>
      </ion-card>
      </ion-col>
```

Ilustración 34 Extracto código Componentes Ionic

Como se puede ver en la figura anterior, ionic nos proporciona componentes UI (ion-content, ion-grid, ion-col, ion-card,…) para facilitarnos el desarrollo de una interfaz limpia.

Por otra parte para el desarrollo de la sección de '*Default Routes'* se ha tenido que programar una directiva propia para mediante una etiqueta HTML nos genere una ruta en el mapa utilizando la API de Google Maps. A continuación podemos ver un fragmento de la directiva.

```
@Directive({
  selector: 'sebm-google-map-directions'
})
export class CustomMapDirective {
  @Input() origin;
  @Input() destination;
  @Input() waypoints;
  @Input() travelMode;
  @Input() location;
  constructor (private gmapsApi: GoogleMapsAPIWrapper) {}
  ngAfterViewInit(){
    this.calculateRoute();
  }

  calculateRoute(){
        this.gmapsApi.getNativeMap().then(map => {
            var directionsService = new google.maps.DirectionsService;
            var directionsDisplay = new google.maps.DirectionsRenderer({suppressMarkers : true});
            directionsDisplay.setMap(map);
```

Ilustración 35 Extracto código directiva

Para aplicar esta directiva en nuestra plantilla será tan fácil como llamarla de la siguiente manera:

```
<sebm-google-map-directions [origin]="origin" [waypoints]="points" [travelMode]="travelMode"
 [destination]="destination"></sebm-google-map-directions>
```

Ilustración 36 Extracto código uso directiva

De esta manera mediante la utilización de estas etiquetas nos generará la ruta deseada.

3.1.2.4. Estilos

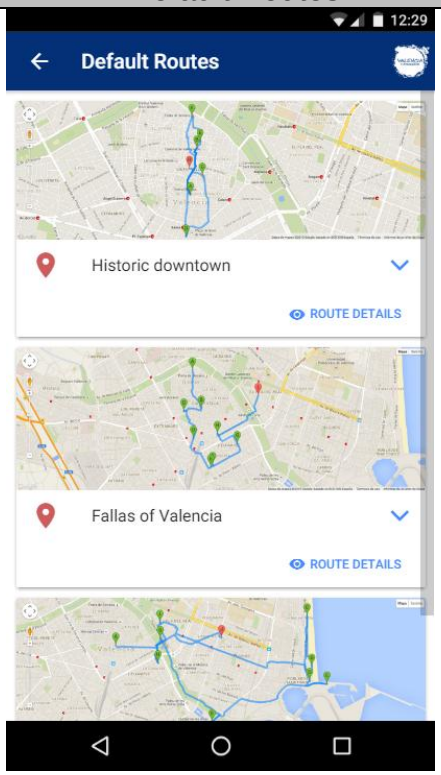
En la aplicación tendremos una hoja de estilos común a todos los controladores y aparte para cada controlador tendremos una hoja de estilos individual, a continuación se muestra un extracto de la hoja de estilos principal.

```
.toolbar-background{

    background:#033076;
    color:white;
}

.logo{
  height: 39px;
    width: auto;
    margin-top: 5px;
}

.card-background-page {

  ion-card {
    position: relative;
    text-align: center;

  }

  .card-title {
    position: absolute;
    top: 36%;
    font-size: 2.0em;
    width: 100%;
    font-weight: bold;
    color: #fff;
  }
```

Ilustración 37 Extracto código estilo scss

Estos estilos scss se compilaran a la hora de lanzar la aplicación con un gestor de tareas llamado *gulp* en el que le daremos las indicaciones para que a la hora de construir la aplicación nos compile y minimice los scss en css

3.1.2.5. Pantallazos finales

Menú Inicio	Default Routes
	
Ilustración 38 Captura menú inicio	Ilustración 39 Captura Def. Routes
Detalle Default Routes (I)	**Detalle Default Routes (II)**

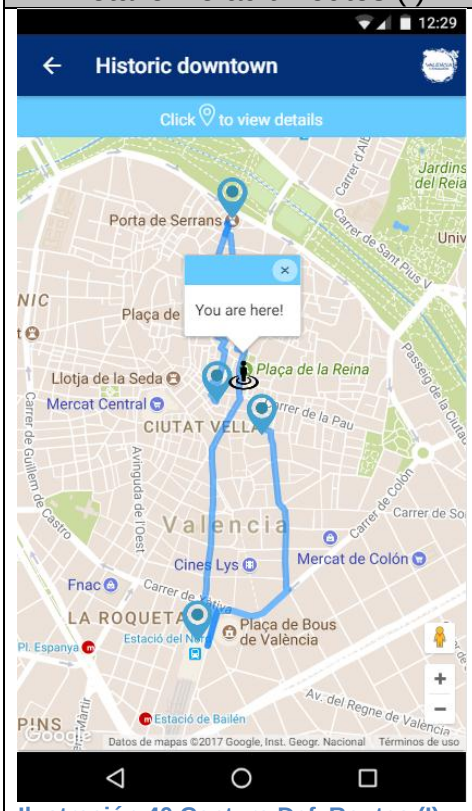
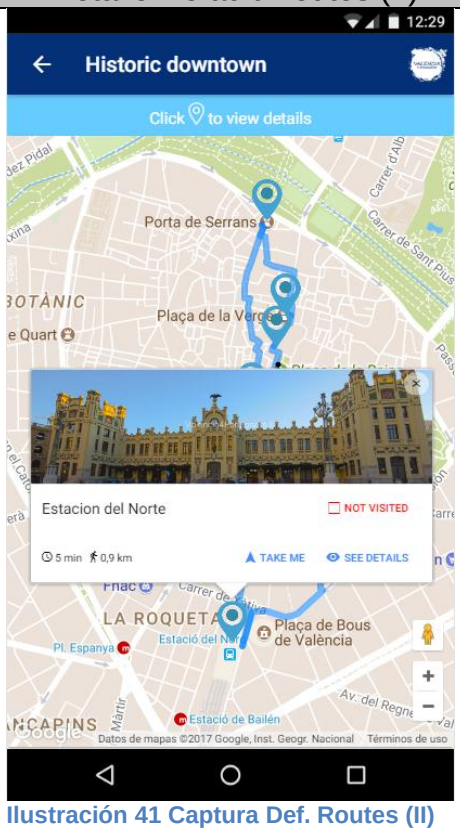

	

Ilustración 40 Captura Def. Routes (I)	Ilustración 41 Captura Def. Routes (II)

Ficha Monumento	Listado Monumentos

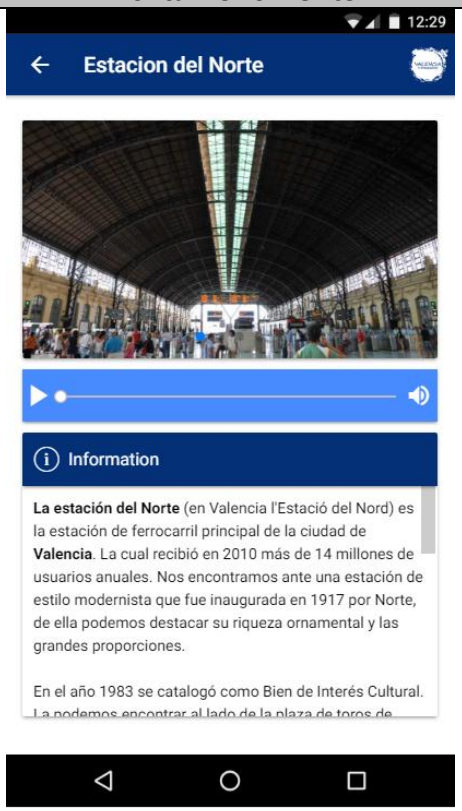
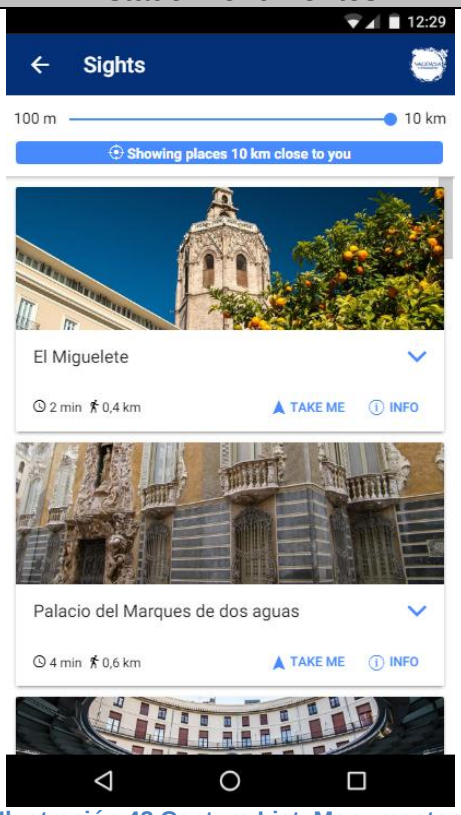

	

Ilustración 42 Captura Ficha Monumento	Ilustración 43 Captura List. Monumentos
Mapa lugares (I)	Mapa lugares (II)

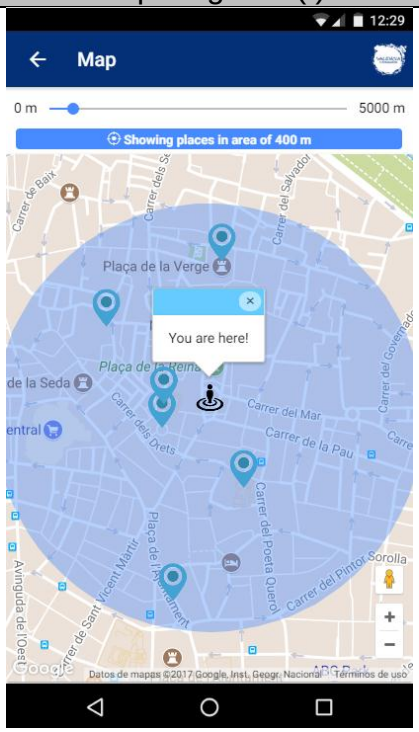
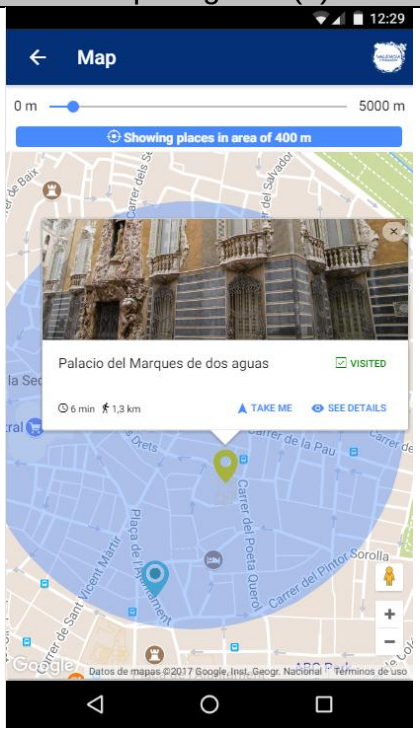

	

Ilustración 44 Captura Mapa Lugares (I)	Ilustración 45 Captura Mapa Lugares (II)

3.2. Seguridad

Dentro de nodeJS encontramos la librería postgresql la cual va a ser la encargada de establecer la conexión con la base de datos, esta librería tiene métodos de seguridad como puede ser el *binding* de los parámetros de la query, con esto nos evitaremos que pueda haber una inyección SQL.

Por otra parte para asegurar una seguridad contra ataques a la base de datos y que pudiera borrar o dañar información el usuario que se ha creado para la aplicación ha sido un usuario con escasos privilegios, exactamente uno, el cual es el realizar 'Select'.

Por otra parte el propio Angular 4 nos provee de elementos de seguridad como 'Sanitization' el cual se encarga de prevenir fallos en el *Cross Site Scripting Security (XSS)* por lo que se verifican los valores para que sean seguros en cualquier contexto del DOM.

Si en futuro se ampliara la aplicación y se añadieran roles de usuarios diferentes, se podría utilizar el sistema de JSON *Web Tokens* el cual contendrá los roles que han sido garantizados para el usuario.

3.3. Test

Este apartado vamos a dividirlo en dos subapartados por una parte vamos a analizar desde una visión heurística la web basándonos en los 10 principios heurísticos de Jakob Nielsen, los cuales están basados en amplias reglas generales y no específicas directrices de usabilidad.

También se realizará un test con las funcionalidades principiales que nos aporta la aplicación.

Por otra parte vamos a diseñar una serie de pruebas que los usuarios elegidos van a ir realizando donde anotaremos las impresiones, opiniones y si se pierden en el camino de realizar alguna acción.

3.3.1. Heurística principios de Jakob Nielsen:

- **Principio 1:** Visibilidad del estado del sistema

  En este principio se nos indica que siempre tenemos que tener informado al usuario de lo que está pasando en nuestra web en todo momento, por ejemplo si está cargando algún dato, que el usuario pueda ver una barra de progreso, si se envía un formulario ver una ventana de confirmación, etc.

A continuación podemos ver en la captura, mediante el *spinner* de carga estamos indicando al usuario que se están cargando los datos,por lo que en todo momento el usuario está informado.

Ilustración 46 Principio 1 Ejemplo

- **Principio 2:** Relación entre el sistema y el mundo real
  Con este principio se refiere a que el sistema ha de hablar el lenguaje del usuario con frases cotidianas para que pueda reconocerlas con facilidad, como podemos ver en este ejemplo.

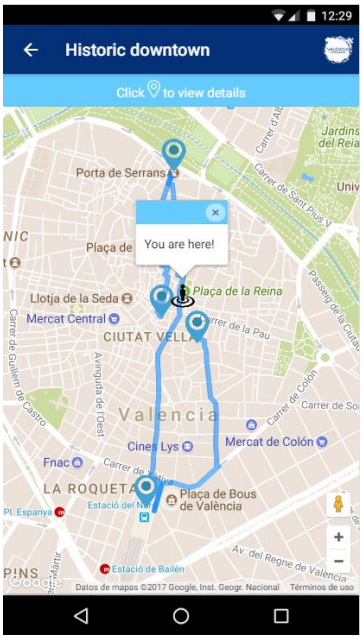


Ilustración 47 Principio 2 Ejemplo

- **Principio 4**: Consistencia y estándares
  Este principio viene a decir que tenemos que tener en cuenta en seguir los convenios establecidos para ciertos iconos. Como podemos ver en la siguiente captura del menú principal al utilizar los íconos que Ionic nos proporciona, nos aseguramos seguir unos estándares y mantener una consistencia en la aplicación.

Ilustración 48 Principio 4 Ejemplo

- **Principio 8:** Diseño estético y minimalista
  Las aplicaciones no tienen que tener información innecesaria por lo que para cumplir este principio se ha de diseñar una aplicación con un diseño estético y minimalista, como podemos ver en la siguientes capturas, podemos ver que la aplicación solo muestra la información necesario y útil para el usuario.

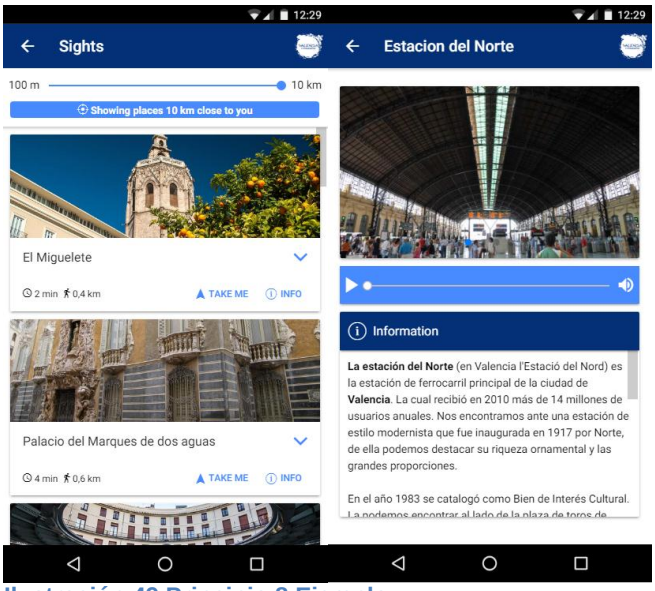


Ilustración 49 Principio 8 Ejemplo

3.3.2. Prueba Funcionalidad

Identificador	Test_001		
Objetivo	Comprobar listado Monumentos (Ordenado cercanía)		
Prerrequisitos			
Procedimientos	**Paso 1**	**Entrada**	Hacer click sobre la opción “Sights”
		Salida esperada	Se muestra el listado de monumentos ordenado por cercanía

Identificador	Test_002		
Objetivo	Comprobar Ficha Monumento (Repr. Audioguia)		
Prerrequisitos			
Procedimientos	**Paso 1**	**Entrada**	Hacer click sobre un monumento
		Salida esperada	Se muestra la ficha con sus imágenes, texto y se reproduce correctamente la audioguia

Identificador	Test_003		
Objetivo	Comprobar listado Rutas		
Prerrequisitos			
Procedimientos	**Paso 1**	**Entrada**	Hacer click sobre la opción “Default Routes”
		Salida esperada	Se muestra el listado de rutas predefinidas

Identificador	Test_004		
Objetivo	Comprobar Opciones Ruta		
Prerrequisitos			
Procedimientos	**Paso 1**	**Entrada**	Hacer click sobre la ruta
		Salida esperada	Se muestra un mapa con la ruta representado, pudiendo interactuar con los “markers” y viendo detalles de cada lugar.

Identificador	Test_005		
Objetivo	Comprobar Mapa		
Prerrequisitos			
Procedimientos	**Paso 1**	**Entrada**	Hacer click sobre la opción “Mapa”
		Salida esperada	Se muestra un mapa con todos los monumentos disponibles en la aplicación filtrados por radio de acción

3.3.3. Pruebas Test Usuarios

- **Nombre:** Miguel Rodríguez

- **Edad:** 23 años

- **Tarea 1: Busca una ruta predefinida**
  Previa traducción del menú el usuario no ha tenido ningún problema en encontrar la sección, después de elegir la ruta en cuestión el usuario ha encontrado intuitivo el mapa, ya que con un simple clic puede acceder a la información.

- **Tarea 2: Busca un lugar en el mapa interactivo**
  El menú lo ha encontrado sin problemas, después durante la navegación dentro del mapa al usuario le ha costado interactuar debido a un *bug* que está pendiente de solucionar donde no se centra bien la ventana

- **Tarea 3: Busca un lugar en listado que esté a menos de 500 m**
  No ha tenido ningún problema en activar el filtrado por lejanía, aunque le ha costado un poco ajustar la barra justo a 500m
- **Tarea 4: Acceder Ficha y reproducir audioguia**
  No ha tenido problemas en realizar la tarea de manera satisfactoria

- **Nombre:** Ana Navarro

- **Edad:** 60 años

- **Tarea 1: Busca una ruta predefinida**
  Ha conseguido realizar la tarea correctamente (previa traducción), aunque en un tiempo algo mayor que el usuario nº1

- **Tarea 2: Busca un lugar en el mapa interactivo**
  Ha tenido cierta dificultad al orientarse en el mapa y buscar información sobre un monumento, ya que no está familiarizada con el uso de google maps, aun así ha conseguido realizar la tarea correctamente

- **Tarea 3: Busca un lugar en listado que esté a menos de 500 m**
  Ha resuelto la tarea de forma satisfactoría siendo mucho más intuitivo y sencillo esta forma de mostrar los monumentos

- **Tarea 4: Acceder Ficha y reproducir audioguia**
  El usuario ha elegido acceder a la ficha mediante la opción del listado de monumentos, una vez dentro de la ficha ha asociado intuitivamente el ícono del “play” con la reproducción de la audioguia.

3.4. Versiones de la aplicación/servicio

- Alpha/PEC 2
- Beta/PEC 3
- 1.0 / Entrega Final

3.5. Bugs

Después de haber realizado diferentes tipos de test hemos sacado las siguientes vulnerabilidades o fallos

- En la ficha de los monumentos la barra de reproducción no funciona correctamente
- La barra para configurar la lejanía no es del todo precisa
- En el mapa cuando hacemos clic sobre un punto el mapa no se centra correctamente

4. Conclusiones

El reto principal de este proyecto, era adquirir conocimientos de nuevas tecnologías basadas en el lenguaje typescript, como en este caso Angular 4, aunque previamente tenía conocimiento sobre AngularJS 1.5.X, el cambio a la hora de programar entre una versión y otra es notable debido al cambio de lenguaje JavaScript (ES5) VS TypeScript. La curva de aprendizaje ha sido algo empinada al principio siendo esta la parte más dura ya que estructuralmente es totalmente diferente, una vez aprendido como se estructura, la curva se estabiliza. Otros conocimientos aprendidos durante el desarrollo ha sido la configuración de un servidor gratuito basado en nodeJS en la nube a través de la plataforma Heroku. Por último también se ha aprendido el uso de componentes que nos ofrece Ionic para el desarrollo de la interfaz de la aplicación.

Los objetivos propuestos son a su vez requisitos de funcionales de la aplicación, por lo que se puede decir que se han cumplido satisfactoriamente, hay que decir que algunas tecnologías pensadas en un principio no se han podido implementar debido a las limitaciones del servidor gratuito donde se aloja la API, en concreto lo que no se ha podido implementar ha sido una base de datos con motor MySql ya que el “plugin” en la plataforma era de pago por lo que se ha tenido que cambiar a una base de datos con motor Postgresql, este cambio no ha tenido ninguna repercusión a la hora de desarrollar la aplicación, por último durante el transcurso del desarrollo se cambió el almacenamiento de forma local, en vez de crear una base de datos con SqlLite, se ha creado un almacenamiento con LocalStorage.

Para la planificación de este proyecto se eligió un modelo lineal secuencial en cascada, el hecho de haber elegido este modelo en vez de otros como pueden ser las metodologías ágiles es debido a que las entregas son cerradas e inflexibles en el tiempo, aparte que para utilizar una metodología ágil, sería muy recomendable que el cliente estuviera “in situ” durante el desarrollo. El uso de una metodología tradicional nos priva de realizar muchos cambios durante el proyecto. En la planificación no ha habido ningún problema ya que en el momento de gestionar los hitos se tuvo en cuenta la jornada laboral y festivos, todo esto ha ayudado a que no se quede ningún hito por cumplir y no se haya tenido que introducir ningún cambio en la planificación, sí que es verdad que si se hubiera contado con una metodología ágil, se hubiera tenido más flexibilidad si algún contratiempo no planificado hubiera ocurrido.

Durante el desarrollo de este proyecto, se han ido anotando en un bloc de notas las nuevas ideas que han surgido para la incorporación en un futuro, las posibles nuevas funcionalidades son las siguientes:

1. Extracción de datos API externas (Yelp,TripAdvisor)
   Aparte de nuestra propia base de datos donde se almacena la información sobre los monumentos, se añadirá más información de TripAdvisor o Yelp.
2. Añadir nuevos idiomas (Castellano,Alemán…)
   Se añadirá diferentes idiomas aparte del inglés, como es el castellano o el alemán
3. Añadir nuevas ciudades
   Se insertarán en nuestra base de datos más ciudades aparte de Valencia y se complementará la información con los datos que nos proporciones las API externas.
4. Añadir nuevas categorías
   Mediante las APIs externas añadiremos nuevas categorías al menú como puede ser restaurantes y bares de ocio

5. Glosario

5.1. Términos

- **Framework**: es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
- **API:** conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- **Front - end:** es la parte del software que interactúa con los usuarios
- **Back – end**: es la parte que procesa la entrada desde el front-end
- **Bootstrap:** framework web o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web
- **nodeJS:** entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación ECMAScript, asíncrono
- **MySql:** sistema de gestión de BD relacional bajo licencia dual GPL
- **SqlLite:** sistema de gestión de BD relacional compatible con ACID
- **LocalStorage:** memoria del navegador para guardar información de forma permanente.

5.2. Acrónimos

- SPA: acrónimo de Single Page Application o aplicación de una sola página es un sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios como una aplicación de escritorio

5. Bibliografía

[1]. (s.f.). Recuperado el 22 de Septiembre de 2017, de https://ionicframework.com/

[2]. (s.f.). Recuperado el 22 de Septiembre de 2017, de http://getbootstrap.com/

[3]. (s.f.). Recuperado el 22 de Septiembre de 2017, de https://nodejs.org/es/

[4]. (s.f.). Recuperado el 22 de Septiembre de 2017, de http://ngcordova.com/docs/plugins/sqlite/