Universidad de Valladolid

E.U. DE INFORMÁTICA (SEGOVIA)

Grado en Ingeniería Informática de Servicios y Aplicaciones

mySGuide: Aplicación Android para un recorrido monumental virtual por Segovia

Alumno: Ángel Barroso Sanz

Tutor: Jesús Cordobés Puertas

Tutor: Diego Martín de Andrés

RESUMEN

La sociedad actual gira en torno a la tecnología. Cada vez está más extendido el uso de dispositivos móviles tales como *tablets* o *smatrphones* que permiten a las personas hacer una infinidad de cosas, desde compras o consultas en la web hasta en algunos casos trabajar desde ellos.

Entre esos usos está el acceso continuo a información. Ya sea mediante el navegador web del dispositivo o a través de aplicaciones, a día de hoy estamos continuamente bombardeados por información.

Además, una pieza clave de ello es la geolocalización. Mediante la posición en la que se encuentre el usuario, puede tener acceso a diferentes tipos de información. Este sistema es aplicado tanto en *marketing* como en juegos, haciendo que el usuario del dispositivo tenga una experiencia satisfactoria.

Este proyecto propone una aplicación *Android* que sirva de guía al turista ofreciendo información sobre monumentos, leyendas festivales y museos. Cuenta con el añadido de que, en base a la geolocalización, el usuario recibirá notificaciones sobre monumentos o museos cercanos a su posición.

Índice de contenido

Índice de tablas

Índice de Ilustraciones

1. INTRODUCCIÓN

"Viajamos para cambiar, no de lugar, sino de ideas"

Hippolyte Taine(1828-1893) Filósofo francés.

Segovia es una ciudad con mucha historia. Está situada en un enclave óptimo de nuestra geografía, lo que ha provocado que prácticamente todas las culturas que han habitado la Península Ibérica hayan dejado marcada la ciudad, ya sea en forma de costumbres, de gastronomía o de monumentos. Todo comenzó con los primitivos habitantes celtas, que dejaron paso a la ocupación romana, cuya obra más representativa y que hace famosa nuestra ciudad a nivel mundial es el Acueducto, que sigue intacto a través de los años para admiración de los turistas que nos visitan. Pero en Segovia no sólo podemos encontrar obras de estas dos civilizaciones: dando un paseo por el casco antiguo encontramos restos de otras culturas como la judía, la árabe e incluso obras esplendorosas del cristianismo como nuestra Catedral,

Además de monumentos, Segovia es el lugar idóneo para disfrutar de una de las mejores gastronomías de nuestro país. Sus asados son antológicos y tienen fama mundial, gracias a personajes tan conocidos como el famoso Cándido y sus cochinillos. La multitud de asadores, restaurantes, bares y mesones que encontramos por sus calles suponen el complemento perfecto para cualquier turista, todo ello a un precio asequible.

Segovia cuenta además con la ventaja de estar muy bien comunicada. Al encontrarse prácticamente en el medio de la Península Ibérica, sus comunicaciones con la capital del estado, Madrid, son excelentes, ya sea mediante autobús, tren de alta velocidad (AVE) o incluso en coche.

Pero, ¿qué pretende este trabajo de fin de grado? Pues pretende que cualquier persona que visite la ciudad de Segovia tenga la oportunidad de acceder a toda la información posible sobre la parte cultural de la ciudad, más concretamente sus monumentos, mediante el uso de las nuevas tecnologías. Basándonos en la posición geográfica del usuario, se pretende informar al mismo de los lugares de interés que se encuentren a su alrededor. Bajo el nombre "***mySGuide***", a continuación describiremos en qué se basa este proyecto, como está construido y que uso tendrá.

1.1 Motivación

¿Qué es la geolocalización? Esta tecnología consiste en conocer la ubicación geográfica del usuario en todo momento. Una funcionalidad que cada vez es más importante, utilizada sobre todo en el mundo de los dispositivos móviles y en el desarrollo de aplicaciones para los mismos.

Es por esta razón que cada vez más *smartphones*[1], y no sólo los de gama alta, incluyen una funcionalidad denominada *GPS*[2] que sirve para poder determinar la posición localizada sobre el mapa.

Por ejemplo, utilizando la geolocalización del dispositivo móvil mediante el *GPS*, el usuario del dispositivo puede encontrar comercios cercanos, tiendas, cines, etc., o algo tan simple como determinar la franja horaria en la que se encuentra, información que el dispositivo obtiene de los satélites que orbitan la tierra.

Aunque también puede recibir información a través del *wi-fi* y *bluetooth* además del *GPS*, la localización final es una combinación de los datos obtenidos con esas herramientas. Es por eso que la ubicación puede que no siempre sea exacta, sino que será una ubicación aproximada.

¿Cómo aplicar esto a nuestro proyecto? Se pretende crear una aplicación *Android* que, haciendo uso de la geolocalización, informe al usuario sobre los lugares de interés, más concretamente monumentos y museos, mostrando información sobre los mismos. Se pretende también que *mySGuide* proporcione información sobre las leyendas locales más famosas, y que informe sobre los festivales más famosos de la ciudad, como Titirimundi o Muces.

Además de todo esto, una de las principales motivaciones es el aprendizaje de nuevas tecnologías, que se describirán en apartados siguientes, y profundizar en el estudio y desarrollo de tecnologías móviles.

1.2 Objetivos

Con esta aplicación se pretende que cualquier usuario que visite la ciudad de Segovia tenga una guía rápida en el bolsillo. Pretende informar sobre monumentos, contando su historia, construcción o anécdotas, y además, que la aplicación lance un aviso cuando el usuario se encuentre cerca de uno, en forma de notificación, como hace la muy conocida aplicación de mensajería *Whatsapp*. Pero no se queda ahí, dado que la aplicación quiere abarcar mucho más. Pretende también crear una guía de los museos de la ciudad, y al igual que los monumentos, cuando el usuario se encuentre cerca de uno, que se le avise mediante una notificación, y la aplicación contendrá información sobre el museo y

[1] Smartphone: tipo de teléfono móvil con mayor capacidad de almacenar datos y realizar actividades.
[2] GPS: Global Positioning System (Sistema de Posicionamiento Global)

la exposición (si la hubiera) de la que dispone en ese momento. También contará con información sobre festivales, como el famoso Titirimundi, con las fechas y programación del festival en su descripción. Y por último, para los más curiosos, la aplicación dispondrá de una sección dedicada a leyendas populares, como por ejemplo la más famosa de todas sobre la construcción del Acueducto por el diablo entre otras. Como se puede ver, esta aplicación pretende ser una guía cultural en el bolsillo. Y lo más ambicioso del proyecto es que con una sola instalación de la aplicación se pretende que el contenido de la misma se actualice solo, sin necesidad de actualizaciones, para que cuando el usuario la inicie, disponga de la nueva información (si la hubiera).

Esta aplicación está dirigida principalmente a cualquier usuario que visite la ciudad de Segovia. Está orientada a dispositivos móviles *Android*, en un rango de versiones que abarque a la mayoría de usuarios. Los objetivos principales del sistema son:

- **OB-01**: Ofrecer a toda persona que visite la ciudad de Segovia una alternativa virtual a los folletos informativos impresos en papel.
- **OB-02**: Ofrecer toda la información cultural relativa a monumentos, leyendas, festivales y museos de Segovia en una aplicación móvil para el Sistema Operativo Android.
- **OB-03**: Enviar notificaciones push al usuario cuando se encuentre cerca de un museo o monumento, ofreciéndole información relevante al mismo.
- **OB-04**: Desarrollar una aplicación que no necesite actualizaciones para actualizar su contenido.

Para el correcto cumplimiento de estos objetivos, se han establecido una serie de reglas de negocio:

- **RN-01**: El sistema debe funcionar en dispositivos móviles con Sistema Operativo Android.
- **RN-02**: La aplicación Android deberá ejecutarse sin problemas en dispositivos móviles con mínimo el API 16 (Android 4.1 Jelly Bean).
- **RN-03**: El usuario deberá disponer de una conexión a internet continua para poder visualizar los contenidos.
- **RN-04**: El sistema debe restringir que el usuario sólo realice las funciones permitidas.

1.3 Estado del arte

A lo largo de estos años han surgido multitud de aplicaciones sobre turismo en la ciudad de Segovia. A continuación, haremos un breve análisis sobre las aplicaciones que hacen competencia directa a nuestra propuesta de aplicación.

- **TourKhana Segovia**

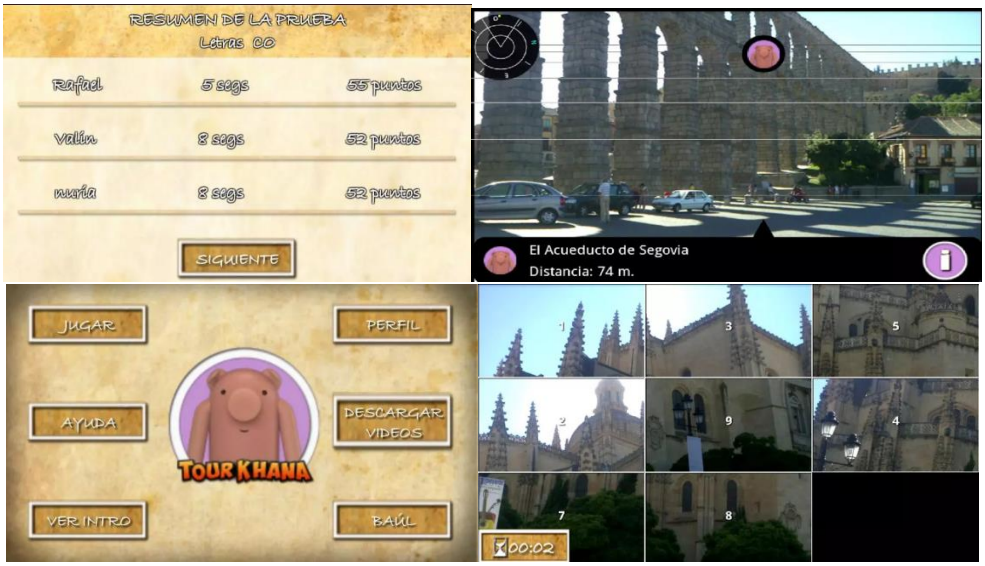

Ilustración 1.1 Imágenes aplicación TourKhana

Esta aplicación tiene por protagonistas a dos cochinillos y se sirve de la realidad aumentada para hacer un tour por la ciudad. Cuenta con retos, pasatiempos recompensas y juegos para que el usuario pueda conocer Segovia de una manera divertida. El recorrido de la aplicación comienza en el Acueducto y termina en la Vera Cruz, y cada parada cuenta con un vídeo y una prueba. Al completar todas las pruebas, la aplicación revela un secreto. Esta aplicación está disponible para *Android* e *iOS*.

- **App Segovia Guía Segovia**

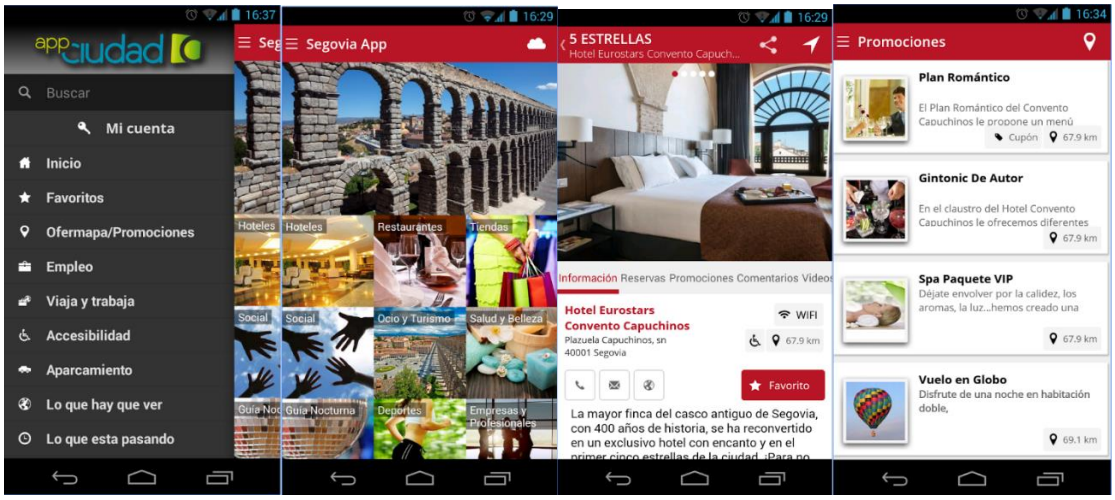

Ilustración 1.2 Imágenes aplicación App Segovia Guía Segovia

App Segovia Guía Segovia es una aplicación que ofrece información sobre promociones, eventos, ocio y restauración entre otros. Básicamente se centra en todo lo que ofrece la ciudad de Segovia. Es muy fácil de usar y tiene un diseño atractivo.

- **Segovia para todos**

Ilustración 1.3 Imágenes aplicación Segovia para todos

Es la aplicación oficial del organismo público "Turismo de Segovia". Esta ofrece una guía multimedia oficial, con visita guiada incluida. Ofrece un listado de puntos de interés donde el usuario puede elegir si visitarlos o no. Además, los contenidos de esta aplicación son accesibles para personas con discapacidad visual y auditiva.

- **Segovia Guía de turismo – S2go**

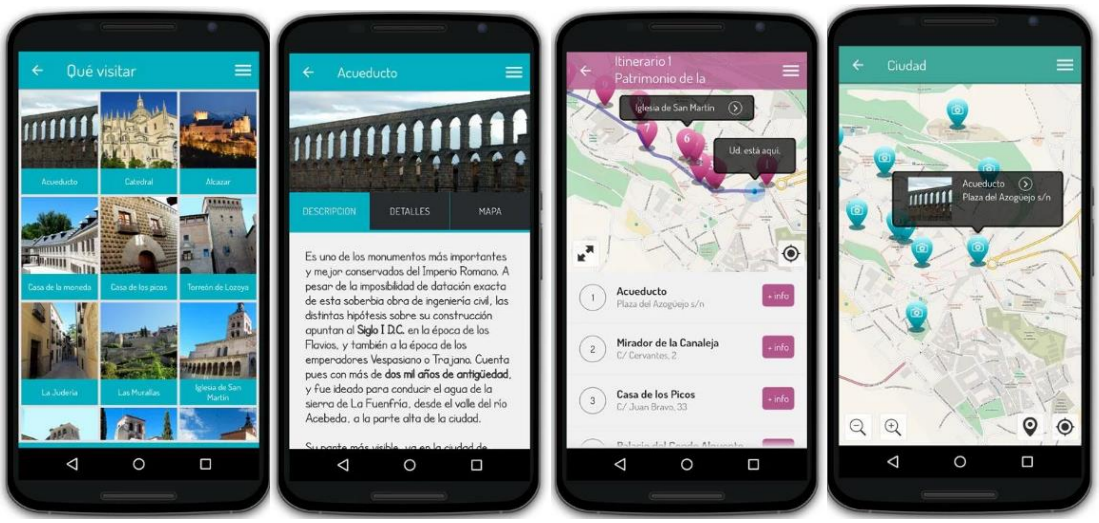

Ilustración 1.4 Imágenes aplicación S2go

Por último, y no por ella menos importante, tenemos la aplicación S2go. Esta aplicación es la más similar a la que nosotros queremos implementar. Muestra información sobre todos los monumentos, un mapa con su localización, y se basan en la geolocalización para ubicar al usuario en un mapa, señalando a su alrededor los monumentos y lugares de interés cercanos.

Habiendo instalado estas aplicaciones y habiéndolas probado, podemos realizar un análisis crítico de todas ellas:

- La mayoría de ellas están desactualizadas. Eso es un contra si estás ofreciendo un servicio que oferta actividades.
- Dos de ellas se centran sólo en monumentos. Una de ellas mediante juegos y otra mediante guías.
- De todas ellas solo la aplicación S2go dispone de geolocalización. Sin embargo, sólo la utiliza para situarte en un mapa, por lo que nuestra aplicación ofrece algo que las demás no.

Dicho esto, ¿qué puede ofrecer nuestra aplicación que las anteriores no? Pues ofrece lo siguiente:

- Información basada en la posición. Además de una guía que permitirá al lector ver información sobre los monumentos, si en un momento dado el usuario se encuentra en un lugar de interés, la aplicación dará un aviso al usuario dando la opción de ver la información.
- Información sobre los museos, con la exposición mostrada en el momento en que el usuario este en la ciudad.
- Uno de los puntos clave de la aplicación es que se actualiza sola. Con una única instalación, en caso de modificación del contenido, el usuario podrá disponer de esa nueva información.

1.4 Metodología

A la hora de elegir la metodología a utilizar, para este tipo de proyecto se llevará a cabo un modelo incremental[3], dado que no se conoce toda la información necesaria del proyecto al principio del mismo, y será necesario realizar varias iteraciones. Al final de cada iteración obtendremos un prototipo de la aplicación, que se irá completando según avance el ciclo de vida.

Se ha definido una iteración cada dos semanas, y cada iteración consta de 5 fases:

- Análisis
- Diseño
- Implementación
- Pruebas
- Despliegue

Añadimos la fase despliegue porque al ser una aplicación *Android*, necesitamos instalarla en el dispositivo y probar que todo lo introducido cumple los requisitos establecidos. Además, la primera iteración cuenta con una fase extra de planificación.

Este modelo, además de ser el más utilizado, cuenta con la ventaja de que la primera iteración cumple los requisitos críticos, es decir, la primera iteración proporciona un prototipo funcional, y según se van añadiendo iteraciones, vamos complementando ese prototipo inicial.

Como desventaja, muchos de los requisitos que definimos en una primera iteración no son válidos al inicio de la siguiente, lo que supone que para cada nueva iteración habrá que definir de nuevo los requisitos, o modificar los existentes.

1.5 Organización del documento

El siguiente documento tendrá la siguiente organización:

- Un primer capítulo de introducción, donde se explica al lector el origen del proyecto, el estado del arte y la metodología utilizada.
- Un segundo capítulo dedicado a la planificación del proyecto y al coste.
- Un tercer capítulo donde se explicará al lector la tecnología principal de desarrollo del proyecto.
- Un cuarto capítulo donde se analizará el sistema y se dispondrán los diagramas necesarios para su entendimiento.

[3] Fuente: apuntes de la asignatura Proceso de Desarrollo de Software de F. José González Cabrera.

- Un quinto capítulo de diseño donde se explicará al lector el tipo de arquitectura que tiene el proyecto, funcionamiento del mismo y donde se mostrarán los prototipos del proyecto.
- Un sexto capítulo dedicado a la implementación, donde se explicará cómo se hizo el proyecto desde cero.
- Un séptimo capítulo dedicado a manuales.
- Un octavo capítulo dedicado a conclusiones y futuras mejoras
- Un noveno y último capítulo dedicado a poner referencias y enlaces de interés

1.6 Contenido del CD-ROM

El CD-ROM entregado con esta documentación contendrá dos carpetas, una llamada "Documentación" y otra llamada "Software".

- Carpeta documentación: contendrá la versión digital en PDF de esta documentación.
- Carpeta Software: contendrá el proyecto entero, a saber
    - Api Rest: todos los archivos que conforman el API Rest en formato comprimido .rar.
    - mySGuide: todos los archivos necesarios que conforman la aplicación en formato comprimido .rar.
    - mySGuide.apk: archivo de instalación de Android

2. PLANIFICACIÓN Y PRESUPUESTO

Como se ha comentado en el apartado anterior, en la metodología vamos a utilizar un modelo de desarrollo incremental. Para ello, vamos a establecer una serie de iteraciones con sus correspondientes intervalos de tiempo. Para ello, utilizaremos un diagrama de Gantt.

Nombre	Fecha de inicio	Fecha de fin
mySGuide	5/04/16	23/06/16
Iteración 1	5/04/16	19/04/16
Planificación	5/04/16	6/04/16
Análisis	7/04/16	8/04/16
Diseño	11/04/16	14/04/16
Implementación	14/04/16	18/04/16
Pruebas	18/04/16	18/04/16
Despliegue	18/04/16	19/04/16
Iteración 2	20/04/16	3/05/16
Análisis	20/04/16	21/04/16
Diseño	22/04/16	25/04/16
Implementación	26/04/16	29/04/16
Pruebas	2/05/16	2/05/16
Despliegue	2/05/16	3/05/16
Iteración 3	4/05/16	18/05/16
Análisis	4/05/16	6/05/16
Diseño	9/05/16	10/05/16
Implementación	11/05/16	13/05/16
Pruebas	16/05/16	16/05/16
Despliegue	17/05/16	18/05/16
Iteración 4	20/05/16	3/06/16
Análisis	20/05/16	20/05/16
Diseño	23/05/16	23/05/16
Implementación	24/05/16	30/05/16
Pruebas	31/05/16	31/05/16
Despliegue	2/06/16	3/06/16
Iteración 5	6/06/16	23/06/16
Análisis	6/06/16	8/06/16
Diseño	9/06/16	12/06/16
Implementación	13/06/16	21/06/16
Pruebas	22/06/16	22/06/16
Despliegue	22/06/16	23/06/16

Ilustración 2.1 Planificación de iteraciones

Y por otro lado, el diagrama de Gantt:

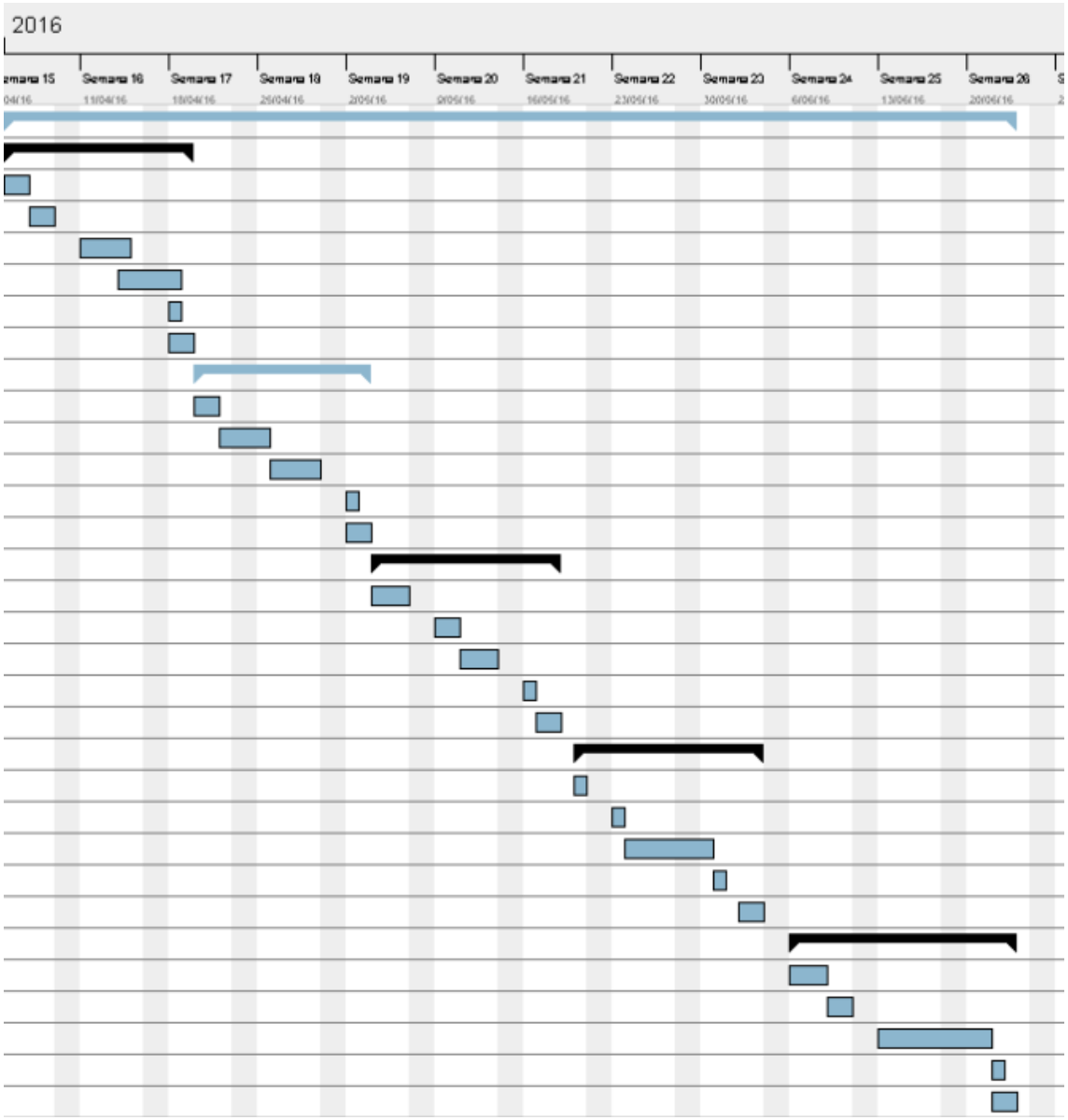

Ilustración 2.2 Diagrama de Gantt de las iteraciones del proyecto

Como se observa en el gráfico anterior, la planificación de desarrollo del proyecto dura desde el 4 de Abril de 2016 hasta el 23 de Junio. Eso supone un total de **79 días** de desarrollo. Suponiendo que cada día tiene 8 horas laborales, se estima que la realización del proyecto tendrá un total de **632 horas** de desarrollo.

2.2 Presupuesto

En este apartado realizaremos la estimación de presupuesto mediante diferentes métodos.

- **Estimación de costes**

Este método de estimación ha de tener en cuenta los medios *Hardware* y *Software*, cuyo coste será proporcional al uso que se le haya dado en el proyecto. También habrá que tener en cuenta los costes en recursos humanos. Dado que se van a realizar tareas de analista, diseñador, programador y documentador, se utilizará el perfil de un programador *Fullstack*. Los desarrolladores *fullstack* ofrecen el paquete completo, ya que tienen capacidad para realizar todas las tareas mencionadas anterirormente. Dado que solo una persona se va a encargar de todo, no hace falta desglosar el número de horas que se dedican a cada parte.

- Un desarrollador Fullstack cobra 26.000 € brutos al año, lo que significa que a la empresa ese trabajador le cuesta 33.800 € al año(incluyendo el pago a la seguridad social). Si dividimos ese salario entre 14 pagas y calculamos el precio por hora de ese trabajador, el coste aproximado por hora nos sale a unos **15,08 € / hora**.
- La estimación de horas del proyecto es de **632 horas.**

	Tiempo	**Coste**
Desarrollador Fullstack	632 horas	15,08 € / hora
TOTAL	632 horas	**9.530,56 €**

Tabla 2.1 Estimación de RRHH

Además del trabajo humano, se deben disponer de las herramientas necesarias para ello. A continuación, mostramos dos tablas que indican los costes de *software* y *hardware* del proyecto.

Hardware	Uso (%)	Coste total (€)	Coste(€)
Acer Aspire 771G	7%	600	42
Xiaomi MI3 W	9%	325	29,25
Internet	15%	55	8,25
TOTAL			**79,50 €**

Tabla 2.2 Estimación de costes de hardware

Software	Uso (%)	Coste total (€)	Coste(€)
Atom	10%	0	0,00
Windows 10	9%	0	0,00
Apache Cordova	75%	0	0,00
Google Chrome	5%	0	0,00
TOTAL			**0,00 €**

Tabla 2.3 Estimación de costes de software

También se debe tener en cuenta que para publicar la aplicación en los markets de Google hay que tener una licencia de desarrollador. Esta licencia solo se paga una vez y es de por vida, por lo que se incluye en el presupuesto.

Otros	Uso (%)	Coste total (€)	Coste(€)
Licencia Google	100%	25	25,00
TOTAL			**25,00 €**

Tabla 2.4 Otros gastos

Una vez calculadas todas las estimaciones, para obtener el presupuesto total del proyecto por estimación de costes, basta con sumar los valores obtenidos anteriormente

Estimación del coste del proyecto = Coste estimado RRHH + Coste estimado Hardware + Coste estimado Software + Otros gastos

Estimación del coste del proyecto= 9.530,56 € + 79,50 € + 0,00 € + 25,00 €

Estimación del coste del proyecto = 9.635,06 €

- **Estimación mediante puntos de función**

Este método de aproximación se utiliza para medir el tamaño del software. Pretende medir la funcionalidad independientemente de la tecnología utilizada. Los parámetros que sirven para medir esa funcionalidad son los siguientes:

- *Número de entrada*s: Datos que el usuario aporta al sistema (nombres de ficheros, menús de selección).
- *Número de salidas*: Datos que el sistema aporta al usuario (informes, mensajes).
- *Número de ficheros lógicos internos*: Ficheros o bases de datos internos del sistema.
- *Número de ficheros externos*: Ficheros o bases de datos externos al sistema.
- *Número de consultas externas*: Entradas que requieren de una respuesta por parte del sistema.

Para hallar los PFNA[4] se debe contar el número de elementos de cada clase descrita anteriormente. Para ello, nos fijamos en la siguiente tabla:

Ficheros lógicos externos e internos				**Salidas y consultas**				**Entradas**			
Registros Elementales	Datos Elementales			Tipos de Ficheros	Datos Elementales			Tipos de Ficheros	Datos Elementales		
	1-9	20-50	>51		1-9	6-19	>20		1-4	5-15	>16
1	Baja	Baja	Media	0-1	Baja	Baja	Media	0-1	Baja	Baja	Media
2-5	Baja	Media	Alta	2-3	Baja	Media	Alta	2-3	Baja	Media	Alta
>6	Media	Alta	Alta	>4	Media	Alta	Alta	>3	Media	Alta	Alta

Tabla 2.5 Grados de complejidad de PF

[4] Puntos de Función no Ajustados

	Complejidad Baja	Complejidad Media	Complejidad Alta	TOTAL
Entradas	4x3	1x4	0x6	16
Salidas	0x4	4x5	0x7	20
Consultas	40x3	4x4	1x6	142
Ficheros Internos	0x7	0x10	0x15	0
Ficheros Externos	4x5	0x7	0x10	20
TOTAL				**198**

Tabla 2.6 Valores de complejidad de PF

A través de las dos tablas anteriores, se obtienen los PFNA, que para este proyecto son **198**. Ahora, a través de la siguiente tabla se consigue el factor de complejidad para obtener los **PFA**[5].

[5] Puntos de Función Ajustados

Factor de Complejidad (FC)	0-5
Comunicación de datos	4
Rendimiento	5
Frecuencia de transacciones	5
Requisitos de manejo del usuario final	1
Procesos complejos	2
Facilidad de mantenimiento	1
Instalación en múltiples lugares	0
Funciones distribuidas	0
Gran carga de trabajo	1
Entrada on-line de datos	0
Actualizaciones on-line	3
Interacción con otros sistemas	3
Facilidad de operación	2
Facilidad de cambio	2

Tabla 2.7 Factores de complejidad

Aplicando la siguiente fórmula, obtenemos el factor de ajuste

$$\mathbf{FA} = (0{,}01 * \sum \mathrm{FC}) + 0{,}65$$

$$\mathbf{FA} = \mathbf{0{,}94}$$

Una vez obtenido el factor de ajuste, se calculan los **PFA** :

$$\mathbf{PFA} = \mathbf{PFNA} * \mathbf{FA}$$

$$\mathbf{PFA} = \mathbf{186{,}12}$$

Teniendo en cuenta que la aplicación va a ser programada en *Javascript*, y que el coste de la línea de código por punto de función[6] es de una media de 47 líneas de código

186,12 *47 = **8.748 líneas de código (LDC)**

[6] Fuente http://www.qsm.com/resources/function-point-languages-table

- **Estimación mediante COCOMO**

Para realizar el modelado algorítmico de costes se va a llevar a cabo un COCOMO **Empotrado**, ya que a pesar de que el proyecto tiene requisitos restrictivos y existen presiones de tiempo, tenemos experiencia en el entorno a desarrollar.

Proyecto Software	a	b	c	d
Orgánico	2,40	1,05	2,50	0,38
Semi-acoplado	3,0	1,12	2,5	0,35
Empotrado	3,20	1,20	2,50	0,32

Tabla 2.8 Tipos de COCOMO

Primero calculamos el esfuerzo nominal :

Esfuerzo nominal (En) = $a*(KLDC)^b$

En = 43,19

Necesitamos calcular también los factores conductores de coste (FCC). Mediante la siguiente tabla, obtenemos el valor deseado:

Factores Conductores del Coste		Valor de los factores					
		Muy bajo	Bajo	Medio	Alto	Muy alto	Extra
Software	Fiabilidad del software requerido	0,75	0,88	1,00	1,15	1,4	
	Tamaño de la base de datos		0,94	1,00	1,08	1,16	
	Complejidad del software	0,70	0,85	1,00	1,15	1,30	1,65
Hardware	Restricciones de rendimiento en tiempo de ejecución			1,00	1,11	1,30	1,66
	Restricciones de memoria			1,00	1,06	1,21	1,56
	Volatilidad del entorno de la máquina virtual		0,87	1,00	1,15	1,30	
	Tiempo de respuesta requerido		0,87	1,00	1,07	1,15	
Personal	Capacidad de los analistas	1,46	1,19	1,00	0,86	0,71	
	Experiencia con el tipo de aplicación	1,29	1,13	1,00	0,91	0,82	
	Experiencia con el hardware	1,21	1,10	1,00	0,90		
	Experiencia con el lenguaje de programación	1,14	1,07	1,00	0,95		
	Capacidad de los programadores	1,42	1,17	1,00	0,86	0,70	
Proyecto	Técnicas modernas de programación	1,24	1,10	1,00	0,91	0,82	
	Utilización de herramientas software	1,24	1,10	1,00	0,91	0,83	
	Restricciones en la planificación temporal del desarrollo	1,23	1,08	1,00	1,04	1,10	

Ilustración 2.1 Factores de coste COCOMO

FCC = 1,15*0,94*1*1*1*1*1,07*1*1*0,90*0,95*0,86*0,91*0,91*1,04

FCC=0,73

Con estos valores, ya se puede calcular el esfuerzo y el tiempo de desarrollo

Esfuerzo (E) = En * FCC

E = 31,63 personas/ mes

Tiempo de Desarrollo (TD) = $c*(E)^d$

TD = 7,55 meses

El nº medio de personas es **E/TD** = 31,63 / 7,55 = 4,18, o lo que es lo mismo, **4 personas realizan el trabajo en 7 meses.**

2.2 Coste final

Partiendo de una planificación inicial con **79 días** estimados de desarrollo, a la hora de la verdad sólo se han necesitado **58 días**. Por tanto, lo único que varía en el presupuesto inicial son las horas trabajadas, que suman un total de **464 horas** frente a las 632 planteadas inicialmente. Por tanto, teniendo en cuenta los datos definidos en el apartado anterior, los costes reales del proyecto son:

- Un desarrollador Fullstack cobra **15,08 € / hora**.
- Las horas reales del proyecto son **264 horas.**

	Tiempo	**Coste**
Desarrollador Fullstack	464 horas	15,08 € / hora
TOTAL	464 horas	**6.997,12 €**

Tabla 2.10 Coste real de RRHH

Hardware	**Uso (%)**	**Coste total (€)**	**Coste(€)**
Acer Aspire 771G	6%	600	36
Xiaomi MI3 W	10%	325	32,25
Internet	25%	55	13,75
TOTAL			**82 €**

Tabla 2.14 Coste real de hardware

Software	Uso (%)	Coste total (€)	Coste(€)
Atom	10%	0	0,00
Windows 10	9%	0	0,00
Apache Cordova	75%	0	0,00
Google Chrome	5%	0	0,00
TOTAL			**0,00 €**

Tabla 2.15 Coste real de software

Otros	Uso (%)	Coste total (€)	Coste(€)
Licencia Google	100%	25	25,00
TOTAL			**25,00 €**

Tabla 2.16 Coste real de otros gastos

Una vez calculadas todos los costes reales, para obtener el coste real total del proyecto, basta con sumar los valores obtenidos anteriormente

Coste real del proyecto = Coste real RRHH + Coste real Hardware + Coste real Software + Coste real otros gastos

Coste real del proyecto= 6.997,12 € + 82 € + 0,00 € + 25,00 €

Coste real del proyecto = 7.104,12 €

Una vez terminado el proyecto se ha visto que los días estimados eran demasiados, lo que ha hecho que el coste en recursos humanos haya bajado considerablemente.

3. APACHE CORDOVA

Apache Cordova es la herramienta principal sobre la que desarrollaremos nuestro proyecto. Tras una primera versión lanzada en el año 2009, este marco de desarrollo no ha hecho más que crecer y evolucionar hasta llegar a día de hoy como uno de los métodos de desarrollo de aplicaciones más rápido y asequible para las empresas.

3.1 Descripción

Apache Cordova es un *framework*[7] de código abierto que sirve para desarrollar aplicaciones para móviles. Cada vez está tomando más importancia, ya que con una sola vez que se programe la aplicación, se podrá disponer de ella en la gran mayoría de dispositivos móviles. Apache Cordova a día de hoy permite el desarrollo de aplicaciones en las siguientes plataformas:

- Amazon Fire OS.
- Android.
- BlackBerry 10.
- Firefox OS.
- iOS.
- Ubuntu.
- Windows Phone.
- Windows 8.
- Tizen.

Todo esto es posible gracias a su manera de construir aplicaciones. La aplicación se construye como una página web, con archivos *HTML*, *CSS* y *Javascript*. Cordova genera un *WebView*[8] a través del *SDK*[9] de la plataforma deseada, lo que permite la visualización de nuestra aplicación en los distintos sistemas operativos.

¿Por qué hemos elegido Cordova para construir una aplicación *Android*? Además de porque *Android* es uno de los sistemas operativos móviles más extendidos entre los

[7] Framework: es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular.
[8] WebView: es un navegador web integrado dentro de una aplicación.
[9] SDK: conjunto de herramientas software que permite al programador crear aplicaciones para un sistema concreto.

usuarios, Cordova nos permite desarrollar una aplicación móvil para un dispositivo en un tiempo bastante menor que si se tratase del lenguaje nativo del sistema operativo móvil concreto (*Java* en caso de *Android*, *Objetive-C* en caso de *iOS*, por ejemplo). Otra de las razones que se ha mencionado anteriormente, es que con una sola programación de la aplicación, en caso de que nuestro proyecto tuviera éxito comercial y quisiéramos disponer de esa aplicación en más sistemas operativos, bastaría con añadir una plataforma más y construir la aplicación, de tal manera que los costes de desarrollo que supondrían migrar una aplicación de un sistema operativo a otro serían prácticamente nulos, y podríamos disfrutar de la aplicación en un plazo inferior a una semana, cosa que a día de hoy supone un gasto de recursos enormes.

Sin embargo, como contrapartida a tantos beneficios, cabe decir que las aplicaciones Cordova no ofrecen los mismos rendimientos que una aplicación nativa,y lo respectivos *"stores*[10]*"* de cada sistema operativo suelen ser más estrictos y ponen más trabas con este tipo de aplicaciones, ya que muchas veces no queda clara la funcionalidad de la misma.

3.2 Estructura de un proyecto Cordova

Todas las aplicaciones Cordova se construyen en torno a un archivo común, denominado *config.xml*. Este archivo proporciona información acerca de la versión de la aplicación, autor de la misma, y sistemas operativos en los que se construye, así como los parámetros que afectan a cómo funciona, como responde dependiendo de la orientación del dispositivo o que hace cuando el dispositivo está en movimiento. También este archivo indica desde dónde se iniciará la aplicación.

Además de este archivo, un proyecto Cordova está formado por varias carpetas, cada una con su función específica:

- Carpeta *Hooks*: contiene archivos que representan secuencias de comandos especiales que podrían ser añadidos por la aplicación y desarrolladores de plugins o incluso por su propio sistema de construcción para personalizar los comandos de Cordova.
- Carpeta *Platforms*: contiene todos los sistemas operativos en los que dispondremos de la aplicación. Cada sistema operativo crea una carpeta con los archivos necesarios para generar el paquete instalador en el dispositivo móvil.
- Carpeta *Plugins*: contiene los plugins adicionales que queramos añadir a nuestra aplicación. En un apartado posterior se explicará más sobre esta funcionalidad.
- Carpeta *Www*: contiene toda la parte de programación de la aplicación y su funcionalidad, es decir, en esta carpeta se encuentran todos los archivos que se programarán.

3.3 Apache Cordova en mySGguide

[10] Store: tienda virtual desde la que se distribuyen aplicaciones en los diferentes sistemas operativos.

Una vez explicado que es Apache Cordova, cómo funciona y de qué se compone, es momento de explicar cómo afecta a nuestro proyecto. Como se ha descrito en un apartado anterior, un proyecto Cordova está compuesto por diferentes elementos, uno de ellos la carpeta *"www"*. En esta carpeta introduciremos el código de nuestra aplicación, de manera que programando una página web adaptada para móviles, Cordova se encargará de transformar esa página en una aplicación funcional.

El tipo de página web elegido es una de tipo *MEAN* , cuyas siglas representan a las tecnologías con las que está implementada (*MongoDB, Express, Angular* y *Node)* y que se explicarán en apartados posteriores. El lado cliente (*Android*) serán los archivos que contenga la aplicación Cordova en su carpeta. Se ha elegido este tipo de página porque funciona muy bien con la tecnología de Cordova, porque son lenguajes "punteros" con no más de 10 años de existencia y porque para nuestro tipo de aplicación ofrece un rendimiento óptimo, además de que los gastos de datos son muy bajos.

3.4 Plugins utilizados

Uno de los principales atractivos de Cordova es su extensa cantidad de plugins[11]. Estos plugins dotan de funcionalidades extra a las aplicaciones que por defecto una aplicación Cordova no tiene. Pueden ser oficiales (creados por los desarrolladores de Cordova) o creados por la extensa comunidad de usuarios de Cordova, ya que al ser una tecnología de código abierto cualquiera puede aportar su granito de arena. Estos plugins además tienen la peculiaridad de que permiten a la aplicación comunicarse con los componentes nativos del dispositivo, tales como cámara de fotos, acelerómetro o GPS entre otros.

Para la realización de este proyecto ha sido necesaria la inclusión de varios plugins, ya que por defecto la aplicación Cordova no proporcionaba las herramientas suficientes para los problemas planteados. Los plugins seleccionados para la realización del proyecto han sido los siguientes:

- *Plugin Device*: plugin instalado de manera predeterminada a la hora de crear un nuevo proyecto Cordova. Este plugin es el encargado de reconocer el dispositivo móvil desde el que se ejecuta la aplicación y puede acceder a información como el *UUID*[12] o el sistema operativo bajo el que está funcionando.
- *Plugin WhiteList*: otro de los plugins instalado de manera predeterminada a la hora de crear un proyecto. Este plugin es el encargado de autorizar los links por los que está formada la aplicación, de manera que no nos dirija (sin permiso del usuario) a sitios webs de terceros.

[11] Plugin: complemento de una aplicación que se relaciona con otra para aportarle una funcionalidad nueva y generalmente muy específica.
[12] UUID: Universally Unique Identifier (Identificador Único Universal)

- *Plugin Local-Notification*: plugin añadido al proyecto. Su función es enviar una notificación *push*[13] cuando el usuario se encuentre cerca de un lugar de interés. Este plugin ha sido creado por un desarrollador externo al proyecto Cordova.

[13] Notificación push: forma de comunicación en la que una aplicación envía un mensaje a un cliente.

4. ANÁLISIS DEL SISTEMA

4.1 Características principales

En este apartado se describen las características que conforman nuestro proyecto. Para ello, hay que ponerse en la piel del usuario y en cómo y de qué manera lo utilizarán para así poder abarcar todas las posibilidades que puede ofrecer el proyecto. A continuación, se listan las características del software a desarrollar

- **C-01**: Gestión de monumentos
- **C-02**: Gestión de museos
- **C-03**: Gestión de festivales
- **C-04**: Gestión de leyendas
- **C-05**: Gestión de notificaciones

Para hacerlo de manera más visual para el lector, a continuación se adjunta un diagrama que muestra el árbol de características del sistema, para crear una imagen global del sistema y de todos los subsistemas que lo conforman.

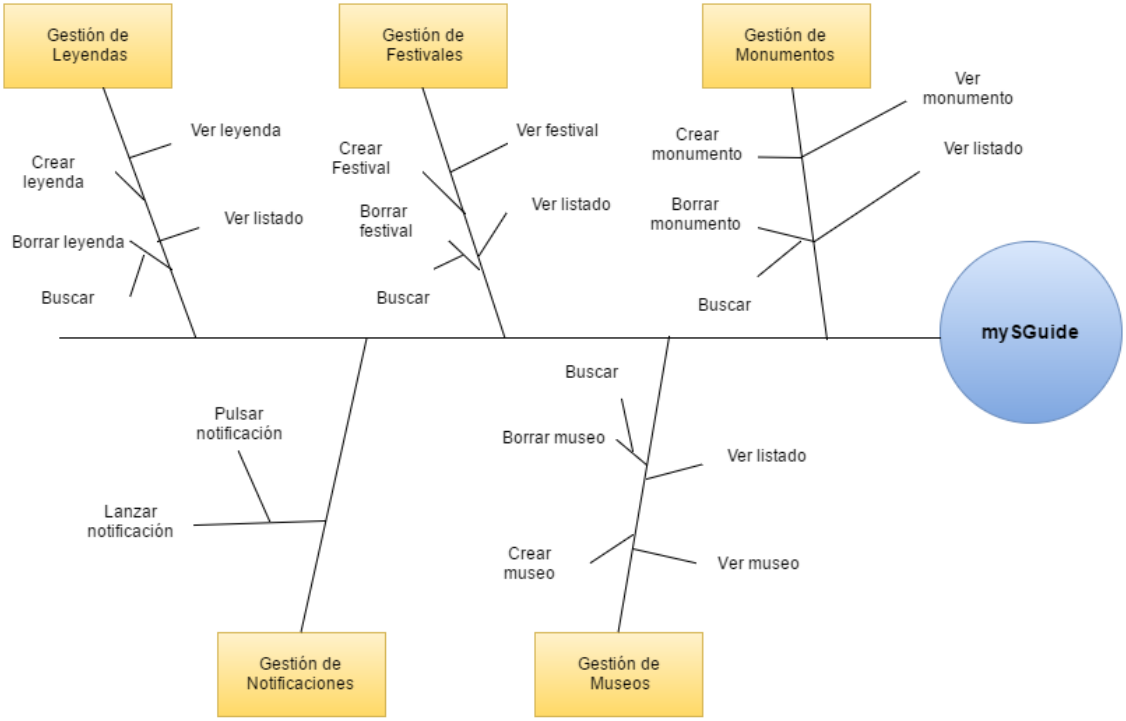

Ilustración 4.1 Árbol de características del sistema

Como se aprecia en la Ilustración anterior, tenemos 5 subsistemas diferenciados. En cada uno de ellos, las acciones de "Ver información" (de cada respectivo subsistema) y la

acción “Ver listado” corresponden a la parte cliente, ya que es el usuario el que las realiza. Por otro lado, todas las acciones de “Crear” y “Borrar” y “Buscar” pertenecen al lado servidor, y son realizadas por el administrador del sistema. El último subsistema llamado “Gestión de notificaciones” pertenece al lado cliente.

4.2 Identificación de los actores del sistema

Denominamos actor del sistema a toda persona u entidad externa al sistema que interactúa con el mismo y realiza casos de uso para efectuar tareas. Normalmente cada actor corresponde a una clase de usuario con diferentes roles. A continuación, se especifican los actores que formarán parte del sistema.

ACT–01	***Usuario***
Versión	*1.0*
Autores	*Ángel Barroso Sanz*
Descripción	*Usuario que hace uso normal de la aplicación.*
Comentarios	

Tabla 4.1 Tabla de actor 1

ACT–02	***Cliente de Api Rest***
Versión	*1.0*
Autores	*Ángel Barroso Sanz*
Descripción	*Entidad que recibe peticiones y devuelve información.*
Comentarios	

Tabla 4.2 Tabla de actor 2

ACT–03	***Usuario Administrador***
Versión	*1.0*
Autores	*Ángel Barroso Sanz*
Descripción	*Tipo de usuario que se encarga del mantenimiento de la base de datos del sistema*.
Comentarios	*Este tipo de usuario hereda todas las características del actor 01*

Tabla 4.3 Tabla de actor 3

4.3 Requisitos de usuario

Se ha decidido modelar los requisitos de usuario en forma de casos de uso.

4.3.1 Casos de uso

A continuación se mostrará una lista de los casos de uso, divididos por actores.

- Actor 01 – Usuario
  - **UC-01** Ver listado de monumentos
  - **UC-02** Ver información de monumento
  - **UC-03** Ver listado de museos
  - **UC-04** Ver información de museo
  - **UC-05** Ver listado de festivales
  - **UC-06** Ver información de festival
  - **UC-07** Ver listado de leyendas
  - **UC-08** Ver información de leyenda
  - **UC-09** Solicitar información recibida
- Actor 02 – Cliente API Rest
  - **UC-10** Obtener JSON Listado monumentos
  - **UC-11** Obtener JSON Monumento
  - **UC-12** Obtener JSON Listado Museos
  - **UC-13** Obtener JSON Museo
  - **UC-14** Obtener JSON Listado Festivales
  - **UC-15** Obtener JSON Festival
  - **UC-16** Obtener JSON Listado Leyendas
  - **UC-17** Obtener JSON Leyenda
- Actor 03 – Usuario administrador
  - **UC-18** Login
  - **UC-19** Logout
  - **UC-20** Crear monumento
  - **UC-21** Consulta listado de Monumentos
  - **UC-22** Buscar monumento
  - **UC-23** Borrar monumento
  - **UC-24** Crear festival
  - **UC-25** Consulta listado de festivales
  - **UC-26** Buscar festival
  - **UC-27** Borrar festival
  - **UC-28** Crear museo
  - **UC-29** Consulta listado de museos
  - **UC-30** Buscar museo
  - **UC-31** Borrar museo
  - **UC-32** Crear leyenda
  - **UC-33** Consulta listado de leyendas
  - **UC-34** Buscar leyenda
  - **UC-35** Borrar leyenda

4.3.2 Diagrama de casos de uso

A continuación, se muestra el diagrama de casos de uso del sistema

Ilustración 4.2 Diagrama de casos de uso del sistema

Una vez con el diagrama, podemos dividir los actores en dos partes. El actor Usuario será el que interactúe en la parte cliente, mientras que el actor usuario administrador y el actor cliente API Rest actuarán en la parte servidor. Eso no quiere decir que el administrador no pueda actuar en la parte cliente. Si dispone de un dispositivo con la aplicación instalada, puede realizar todas las funciones de un usuario normal, de ahí que herede los casos de uso de usuario.

4.3.3 Especificación de casos de uso

En esta sección se incluirán las tablas con las especificaciones de los casos de uso definidos anteriormente.

UC-01	***Ver listado monumentos***	
Versión	1.0	
Actor principal	ACT-01, ACT-02	
Actor secundario	-	
Trigger	• TR-01 Pulsar botón “Monumentos”	
Descripción	El usuario visualiza un listado de los monumentos registrados en la aplicación	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Monumentos”.
	p3	La aplicación solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La aplicación muestra el listado de monumentos.
Postcondiciones	El usuario visualiza la lista de monumentos	
Excepciones	**Paso**	**Acción**
	p4	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Muy alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.4 Especificación del caso de uso “Ver listado de monumentos”

UC-02	***Ver información de monumento***	
Versión	1.0	
Actor principal	ACT-01, ACT-02	
Actor secundario	-	
Trigger	• TR-01 - Pulsar monumento • TR-02 - Pulsar notificación	
Descripción	El usuario visualiza información sobre el monumento seleccionado	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Monumentos”.
	p3	La aplicación muestra el listado de monumentos.
	p4	El usuario selecciona un monumento
	p5	La aplicación solicita información sobre ese monumento al API Rest
	p6	El API Rest envía la información solicitada
	p7	La aplicación muestra la información sobre el monumento
Postcondiciones	El usuario visualiza información sobre un monumento	
Excepciones	**Paso**	**Acción**
	p5	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Secuencia alternativa	**Paso**	**Acción**
	p1	El usuario recibe una notificación
	p2	El usuario pulsa la notificación
	P3	*La aplicación solicita información sobre ese monumento al API Rest*
	P4	*El API Rest envía la información solicitada*
	P5	*La aplicación muestra la información sobre el monumento*
Frecuencia	*Muy alta*	
Importancia	*Muy Alta*	
Comentarios	Este caso de uso extiende de UC-01, por lo que puede no realizarse.	

Tabla 4.5 Especificación de caso de uso “Ver información de monumento”

UC-03	***Ver listado de museos***	
Versión	1.0	
Actor principal	ACT-01, ACT-02	
Actor secundario	-	
Trigger	• TR-01 Pulsar botón “Museos”	
Descripción	El usuario visualiza un listado de los Museos registrados en la aplicación	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Museos”.
	p3	La aplicación solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La aplicación muestra el listado de Museos.
Postcondiciones	El usuario visualiza la lista de museos	
Excepciones	**Paso**	**Acción**
	p4	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Muy alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.6 Especificación del caso de uso “Ver listado de museos”

UC-04	***Ver información de museo***	
Versión	1.0	
Actor principal	ACT-01, ACT-02	
Actor secundario	-	
Trigger	• TR-01 - Pulsar museo • TR-02 - Pulsar notificación	
Descripción	El usuario visualiza información sobre el museo seleccionado	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Museos”.
	p3	La aplicación muestra el listado de museos.
	p4	El usuario selecciona un museo
	p5	La aplicación solicita información sobre ese monumento al API Rest
	p6	El API Rest envía la información solicitada
	p7	La aplicación muestra la información sobre el museo
Postcondiciones	El usuario visualiza información sobre un museo	
Excepciones	**Paso**	**Acción**
	p5	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Secuencia alternativa	**Paso**	**Acción**
	p1	El usuario recibe una notificación
	p2	El usuario pulsa la notificación
	P3	La aplicación solicita información sobre ese museo al API Rest
	P4	El API Rest envía la información solicitada
	P5	La aplicación muestra la información sobre el museo
Frecuencia	*Muy alta*	
Importancia	*Muy Alta*	
Comentarios	Este caso de uso extiende de UC-03, por lo que puede no realizarse.	

Tabla 4.7 Especificación de caso uso “Ver información de museo”

UC-05	***Ver listado de festivales***	
Versión	1.0	
Actor principal	ACT-01, ACT-02	
Actor secundario	-	
Trigger	• TR-01 Pulsar botón “Festivales”	
Descripción	El usuario visualiza un listado de los festivales registrados en la aplicación	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Festivales”.
	p3	La aplicación solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La aplicación muestra el listado de festivales.
Postcondiciones	El usuario visualiza la lista de festivales	
Excepciones	**Paso**	**Acción**
	p4	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Muy alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.8 Especificación del caso de uso “Ver listado de festivales”

UC-06	***Ver información de festival***	
Versión	1.0	
Actor principal	ACT-01, ACT-02	
Actor secundario	-	
Trigger	• TR-01 - Pulsar festival	
Descripción	El usuario visualiza información sobre el festival seleccionado	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Festivales”.
	p3	La aplicación muestra el listado de festivales.
	p4	El usuario selecciona un festival
	p5	*La aplicación solicita información sobre ese monumento al API Rest*
	p6	*El API Rest envía la información solicitada*
	p7	*La aplicación muestra la información sobre el festival*
Postcondiciones	El usuario visualiza información sobre un festival	
Excepciones	**Paso**	**Acción**
	p5	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Muy alta*	
Importancia	*Muy Alta*	
Comentarios	Este caso de uso extiende de UC-05, por lo que puede no realizarse.	

Tabla 4.9 Especificación de caso de uso “Ver información de festival”

UC-07	***Ver listado de leyendas***	
Versión	1.0	
Actor principal	ACT-01, ACT-02	
Actor secundario	-	
Trigger	• TR-01 Pulsar botón “Leyendas”	
Descripción	El usuario visualiza un listado las leyendas registradas en la aplicación	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Leyendas”.
	p3	La aplicación solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La aplicación muestra el listado de leyendas.
Postcondiciones	El usuario visualiza la lista de leyendas	
Excepciones	**Paso**	**Acción**
	p4	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Muy alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.10 Especificación del caso de uso “Ver listado de leyendas”

UC-08	***Ver información de leyenda***	
Versión	1.0	
Actor principal	ACT-01, ACT-02	
Actor secundario	-	
Trigger	• TR-01 - Pulsar leyenda	
Descripción	El usuario visualiza información sobre la leyenda seleccionado	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Leyendas”.
	p3	La aplicación muestra el listado de leyendas.
	p4	El usuario selecciona una leyenda
	p5	La aplicación solicita información sobre ese monumento al API Rest
	p6	El API Rest envía la información solicitada
	p7	La aplicación muestra la información sobre la leyenda
Postcondiciones	El usuario visualiza información sobre una leyenda	
Excepciones	**Paso**	**Acción**
	p5	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Muy alta*	
Importancia	*Muy Alta*	
Comentarios	Este caso de uso extiende de UC-07, por lo que puede no realizarse.	

Tabla 4.11 Especificación de caso de uso “Ver información de leyenda”

UC-09	***Solicitar información***	
Versión	1.0	
Actor principal	ACT-01- ACT-02	
Actor secundario	-	
Trigger	• TR-01 La aplicación ha detectado un monumento o un museo cercano y lanza una notificación	
Descripción	El usuario recibe una notificación de monumento o museo cercano y la pulsa	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario recibe una notificación.
	p2	El usuario pulsa la notificación recibida.
	p3	El sistema se conecta con la API Rest y solicita información sobre el monumento o museo localizado
	p4	El API Rest devuelve la información solicitada
	p5	La aplicación muestra la información
Postcondiciones	El usuario visualiza la información del monumento o museo localizado	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.12 Especificación de caso de uso "Solicitar información"

UC-10	***Obtener JSON Listado monumentos***	
Versión	1.0	
Actor principal	ACT-02	
Actor secundario	ACT-01, ACT-03	
Trigger	• TR-01 El usuario pulsa el botón “Monumentos” • TR-02 El usuario administrador pulsa “Consultar monumentos”	
Descripción	El Cliente API Rest recibe una petición de listado de monumentos	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario pulsa el botón “Monumentos”
	p2	La aplicación envía al API Rest la petición de listado de monumentos
	p3	El API Rest obtiene los datos solicitados de la base de datos
	p4	El API Rest devuelve la información solicitada
Postcondiciones	El API Rest obtiene el JSON con el listado de monumentos	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.13 Especificación de caso de uso “Obtener JSON listado monumentos”

UC-11	***Obtener JSON monumento***	
Versión	1.0	
Actor principal	ACT-02	
Actor secundario	ACT-01, ACT-03	
Trigger	• TR-01 El usuario selecciona un monumento del listado • TR-02 El usuario administrador selecciona un monumento del listado • TR-03 Pulsar la notificación recibida manda una petición de JSON	
Descripción	El Cliente API Rest recibe una petición de monumento	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario pulsa un monumento de los mostrados en el listado
	p2	La aplicación envía al API Rest la petición de información de monumento
	p3	El API Rest obtiene los datos solicitados de la base de datos
	p4	El API Rest devuelve la información solicitada
Postcondiciones	El API Rest obtiene el JSON con la información del monumento	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.14 Especificación de caso de uso "Obtener JSON monumento"

UC-12	***Obtener JSON Listado museos***	
Versión	1.0	
Actor principal	ACT-02	
Actor secundario	ACT-01, ACT-03	
Trigger	• TR-01 El usuario pulsa el botón “Museos” • TR-02 El usuario administrador pulsa “Consultar museos”	
Descripción	El Cliente API Rest recibe una petición de listado de museos	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario pulsa el botón “Museos”
	p2	La aplicación envía al API Rest la petición de listado de museos
	p3	El API Rest obtiene los datos solicitados de la base de datos
	p4	El API Rest devuelve la información solicitada
Postcondiciones	El API Rest obtiene el JSON con el listado de museos	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.15 Especificación de caso de uso “Obtener JSON listado museos”

UC-13	***Obtener JSON museo***	
Versión	1.0	
Actor principal	ACT-02	
Actor secundario	ACT-01, ACT-03	
Trigger	• TR-01 El usuario selecciona un museo del listado • TR-02 El usuario administrador selecciona un museo del listado • TR-03 Pulsar la notificación recibida manda una petición de JSON	
Descripción	El Cliente API Rest recibe una petición de museo	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario pulsa un museo de los mostrados en el listado
	p2	La aplicación envía al API Rest la petición de información de museo
	p3	El API Rest obtiene los datos solicitados de la base de datos
	p4	El API Rest devuelve la información solicitada
Postcondiciones	El API Rest obtiene el JSON con la información del museo	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.16 Especificación de caso de uso "Obtener JSON museo"

UC-14	***Obtener JSON Listado festivales***	
Versión	1.0	
Actor principal	ACT-02	
Actor secundario	ACT-01, ACT-03	
Trigger	• TR-01 El usuario pulsa el botón “Festivales” • TR-02 El usuario administrador pulsa “Consultar festivales”	
Descripción	El Cliente API Rest recibe una petición de listado de festivales	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario pulsa el botón “Festivales”
	p2	La aplicación envía al API Rest la petición de listado de festivales
	p3	El API Rest obtiene los datos solicitados de la base de datos
	p4	El API Rest devuelve la información solicitada
Postcondiciones	El API Rest obtiene el JSON con el listado de festivales	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.17 Especificación de caso de uso “Obtener JSON listado festivales”

UC-15	***Obtener JSON festival***	
Versión	1.0	
Actor principal	ACT-02	
Actor secundario	ACT-01, ACT-03	
Trigger	• TR-01 El usuario selecciona un festival del listado • TR-02 El usuario administrador selecciona un festival del listado	
Descripción	El Cliente API Rest recibe una petición de festival	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario pulsa un festival de los mostrados en el listado
	p2	La aplicación envía al API Rest la petición de información de festival
	p3	El API Rest obtiene los datos solicitados de la base de datos
	p4	El API Rest devuelve la información solicitada
Postcondiciones	El API Rest obtiene el JSON con la información del festival	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.18 Especificación de caso de uso "Obtener JSON festival"

UC-16	***Obtener JSON Listado leyendas***	
Versión	1.0	
Actor principal	ACT-02	
Actor secundario	ACT-01, ACT-03	
Trigger	• TR-01 El usuario pulsa el botón “Leyendas” • TR-02 El usuario administrador pulsa “Consultar leyendas”	
Descripción	El Cliente API Rest recibe una petición de listado de leyendas	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario pulsa el botón “Leyendas”
	p2	La aplicación envía al API Rest la petición de listado de leyendas
	p3	El API Rest obtiene los datos solicitados de la base de datos
	p4	El API Rest devuelve la información solicitada
Postcondiciones	El API Rest obtiene el JSON con el listado de leyendas	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.19 Especificación de caso de uso “Obtener JSON listado leyendas”

UC-17	***Obtener JSON leyenda***	
Versión	1.0	
Actor principal	ACT-02	
Actor secundario	ACT-01, ACT-03	
Trigger	• TR-01 El usuario selecciona una leyenda del listado • TR-02 El usuario administrador selecciona una leyenda del listado	
Descripción	El Cliente API Rest recibe una petición de leyenda	
Precondición	PRE-01 · API Rest operativa	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario pulsa una leyenda de los mostrados en el listado
	p2	La aplicación envía al API Rest la petición de información de leyenda
	p3	El API Rest obtiene los datos solicitados de la base de datos
	p4	El API Rest devuelve la información solicitada
Postcondiciones	El API Rest obtiene el JSON con la información de la leyenda	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Muy Alta*	
Importancia	*Muy Alta*	
Comentarios	-	

Tabla 4.20 Especificación de caso de uso "Obtener JSON leyenda"

UC-18	***Login***	
Versión	1.0	
Actor principal	ACT-03	
Actor secundario	-	
Trigger	• TR-01 El administrador entra dentro de la interfaz web de administración	
Descripción	El administrador se identifica en la interfaz web para poder realizar modificaciones sobre la base de datos	
Precondición	PRE-01 · Base de datos disponible	
Secuencia normal	**Paso**	**Acción**
	p1	El administrador entra en la interfaz web de administración
	p2	El administrador introduce su nombre de usuario y su contraseña y pulsa enviar
	p3	El sistema interfaz web confirma los datos introducidos con la base de datos
Postcondiciones	El administrador inicia sesión	
Excepciones	**Paso**	**Acción**
	p3	Si los datos introducidos son erróneos, la interfaz web volverá a la pantalla de login
Frecuencia	*Alta*	
Importancia	*Alta*	
Comentarios	-	

Tabla 4.21 Especificación de caso de uso "Login"

UC-19	***Logout***	
Versión	1.0	
Actor principal	ACT-03	
Actor secundario	-	
Trigger	• TR-01 El administrador pulsa el botón de logout	
Descripción	El administrador cierra sesión en la interfaz web de administración	
Precondición	PRE-01 · Usuario administrador logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario administrador pulsa el botón de logout
Postcondiciones	El administrador cierra sesión	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Alta*	
Importancia	*Alta*	
Comentarios	-	

Tabla 4.22 Especificación de caso de uso "Logout"

UC-20	***Crear monumento***	
Versión	1.0	
Actor principal	ACT-03	
Actor secundario	-	
Trigger	• TR-01 El administrador pincha la opción “Crear monumento”	
Descripción	El administrador puede añadir nuevos monumentos a la base de datos	
Precondición	PRE-01 · Base de datos disponible PRE-02 · Administrador logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El administrador pulsa el botón “Crear monumento”
	p2	La interfaz web muestra los campos a rellenar, a saber: nombre, localización, foto y descripción.
	p3	El administrador rellena los datos necesarios y pulsa el botón enviar
Postcondiciones	Monumento agregado a la base de datos	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Baja*	
Importancia	*Baja*	
Comentarios	-	

Tabla 4.23 Especificación de caso de uso “Crear monumento”

UC-21	***Consultar listado monumentos***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	• TR-01 Pulsar botón “Consultar monumentos”	
Descripción	El usuario administrador visualiza un listado de los monumentos registrados en la base de datos	
Precondición	PRE-01 · API Rest operativa PRE-02 Usuario logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar monumentos ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de monumentos.
Postcondiciones	El usuario administrador visualiza la lista de monumentos	
Excepciones	**Paso**	**Acción**
	p4	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	-	

Tabla 4.24 Especificación del caso de uso “Consultar listado de monumentos”

UC-22	***Buscar monumento***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	• TR-01 Introducir en la barra de búsqueda el nombre del monumento	
Descripción	El usuario administrador podrá buscar en la lista de monumentos un monumento específico introduciendo su nombre	
Precondición	PRE-01 API Rest operativa PRE-02 Usuario logueado PRE-03 El usuario debe encontrarse en el listado de monumentos	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar monumentos ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de monumentos.
	p6	El usuario introduce el nombre en la barra de búsqueda
	p7	El sistema busca coincidencias con el nombre introducido
	p8	El sistema muestra en el listado el monumento introducido en la barra de busqueda
Postcondiciones	El usuario administrador visualiza en la lista de monumentos únicamente el monumento introducido	
Excepciones	**Paso**	**Acción**
	P7	Si el monumento introducido no se encuentra en la lista de monumentos, el sistema mostrará una lista vacía
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	Este caso de uso se extiende de UC-21, por lo que el usuario no está obligado a realizarlo	

Tabla 4.25 Especificación del caso de uso “Buscar monumento”

UC-23	***Borrar monumento***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	**-**	
Descripción	El usuario administrador podrá borrar un monumento registrado en el sistema	
Precondición	PRE-01 API Rest operativa PRE-02 Usuario logueado PRE-03 El usuario debe encontrarse en el listado de monumentos	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar monumentos ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de monumentos.
	p6	El usuario pulsa el botón borrar que aparecerá a la derecha del monumento
	p7	El sistema borra el monumento
Postcondiciones	El usuario administrador borrará un monumento	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	Este caso de uso se extiende de UC-21, por lo que el usuario no está obligado a realizarlo	

Tabla 4.26 Especificación del caso de uso “Borrar monumento”

UC-24	***Crear festival***	
Versión	1.0	
Actor principal	ACT-03	
Actor secundario	-	
Trigger	• TR-01 El administrador pincha la opción “Crear festival”	
Descripción	El administrador puede añadir nuevos festivales a la base de datos	
Precondición	PRE-01 · Base de datos disponible PRE-02 · Administrador logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El administrador pulsa el botón “Crear festival”
	p2	La interfaz web muestra los campos a rellenar, a saber: nombre, fecha, foto y descripción.
	p3	El administrador rellena los datos necesarios y pulsa el botón enviar
Postcondiciones	Festival agregado a la base de datos	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Baja*	
Importancia	*Baja*	
Comentarios	-	

Tabla 4.27 Especificación de caso de uso “Crear festival”

UC-25	***Consultar listado de festivales***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	• TR-01 Pulsar botón “Consultar festivales”	
Descripción	El usuario administrador visualiza un listado de los festivales registrados en la base de datos	
Precondición	PRE-01 · API Rest operativa PRE-02 Usuario logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar festivales ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de festivales.
Postcondiciones	El usuario administrador visualiza la lista de festivales	
Excepciones	**Paso**	**Acción**
	p4	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	-	

Tabla 4.28 Especificación del caso de uso “Consultar listado de festivales”

UC-26	***Buscar festival***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	• TR-01 Introducir en la barra de búsqueda el nombre del festival	
Descripción	El usuario administrador podrá buscar en la lista de festivales un festival específico introduciendo su nombre	
Precondición	PRE-01 API Rest operativa PRE-02 Usuario logueado PRE-03 El usuario debe encontrarse en el listado de festivales	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar festivales ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de festivales.
	p6	El usuario introduce el nombre en la barra de búsqueda
	p7	El sistema busca coincidencias con el nombre introducido
	p8	El sistema muestra en el listado el festival introducido en la barra de busqueda
Postcondiciones	El usuario administrador visualiza en la lista de festivales únicamente el festival introducido	
Excepciones	**Paso**	**Acción**
	P7	Si el festival introducido no se encuentra en la lista de festivales, el sistema mostrará una lista vacía
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	Este caso de uso se extiende de UC-25, por lo que el usuario no está obligado a realizarlo	

Tabla 4.29 Especificación del caso de uso “Buscar festival”

UC-27	***Borrar festival***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	**-**	
Descripción	El usuario administrador podrá borrar un festival registrado en el sistema	
Precondición	PRE-01 API Rest operativa PRE-02 Usuario logueado PRE-03 El usuario debe encontrarse en el listado de monumentos	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar festivales ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de festivales.
	p6	El usuario pulsa el botón borrar que aparecerá a la derecha del festival
	p7	El sistema borra el festival
Postcondiciones	El usuario administrador borrará un festival	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	Este caso de uso se extiende de UC-25, por lo que el usuario no está obligado a realizarlo	

Tabla 4.30 Especificación del caso de uso “Borrar festival”

UC-28	***Crear museo***	
Versión	1.0	
Actor principal	ACT-03	
Actor secundario	-	
Trigger	• TR-01 El administrador pincha la opción “Crear museo”	
Descripción	El administrador puede añadir nuevos museos a la base de datos	
Precondición	PRE-01 · Base de datos disponible PRE-02 · Administrador logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El administrador pulsa el botón “Crear museo”
	p2	La interfaz web muestra los campos a rellenar, a saber: nombre, localización, foto y descripción.
	p3	El administrador rellena los datos necesarios y pulsa el botón enviar
Postcondiciones	Museo agregado a la base de datos	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Baja*	
Importancia	*Baja*	
Comentarios	-	

Tabla 4.31 Especificación de caso de uso “Crear museo”

UC-29	***Consultar listado de museos***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	• TR-01 Pulsar botón “Consultar museos”	
Descripción	El usuario administrador visualiza un listado de los museos registrados en la base de datos	
Precondición	PRE-01 · API Rest operativa PRE-02 Usuario logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar museos ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de museos.
Postcondiciones	El usuario administrador visualiza la lista de museos	
Excepciones	**Paso**	**Acción**
	p4	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	-	

Tabla 4.32 Especificación del caso de uso “Consultar listado de museos”

UC-30	***Buscar museo***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	• TR-01 Introducir en la barra de búsqueda el nombre del museo	
Descripción	El usuario administrador podrá buscar en la lista de museos un museo específico introduciendo su nombre	
Precondición	PRE-01 API Rest operativa PRE-02 Usuario logueado PRE-03 El usuario debe encontrarse en el listado de museos	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar museos ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de muses.
	p6	El usuario introduce el nombre en la barra de búsqueda
	p7	El sistema busca coincidencias con el nombre introducido
	p8	El sistema muestra en el listado el museo introducido en la barra de busqueda
Postcondiciones	El usuario administrador visualiza en la lista de museos únicamente el museo introducido	
Excepciones	**Paso**	**Acción**
	P7	Si el festival introducido no se encuentra en la lista de museos, el sistema mostrará una lista vacía
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	Este caso de uso se extiende de UC-29, por lo que el usuario no está obligado a realizarlo	

Tabla 4.33 Especificación del caso de uso “Buscar museo”

UC-31	***Borrar museo***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	**-**	
Descripción	El usuario administrador podrá borrar un museo registrado en el sistema	
Precondición	PRE-01 API Rest operativa PRE-02 Usuario logueado PRE-03 El usuario debe encontrarse en el listado de museos	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar museos ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de museos.
	p6	El usuario pulsa el botón borrar que aparecerá a la derecha del museo
	p7	El sistema borra el museo
Postcondiciones	El usuario administrador borrará un museo	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	Este caso de uso se extiende de UC-29, por lo que el usuario no está obligado a realizarlo	

Tabla 4.34 Especificación del caso de uso “Borrar museo”

UC-32	***Crear leyenda***	
Versión	1.0	
Actor principal	ACT-03	
Actor secundario	-	
Trigger	• TR-01 El administrador pincha la opción “Crear leyenda”	
Descripción	El administrador puede añadir nuevas leyendas a la base de datos	
Precondición	PRE-01 · Base de datos disponible PRE-02 · Administrador logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El administrador pulsa el botón “Crear leyenda”
	p2	La interfaz web muestra los campos a rellenar, a saber: nombre, foto y descripción.
	p3	El administrador rellena los datos necesarios y pulsa el botón enviar
Postcondiciones	Leyenda agregado a la base de datos	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Baja*	
Importancia	*Baja*	
Comentarios	-	

Tabla 4.35 Especificación de caso de uso “Crear leyenda”

UC-33	***Consultar listado de leyendas***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	• TR-01 Pulsar botón “Consultar leyendas”	
Descripción	El usuario administrador visualiza un listado de las leyendas registrados en la base de datos	
Precondición	PRE-01 · API Rest operativa PRE-02 Usuario logueado	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar leyendas ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de leyendas.
Postcondiciones	El usuario administrador visualiza la lista de leyendas	
Excepciones	**Paso**	**Acción**
	p4	Si el API Rest no está disponible la aplicación lanzará una alerta diciendo “El sistema no puede mostrar la información solicitada”
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	-	

Tabla 4.36 Especificación del caso de uso “Consultar listado de leyendas”

UC-34	***Buscar leyenda***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	• TR-01 Introducir en la barra de búsqueda el nombre de la leyenda	
Descripción	El usuario administrador podrá buscar en la lista de leyendas una leyenda específica introduciendo su nombre	
Precondición	PRE-01 API Rest operativa PRE-02 Usuario logueado PRE-03 El usuario debe encontrarse en el listado de leyendas	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar leyendas ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de leyendas.
	p6	El usuario introduce el nombre en la barra de búsqueda
	p7	El sistema busca coincidencias con el nombre introducido
	p8	El sistema muestra en el listado la leyenda introducida en la barra de busqueda
Postcondiciones	El usuario administrador visualiza en la lista de leyendas únicamente la leyenda introducida	
Excepciones	**Paso**	**Acción**
	P7	Si la leyenda introducida no se encuentra en la lista de leyendas, el sistema mostrará una lista vacía
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	Este caso de uso se extiende de UC-33, por lo que el usuario no está obligado a realizarlo	

Tabla 4.37 Especificación del caso de uso “Buscar leyenda”

UC-35	***Borrar Leyenda***	
Versión	1.0	
Actor principal	ACT-02, ACT-03	
Actor secundario	-	
Trigger	**-**	
Descripción	El usuario administrador podrá borrar una leyenda registrada en el sistema	
Precondición	PRE-01 API Rest operativa PRE-02 Usuario logueado PRE-03 El usuario debe encontrarse en el listado de leyendas	
Secuencia normal	**Paso**	**Acción**
	p1	El usuario entra en la aplicación.
	p2	El usuario pulsa el botón “Consultar leyendas ”.
	p3	La interfaz web solicita la información a la API Rest.
	p4	El API Rest devuelve el JSON con los datos solicitados.
	p5	La interfaz web muestra el listado de leyendas.
	p6	El usuario pulsa el botón borrar que aparecerá a la derecha de la leyenda
	p7	El sistema borra la leyenda
Postcondiciones	El usuario administrador borrará una leyenda	
Excepciones	**Paso**	**Acción**
	-	-
Frecuencia	*Media*	
Importancia	*Media*	
Comentarios	Este caso de uso se extiende de UC-33, por lo que el usuario no está obligado a realizarlo	

Tabla 4.38 Especificación del caso de uso “Borrar leyenda”

4.4 Requisitos funcionales

Diferenciaremos dos grupos de requisitos funcionales, los del lado cliente (aplicación *Android*) y los del lado servidor (*Api Rest*).

Lado Cliente

- **RF-01** La aplicación accederá al sistema GPS del dispositivo.
- **RF-02**: La aplicación será accesible a cualquier usuario que disponga de ella sin necesidad de registro o autenticación.
- **RF-03** La aplicación obtendrá las coordenadas GPS del usuario.
- **RF-04** La aplicación accederá al sistema *WiFi* del dispositivo.
- **RF-05** La aplicación accederá al sistema de internet móvil del dispositivo.
- **RF-06** La aplicación se conectará a una API Rest externa para obtener información sobre los monumentos.
- **RF-07** La aplicación se conectará a una API Rest externa para obtener información sobre los museos.
- **RF-08** La aplicación se conectará a una API Rest externa para obtener información sobre los festivales.
- **RF-09** La aplicación se conectará a una API Rest externa para obtener información sobre las leyendas.
- **RF-10** La aplicación permitirá ver una lista de los monumentos registrados.
- **RF-11** La aplicación permitirá ver una lista de los museos registrados.
- **RF-12** La aplicación permitirá ver una lista de los festivales registrados.
- **RF-13** La aplicación permitirá ver una lista de las leyendas registradas.
- **RF-14** La aplicación permitirá ver información sobre los monumentos almacenados en la base de datos.
- **RF-15** La aplicación permitirá ver información sobre los museos almacenados en la base de datos.
- **RF-16** La aplicación permitirá ver información sobre los festivales almacenados en la base de datos.
- **RF-17** La aplicación permitirá ver información sobre las leyendas almacenadas en la base de datos.
- **RF-18** La aplicación permitirá ver información sobre los monumentos almacenados en la base de datos.
- **RF-19** La aplicación avisará al usuario de la localización de un monumento cercano mediante una notificación push.
- **RF-20** La aplicación avisará al usuario de la localización de un museo cercano mediante una notificación push.
- **RF-21** La aplicación permitirá al usuario visualizar información sobre un monumento localizado pulsando la notificación push recibida.
- **RF-22** La aplicación permitirá al usuario visualizar información sobre un museo localizado pulsando la notificación push recibida.
- **RF-23** La aplicación permitirá al usuario confirmar su salida de la misma mediante un cuadro de diálogo.

Lado Servidor

- **RF-24** El sistema dispondrá de una interfaz web para modificar toda la información mostrada en la aplicación.

- **RF-25** El sistema permitirá a un usuario administrador iniciar sesión en la interfaz web.
- **RF-26** El sistema permitirá a un usuario administrador cerrar sesión en la interfaz web.
- **RF-27** El sistema permitirá la creación de nuevos monumentos.
- **RF-28** El usuario administrador podrá crear nuevos monumentos.
- **RF-29** El sistema permitirá mostrar una lista de monumentos.
- **RF-30** El usuario administrador podrá acceder a una lista de monumentos.
- **RF-31** El sistema permitirá borrar un monumento.
- **RF-32** El usuario administrador podrá borrar un monumento.
- **RF-33** El sistema permitirá buscar un monumento en el listado mostrado.
- **RF-34** El usuario administrador podrá buscar monumentos.
- **RF-35** El sistema permitirá la creación de nuevos museos.
- **RF-36** El usuario administrador podrá crear nuevos museos.
- **RF-37** El sistema permitirá mostrar una lista de museos.
- **RF-38** El usuario administrador podrá acceder a una lista de museos.
- **RF-39** El sistema permitirá borrar un museo.
- **RF-40** El usuario administrador podrá borrar un museo.
- **RF-41** El sistema permitirá buscar un monumento en el listado museos.
- **RF-42** El usuario administrador podrá buscar museos.
- **RF-43** El sistema permitirá la creación de nuevos festivales.
- **RF-44** El usuario administrador podrá crear nuevos festivales.
- **RF-45** El sistema permitirá mostrar una lista de festivales.
- **RF-46** El usuario administrador podrá acceder a una lista de festivales.
- **RF-47** El sistema permitirá borrar un festival.
- **RF-48** El usuario administrador podrá borrar un festival.
- **RF-49** El sistema permitirá buscar un monumento en el listado festivales.
- **RF-50** El usuario administrador podrá buscar festivales.
- **RF-51** El sistema permitirá la creación de nuevas leyendas.
- **RF-52** El usuario administrador podrá crear nuevas leyendas.
- **RF-53** El sistema permitirá mostrar una lista de leyendas.
- **RF-54** El usuario administrador podrá acceder a una lista de leyendas.
- **RF-55** El sistema permitirá borrar una leyenda.
- **RF-56** El usuario administrador podrá borrar una leyenda.
- **RF-57** El sistema permitirá buscar un monumento en el listado leyendas.
- **RF-58** El usuario administrador podrá buscar leyendas.

4.5 Requisitos de información

Dado que el sistema propuesto en este proyecto va a manejar una gran cantidad de información, se ha elegido un modelo de base de datos no relacional. Este modelo deberá almacenar la siguiente información.

- **RI-01** Información relativa a los monumentos, a saber nombre, foto, latitud, longitud y descripción del mismo.
- **RI-02** Información relativa a los museos, a saber nombre, foto, latitud, longitud y descripción del mismo, detallando en esta su exposición actual.

- **RI-03** Información relativa a leyendas populares, a saber nombre, descripción y foto.
- **RI-04** Información relativa a los festivales, a saber nombre, fecha, imagen y descripción del mismo.
- **RI-05** Información relativa al usuario administrador, a saber nombre y contraseña.

4.5.1 Sistema de base de datos

En este proyecto se utilizará el sistema de base de datos MongoDB. Este es una base de datos no relacional, basada en colecciones o documentos y de código abierto. Soporta varios tipos de datos, entre ellos:

`String` - Cadenas de caracteres.
`Integer` - Números enteros.
`Double` - Números con decimales.
`Boolean` - Booleanos verdaderos o falsos.
`Date` - Fechas.
`Timestamp` - Estampillas de tiempo.
`Null` - Valor nulo.
`Array` - Arreglos de otros tipos de dato.
`Object` - Otros documentos embebidos.
`ObjectID` - Identificadores únicos creados por MongoDB al crear documentos sin especificar valores para el campo `_id`.
`Data Binaria` - Punteros a archivos binarios.
`Javascript` - código y funciones Javascript.

Ilustración 4.3 Tipos de datos MongoDB

Se ha elegido este sistema de base de datos, además de por profundizar más en el conocimiento de este tipo de sistemas de almacenamiento de datos, por varias características:

- Proporciona un esquema flexible: significa que los documentos de la misma colección no necesitan tener el mismo número de campos o estructura.
- Posee un alto rendimiento: proporciona la persistencia de datos de alto rendimiento
- Ofrece escalado automático: la inserción de nuevos datos sólo ampliará la colección.

A diferencia de las bases de datos relacionales, MongoDB se centra en el rendimiento y la escalabilidad

4.5.2 Modelo de datos

Dado que vamos a utilizar el modelo de base de datos no relacional MongoDB, vamos a explicar una serie de conceptos. Este sistema no posee un modelo entidad-relación, ya que es una base de datos no relacional. MongoDB no guarda tablas, si no documentos en colecciones, es decir, lo que en modelos relacionales llamamos tablas ahora son colecciones, y lo que en un modelo relacional es un elemento de información ahora es

un documento. En MongoDB se utilizan los esquemas .La estructura que poseen estos documentos o colecciones es la siguiente:

```
{
        Nombre : "Angel",
        Apellido : "Barroso",
        Edad : 24
}
```

Por tanto, para nuestro proyecto, se modelarán 4 colecciones de datos, cada una referida a cada sección de la aplicación. Además, se modelará una tabla extra para introducir al usuario administrador. Todas las colecciones partirán de unos atributos base. Dependiendo de la colección, se añadirán más atributos.

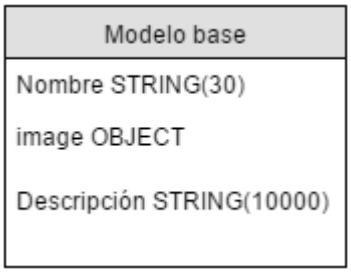

Ilustración 4.4 Esquema de modelo de datos general

A partir de ese modelo base, obtenemos los siguientes modelos de las diferentes colecciones.

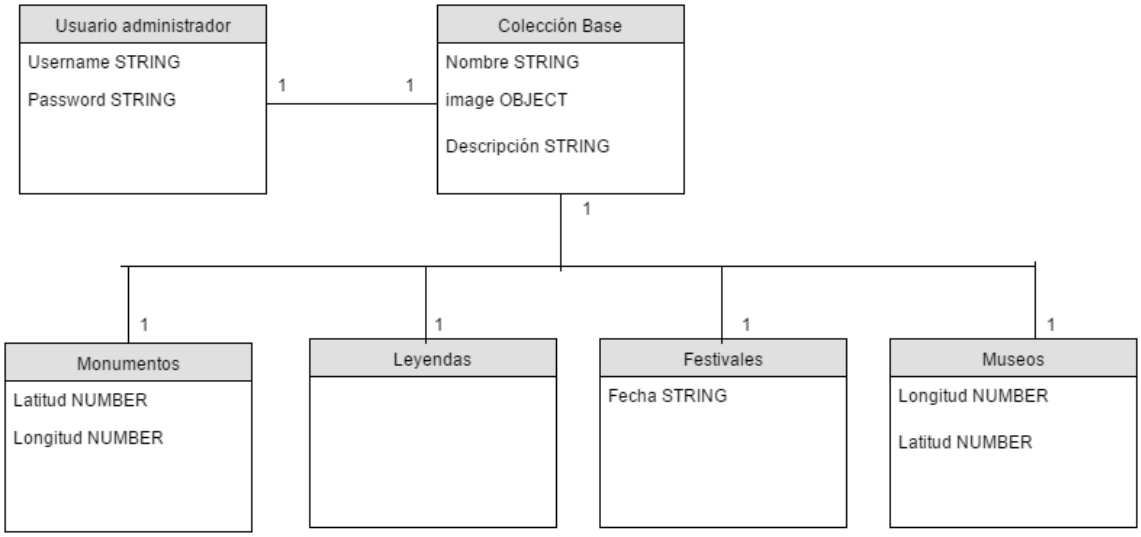

Ilustración 4.5 Esquema de modelo de datos del sistema

Con el diagrama anterior se quiere reflejar cómo funcionaría el sistema de base de datos. Un usuario Administrador manejará todas las colecciones. De cada colección base, derivará una colección concreta, añadiendo atributos si fuera necesario. Además, todo elemento de cada colección contará con un campo *Id*, generado automáticamente por MongoDB.

4.6 Requisitos de interfaces externas

- **RIE-01** El sistema obtendrá la información de los monumentos conectándose a una API Rest externa que tendrá almacenados todos los datos registrados en la misma sobre monumentos, museos, festivales y leyendas.

4.7 Requisitos no funcionales

- **RNF-01** El sistema debe tener un diseño sencillo y funcional.
- **RNF-02** El sistema debe ser capaz de registrar todos los posibles fallos que puedan ocurrir durante el normal funcionamiento de la aplicación.
- **RNF-03** El sistema debe ser eficiente a la hora de realizar consultas con el objetivo de tener un alto rendimiento.
- **RNF-04** El sistema debe estar disponible 24/7.
- **RNF-05** El sistema debe ser fácilmente mantenible.
- **RNF-06** El sistema debe tener un diseño y una estructura que le permita ser escalable.
- **RNF-07** El sistema no debe corromper los datos utilizados en la aplicación.

4.8 Requisitos de internacionalización y localización

- **RIL-01** La primera versión (1.0) sólo soportará el idioma español.

5. DISEÑO DEL SISTEMA

5.1 Tecnologías

Para la realización de este proyecto se han utilizado varias tecnologías.

- **Apache Cordova**

Ilustración 5.1 Tecnología Apache Cordova

Como ya se ha explicado anteriormente, el marco de desarrollo móvil que se utilizará en este proyecto es apache Cordova.

- **AngularJS**

Ilustración 5.2 Tecnología Angularjs

AngularJS es un *framework* MVC desarrollado por *Google* que sirve para crear aplicaciones simples. Utilizaremos este lenguaje para la construcción de la parte cliente de nuestro proyecto, es decir, para crear el contenido de la aplicación Cordova.

- **Bootstrap**

Ilustración 5.3 Tecnología Bootstrap

Bootstrap es un conjunto de herramientas o *framework* de código abierto que sirve para dotar a los sitios web de estilo y de diseño. Se utilizará para maquetar nuestra aplicación y la interfaz web de nuestro proyecto.

- **NodeJS y ExpressJS**

Ilustración 5.4 Tecnología NodeJS y ExpressJS

Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles. *Node* es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor. Utilizaremos estos dos lenguajes para programar la parte de servidor de nuestra aplicación.

- **MongoDB**

Ilustración 5.5 Tecnología MongoDB

MongoDB es el sistema de base de datos elegido para nuestro proyecto. Es un modelo de base de datos no relacional basado en colecciones de datos. Para nuestro proyecto tenemos 4 grandes colecciones de datos: monumentos, museos, festivales y leyendas.

- **SDK de *Android***

Ilustración 5.6 Tecnología SDK Android

El SDK de *Android* sirve para construir aplicaciones para el sistema operativo *Android*. En nuestro proyecto será de gran utilidad, ya que gracias a esto Cordova podrá crear nuestra aplicación.

5.2 Arquitectura lógica

Una aplicación Cordova está formada por varios componentes. La siguiente Ilustración muestra una vista a alto nivel de como esta compuesta una aplicación de este tipo.

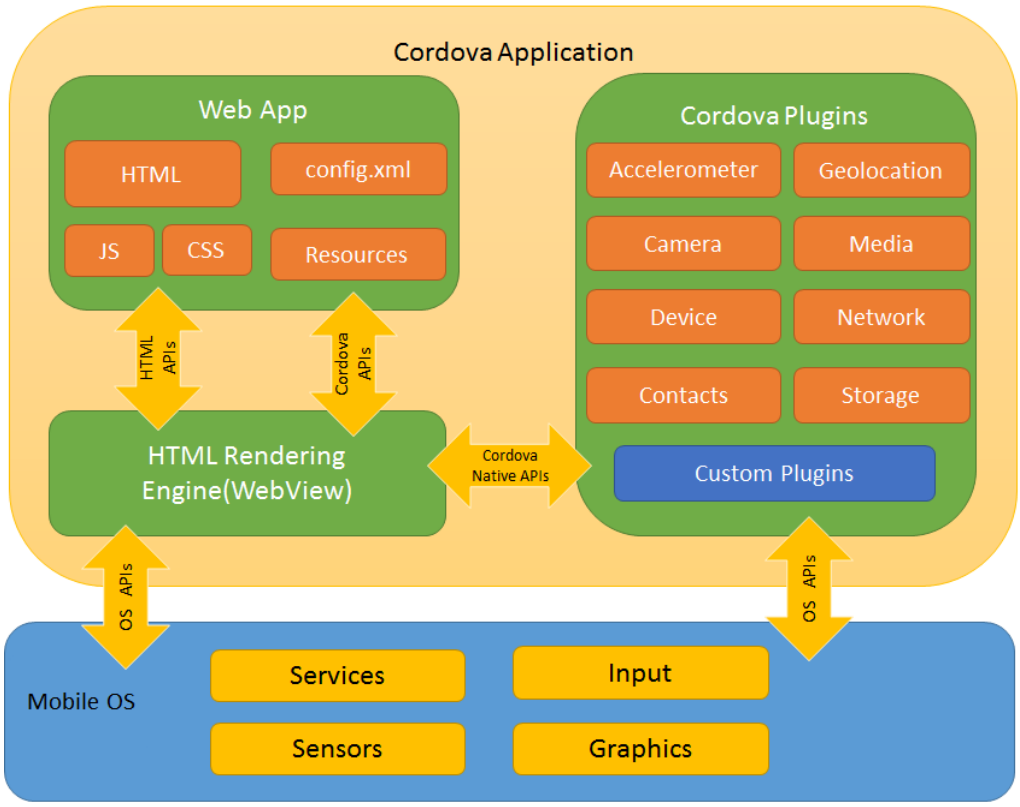

Ilustración 5.7 Arquitectura lógica de una aplicación Cordova

Como se puede observar en la ilustración, una aplicación está compuesta por 4 elementos principales:

- *WebApp*: contiene todo el código que hace de la aplicación lo que es. Este le dota de funcionalidad y ofrece la parte visual de cara al usuario
- *WebView*: es la tecnología que renderiza la parte contenida en la *webapp*, es decir, proporciona un navegador incrustado en la aplicación que permite ver los archivos programados como si de una página web se tratara.
- *Plugins*: sirven para comunicar la aplicación con los elementos que conforman el dispositivo o componentes hardware del mismo.
- *OS APIS*: son los mecanismos proporcionados por el *SDK* de cada sistema operativo móvil que hace funcionar la aplicación en los diferentes sistemas operativos.

En cuanto al servidor, en los siguientes apartados se explicará su funcionamiento.

5.3 Arquitectura física

En este apartado se explicará el funcionamiento del proyecto y como se comunican las diferentes partes entre sí. Partiendo de la siguiente Ilustración como base, se procederá a detallar el funcionamiento del proyecto.

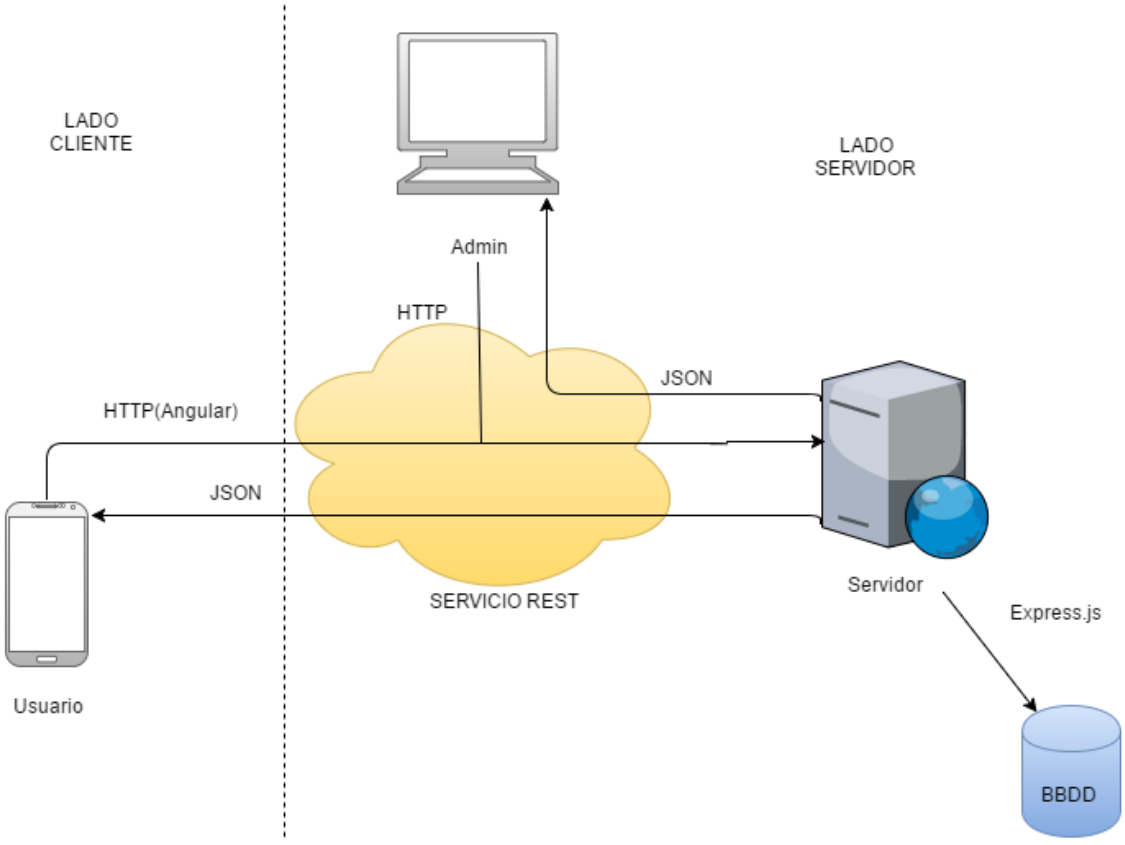

Ilustración 5.8 Arquitectura física del proyecto

Como se observa en el diagrama anterior, tenemos dos partes bien diferenciadas. La parte cliente es aquella que engloba al dispositivo móvil, es decir, la aplicación. Se comunica con el lado servidor mediante *AngularJS*, haciendo una petición de tipo HTTP al servicio REST.

Por otro lado, el servidor recibe esa petición HTTP, y devuelve la información solicitada en un archivo en formato JSON, con el que la aplicación es capaz de obtener la información deseada y mostrársela al usuario.

El servidor estará programado siguiendo el patrón de MVC[14], donde el modelo será la base de datos, el controlador será *Node* y *Express*, y se creará una interfaz para interactuar con la base de datos mediante el uso de lenguajes web como *HTML* y *CSS*.

El funcionamiento del servidor es similar a de la aplicación móvil. Este envía peticiones HTTP desde la interfaz y se le devuelve la información en formato JSON, donde el controlador de la interfaz deberá ser capaz de extraer esos datos y mostrárselos al administrador de manera adecuada.

En los siguientes apartados se mostrarán diagramas que ayudarán y mejorarán el entendimiento del proyecto.

5.4 Diagrama de componentes

Los diagramas de componentes se utilizan para modelar la vista de implementación estática de un sistema. Sirven para reforzar la explicación de la arquitectura física. En este caso, se utilizará para modelar los componentes que forman la parte cliente (*Cordova app*) y la parte servidor (*API Rest*).

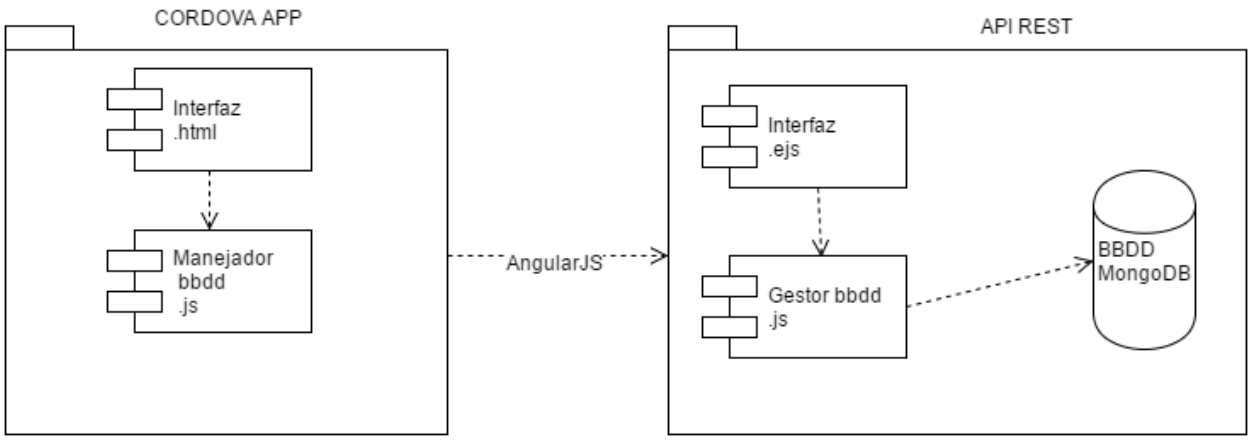

Ilustración 5.9 Diagrama de componentes del sistema

[14] MVC : Modelo Vista Controlador

Como se puede observar, en la parte cliente tenemos dos componentes principales. Uno de ellos es la interfaz, que se proporciona a partir de archivos .html. Este se relaciona con el manejador de base de datos. Es un archivo de tipo *javacript* cuya función principal es conectarse al *API Rest* y obtener los datos que mostrará la interfaz.

Por otro lado, en los componentes del servidor se observa claramente como está desarrollado el MVC. Por un lado, tenemos nuestro modelo, que en este caso es la base de datos MongoDB. El controlador, que se encargará de comunicarse con la base de datos y de mostrar adecuadamente el contenido de la misma en la interfaz del servidor es el gestor bbdd, un archivo de tipo *javascript* programado con *NodeJS* y *Express*. Por último, la vista que se encargará de mostrar todos los datos de la bbdd es la interfaz, programada con archivos de tipo .ejs, una variable del lenguaje HTML que permite la inclusión de comandos enviados por el controlador.

5.5 Diagrama de despliegue

El diagrama de despliegue representa la arquitectura de ejecución de los sistemas. Muestran la topología del hardware mediante nodos y conexiones. Los nodos pueden representar tanto dispositivos hardware como entornos de ejecución software. Se realiza este tipo de diagrama para reforzar el modelo de arquitectura física descrito anteriormente.

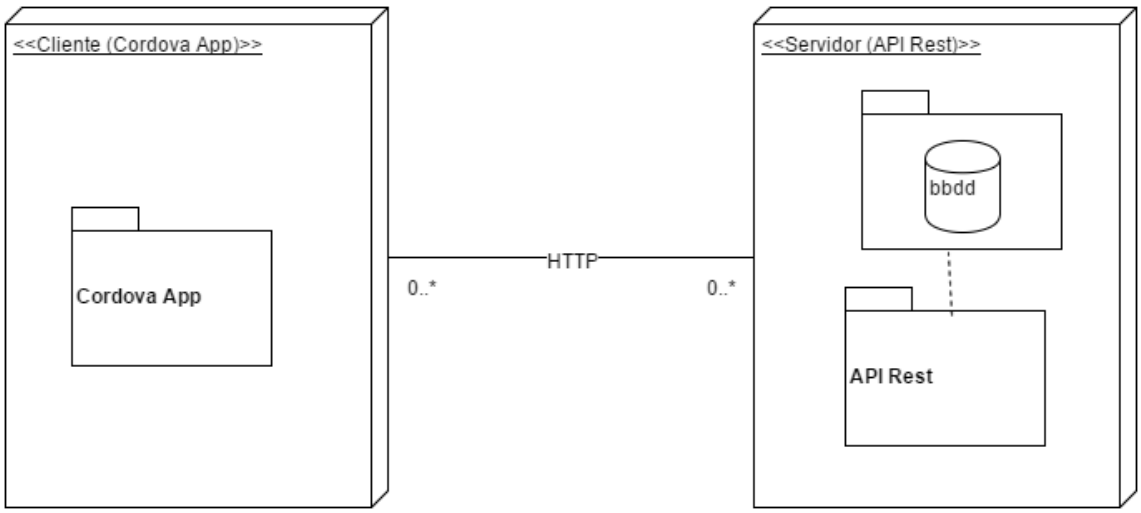

Ilustración 5.10 Diagrama de despliegue del sistema

En el diagrama anterior se observan claramente dos nodos diferenciados: uno de ellos es el cliente, que contiene la aplicación Cordova, y que por tanto sería cualquier dispositivo móvil con capacidad para soportarla; por otro lado tenemos el segundo nodo o servidor, que contiene la *API Rest* y la base de datos. La comunicación entre ellos se realiza a través del protocolo HTTP, y tiene una multiplicidad de 0 a *. Esto significa que el cliente puede enviar entre 0 e infinitas peticiones al servidor, y que el servidor puede responder entre 0 e infinitas peticiones.

5.6 Diagramas de actividad

Para un mayor entendimiento del sistema, a continuación se muestran dos diagramas de actividad. Uno de ellos representa el lanzamiento de una notificación push y otro de ellos muestra el proceso que llevaría a cabo el usuario de la aplicación para ver la información de un monumento. Este segundo diagrama es extensible a los casos ver museo, ver festival y ver leyenda.

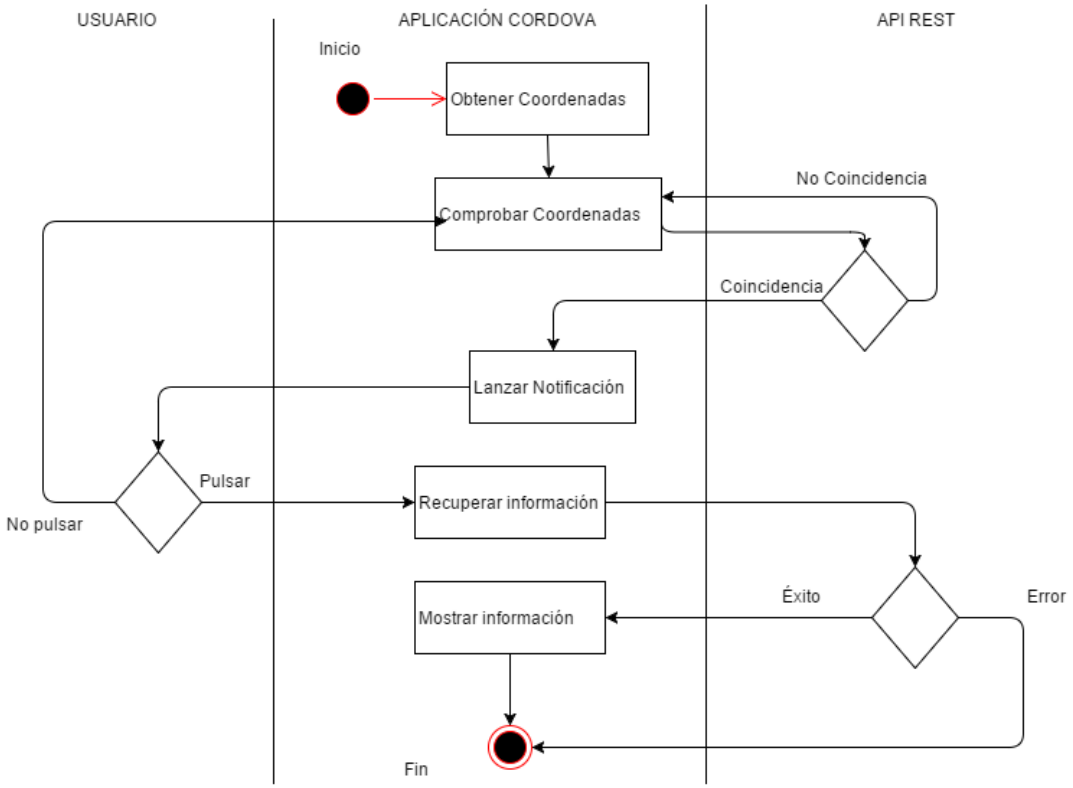

Ilustración 5.11 Diagrama de actividad recibir notificación

Este diagrama muestra la secuencia que sigue el sistema para el lanzamiento de una notificación. A medida que la aplicación va obteniendo coordenadas, va comprobando con la base de datos posibles coincidencias. En caso de encontrar una coincidencia, lanza una notificación al usuario, que tiene dos opciones, pulsar o no pulsar. Si no la pulsa, la aplicación sigue comprobando coordenadas. Si la pulsa, la aplicación se conecta con la base de datos y obtiene la información del elemento que coincida con las coordenadas del usuario.

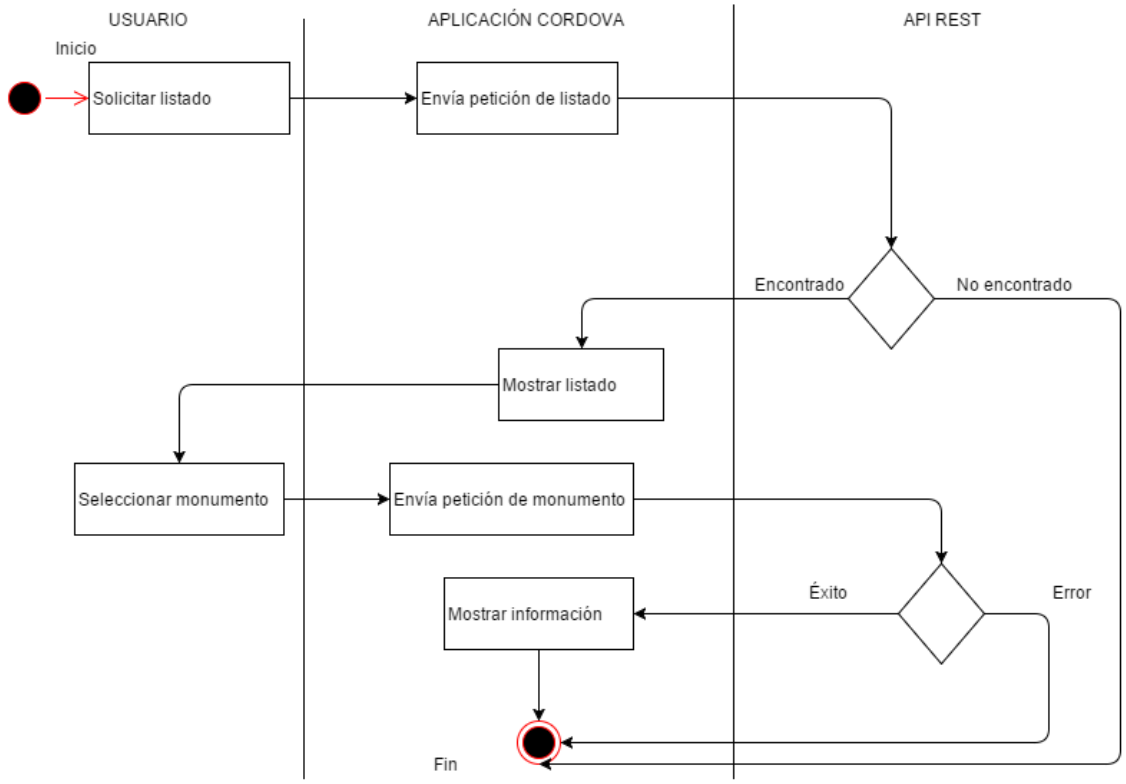

Ilustración 5.12 Diagrama de actividad ver información de monumento

Como se ha explicado anteriormente, este diagrama de actividad sería válido para los casos ver información de museo, ver información de festival o ver información de leyenda. El usuario solicita ver el listado de monumentos registrados en la aplicación, esta se conecta con el *API Rest*, y en caso de éxito muestra en la aplicación el listado. Si el usuario desea ver un monumento en concreto, se repite el proceso.

5.7 Diagramas de secuencia

Los diagramas de secuencia tienen por objetivo mostrar la secuencia de mensajes intercambiados entre un actor y el sistema en un determinado escenario de caso de uso. A continuación, se modelarán cuatro diagramas de secuencia, uno de ellos de la secuencia normal para ver un listado de los monumentos y la información del mismo, otro que indica la secuencia que sigue el proyecto para enviar una notificación, refiriéndose al lado cliente. Los otros dos diagramas referidos al lado servidor serán la creación de un nuevo monumento y el borrado del mismo. Se incluyen estos diagramas para que en lector se haga una idea más aproximada de como funcionará el sistema.

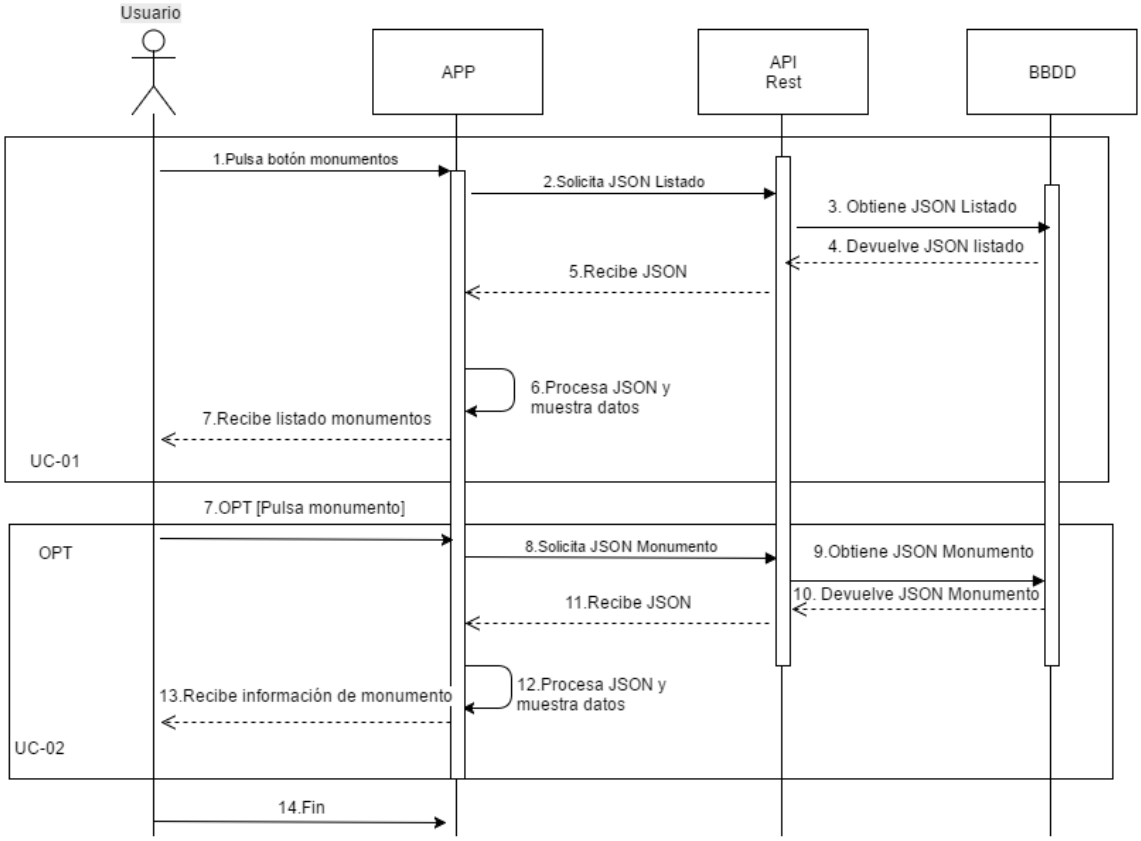

Ilustración 5.13 Diagrama de secuencia para UC-01 y UC-02

En este diagrama que engloba los casos de uso UC-01 y UC-02 se explica la secuencia de mensajes que intercambia el actor usuario con el sistema. Al recibir el listado de monumentos, el sistema da la opción de ver la información de un monumento concreto, por lo que se explica la inclusión de UC-02 dentro de este diagrama. Este diagrama es similar en el resto de casos de uso de leyendas, festivales y museos. Dado que pulsar monumento es opcional, se modela la segunda parte de este diagrama como tal.

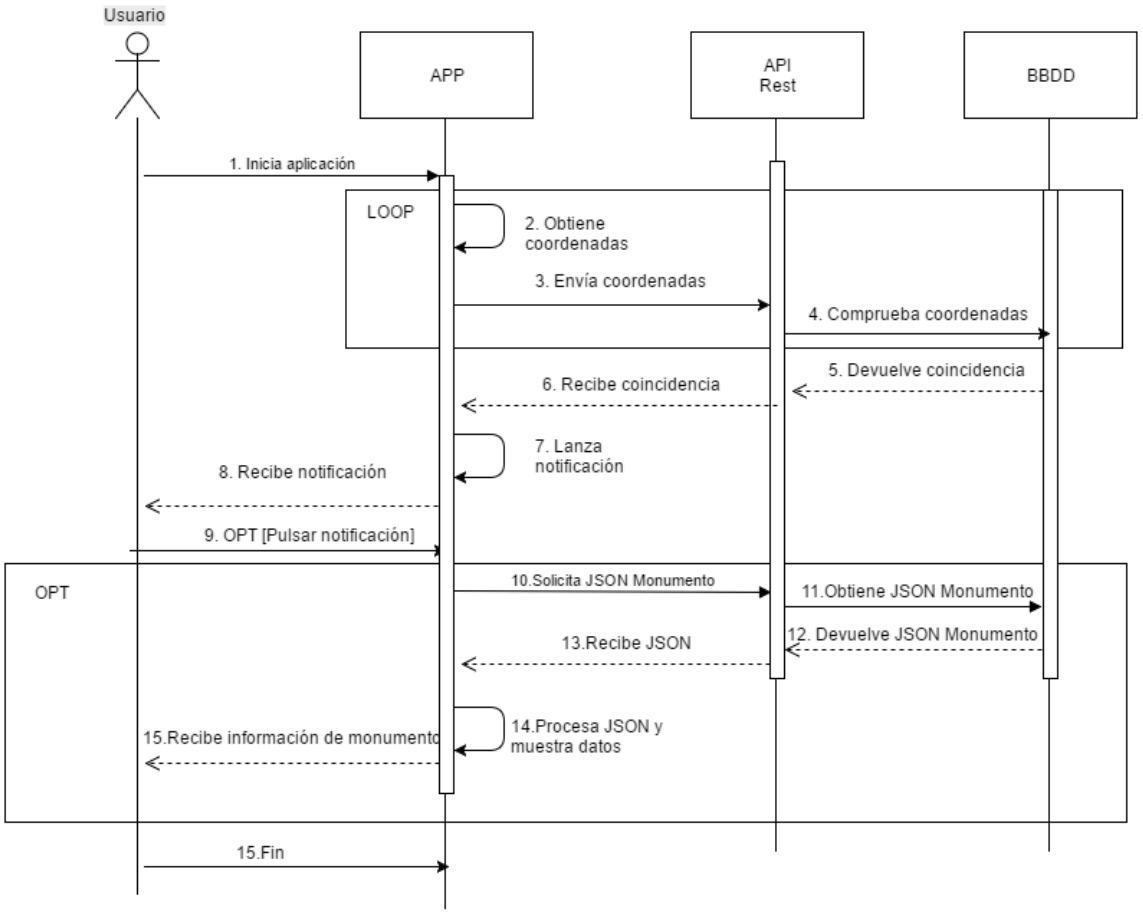

Ilustración 5.14 Diagrama de secuencia para UC-09

En este diagrama se observa la secuencia que sigue una notificación desde el arranque de la aplicación. Se declara un LOOP (bucle) porque a aplicación está comprobando constantemente las coordenadas, y cuando encuentra una coincidencia, lanza la notificación. Cuando el usuario recibe la notificación entra en juego el UC-09 “Solicitar información”. Pueden llegar notificaciones por dos razones: porque hay un monumento cercano o porque hay un museo cercano. Dado que pulsar la notificación que recibe el usuario es opcional, se modela la segunda parte de este diagrama como tal.

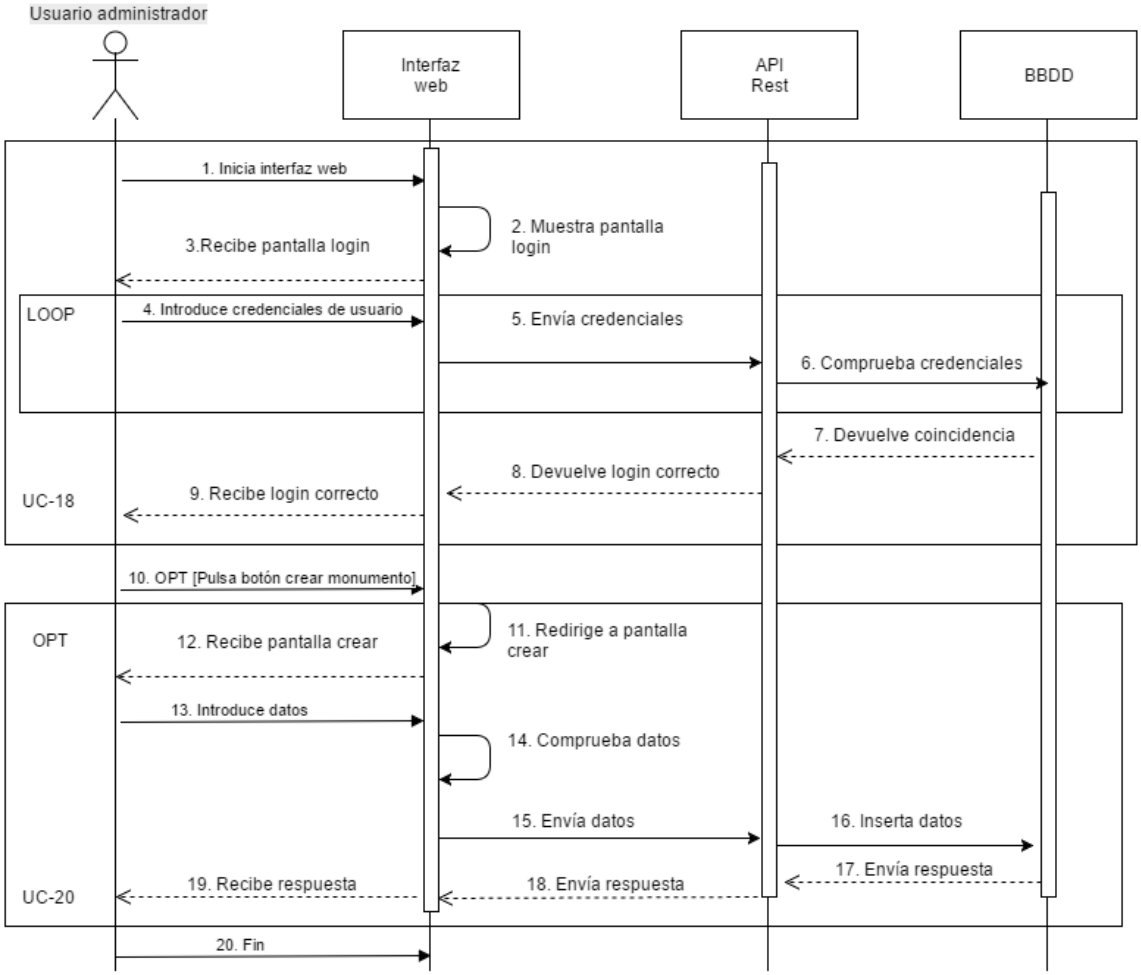

Ilustración 5.15 Diagrama de secuencia para UC-18 y UC-20

En el anterior diagrama se pretende explicar el funcionamiento que sigue la interfaz web a la hora de loguear a un usuario, y un posible caso opcional “Crear monumento”. El Loop que se observa consiste en que, hasta que el usuario administrador no se loguee correctamente, el sistema no avanzará de la pantalla de login, impidiendo así cualquier interacción con la interfaz web de administración.

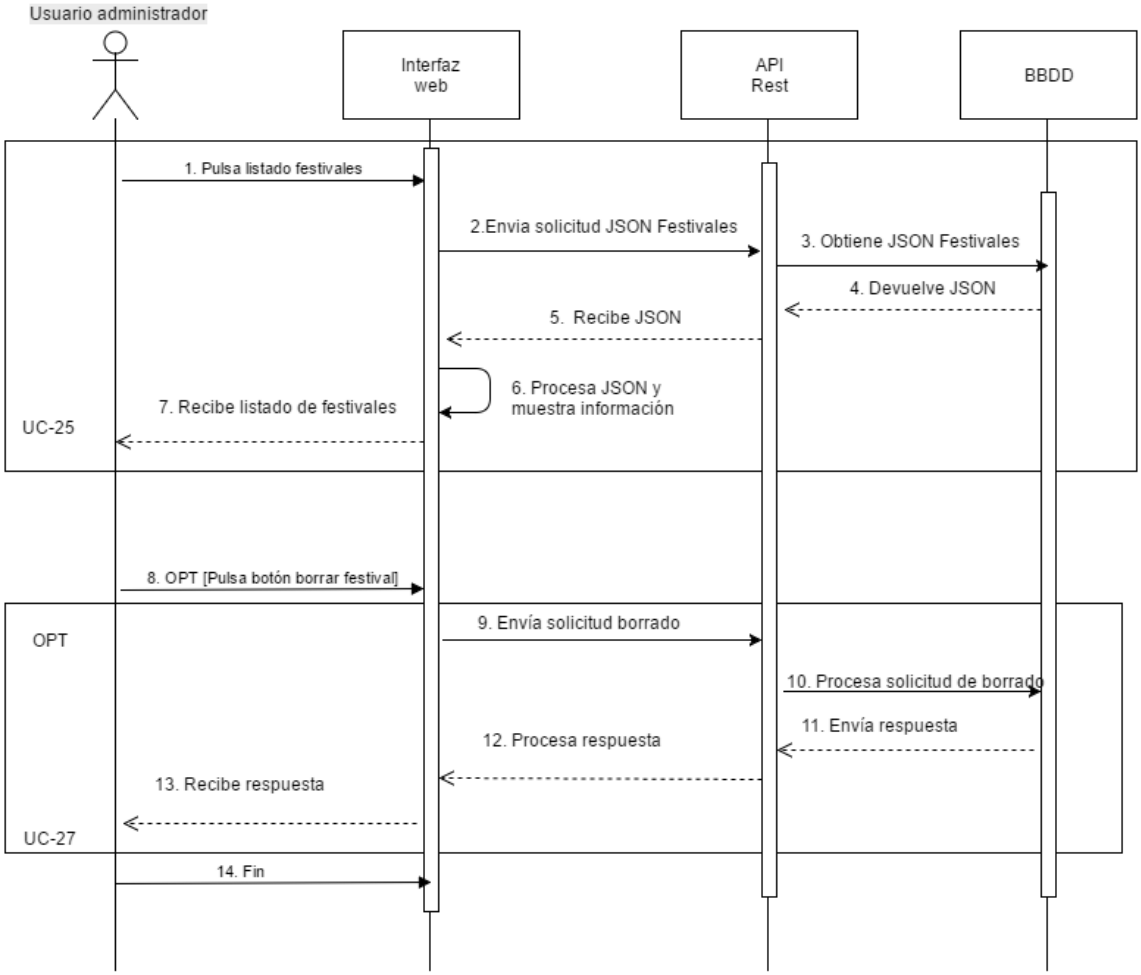

Ilustración 5.16 Diagrama de secuencia para UC-25 y UC-27

Por último, este diagrama muestra la secuencia que sigue el borrado de un festival. Dado que para borrar un festival se ha especificado que antes se debe entrar en el listado, y que borrar un festival no es obligatorio, se modela la segunda parte del diagrama "Borrar festival" como opcional.

5.8 Geolocalizando al usuario

Para la geolocalización del usuario se ha decidido utilizar la tecnología que proporciona *HTML5* para geolocalización. En dispositivos móviles, a través de datos móviles o del dispositivo *GPS* con el que vienen la mayoría de dispositivos de serie, *HTML5* es capaz de obtener las coordenadas del usuario en tiempo real. Mediante dos sencillos comandos, es muy fácil obtener las coordenadas del usuario a través de un script.

Pero obtener las coordenadas en tiempo real del usuario no es suficiente, pues si se quiere obtener una notificación por cada monumento o museo cercano, con geolocalizar al usuario no basta. Para ello, a la hora de introducir un nuevo monumento o museo en la base de datos, se piden los campos latitud y longitud. Esto tiene una finalidad muy simple. Dado que los monumentos y museos permanecen siempre en las mismas coordenadas, sólo deberemos lanzar una notificación cuando el usuario se encuentre cerca de esas coordenadas. Para ello, se definirá un área circular alrededor de las coordenadas del monumento o museo, con un radio de acción de trecientos metros. Cuando el usuario se encuentre dentro de esas coordenadas, la aplicación lanzará una notificación. A continuación se muestra una imagen con un ejemplo de área circular definida.

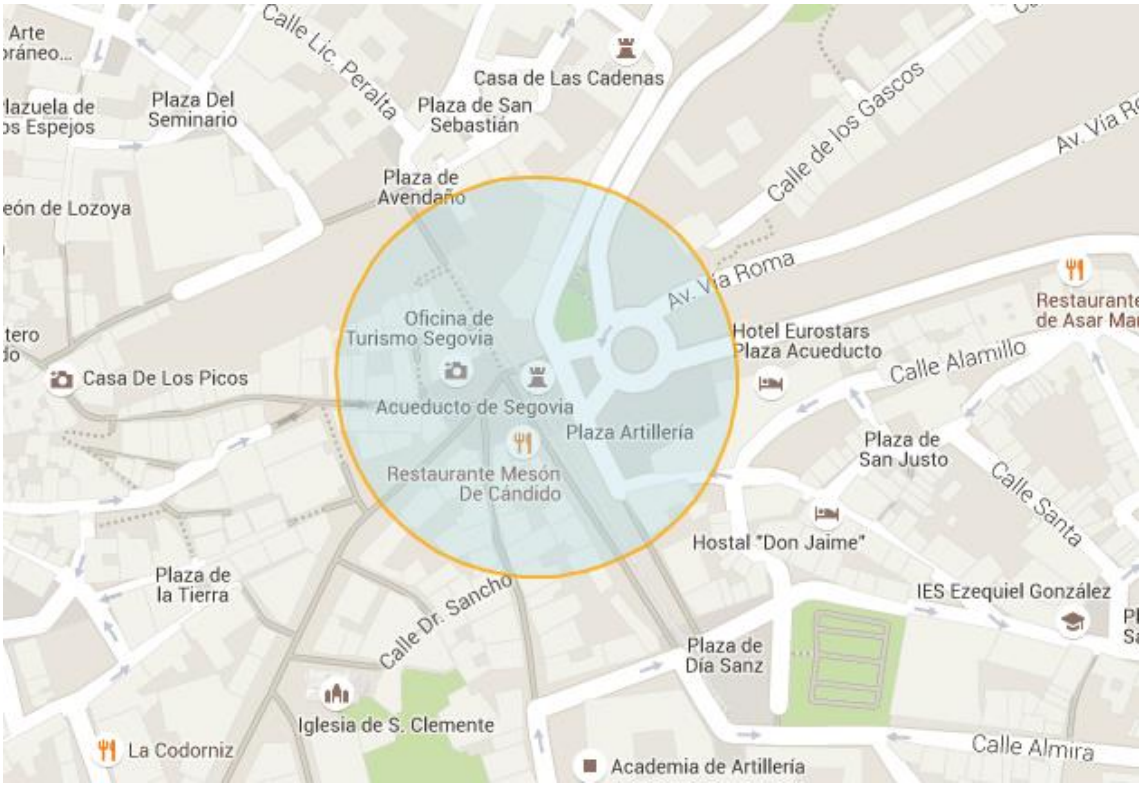

Ilustración 5.17 Ejemplo de área circular definida

El problema actual reside en cómo determinar si un usuario está dentro de esa área. Para ello, *Google* pone a disposición de cualquier usuario una serie de librerías[15]. Mediante el uso de una de esas librerías (concretamente "*geometry spherical"*) y con un poco de imaginación, se puede conseguir. Conociendo el centro de coordenadas (definido al introducir un monumento o museo en la base de datos), el radio (como se ha dicho antes, trecientos metros) y matemáticas, se determinó que si la distancia entre la posición del usuario (obtenida por la aplicación) y el centro de coordenadas del monumento es menor que el radio definido, entonces ese usuario estaría dentro del área circular. Si por el contrario, la distancia entre el usuario y el monumento es mayor que el radio, no estaríamos dentro.

Una vez que se tiene claro cómo hacerlo, encontrar una manera es lo primordial. La librería de google mencionada anteriormente posee una función para calcular distancias entre coordenadas y convertirlas al sistema decimal, para así compararlo con el radio. A esta función se le ha de pasar la posición del usuario y el centro de coordenadas del objetivo. Si esa distancia es menor que el radio, la aplicación lanzará la notificación.

Este sistema supone estar enviando constantemente la posición del usuario, y por cada nueva posición, enviar todos los centros de coordenadas de los monumentos y los museos, y además enviarlos en segundo plano, ya que no se vislumbra que el usuario vaya continuamente con la aplicación abierta y el móvil en la mano. En una primera instancia se supuso que esto conllevaría un gasto de batería y de datos móviles considerable, pero tras varias pruebas se observó que el gasto en una semana de uso habitual de la aplicación supuso un coste de 2,84 MB, y la batería apenas se vio afectada por el uso de la aplicación.

5.9 Patrón MVC

El **MVC (Modelo-Vista-Controlador)** es un patrón de arquitectura de software que divide por un lado los datos y la lógica de negocio y por otro la interfaz. Para ello, este patrón se compone de:

- **Modelo**: es la representación de la información con la que trabaja el sistema.
- **Controlador** : responde a acciones del usuario y se encarga de coger información del modelo cuando es necesario. También puede enviar ordenes a su vista asociada, es decir, hace de intermediario entre la vista y el modelo.
- **Vista**: es la representación del modelo y toda la lógica de negocio.

Este patrón se basa en la reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. En nuestra aplicación, diferenciamos dos partes claras, el lado cliente y el lado servidor. Ambas partes trabajarán con el mismo modelo, y cada una de ellas tendrá sus controladores y sus vistas.

[15] Más información en ANEXO I. API de Google Maps

5.9.1 Relación controladores-vistas

Lado cliente

Todos los controladores se encontrarán en un único archivo llamado script.js. En este archivo se definirá un controlador por cada vista de la aplicación, que se encargará de la obtención de datos y de la ejecución de scripts en cada vista. A continuación, se ofrece una tabla con las relaciones controladores-vistas y una breve descripción del funcionamiento del controlador. En el lado servidor, el lenguaje de desarrollo del controlador será *AngularJS.*

Controlador	Vista Asociada	Descripción
mainController	Home	Encargado de obtener todos los datos de monumentos y museos para el envío de notificaciones push
guideController	guide	Encargado de mostrar la lista de monumentos registrados en el sistema
monumentController	monument	Encargado de mostrar la información de un monumento concreto
legendController	legends	Encargado de mostrar la lista de leyendas registradas en el sistema
leyendaController	leyenda	Encargado de mostrar la información de una leyenda concreta
museumController	museums	Encargado de mostrar la lista de museos registrados en el sistema
museoController	museo	Encargado de mostrar la información de un museo concreto
festivalesController	festivals	Encargado de mostrar la lista de festivales registrados en el sistema

festivalController	festival	Encargado de mostrar la información de un festival concreto

Tabla 5.1 Relaciones controlador-vista lado cliente

Lado Servidor

Todos los controladores se encontrarán en la carpeta *Controllers*. Cada archivo ofrece una serie de métodos de los cuales se beneficiaran las vistas de la interfaz de administración y los controladores del lado cliente. A continuación, se ofrece una tabla con las relaciones controladores-vistas y una breve descripción del funcionamiento del controlador. En el lado servidor, el lenguaje de desarrollo del controlador será *NodeJS.*

Controlador	Vista Asociada	Descripción
index	Index, login	Encargado de la gestión de inicio de sesión y de mostrar los componentes de administración
monumento	Anadirmonumento, guidemonumento, monumento	Encargado de toda la gestión de monumentos, a saber: crear, borrar, listar y ver información. Proporciona los métodos para que el cliente pueda obtener datos del modelo
leyenda	Anadirleyenda, guideleyenda, leyenda	Encargado de toda la gestión de leyendas, a saber: crear, borrar, listar y ver información. Proporciona los métodos para que el cliente pueda obtener datos del modelo
festival	Anadirfestival, guidefestival, festival	Encargado de toda la gestión de festivales, a saber: crear, borrar, listar y ver información. Proporciona los métodos para que el cliente pueda obtener datos del modelo
museo	Anadirmuseo, guidemuseo, museo	Encargado de toda la gestión de museos, a saber: crear, borrar, listar y ver información. Proporciona los métodos para que el cliente pueda obtener datos del modelo

Tabla 5.2 Relaciones controlador-vista lado servidor

5.10 Diseño de interfaz

Diseñar la interfaz sea quizá el mayor problema para un programador. Debe tener un aspecto sencillo, intuitivo y fácil, que con un solo vistazo el usuario perciba todas las opciones que puede realizar. Para realizar este diseño, se utilizará el conjunto de librerías *Bootstrap*. Estas librerías ofrecen una serie de clases prediseñadas y la ventaja que tienen es que la mayoría de esas clases están adaptadas a dispositivos móviles.

A continuación, se muestran los diseños propuestos para la aplicación, divididos en dos partes: lado cliente (aplicación *Android*) y lado servidor (interfaz de administración web)

Lado cliente

Ilustración 5.18 Diseño de pantalla principal de la aplicación

Este será el diseño de nuestra pantalla principal de la aplicación. Se proponen cuatro botones, en los que cada uno de ellos dirigirá a un listado concreto, a saber monumentos, leyendas, museos o mapas. Es un diseño de interfaz sencillo y de fácil entendimiento.

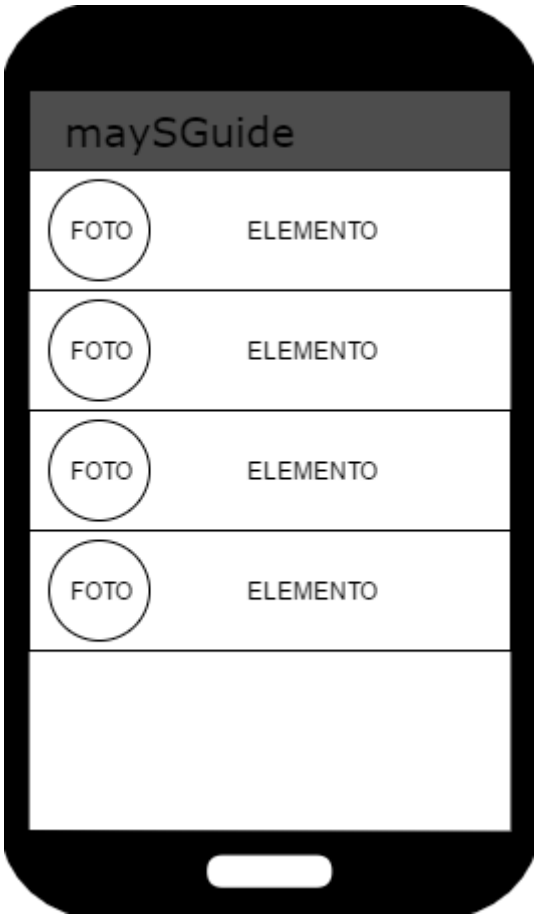

Ilustración 5.19 Diseño de interfaz listado

En la Ilustración anterior se muestra el tipo de listado que queremos que se muestre en la aplicación. Pulsando cualquiera de las opciones de la pantalla principal, nos llevará a un listado con la información respectiva, dependiendo de la opción que se pulse. Si se pulsa la opción monumentos, se mostrará un listado de monumentos y así con las demás opciones.

Ilustración 5.20 Diseño de interfaz de información

Por último, este será el último diseño propuesto para la parte cliente. Al pulsar sobre un elemento de un listado cualquiera, la aplicación dirigirá al usuario a una pantalla con este diseño, en la que mostrará información concreta sobre el elemento pulsado.

Lado servidor

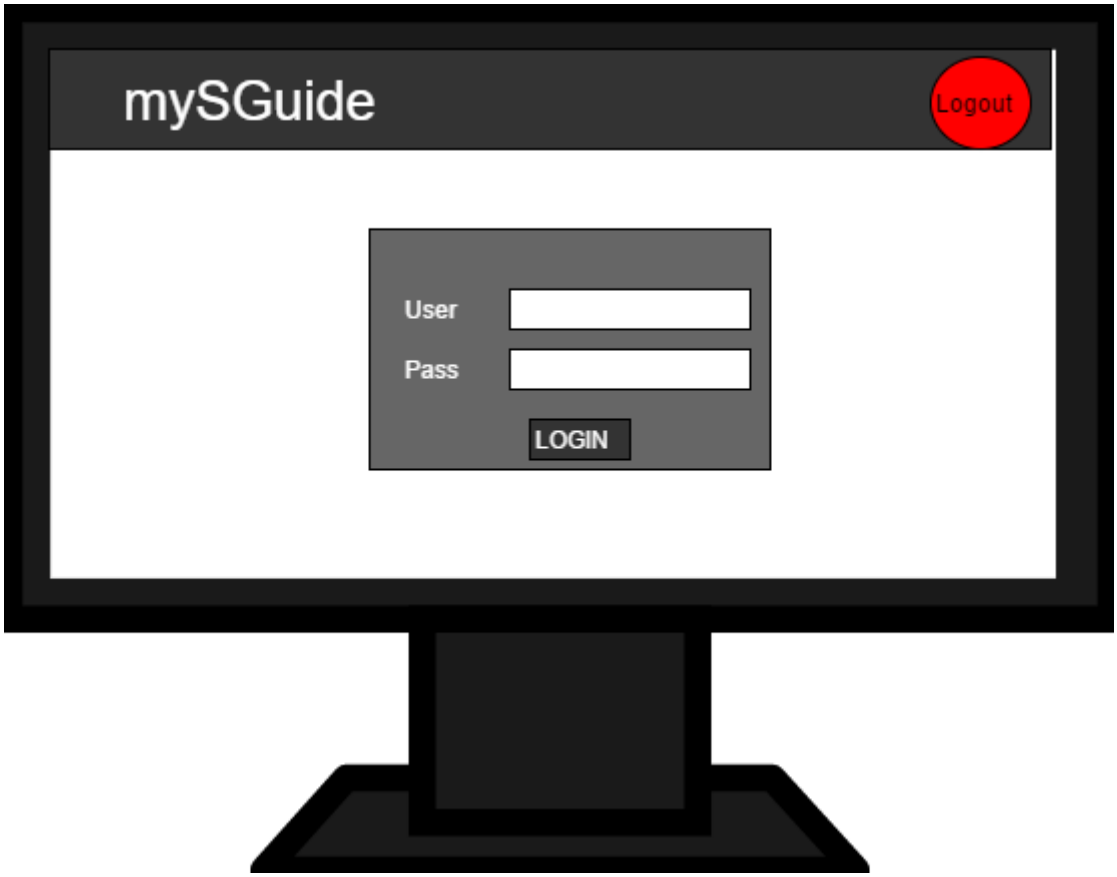

Ilustración 5.21 Diseño de pantalla login

Este diseño se propone para la interfaz web del lado administrador, concretamente para la pantalla a la que el usuario administrador accede por primera vez. En ella se muestra un cuadro con un formulario de login. En la parte superior derecha, el botón de logout.

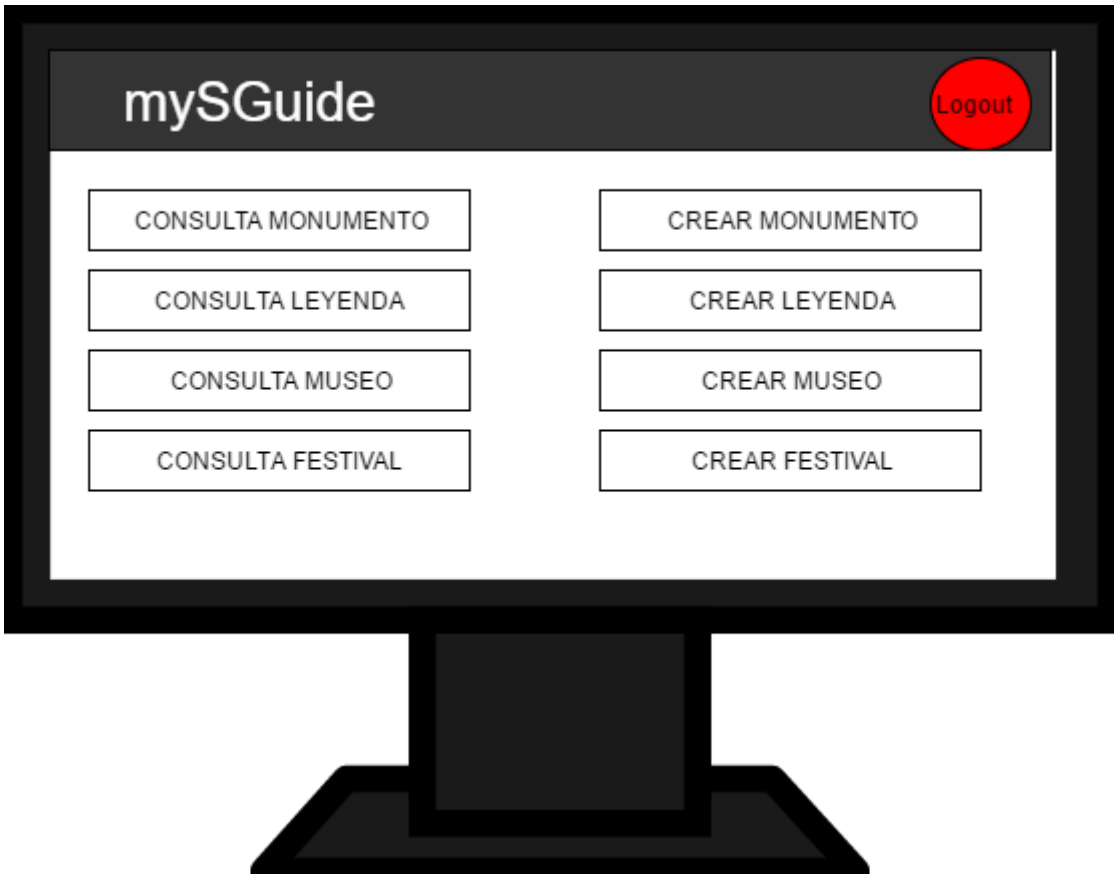

Ilustración 5.22 Diseño de pantalla principal de interfaz web

Se propone el diseño de la pantalla principal de la zona de administración. En ella se pueden observar las opciones disponibles para el usuario administrador. Todas las opciones de consulta mostrarán un listado con la información correspondiente a cada sección, mientras que las opciones de crear llevarán a un formulario de creación. Esta pantalla aparece cuando el usuario administrador se ha identificado correctamente en la pantalla de login.

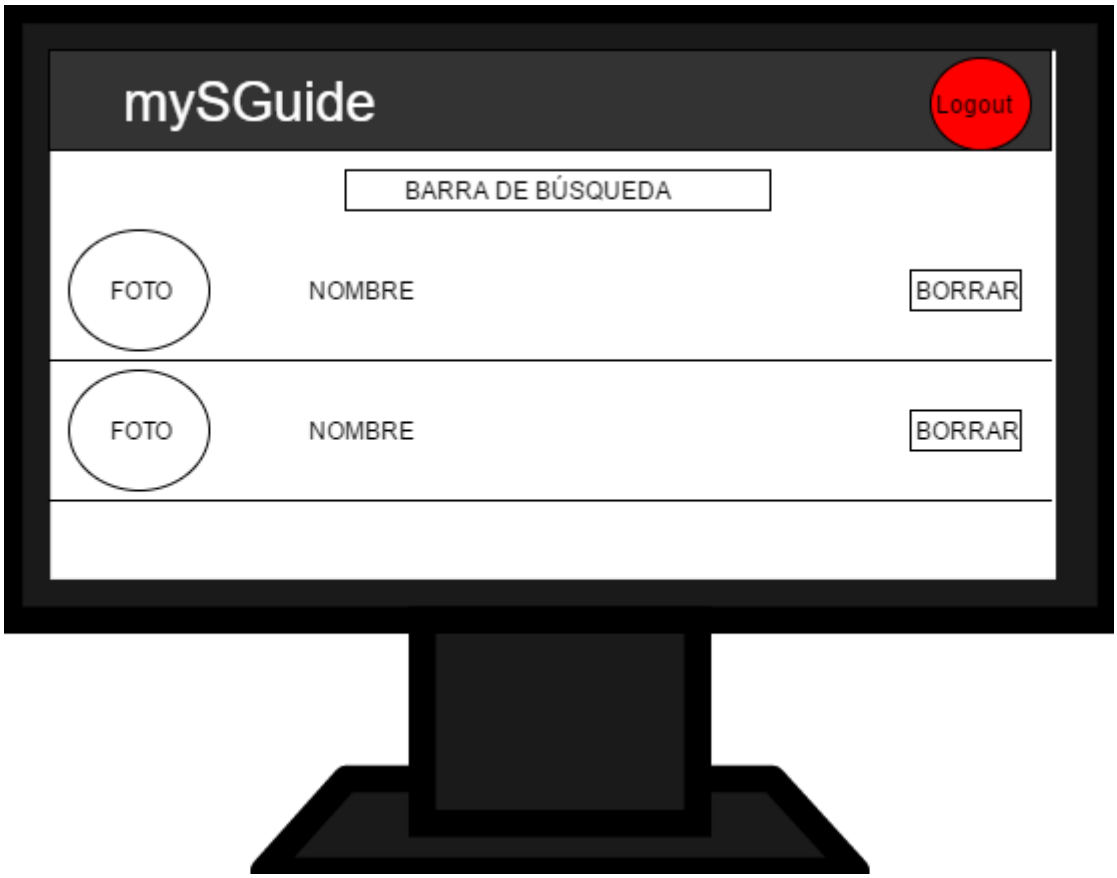

Ilustración 5.23 Diseño de pantalla consulta

El diseño de la pantalla consulta es similar al diseño de la pantalla listado de la aplicación móvil, sólo que esta nos permite además la opción de borrar un elemento. Cuenta además también con una barra de búsqueda en la parte superior. Esta pantalla aparece cuando el usuario administrador pulsa cualquiera de las opciones de consulta.

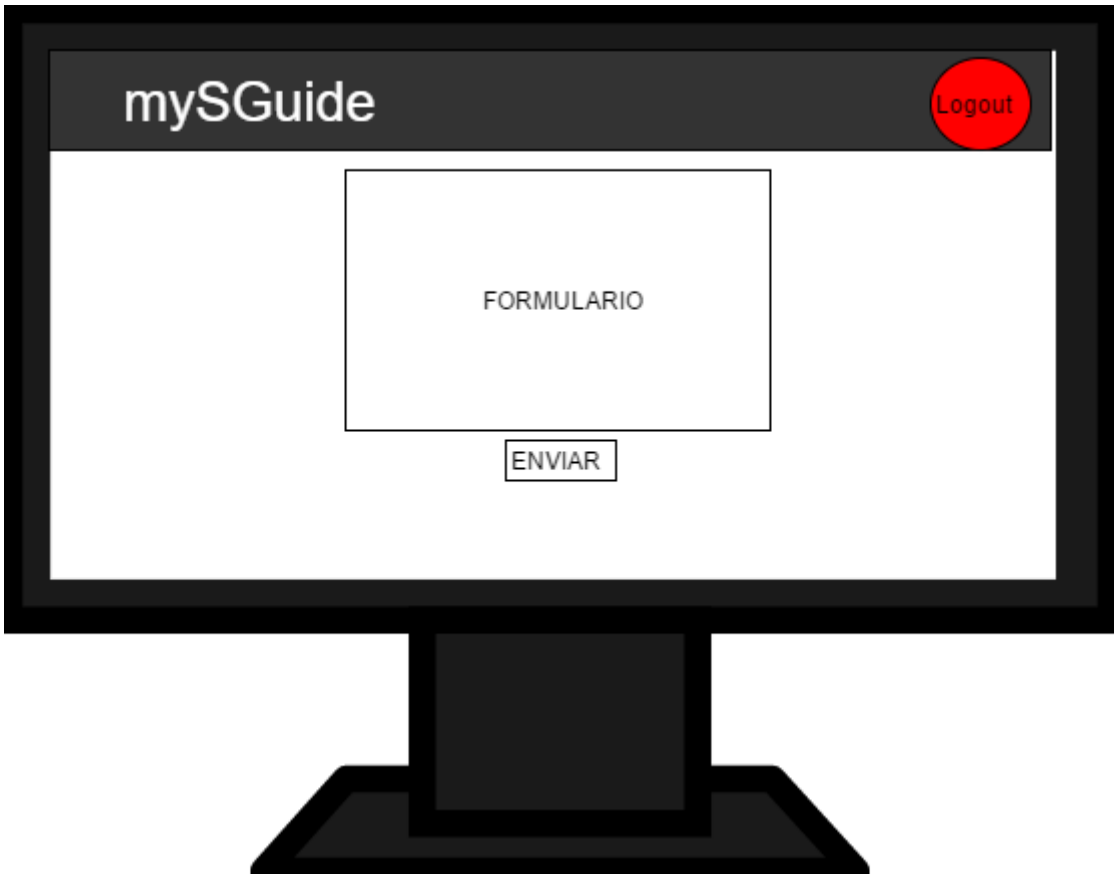

Ilustración 5.24 Diseño de pantalla crear

Por último, este es nuestro diseño para la pantalla de creación. Constará de un simple formulario que, dependiendo del elemento que queramos crear, aparecerá con unos campos u otros. Esta pantalla aparece cuando el usuario administrador pulsa cualquiera de las opciones crear.

6. IMPLEMENTACIÓN Y PRUEBAS

6.1 Consideraciones de implementación

A continuación se explicará el proceso detallado de la implementación del sistema. Toda esta aplicación se ha realizado desde un ordenador con sistema operativo Windows, por lo que es posible que si se sigue esta guía en otro sistema operativo, lo explicado a continuación no funcione. Comenzaremos explicando el desarrollo de la parte cliente (aplicación *Android*) y posteriormente el desarrollo del lado servidor. En ambos casos es necesaria la instalación de la herramienta *NPM*. Esta herramienta es un instalador de paquetes mediante comandos de consola, muy similar a la instalación de paquetes en Linux. Para instalarlo, basta con ir a su página oficial y descargarlo[16]. Una vez instalado, podremos acceder a los comandos desde la consola de Windows. Se decidió subir el servidor a la plataforma Heroku para evitar problemas de sobrecarga de procesos en el portátil y para que, una vez terminado, tenerle accesible desde cualquier parte.

En esta sección se describirá el proceso de implementación de la sección monumentos de la aplicación. Dado que se utiliza el patrón MVC, el desarrollo de las secciones "Leyendas", "Festivales" y "Museos" se basará en la reutilización de código de la sección "Monumentos", adaptándolo a las necesidades de cada sección. De esta manera, ahorramos tiempo de desarrollo y ofrecemos una aplicación más completa. No se ha estipulado ningún convenio de nombres en la implementación. Las variables, clases y métodos en el lado cliente están definidos en inglés, mientras que en el lado servidor están definidos en español.

Lado Cliente

Se replicarán los controladores de listado de monumentos y de información de monumentos, creando así las secciones de Leyendas, Festivales y Museos.

Lado servidor

Se replicarán los controladores y vistas de monumentos adaptadas a las nuevas secciones.

[16] Instalación de npm: https://www.npmjs.com/package/npm

6.2 Implementación lado cliente (*Android)*

Para empezar, deberemos instalar la herramienta con la que desarrollaremos la aplicación, Apache Cordova. Para ello, iniciamos la consola de comandos de Windows y ejecutamos el siguiente comando:

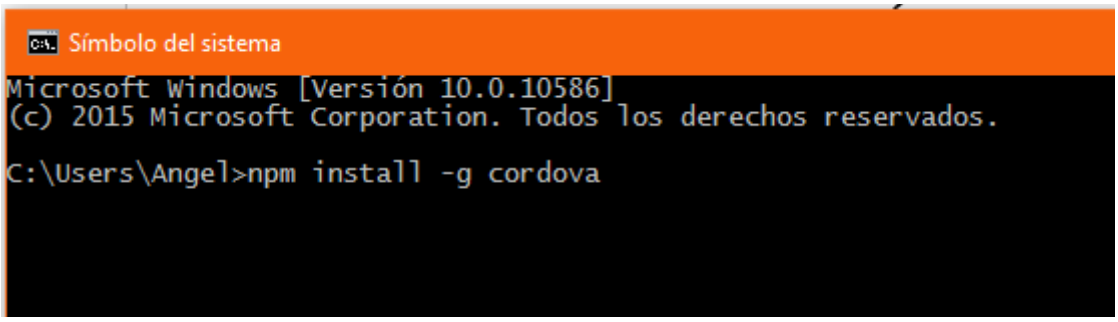

Ilustración 6.1 Instalación de Apache Cordova

Con este sencillo comando, el instalador de paquetes npm se encargará de localizar los archivos de Apache Cordova en su repositorio y de instalarlos en nuestro ordenador. El "-g" indica que queremos instalarlo de manera global, para así poder utilizarlo en cualquier parte de nuestro ordenador.

Una vez que tenemos instalado Cordova, procedemos a crear el proyecto. Con el siguiente comando, crearemos un nuevo proyecto Cordova:

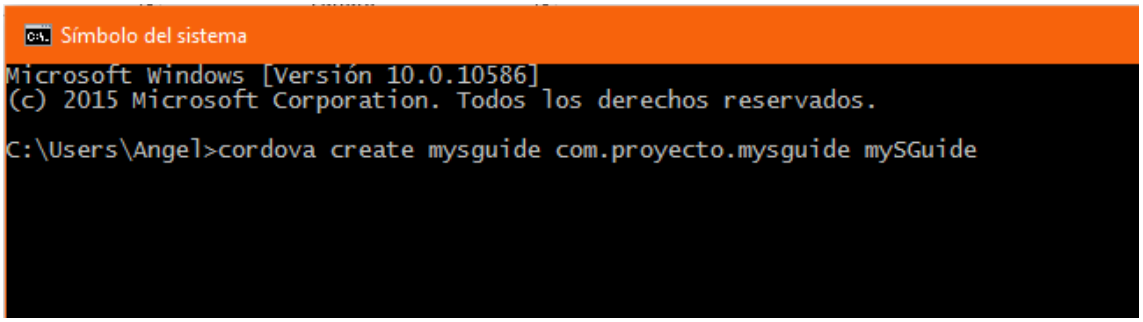

Ilustración 6.2 Creación de un proyecto Cordova

Este comando creará un proyecto Cordova con sus componentes iniciales, descritos en el capítulo 3 de este documento, sección 3.2. Tenemos que indicar el nombre del proyecto (mysguide), el nombre del paquete (com.proyecto.mysguide) y el directorio

en el que alojará todos los archivos (mySGuide). Tras esto, mediante el comando "*cd mySGuide",* nos moveremos dentro del directorio y observamos su contenido:

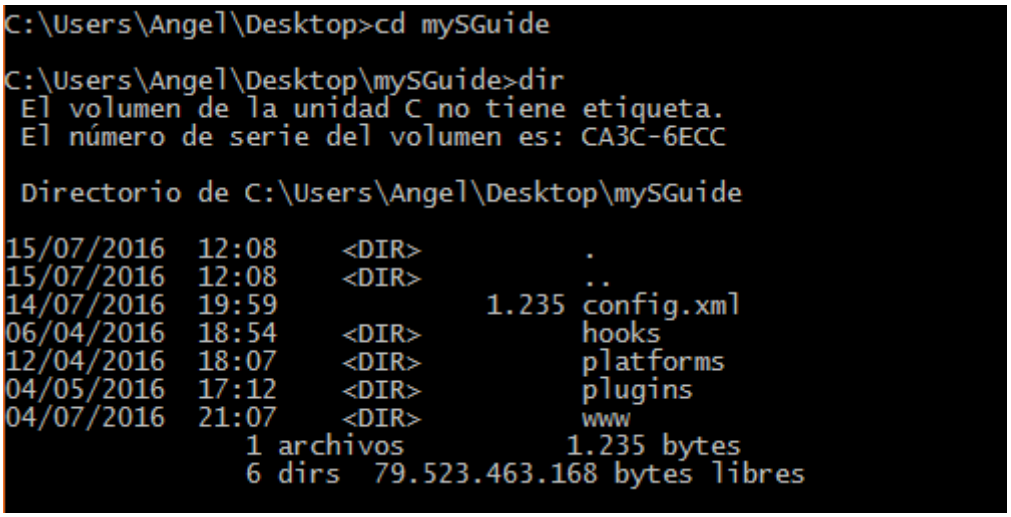

Ilustración 6.3 Estructura de nuevo proyecto Cordova

Una vez que comprobamos que el proyecto se ha creado correctamente, procedemos a incluir las plataformas móviles que queremos para nuestra aplicación. Como esta aplicación va a ser una aplicación *Android,* sólo añadiremos esa.

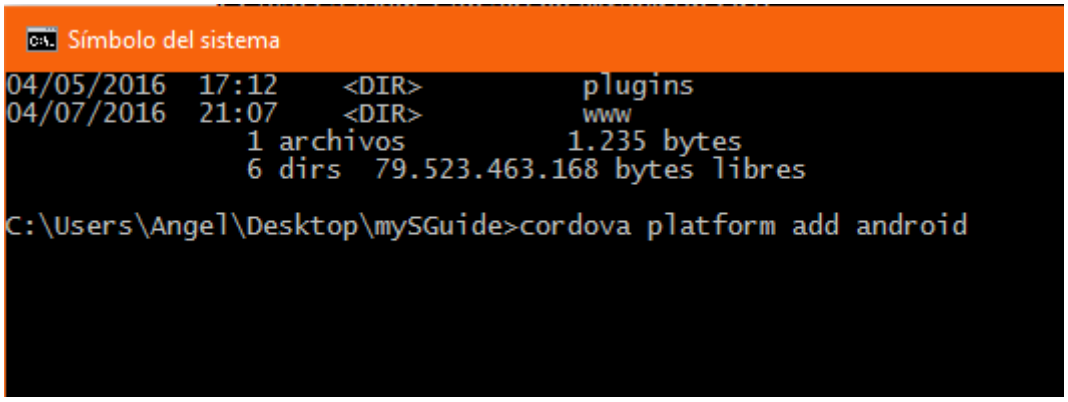

Ilustración 6.4 Inclusión en el proyecto de la plataforma Android

Y por último, para dejar nuestro proyecto listo para trabajar en el, añadimos los *plugins* que necesitaremos para el correcto funcionamiento de la aplicación, explicados en el capítulo 3 de esta documentación, sección 3.4.

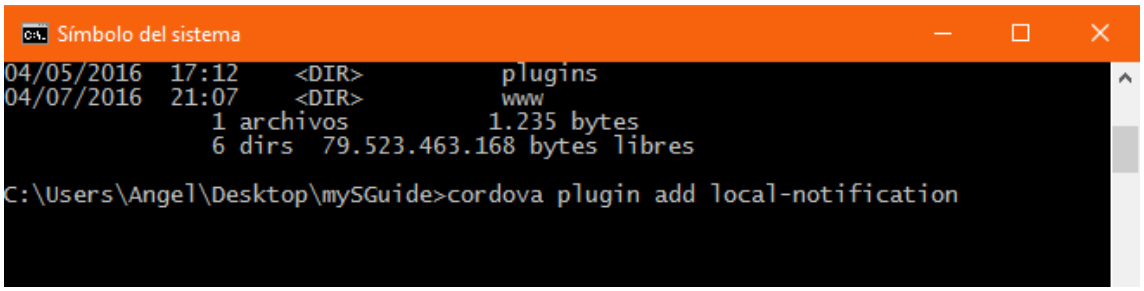

Ilustración 6.5 Inclusión de los plugins necesarios para el proyecto

Con todo esto, tenemos la plataforma lista para empezar a trabajar con ella.

Lo primero que se implementó fue la estructura de la aplicación. Como se explicó en el apartado 3 de este documento, todo el código de la aplicación reside en la carpeta "*www*".

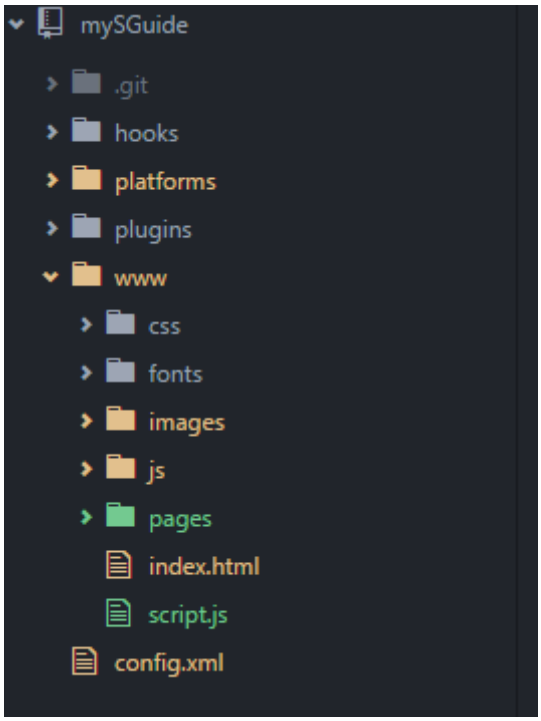

Ilustración 6.6 Estructura del proyecto mySGuide

En la carpeta *www* encontramos varias carpetas a su vez:

- Carpeta *CSS*: contendrá los archivos que darán estilo a la aplicación.
- Carpeta *Fonts*: contendrá los archivos con la tipografía de fuente elegida.
- Carpeta *images:* contendrá las imágenes necesarias para el diseño de la aplicación.
- Carpeta *JS*: contendrá los archivos *Javascript* para el diseño de la aplicación.
- Carpeta *Pages*: contendrá todas las vistas de la aplicación.
- Archivo *index.html*: archivo inicial al que entra la aplicación cuando arranca.
- Archivo *script.js*: contiene los scripts necesarios para la redirección entre páginas, conexión con el *API Rest* y para el comportamiento de la aplicación.

Se ha decidido crear una aplicación con un solo archivo principal en el que cargamos vistas para así aumentar su rendimiento. En la pantalla principal, se va inyectando el contenido en concreto. Además, así evitamos tener que cargar librerías, estilos y archivos cada vez que cambiamos de pantalla, dado que al inyectar una vista, lo que mantenemos constante es el marco de la aplicación en la que vamos cambiando el contenido mostrado.

```
<!-- Inicio de aplicación angular -->
<html ng-app="myApp">
<head>
  <!-- SCROLLS -->
  <!-- Carga de boostrap y estilos-->
  <link rel="stylesheet" href="http://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css" />
  <link rel="stylesheet" href="http://netdna.bootstrapcdn.com/font-awesome/4.0.0/css/font-awesome.css" />
  <link href="css/bootstrap.css" rel='stylesheet' type='text/css' />
  <link href="css/style.css" rel='stylesheet' type='text/css' />
  <script src="js/jquery-1.11.1.min.js"> </script>
  <script src="js/bootstrap.js"></script>
  <script type="text/javascript" src="cordova.js"></script>

  <!-- SPELLS -->
  <!-- Carga de las librerías de angular-->
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.7/angular.min.js"></script>
  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.5.7/angular-route.js"></script>
  <script src="script.js"></script>
  <script src="js/ng-map.min.js"></script>
```

Ilustración 6.7 Carga de librerías y recursos de la aplicación

En este archivo principal es donde se inyectan las vistas.

```
<!-- angular controller -->
<body ng-controller="mainController">
  <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDro9WMsSyL6RDfKqB0uI4cmrb_MdaHIiU&signed_in=true&libraries=geometry"
      async defer>
  </script>



<!-- Contenido principal y vistas inyectadas-->
<div id="main">

    <!-- Plantillas angular -->
    <!-- aquí es donde se inyecta el contenido -->
    <div ng-view></div>

</div>

</body>
```

Ilustración 6.8 Zona de inyectado de vistas html

En "*div ng-view"* es donde inyectaremos las vistas. Hasta ahora lo único que tenemos de la aplicación es la cabecera y nada más. Se ha decidido inyectar una serie de vistas, alojadas en la carpeta "*Pages"*. Esta carpeta contendrá las vistas de pantalla principal, las páginas de cargado de los diferentes listados y la ficha de cada uno de los elementos de esos listados.

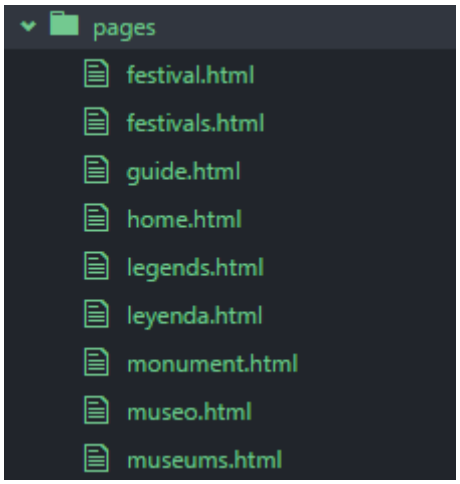

Ilustración 6.9 Vistas creadas para la aplicación mySGuide

La vista principal, que servirá para desplazarse por todas las demás, es *home.html*. Es la vista inicial que se carga en la pantalla principal. Contiene cuatro botones que servirán para dirigir al usuario por las diferentes secciones.

```
<!--Primer ban Monumentos-->
<div class="col-md-6 banner-grid">
  <a href="#guide"><!--LINK-->
  <div class="banner-left-grid blue">
    <div class="banner-left-icon">
      <div class="services-icon">
        <span class="glyphicon glyphicon-tower" aria-hidden="true"></span>
      </div>
    </div>
    <div class="banner-grid-info">
      <h3>Monumentos</h3>
      <p>Entra y descubre rincones que jamás habrías imaginado</p>
    </div>
    <div class="clearfix"> </div>
  </div>
</a>
</div>
```

Ilustración 6.10 Botón de home “monumentos”

Por tanto, tenemos la vista *Guide.html*, que se encargará de mostrar un listado de monumentos, con su correspondiente ficha de monumento, llamado *monument.html*. En los demás casos es de manera similar. Estos archivos *html* se inyectan en la vista principal y muestran información al usuario. Todo esto es posible gracias a las rutas definidas en el archivo *script.js*.

```
(function(){
    var app = angular.module('myApp', ['ngRoute']);

    // configure our routes
    app.config(function($routeProvider) {
        $routeProvider

            // ruta para index
            .when('/', {
                templateUrl : 'pages/home.html',
                controller  : 'mainController'
            })

            // ruta para guia monumentos
            .when('/guide', {
                templateUrl : 'pages/guide.html',
                controller  : 'guideController'
            })
            // ruta para guia leyendas
            .when('/legends', {
                templateUrl : 'pages/legends.html',
                controller  : 'legendController'
            })
```

Ilustración 6.11 Rutas creadas para la aplicación mySGuide

Esto significa que cuando se pinche determinado enlace, por ejemplo "*/guide*", este enrutador cargará en el *div ng-route* la plantilla indicada en el mismo, que pare este caso de ejemplo sería "*pages/guide.html*". Este enrutador también asigna un controlador, es decir, un "método" que se encarga de definir el comportamiento de esa plantilla. Para este ejemplo concreto, el controlador de *guide* sería el siguiente.

```
//página de guide
app.controller('guideController', function($scope,$http) {
  var myurl = 'https://mysguide.herokuapp.com/monumentos/json';
  $scope.monumentos=[];
  $http({
    method: 'GET',
    url: myurl
  }).then(function successCallback(response) {

      $scope.monumentos=response.data.monumentos;
      console.log(response.data.monumentos);

    }, function errorCallback(response) {

      console.log('ERROR!!!');
    });
});
```

Ilustración 6.12 Controlador de página Guide.html

Este controlador obtiene los datos del *API Rest* mediante HTTP. Una vez coge esa información, la introduce en un "*scope",* es decir, un lanzador, que enviará esa información a la página *guide.html* para que esta trabaje con ella. Como se puede observar, con angular es una tarea bastante sencilla. Pero la obtención de datos no es lo único que debe hacer la aplicación, sino que debe representarlos y mostrarlos adecuadamente. Para ello, debemos ir al archivo *guide.html*. Este archivo, al igual que los demás de listado, son una plantilla vacía, que necesitan la información que obtienen del *API Rest* para rellenarse.

```
<!-- /{{monumento._id}}-->
<div class="box" ng-controller="guideController" style="padding-top: 0;margin-top: -12px;" ng-repeat="monumento in monumentos">
<a ng-href="#monument/{{monumento._id}}" ><p><img class="alineadoTextoImagenCentro" ng-src="https://mysguide.herokuapp.com/monumentos/image/{{
monumento._id }}" /><span class="etiqueta">{{ monumento.Nombre }}</span></p></a>
</div>
```

Ilustración 6.13 Página Guide.html

Como se puede observar en la Ilustración anterior, a la página *guide.html* le asignamos el controlador definido en la Ilustración 6.11. Esto significa que a este div en concreto, le llegara la información que hemos almacenado en el *scope* anterior. Mediante angular, creamos un bucle que irá pintando una estructura base tantas veces como información (monumentos en ese caso) contenga ese *scope*, de tal manera que se nos irá rellenando esa plantilla definida. Funciona igual para todos los listados.

Para mostrar la información de esos listados, utilizaremos un archivo que haga una petición al servidor y muestre los datos. El código es el siguiente.

```
// ruta para pagina de monumento
.when('/monument/:monumentId', {
    templateUrl : 'pages/monument.html',
    controller  : 'monumentController'
})
```

Ilustración 6.14 Método para obtener el id

Al definir la ruta, para poder obtener la información de un monumento en concreto, y dado que en el lado servidor tenemos programada un método que devuelve información enviando el id, al pinchar un monumento, deberemos coger su id y enviarlo al servidor, de la siguiente manera.

```
var idMonument = $routeParams.monumentId;
var urlMonument = 'https://mysguide.herokuapp.com/monumentos/'+idMonument+'/json';
console.log(urlMonument);
$scope.infoMonumento = [];
```

Ilustración 6.15 Petición para obtener un JSON concreto

De esta manera, obtendremos el JSON de un monumento con un id concreto. Tras esto, mostrar la información de ese JSON es similar al caso del listado.

```
<div ng-controller="monumentController"
  <div align="center">
    <!-- https://apirestsegotour.herokuapp.com/monumentos/image/ -->
    <img  src="https://mysguide.herokuapp.com/monumentos/image/{{ infoMonumento._id }}" >
  </div>
    <br>
    <p class="title"> {{ infoMonumento.Nombre }}</p>
    <hr>
  <div class="text">
    <p style="font-size:16px;">{{ infoMonumento.Descripcion }}</p>
    <hr>
    <br><br>
  </div>
</div>
```

Ilustración 6.16 Mostrar información de monumento

Por último, queda definir como lanzar las notificaciones y geolocalizar al usuario. Se hace todo en un mismo método, que expondremos a continuación. Se usará el sistema de coordenadas decimal, usando grados decimales, para así poder utilizar las funciones de *Google* mencionadas en el apartado anterior.

Mediante la siguiente función, geolocalizaremos al usuario:

navigator.geolocation.getCurrentPosition(success, error, options);

Esto significa lo siguiente. Mediante *navigator.geolocation* accederemos a la geolocalización de *HTML5*, y llamamos *getCurrentPosition* al método que nos irá cogiendo coordenadas. A este método a su vez hay que pasarle otros dos métodos, *success y error* y una variable *options*, que determinará la precisión del *GPS* que utilizará la aplicación, el tiempo máximo de respuesta y el tiempo máximo que podemos usar la información.

```
var options = {
  enableHighAccuracy: true,
  timeout: 5000,
  maximumAge: 0
};
```

Ilustración 6.17 Variable *options*

```
function error(err) {
  console.warn('ERROR(' + err.code + '): ' + err.message);
};
```

Ilustración 6.18 Función error

```
var googlePos = new google.maps.LatLng(pos.coords.latitude, pos.coords.longitude);

for (var i=0; i< locations.length; i++){
  var monument=new google.maps.LatLng(locations[i].Latitud,locations[i].Longitud);
  if (google.maps.geometry.spherical.computeDistanceBetween(googlePos, monument) <= 300){
```

Ilustración 6.19 Geolocalización del usuario

En la última Ilustración mostrada, se muestra como geolocalizar al usuario. Mediante *pos.coords.latitude y pos.coords.longitude*, la aplicación es capaz de acceder al *GPS* y obtener las coordenadas del usuario en tiempo real. Para comprobar que se encuentre cerca de un monumento o museo, todos los datos obtenidos del *API Rest* se introducen en un array llamado “locations”. Teniendo la posición del usuario y las coordenadas en un array, vamos comparando, y cuando encuentre una coincidencia, lanzará la notificación.

```
//Granted Permission for Push notifications
cordova.plugins.notification.local.hasPermission(function (granted) {
    //console.log('Permission has been granted: ' + granted);
});

cordova.plugins.notification.local.registerPermission(function (granted) {
    //console.log('Permission has been registered: ' + granted);
});

//Generate a push notification
cordova.plugins.notification.local.schedule({
  title: "Atención : "+locations[i].Nombre+" cerca !",
  message: "Pulsa para ver más información"
});

//Redirecting push to the information
var urlId=locations[i]._id;
var url="#monument/"+urlId;
cordova.plugins.notification.local.on("click", function (notification, state) {
  window.location = url;
}, this);
```

Ilustración 6.20 Código que genera una notificación push

Una vez que se tiene la aplicación construida y completamente funcional, llega el momento de generar el archivo de instalación en dispositivos Android.

```
C:\Users\Angel\Desktop\mySGuide>cordova build
```

Ilustración 6.21 Construcción del archivo .apk

Mediante el comando "*cordova build*", la plataforma apache cordova se encarga de convertir todo lo que hemos programado en un archivo .apk, que permitirá la instalación de la aplicación en dispositivos Android.

6.3 Implementación lado servidor

La implementación del lado servidor seguirá un proceso similar al de la creación de la aplicación. Lo primero, debemos instalar la base de datos *MongoDB*, descargando el paquete de su página oficial[17]. Una vez descargado el paquete, se instala de manera normal en un entorno Windows. Para el desarrollo de la *API Rest*, instalaremos un gestor de instaladores de proyectos llamado Yeoman. Su instalación desde consola es muy sencilla. Partiendo de un directorio base, se realiza lo siguiente:

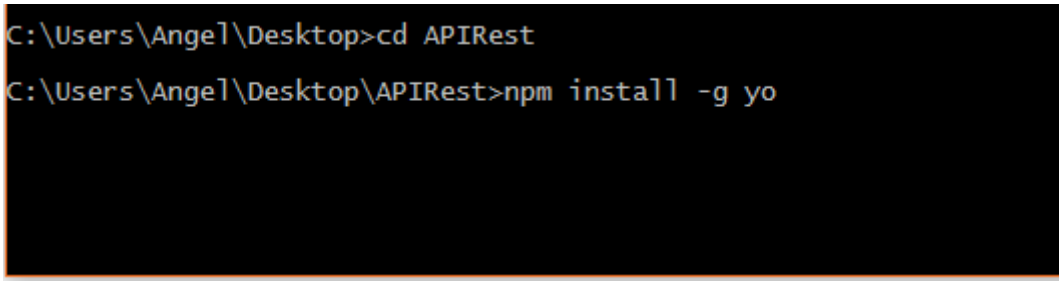

Ilustración 6.22 Instalación de Yeoman

A continuación, instalaremos también un generador de proyectos, concretamente generartor-express, de la misma manera que Yeoman:

Npm install –g generator-express

Una vez instaladas estas dos herramientas, tecleando el comando "**yo express**" se nos creará un proyecto nuevo. Pero antes deberemos responder a una serie de preguntas que nos hará Yeoman:

```
? Would you like to create a new directory for your project? No
? Select a version to install: MVC
? Select a view engine to use: EJS
? Select a css preprocessor to use (Sass Requires Ruby): None
? Select a database to use: MongoDB
? Select a build tool to use: Grunt
```

Con las respuestas marcadas, Yeoman nos creará un proyecto con el siguiente contenido:

[17] Descarga de mongoDB: https://www.mongodb.com/

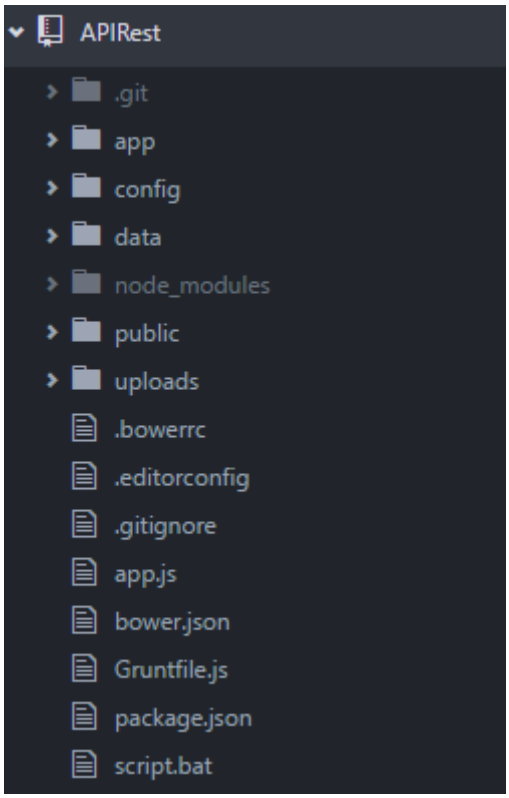

Ilustración 6.23 Estructura del proyecto

El código del servidor irá alojado en la carpeta "App", que a su vez está conformada por tres carpetas, siguiendo el patrón MVC.

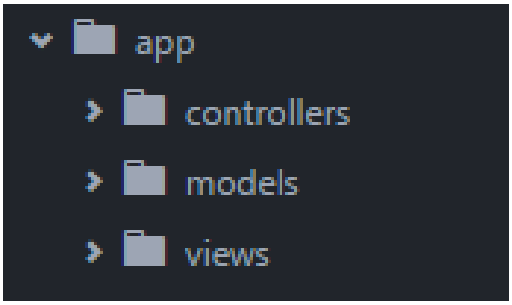

Ilustración 6.24 Contenido carpeta app del servidor

Tras esto, empezamos a programar. Lo primero es arrancar la base de datos, que se hará mediante un script en formato .bat (archivo por lotes ejecutable), cuyo contenido será la ruta en la que se encuentra instalada:

"C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe" --dbpath ./data

Una vez arrancada la base de datos, arrancamos el servidor. Mediante el comando "*Grunt"*, el servidor entra en funcionamiento, y podremos acceder a él a través de una URL en el navegador, concretamente http://localhost:3000/ .

Con el servidor arrancado, ya se puede empezar a trabajar en él. Primero crearemos los modelos de datos. Estos serán las colecciones en la base de datos que utilizaremos.

```
var mongoose = require('mongoose'),
  Schema = mongoose.Schema;

var MonumentoSchema = new Schema({
  Nombre:             { type: String },
  Latitud:            { type: Number },
  Longitud:           { type: Number },
  image:              { data: Buffer, contentType: String },
  Descripcion:        { type: String },
});

mongoose.model('Monumento', MonumentoSchema);
```

Ilustración 6.25 Modelo de datos para colección monumentos

Tras crear el modelo, lo siguiente es implementar el controlador. En él se definirán las rutas que tendrá el servidor, los métodos get para obtener información del mismo y los métodos para obtener información concreta de la base de datos.

```
//Return  JSON
router.get('/json', function(req, res, next) {
  monumentos.find({},{image:0},function(err, monumentos){
    if(err) return next(err);
    res.json({monumentos});
  });
});
```

Ilustración 6.26 Método JSON

El método anterior devuelve un JSON con toda la información sobre los monumentos.

```
router.get('/:ID/json', function(req, res, next) {
  monumentos.findOne({ _id: req.params.ID },{image:0},function(err, monumento){
    if(err) return next(err);
    res.json(monumento);
  });
});
```

Ilustración 6.27 Método JSON :id

Y este otro método devuelve el JSON de un monumento con una ID concreta. Estos dos métodos son los que utilizará la aplicación para obtener los datos, pero la programación del servidor no se queda ahí. Para poder interactuar con la base de datos, se creará una interfaz web, en la que se podrán añadir nuevas entradas en la base de datos, o por el contrario, borrar una tabla existente. Mediante los métodos post y delete, se crearán o borrarán entradas de la base de datos.

```
//Post
router.post('/',isLoggedIn,upload.single('imageupload'),function(req,res,next){

    console.log('El REQ.FILE:', req.file);

    var data = fs.readFileSync(req.file.path);

    var monumento = new monumentos({

      Nombre:    req.body.Nombre,
      Latitud:     req.body.Latitud,
      Longitud:     req.body.Longitud,
      Descripcion:     req.body.Descripcion,


  });

  monumento.image.data =  fs.readFileSync(req.file.path);
  monumento.image.contentType = req.file.mimetype;
  monumento.save(function (err) {
    if (err) return handleError(err);
  });
  res.redirect('/monumentos/anadir');
});
```

Ilustración 6.28 Método post

Este método permite incluir un nuevo dato en la base de datos.

```
router.delete('/:ID', function(req, res){

  monumentos.remove({ _id: req.params.ID }, function (err) {
    if (err) return handleError(err);
    res.redirect('/');
  });
});
```

Ilustración 6.29 Método delete

Y este otro método permite borrar una entrada.

Una vez programadas las funciones básicas del controlador, es hora de implementar las vistas. Node permite utilizar un tipo de archivos de extensión .ejs. Esto significa que podemos dejar archivos con un contenido determinado e incluirlos en otro archivo, algo así como un #include en C, por lo que las líneas de código comunes (como el head o el header de los archivos .html) irán en archivos aparte, y se añadirán en los archivos que sea necesarios.

La primera vista implementada será un index. A partir de aquí, el usuario administrador tendrá acceso a todas las opciones disponibles. Estará formada por una serie de botones, con un formato similar al de la aplicación.

```
<div class="col-md-6 banner-grid">
  <a href="/monumentos/guide"><!--LINK-->
  <div class="banner-left-grid green">
    <div class="banner-left-icon">
      <div class="services-icon">
        <span class="glyphicon glyphicon-th-list" aria-hidden="true"></span>
      </div>
    </div>
    <div class="banner-grid-info">
      <h3>Consulta de monumentos</h3>
      <p>Visualiza y borra monumentos</p>
    </div>
    <div class="clearfix"> </div>
  </div>
</a>
</div>
```

Ilustración 6.30 Botón Consulta de monumentos

La interfaz constará de ocho botones, como se ha indicado en el diseño de la interfaz. Cada botón redirigirá a una pantalla, habiendo dos pantallas tipo: listados y formularios de creación. El formulario de creación es simple, tal y como se indica a continuación.

```
<html>
<% include head %>
<body>

  <% include header %>
  <article style="margin-top:80px">

    <center>
        <form action="/monumentos" method="post" enctype="multipart/form-data" >
          Nombre
          <input type="text" name="Nombre">
          <br><br><br>
          Latitud
          <input type="Number" step="any" name="Latitud">
          <br><br><br><br>
          Longitud
          <input type="Number" step="any" name="Longitud">
          <br><br><br><br>
          <input name="imageupload" id="imageupload"  type="file" accept="image/*" />
          <br><br>
          Descripción
          <textarea name="Descripcion"></textarea>
          <!-- <input type="text" name="Descripcion"> -->
          <br><br><br>

          <input type="submit" value="Aceptar">
          <br>
```

Ilustración 6.31 Formulario de creación de monumentos

En este archivo se puede observar la inclusión de archivos externos, como se ha explicado antes, mediante las etiquetas "*<%include nombre_archivo%>*". El formulario es un típico formulario html,

Sin embargo, el archivo de listado es un poco más complejo. A continuación, se muestra la implementación.

```
    <FORM method=GET action="/monumentos">
      <TABLE bgcolor="#FFFFFF"><tr><td>
        <span class="glyphicon glyphicon-search" aria-hidden="true"></span>
        <INPUT TYPE=text name=q size=31 maxlength=255 value="">
          <INPUT type=submit  VALUE="Búsqueda">
          </td></tr></TABLE>
    </FORM>
</center>

<% for(var i=0; i<monumentos.length; i++) { %>
  <div class="box">

      <a href="/monumentos/<%= monumentos[i]._id %>"><img class="alineadoTextoImagenCentro" src=/monumentos/image/<%= monumentos[i]._id %> ><span
      class="etiqueta"><%= monumentos[i].Nombre %></span></p></a>
      <form action="/monumentos/<%= monumentos[i]._id %>?_method=delete" method="post" >
        <button class="alineado" type="submit">Borrar</button>
      </form>


  </div>
  <% } %>
```

Ilustración 6.32 Listado de monumentos

Al principio del todo vemos un formulario. Es el formulario de la barra de búsqueda, que permitirá buscar información si el listado es demasiado grande. Tras eso, mediante un bucle for, recorremos la base de datos y mostramos todo el contenido de la misma, de manera similar a la aplicación. Este diseño cuenta además con la inclusión de un botón borrar, que en un momento dado, nos permite eliminar un elemento de la base de datos.

- **Problemas encontrados en el desarrollo**

  - Inicio de sesión

    Uno de los grandes problemas encontrados es todo el tema de iniciar sesión. Mediante el módulo Passport, este problema quedo prácticamente resuelto. *Passport*[18] es un middleware de autenticación para *Node.js*. Es extremadamente flexible y fácil de usar. Su instalación es simple. Basta con estar en el directorio del proyecto y mediante consola, teclear :

    Npm install Passport

    Tras esto, en nuestro archivo de conIlustraciónción *Express.js* incluimos lo indicado en su documentación oficial. En este proyecto se usa *Passport* como sistema de autenticación en lado servidor, lo que significa que para que el administrador utilice cualquiera de las páginas de la zona de administrador, deberá registrarse correctamente. De lo contrario, no llegará más allá de la pantalla de login.

    La conIlustraciónción es sencilla, como se muestra en la imagen a continuación.

[18] http://passportjs.org/

```
passport.use(new LocalStrategy(
  function(username, password, done) {

    User.findOne({ username: username }, function(err, user) {
      if (err) { return done(err); }
      console.log("entro");
      if (!user) {
        console.log("no existe user");
        return done(null, false, { message: 'Incorrect username.' });

      }
      console.log("existe usuario");
      if (user.password != password) {
        console.log("contraseña erronea");
         return done(null, false, { message: 'Invalid password' });

       }
       console.log("existe");
      return done(null, user);
    });
  }
));
```

Ilustración 6.33 Checkeo de usuario registrado con passport

Como se aprecia en el código, este sirve para comprobar si el usuario está o no registrado en la base de datos. La principal ventaja de Passport es que nos proporciona un sistema de gestión de usuarios seguro, protegiendo contraseñas mediante serialización.

```
passport.serializeUser(function(user, done) {
  done(null, user);
});

passport.deserializeUser(function(id, done) {
  User.findById(id, function(err, user) {
    done(err, user);
  });
});
```

Ilustración 6.34 Serialización de contraseñas

Tras conIlustraciónr esto, sólo hace falta añadir una pantalla de login.

```
<html>
<% include head %>
<body>
  <% include header %>
  <center>
  <div class"container-fluid" style="background-color: #cac3c3; margin-top: 100px;
  width: 30%; border-radius: 3%; opacity: 0.8;padding-top: 35px;
    padding-bottom: 35px;">
    <form action="/" method="post">
        <div>
            <label>Username:</label>
            <input type="text" name="username"/>
        </div>
        <br>
        <div>
            <label>Password:</label>
            <input type="password" name="password"/>
        </div>
        <br>
        <div>
            <input type="submit" value="Log In"/>
        </div>
    </form>
  </div>
</center>
</body>
</html>
```

Ilustración 6.35 Pantalla login

Para conIlustraciónr a qué páginas afecta Passport, basta con incluir el método proporcionado por este llamado "*isLoggedIn*" , como se puede observar en la Ilustración 6.28 de esta documentación.

- Método borrar

  Dado que los navegadores actuales a día de hoy no aceptan el método *Delete* en los formularios, se tuvo que buscar otra forma de enviar solicitudes de borrado a la base de datos. Mediante el paquete *methodOverride*, y añadiendo un método, es posible realizarlo. Añadiendo el método en la parte del formulario, este paquete reconoce la solicitud de borrado, y procede a eliminar la entrada de la base de datos.

```
<form action="/monumentos/<%= monumentos[i]._id %>?_method=delete" method="post" >
  <button class="alineado" type="submit">Borrar</button>
```

Ilustración 6.36 Método delete

- Tratamiento de imágenes

  Para subir imágenes, lo primero que se debe hacer es definir un atributo imagen en el modelo. Le debemos pasar el búfer, que es donde se alojará la imagen, y el *content type* (tipo de contenido), que será toda la cadena de caracteres que conformará la imagen.

  image: { data: Buffer, contentType: String }

  Una vez definido esto, pasamos al controlador. Mediante el paquete multer, definimos una carpeta de destino para esas imágenes.

  var multer = require('multer');
  var upload = multer({ dest: './uploads' });

  Según esta declaración, todas las imágenes se guardarán en la carpeta "*uploads"*. Creamos la variable *upload* , que será la que utilizaremos para subir las imágenes. Después, en el método *post* , indicamos lo siguiente.

```
router.post('/',isLoggedIn,upload.single('imageupload'),function(req,res,next){

    console.log('El REQ.FILE:', req.file);

    var data = fs.readFileSync(req.file.path);

    var monumento = new monumentos({

      Nombre:     req.body.Nombre,
      Latitud:      req.body.Latitud,
      Longitud:      req.body.Longitud,
      Descripcion:       req.body.Descripcion,


  });

  monumento.image.data =  fs.readFileSync(req.file.path);//búfer
  monumento.image.contentType = req.file.mimetype; //contentType
```

Ilustración 6.37 Método de subida de imágenes

Utilizando la variable *upload* definida anteriormente, permitimos la subida de imágenes de una en una. Mediante las dos últimas líneas, obtenemos la ruta que lo almacena, es decir, la imagen que subimos desde nuestro ordenador, y con el *content-type* identificamos el tipo de archivo que se guarda. *Multer* se encarga de almacenar esa imagen como cadena de caracteres en base 64.

6.4 Pruebas

A lo largo de la implementación se han ido realizando numerosas pruebas. A continuación se reflejan las más importantes y las más repetidas.

Lado cliente

PRUEBA	ACCIÓN REALIZADA	ACCIÓN ESPERADA	RESULTADO
01	Quitar todo tipo de conexión a internet	Visualizar los botones de inicio, y no obtener nada al pulsar cualquiera de esos botones	Correcto
02	Pulsar los botones de listados con conexión a internet	Visualizar los elementos registrados en la base de datos en cada uno de los listados	Correcto
03	Pulsar un elemento de cualquiera de los listados	Visualizar información sobre el elemento pulsado	Correcto
04	Iniciar la aplicación y navegar por ella	Recibir una notificación indicando que hay un monumento o museo cerca	Correcto
05	Pulsar notificación	Redirección a la página de información del monumento o museo localizado	Correcto
06	Eliminar un monumento de la base de datos	Al pulsar de nuevo en el listado de monumentos, el monumento eliminado no aparece	Correcto

Tabla 6.1 Pruebas lado cliente

Todas estas pruebas se han realizado con un teléfono móvil de la marca Xiaomi, modelo MI3w.

Lado servidor

PRUEBA	ACCIÓN REALIZADA	ACCIÓN ESPERADA	RESULTADO
01	Introducción de datos de login incorrecto	La página redirige de nuevo a la página de login	Correcto
02	Introducción de datos de login correcto	Redirección a la página principal de la web de administración	Correcto
03	Pinchar botón consulta de monumentos	Visualizar listado de monumentos registrados	Correcto
04	Pinchar botón borrar del listado de monumentos	Monumento borrado que desaparece del listado	Correcto
05	Pinchar botón de crear nuevo monumento	Redirección a la página de creación de monumentos	Correcto
06	Rellenar los campos del formulario de creación de monumentos erróneamente	La aplicación no deja insertar los datos	Correcto
07	Rellenar los campos del formulario de creación de monumentos correctamente	La aplicación inserta un nuevo monumento en la base de datos, y este aparece en el listado	Correcto
08	Pulsar botón de cierre de sesión	Se cierra sesión y la página redirige a la pantalla de login	Correcto

Tabla 6.2 Pruebas lado servidor

7. MANUALES

7.1 Manual de instalación

Diferenciaremos dos manuales diferentes de instalación. Por una parte el lado cliente (aplicación *Android*) y por otro lado la parte servidor (servidor e interfaz web).

Lado cliente (aplicación *Android*)

En el CD-ROM, en la carpeta *software*, se encuentra el archivo *mySGuide.apk*. Para instalarle en el dispositivo móvil, hay que seguir una serie de pasos.

1. Introducir el archivo en el dispositivo móvil.
2. Activar orígenes desconocidos. Ya que la aplicación no se encuentra en la tienda virtual de Android, se debe activar esta opción que permite la instalación de aplicaciones sea cual sea su origen. Dependiendo del modelo, esta opción se encontrara en diferentes secciones. Generalmente, siguiendo la ruta Ajustes> Seguridad, se encontrará esta opción.
3. Instalar el archivo .apk . Simplemente accediendo al lugar en el que se introdujo la aplicación, con pulsar encima de ella bastaría. Dependiendo del modelo de dispositivo y de la versión de Android que contenga, nos mostrará dos opciones para instalarlo. En caso de que el dispositivo nos muestre dos opciones, pulsar la opción que dice “Instalar mediante instalador de paquetes de Android”.
4. Tras estos pasos, nos saldrá la pantalla de instalación. En ella aceptamos los permisos y pulsamos el botón instalar.
5. Una vez instalada, ya podremos iniciar la aplicación.

Lado servidor

En el CD-ROM, en la carpeta *software*, se encuentra el archivo comprimido *ApiRest*. Para instalarle, hay que seguir una serie de pasos. Se considera que el usuario que lo instale tiene posee el gestor de paquetes NPM, explicado en la sección “Implementación y pruebas” de esta documentación.

1. Descomprimir el archivo.
2. Acceder a la carpeta mediante consola de comandos de Windows.
3. Instalar (si el usuario no dispone de ello) el paquete de base de datos *MongoDB*, explicado en la sección “Implementación” de esta documentación.
4. Iniciar la base de datos. Pulsando en el archivo *script.bat* contenido en la carpeta descomprimida, se inicia la base de datos.
5. En la consola de comandos, una vez que el usuario se encuentre dentro del directorio descomprimido, pulsar el comando *Grunt*.
6. Tras esto, dispondremos de nuestro servidor en la dirección http://localhost:3000

7.2 Manual de usuario

Lado cliente (aplicación *Android*)

Nada más iniciar la aplicación, nos encontramos con la pantalla principal de la aplicación.

Ilustración 7.1 Pantalla principal aplicación

En ella se nos ofrecen cuatro opciones:

- **Monumentos**: ofrece un listado de los monumentos del sistema. Pulsando sobre el botón obtenemos el listado de monumentos, y si a su vez pinchamos sobre el monumento, obtendremos información sobre el mismo.

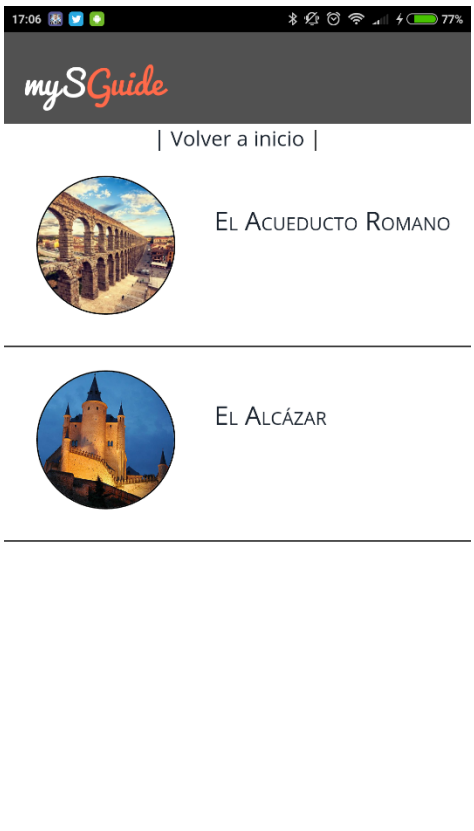

Ilustración 7.2 Sección monumentos

- **Leyendas**: al igual que monumentos, ofrece un listado con las leyendas populares de la ciudad de Segovia. Pulsando sobre cualquiera de ellas, obtendremos la explicación de cada una de ellas.

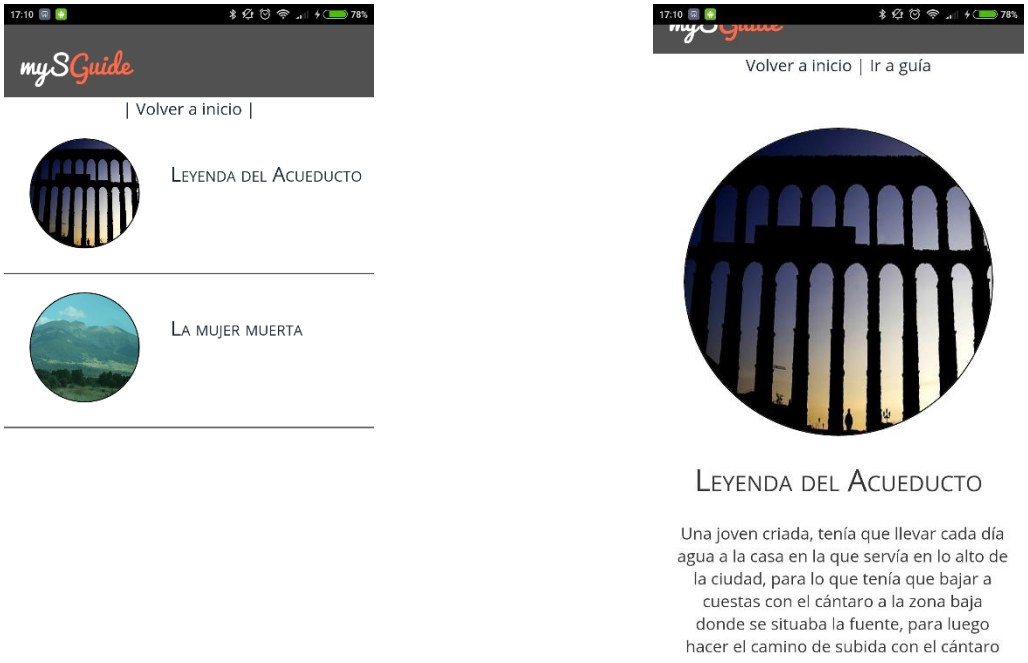

Ilustración 7.3 Sección leyendas

- **Museos**: como en los casos anteriores, la aplicación pondrá a disposición del usuario un listado de museos. Al pinchar en cualquiera de ellos, se ofrecerá información sobre la exposición actual del museo.

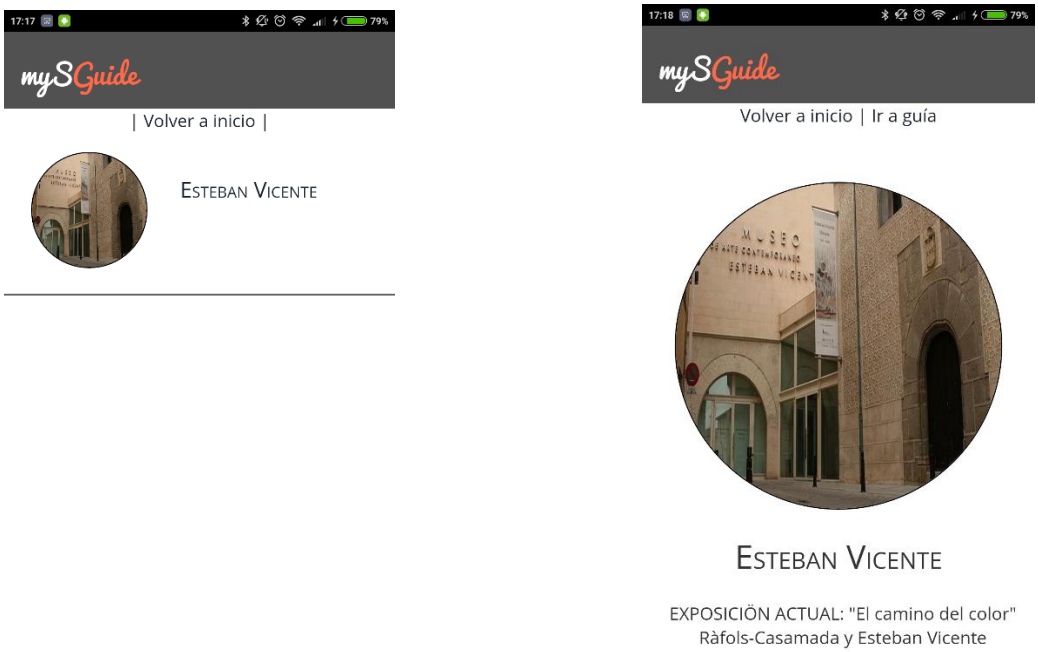

Ilustración 7.4 Sección museos

- **Festivales**: última sección de la aplicación, ofrece información sobre los festivales que se celebran en la ciudad de Segovia durante la estancia del usuario. Pinchando sobre cualquiera de ellos, accederemos a la programación del festival.

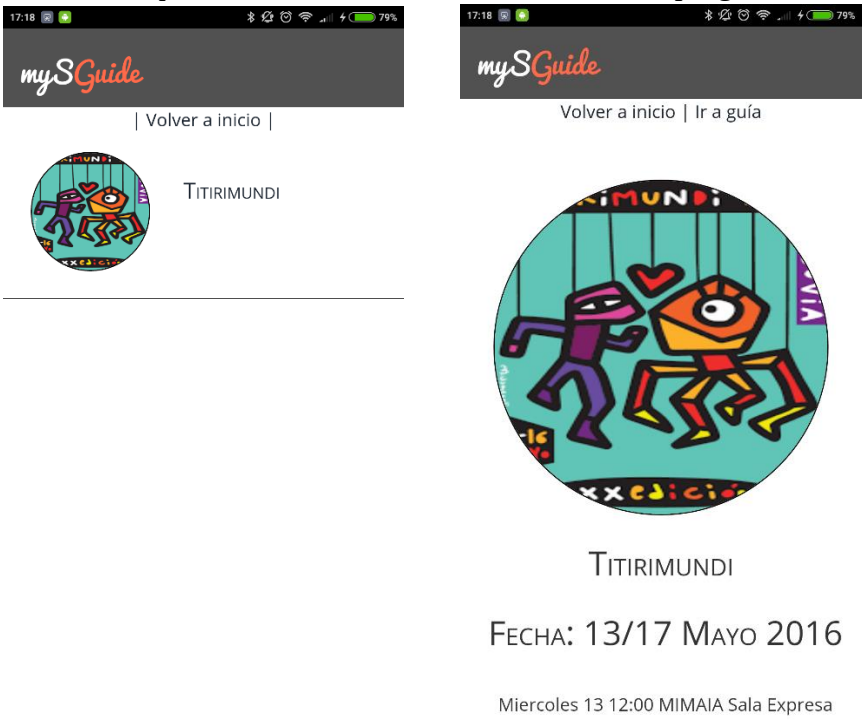

Ilustración 7.5 Sección festivales

Lado Servidor

Nada más entrar a la interfaz web del servidor, nos encontramos la siguiente pantalla.

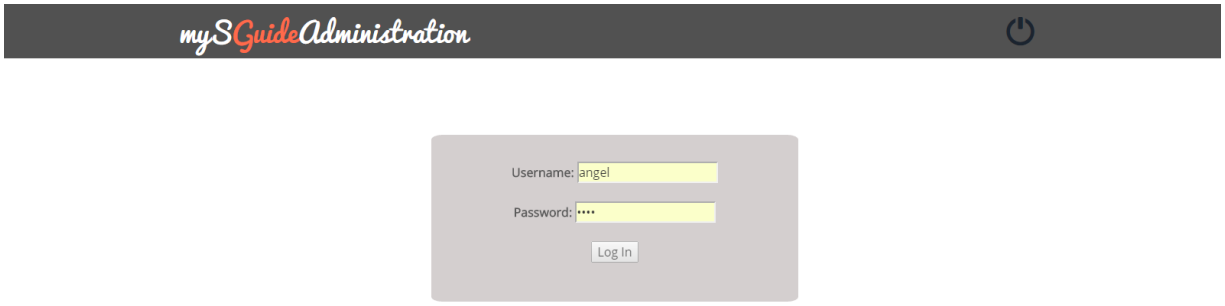

Ilustración 7.6 Página de inicio interfaz web

Una vez que el usuario administrador introduzca correctamente sus datos de acceso, la web redirigirá a la página principal de administración.

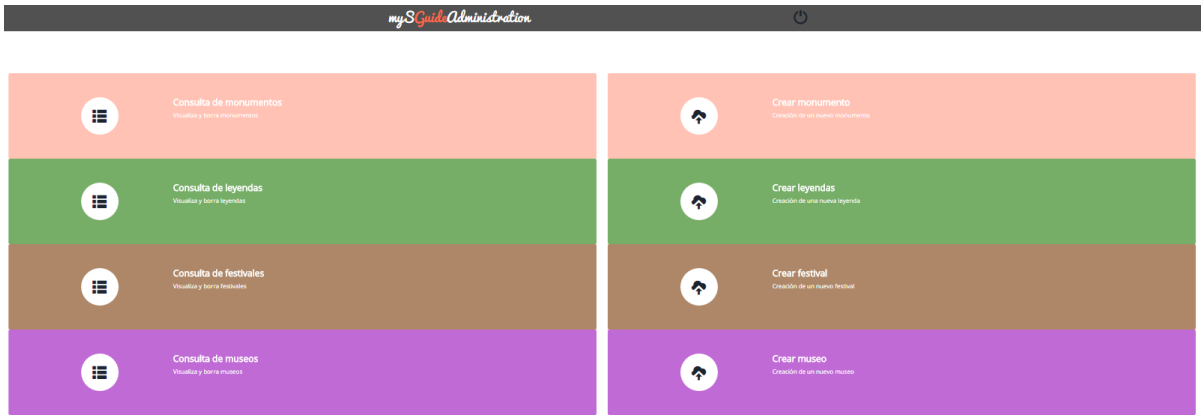

Ilustración 7.7 Página principal de interfaz web

Desde esta interfaz el usuario administrador podrá gestionar toda la información con la que trabaja la aplicación. Se diferencian dos filas de botones principales. A la izquierda, el administrador podrá consultar el listado con los monumentos, museos, festivales o leyendas registrados en la base de datos. Este listado a su vez, permitirá borrar un elemento registrado en la misma. Contará con una barra de búsqueda, que en caso de listados grandes, ayudará al usuario a buscar la información más rápido.

Ilustración 7.8 Listado de monumentos

Como se puede apreciar en la ilustración, en la parte superior aparece la barra de búsqueda, y a la derecha los botones de borrado. Si el usuario introduce un nombre en esa barra de búsqueda, quedará así

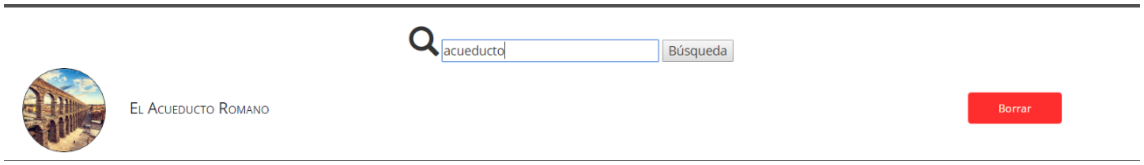

Ilustración 7.9 Ejemplo de búsqueda

Por otro lado, la siguiente fila de botones descrita en la ilustración 7.7 corresponde a los formularios de creación. Mediante estos, el usuario administrador puede añadir nuevas entradas en cada una de las secciones.

Ilustración 7.10 Formulario de creación de nuevo monumento

Por último, si el usuario ha terminado de hacer los cambios pertinentes, puede cerrar sesión pulsando el botón con forma de encendido, que aparece en la esquina superior derecha.

Ilustración 7.11 Botón de cierre de sesión

8. MEJORAS Y CONCLUSIONES

8.1 Mejoras

Se proponen como mejoras los siguientes puntos:

- Inclusión de un módulo llamado eventos, con información actualizada sobre los eventos puntuales realizados en la ciudad de Segovia, como por ejemplo la media maratón, la legua universitaria o la noche de luna llena.
- Inclusión de un mapa de realidad aumentada, donde el usuario pueda ser su avance, y según va avanzando ver la diferente información almacenada sobre monumentos pulsando sobre el la localización del mismo.
- También sería una buena idea la inclusión de vídeos descriptivos de cada uno de los monumentos, además de crear una guía auditiva para ciegos.
- Hacer interactuar al usuario en forma de comentarios. Los usuarios podrían disponer de un hueco en la aplicación para dar su opinión sobre los monumentos o museos, entre otros.
- Además de centrarse en la oferta cultural, mySGuide podría incluir información gastronómica, ya sea creando un módulo restaurantes o incluyendo rutas gastronómicas por la ciudad.
- La inclusión de redes sociales sería un punto que ayudaría a los usuarios a interactuar, dando a conocer la aplicación.
- Una de las mejoras más necesarias sería la expansión de la aplicación a otros sistemas operativos móviles, como pueden ser *iOS* o *Windows Phone*.

8.2 Conclusiones personales

Desde que comencé a cursar la asignatura de Plataformas Software Móviles me llamó mucho la atención enfocar mi vida a la tecnología móvil. Fue con las prácticas en empresa y con el posterior trabajo cuando me di cuenta de que usando tecnología híbrida se podía sacar mucho con muy poco esfuerzo.

A pesar de tener que rehacer el proyecto en muy poco tiempo, yo estoy contento con la aplicación explicada en este documento. Me parece una aplicación útil, que no ocupa apenas espacio ni consume recursos, y que puede satisfacer a muchos curiosos.

Debo dar las gracias de esto a la empresa en la que estuve en Madrid, una startup con 3 años de vida, que me enseñó a realizar el trabajo de cuatro personas de un mes en apenas una semana, con bastantes buenos resultados. Sin eso, esta aplicación habría sido imposible realizar todo lo explicado en este documento en tan poco tiempo.

También mención especial al profesor Francisco Cabrera, que me ha dado esta segunda oportunidad y que me ha comentado todo lo que se podía mejorar. Sin ese empujón, nada de esto habría sido posible.

Por último, gracias sobre todo a mis amigos por el apoyo recibido, y a mi familia, que ha aguantado mis malas caras y me han animado a seguir cuando yo no veía salida posible.

8.3 Conclusiones profesionales

Este trabajo me ha ayudado sobre todo a formarme más como profesional y a poder profundizar más en conocimientos que no se imparten en la carrera.

Partiendo de la base de la carrera con desarrollo web (HTML, CSS y JavaScript) y de los conocimientos de aplicaciones móviles (Android), he podido juntar ambas vías de desarrollo y desarrollar una aplicación en un mercado que está creciendo exponencialmente. Utilizando tecnologías como Apache Cordova o NodeJS siento que estoy más preparado para lo que me encuentre el día de mañana. Ha sido prácticamente empezar de cero con cada nuevo lenguaje, y me siento muy orgulloso de haber podido sacar esto adelante.

También, modelar todo el sistema desde el análisis a la implementación ha hecho que tenga una visión mucho más amplia y detallada del sistema, ayudándome a construir una buena base en todo el ciclo de ingeniería de requisitos y a saber enfrentrarme a problemas similares en el futuro.

Y lo que más me ha enseñado este proyecto es que si trabajas duro puedes conseguir cualquier cosa que te propongas.

9. REFERENCIAS

9.1 Bibliografía

- Apuntes asignatura Proceso de Desarrollo de Software, curso 2015/2016 impartida por el profesor Francisco Cabrera González.
- Apuntes asignatura Modelado Software de Sistemas de Información, curso 2015/2016, impartida por el profesor Miguel Ángel Martínez Prieto.

9.2 Referencias web

- https://cordova.apache.org/ [Última consulta 06/06/2016]
- https://developers.google.com/maps/get-started/?hl=es [Última consulta 01/05/2016]
- http://www.w3schools.com/html/html5_geolocation.asp [Última consulta 03/05/2016]
- http://stackoverflow.com/questions/16222330/geolocation-moving-only-google-maps-marker-without-reload-the-map [Útima consulta 01/05/2016]
- http://stackoverflow.com/questions/16192186/google-maps-api-watchposition [Útima consulta 01/05/2016]
- http://stackoverflow.com/questions/21645329/googlemaps-api-follow-and-center-my-watchpostion-without-reloading-the-map [Útima consulta 01/05/2016]
- http://www.coordenadas-gps.com/ [Útima consulta 04/06/2016]
- https://github.com/phonegap/phonegap-plugin-push/blob/master/docs/API.md[Útima consulta 23/05/2016]
- https://github.com/katzer/cordova-plugin-local-notifications [Útima consulta 23/05/2016]
- http://www.dbsnippets.com/2013/03/12/android-permisos-sobre-la-localizacion/ [Útima consulta 27/04/2016]
- https://github.com/apache/cordova-plugin-dialogs/blob/master/doc/es/index.md [Última consulta 15/05/2016]
- http://casamonedasegovia.es/historia/casa-de-moneda/ [Última consulta 26/05/2016]
- https://es.wikipedia.org/ [Última consulta 26/05/2016]
- http://www.turismoespanagps.com/puntosinteres.php?id=816 [Última consulta 26/05/2016]
- http://triplenlace.com/2012/09/16/la-casa-de-la-quimica-de-segovia-donde-enseno-proust/ [Última consulta 26/05/2016]

- https://github.com/apache/cordova-plugin-device [Última consulta 12/07/2016]
- https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-whitelist/ [Última consulta 12/07/2016]
- http://qode.pro/blog/que-son-las-notificaciones-push/ [Última consulta 12/07/2016]
- http://qode.pro/blog/que-son-las-notificaciones-push/ [Última consulta 13/07/2016]
- http://www.seas.es/blog/informatica/tipos-de-relaciones-en-diagramas-de-casos-de-uso-uml/ [Última consulta 13/07/2016]
- https://developers.google.com/maps/documentation/javascript/geometry?hl=es [Última consulta 18/07/2016]
- https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador [Última consulta 18/07/2016]

ANEXO I. API de Google Maps

- ***Visión General***
  El API de Google Maps es un conjunto de librerías de programación de uso gratuito proporcionadas por la empresa Google Inc. Esta API proporciona todo lo necesario para trabajar con posicionamiento y con los mapas de Google. Ofrece multitud de librerías y ejemplos, pudiendo elegir cualquiera de ellos como base para cualquier proyecto. Dado que una aplicación *Cordova* está basada en un desarrollo web, utilizaremos la API de Google Maps para el lenguaje de programación *Javascript*.

- ***Uso***
  El Api de Google Maps se carga a través de una URL de *arranque* con la forma `https://maps.googleapis.com/maps/api/js`. Esta solicitud de arranque carga todos los archivos y elementos JavaScript principales que se usarán en el API. Algunas funciones de esta *Maps API* también se encuentran disponibles en *bibliotecas* independientes, que no se cargan a menos que se solicite específicamente. Este API proporciona múltiples librerías:
  - ***Rawing***: proporciona al usuario una interfaz para dibujar diferentes figuras geométricas, tal como polígonos, rectángulos, etc..
  - ***Geometry***: incluye funciones para calcular valores como la distancia y el área.
  - ***Places***: permite que la aplicación busque lugares como establecimientos o restaurantes.
  - ***Visualization***: proporciona representaciones visuales de datos.

  Para nuestro proyecto, utilizaremos la librería *Geometry*, ya que es la que nos interesa para calcular la distancia entre el usuario y los monumentos o museos. Mediante el siguiente código, importaremos esta librería a nuestro proyecto.

```
        <script                                    type="text/javascript"
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libr
aries=geometry">
</script>
```

  Donde YOUR_API_KEY será el código proporcionado por Google para utilizar nuestra aplicación. La generación de este código es automática. Basta con entrar en el link:

  https://console.developers.google.com/flows/enableapi?apiid=maps_backend,geocoding_backend,directions_backend,distance_matrix_backend,elevation_backend,places_backend&keyType=CLIENT_SIDE&reusekey=true&hl=es

y seguir los pasos que se indican en la interfaz.

- ***Funciones utilizadas en mySGuide***

  Como se ha indicado anteriormente, la librería utilizada para el proyecto es *Geometry*. Esta librería proporciona funciones de utilidades para el cálculo de datos geométricos de la superficie terrestre. En nuestro caso concreto, una sección de esta librería llamada *spherical*. Esta librería contiene utilidades que permiten calcular ángulos, distancias y áreas a partir de longitudes y latitudes. Utilizaremos esta sección para calcular la distancia entre el monumento o museo y el usuario a través de las coordenadas de los mismos.

  Esta sección proporciona una función en concreto: *computeDistanceBetween*. La distancia entre dos puntos es la extensión del recorrido más corto entre ellos. Con esta función, calcularemos la distancia entre el usuario y el objetivo introduciéndoles dos objetos, posición del usuario y posición del objetivo, y nos devolverá la distancia entre ambos en metros.