Universidad Zaragoza
1542

Trabajo Fin de Grado

Aplicación móvil para la consulta y geolocalización de los espacios de los edificios de la Universidad de Zaragoza a partir de los datos de la Unidad Técnica de Construcciones y Energía

Autor

Jorge Garuz Sánchez

Director

Rubén Béjar Hernández

Escuela de Ingeniería y Arquitectura
Septiembre 2015

Me gustaría dedicarles este trabajo a mis padres y a mi hermano por su apoyo, pero en especial a mi novia Jennifer que siempre ha estado dispuesta a ayudarme. No puedo olvidar a Rubén, sin él, este trabajo no hubiera sido posible.

Aplicación móvil para la consulta y geolocalización de los espacios de los edificios de la Universidad de Zaragoza a partir de los datos de la Unidad Técnica de Construcciones y Energía.

Resumen

Este trabajo consiste en el desarrollo de una aplicación web tanto para dispositivos móviles como sistemas de sobremesa que permite la consulta y geolocalización de los espacios de edificios de la Universidad de Zaragoza, suministrados por la Unidad Técnica de Construcciones y Energía (en adelante UTC). Está desarrollada con tecnologías Web que facilitan que la plataforma sea multiplataforma, sin importar el tamaño de pantalla del usuario.

La aplicación está enfocada tanto para los trabajadores de la UTC, que le ayudará a revisar los espacios de los edificios como a cualquier persona que quiera saber localizaciones de edificios, plantas o estancias.

Al ser una aplicación pedida por la UTC, a lo largo del desarrollo del trabajo, ha habido diversas reuniones con el cliente, añadiendo verosimilitud al trabajo. Se partió de un PowerPoint en el que mostraban sus ideas iniciales. Semanas después les mostramos los requisitos y un mapa de navegación. Después de gustarle nuestra propuesta, ha habido reuniones periódicas para ver el progreso del proyecto así como insistir en la obtención y formato de los datos.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Jorge Garuz Sánchez,

con nº de DNI 25200881B en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Aplicación móvil para la consulta y geolocalización de los espacios de los edificios de la Universidad de Zaragoza a partir de los datos de la Unidad Técnica de Construcciones y Energía.,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 23 de Septiembre 2015

Fdo: Jorge Garuz Sánchez

Índice general

1. Introducción

Esta sección consiste en introducir el contexto del proyecto, así como mi motivación para realizarlo.

1.1 Contexto del proyecto

1.1.1 Colaboración con IAAA

He realizado este proyecto siendo colaborador del Grupo de Sistemas de Información Avanzados (IAAA), un grupo de I+D adscrito al Instituto de Investigación en Ingeniería de Aragón de la Universidad de Zaragoza. Más concretamente, en su área de Sistemas de Información Geográfica (SIG).

1.1.2 Colaboración con UTC

También he colaborado con la Unidad Técnica de Construcciones y Energía (UTC). Contactaron con Rubén para ofrecer el proyecto y Rubén me lo comunicó a mí. Después de aceptar el proyecto, empezamos a reunirnos y a trabajar con una becaria que tenían contratada, la cual se encargaría de proporcionarnos los datos en formato útil para la aplicación. De las primeras reuniones también hubo un intercambio de sugerencias por ambas partes para incluir en la aplicación.

Al final del proyecto, desde mediado de Agosto, hemos dejado de recibir datos y contestación a nuestras dudas.

1.1.3 Contexto tecnológico

Para el desarrollo del proyecto, dado el objetivo de multiplataformidad, hay diversas tecnologías que debían estar presentes, como son las tecnologías Web. Dichas tecnologías Web son HTML, CSS y JavaScript. En los últimos años se ha visto un auge enorme de desarrollos basados en JavaScript, introduciendo en el mercado nuevas librerías y frameworks que aprovechan toda la potencia de JavaScript.

Uno de esos frameworks que más éxito están teniendo en nuestros días es AngularJS. AngularJS añade muchas funcionalidades a JavaScript y permite tener un código mejor mantenido. Además tiene el soporte de Google y es open-source por lo que da más confianza de que va a ser una tecnología con mucho recorrido.

Por encima de AngularJS utilizo el framework de Ionic que añade la posibilidad de realizar desarrollos multiplataforma de manera bastante sencilla, además de facilidades para crear aplicaciones móviles nativas para Android e iOS.

Para la parte del servidor he utilizado Java, más concretamente Spring-boot que facilita la creación de servicios REST apoyada por Gradle, para la organización y utilización de librerías y configuraciones del servidor.

Por último en la Base de Datos se ha utilizado PostgreSQL con el plugin POSTGIS, para permitir la utilización de datos geográficos.

1.2 Motivación

La motivación principal para realizar este trabajo ha sido que está basado en tecnologías Web. Con el auge de los dispositivos móviles se ha visto que estas tecnologías van a estar muy presentes en el futuro. Otra razón importante ha sido la utilidad de la aplicación a realizar, siendo una aplicación con muchas posibilidades en cuanto a utilidad para la Universidad y que se ha echado en falta no tener ya ninguna aplicación disponible similar.

Por otra parte, el poder trabajar con un cliente añade realismo al trabajo y me prepara para el trabajo profesional en el que hay que lidiar con todo tipo de clientes.

1.3 Estructura del documento

La memoria está estructurada en secciones que corresponden a la documentación principal del proyecto, como es el problema del proyecto, el diseño de la solución, la implementación y la gestión del proyecto, además de las conclusiones. Al final de la memoria se encuentra la bibliografía y los anexos, que son: Interfaz de los servicios REST, manual de instalación y manual de usuario, así como el índice de figuras y de tablas.

En la sección de problema (pág. 9) contiene la descripción del problema, los requisitos funcionales y no funcionales de la aplicación y los casos de uso.

En la sección de diseño de la solución (pág. 13) se incluye la documentación del diseño arquitectural del sistema en un diagrama de despliegue, que incluye el de componentes y el mapa de navegación.

En la sección de implementación (pág. 17) están los detalles de la implementación, así como las herramientas utilizadas. También se describe la implementación del cliente, servidor y base de datos.

En la sección de gestión del proyecto (pág. 21) se encuentra la documentación relativa a la gestión del proyecto, esto es, gestión de la configuración, planificación mediante diagrama de Gantt, coste total del proyecto y gestión de riesgos.

En las conclusiones (pág. 25) se muestran las conclusiones obtenidas después de haber realizado el proyecto.

Por último están los anexos de la memoria, que contienen detalles adicionales al proyecto, como interfaz de los servicios REST utilizados en el servidor, el manual de instalación para poner en funcionamiento la aplicación y manual de usuario.

2. Problema

En esta sección se presenta la descripción del problema, la documentación de los requisitos funcionales y no funcionales y los casos de uso del sistema.

2.1 Descripción

El trabajo consiste en desarrollar una aplicación Web para consultar y geolocalizar los espacios de los edificios de la Universidad de Zaragoza, incluyendo Huesca y Teruel. Esto implica mostrar los espacios de los edificios (que son datos proporcionados por la UTC) sobre la base de un gestor de mapas, en este caso Google Maps, a través de un framework de mapas llamado Leaflet. Después de mostrar los edificios, el usuario podrá seleccionar el edificio que él quiera y acceder a los planos de ese edificio seleccionando la planta que desee.

Una vez mostrado el plano de la planta, el usuario podrá escoger las diferentes estancias de las que dispone el plano, así como información asociada a dicha estancia. Opcionalmente según el tipo de estancia, existe la posibilidad de mostrar una imagen de dicha estancia.

Además, se ha desarrollado un formulario de búsqueda para poder buscar estancias individuales de manera más rápida. También se ha desarrollado la aplicación para que pueda ser multilenguaje de manera nativa, cambiando el idioma de la aplicación en Ajustes.

Además de la aplicación cliente, hay que realizar tratamiento de datos, transformando los planos en formato SHP a SQL apto para POSTGIS. Una vez insertados los planos, se trabaja en el servidor con Geoserver, un servidor de intercambio de datos geoespaciales. Una vez conectado Geoserver con la Base de Datos los planos se cargan en el servidor y son mostrados al usuario por los protocolos WMS para la visualización de los planos y WFS para insertar información de los edificios geográficamente.

Por último se necesita la parte de servidor REST desarrollado en Spring Boot para acceder a información de los edificios proporcionada por la UTC. Dicha información fue exportada de Access a SQL y es accedida por el servidor a través de un Pool de conexiones.

2.2 Requisitos

En esta sección se muestran todos los requisitos que debe satisfacer el sistema y que han sido aceptados por el cliente. En la tabla 1 están los requisitos funcionales y en la tabla 2 los requisitos no funcionales.

2.2.1 Requisitos funcionales

Número	Requisito
RF1	La aplicación debe mostrar mapa de Aragón para poder seleccionar una ciudad a mostrar entre Zaragoza, Huesca y Teruel.
RF2	La aplicación debe permitir buscar planos aplicando los siguientes filtros: Ciudad, Campus, Edificio, Planta, Estancia e Identificador de estancia.
RF3	La aplicación debe mostrar información al pulsar sobre el centro universitario deseado. Esta información será: Nombre, dirección, CP.
RF4	La aplicación debe mostrar los distintos edificios que componen el campus seleccionado.
RF5	La aplicación debe permitir filtrar los distintos edificios del campus.
RF6	La aplicación podrá visualizar los edificios por plantas.
RF7	La aplicación deberá mostrar el plano de la planta seleccionada de un edificio.
RF8	La aplicación debe permitir filtrar por tipo de uso o nombre las distintas estancias de la planta seleccionada.
RF9	La aplicación debe mostrar información relativa a cada estancia: Nombre, tipo de uso, superficie, id y ocupación.
RF10	La aplicación debe permitir mostrar un menú lateral con las opciones de la aplicación: Favoritos, mapa, búsqueda, búsqueda avanzada y ajustes.
RF11	La aplicación debe permitir añadir distintas estancias como favoritos.
RF12	La visualización del mapa podrá ser vectorial o por satélite.
RF13	La aplicación permitirá mostrar algunos puntos de interés (PoI).

Tabla 1. Requisitos funcionales.

2.2.2 Requisitos no funcionales

RNF1	La aplicación debe ser una aplicación Web y poder ser mostrada en cualquier dispositivo con navegador moderno.
RNF2	La aplicación debe poder mostrar texto en inglés y español.
RNF3	La aplicación tendrá versiones nativas para iOS y Android fácilmente instalables.
RNF4	La aplicación utilizará la Base de Datos y el formato de datos de la UTC.
RNF5	La interfaz de usuario tiene que tener en cuenta el manual de estilo gráfico de la Universidad.

Tabla 2. Requisitos no funcionales.

2.3 Casos de uso

A continuación se muestra el modelado de los requisitos del sistema mediante un diagrama de casos de uso, para mostrar de manera visual la interacción del usuario con la aplicación.

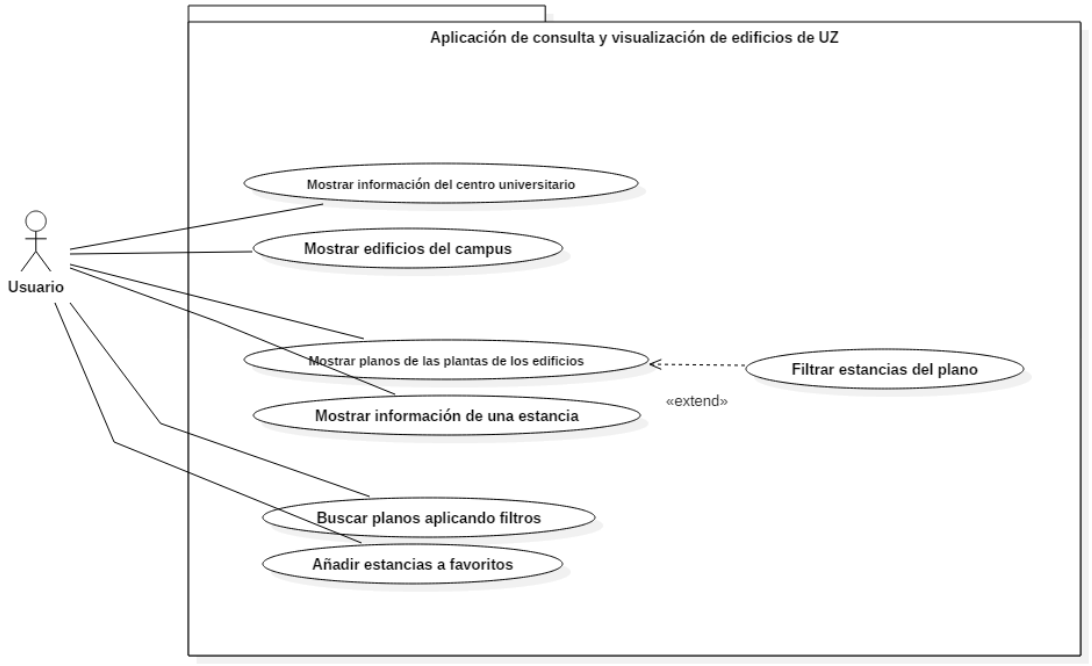

Figura 1. Diagrama de casos de uso.

El diagrama contiene un actor, llamado Usuario. El actor Usuario es el responsable de la interacción con la aplicación y el que podrá realizar las diferentes funcionalidades contenidas en los casos de uso. A continuación se detallan los casos de uso:

- Mostrar información del centro universitario. El caso de uso permite al usuario visualizar la información relacionada con el centro universitario que escoja del mapa. Mostrará la información de Nombre, dirección y CP.

- Mostrar edificios del campus. El caso de uso permite al usuario visualizar los distintos edificios posicionados sobre el mapa.

- Mostrar planos de las plantas de los edificios. El caso de uso permite al usuario visualizar el plano asociado a la planta seleccionada del edificio que quiera. Este caso de uso está relacionado por la relación de extensión con el caso de uso Filtrar estancias del plano, que permite al usuario filtrar las distintas estancias que aparezcan en el plano.

- Mostrar información de una estancia. El caso de uso permite al usuario visualizar información relativa a la estancia que haya seleccionado en el plano o buscado con el formulario. Mostrará la información de Nombre, tipo de uso, superficie y capacidad.

- Mostrar planos aplicando filtros. El caso de uso permite al usuario utilizar un formulario de búsqueda para visualizar el plano correspondiente. Los filtros serán Ciudad, Campus, Edificio, Planta y Estancia. Además de dichos filtros, los

trabajadores de la UTC podrán aplicar el filtro de Identificador de Estancia para buscar un plano añadiendo solamente el Identificador de la Estancia.

- Añadir estancias a favoritos. El caso de uso permite al usuario añadir a favoritos las estancias sobre la que visualiza la aplicación para después poder volver a ellas de manera más rápida.

3. Diseño de la solución

En esta sección se incluye la documentación básica del diseño arquitectural de la aplicación. El diseño se ha ido mejorando a lo largo del transcurso del proyecto.

Debido al tipo de proyecto a realizar, necesitábamos una base de datos que trabajase con tipos de datos geográficos. En este caso la mejor solución era trabajar con PostgreSQL y el plugin POSTGIS.

Para la parte de visualización de mapas, teníamos dos alternativas, Leaflet y Openlayers. En el trabajo se ha usado Leaflet ya que ya lo había utilizado anteriormente además de ser más ligero que Openlayers. Además, las opciones extra que nos proporciona Openlayers no eran necesarias para el trabajo por lo que la decisión fue utilizar Leaflet.

Por otra parte, para el servidor de mapas solamente se planteó Geoserver, ya que era conocido y se integraba fácilmente en el proyecto. En cuanto al servidor de aplicación, existían dos alternativas, usar Spring boot o usar JAX-RS. Con ambos servidores había trabajado, sin embargo escogí utilizar Spring boot por proporcionar mejores recursos y guías y porque actualmente ha crecido su uso.

Por último, en cuanto al cliente se propusieron varias opciones, siendo lo más importante que fuese una aplicación multiplataforma. La primera fue utilizar JavaScript con PhoneGap. Sin embargo esta solución era demasiado arcaica y se desechó rápidamente ya que el resultado de multiplataforma era demasiado costoso de conseguir. Otra alternativa fue utilizar jQuery Mobile + Backbone, cumpliendo correctamente con los requisitos, sin embargo fue la última alternativa, Ionic, la que tuvo más éxito, ya que ofrecía mejores guías al usuario, además de cumplir requisitos correctamente e incluso añadía opciones extra, como la creación de aplicaciones nativas para móvil de manera sencilla. Por último también se buscó información de utilizar Salesforces para realizar la aplicación cliente, pero como era un medio de pago se desechó al instante.

Con todo esto comentado, alternativas y opciones escogidas, la arquitectura que mejor se acomoda a la aplicación es la Arquitectura en 3 niveles, siendo un estilo arquitectural cliente-servidor. En el que el primer nivel está la aplicación Cliente y Leaflet. En el segundo nivel nos encontramos con el servidor Spring y Geoserver. Y en la última capa nos encontramos con la base de datos.

La documentación arquitectural de la aplicación está compuesta por 3 vistas, la vista de componentes, vista de módulos y vista de despliegue, aportándonos cada una de ellos información diferente sobre el sistema.

3.1 Vista de despliegue

A continuación se encuentra el diagrama de despliegue, el cual sirve para describir la configuración del sistema para su ejecución en un ambiente real.

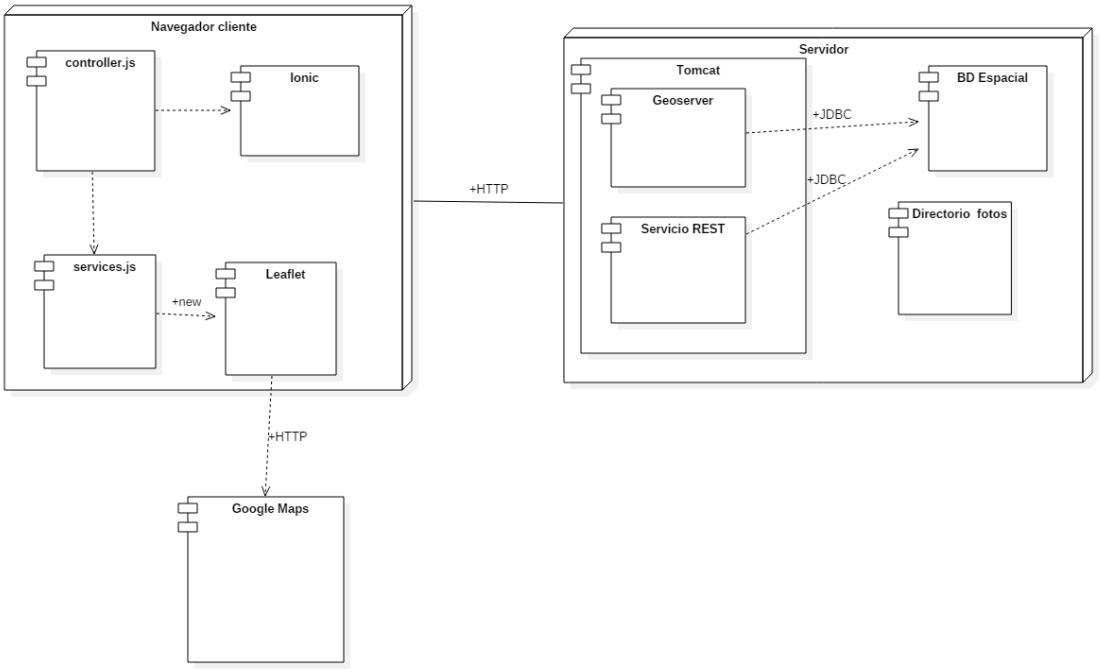

Figura 2. Diagrama de despliegue.

Como se puede apreciar en el diagrama, los componentes están distribuidos en dos nodos, conectados por el protocolo HTTP, además del componente Google Maps que es externo de la aplicación. En el nodo Cliente (accedido por un navegador) encontramos los componentes Ionic y Leaflet, que son frameworks y controller.js y services.js que representan partes de la aplicación cliente y que son los ficheros JavaScript que interaccionan con los framework.

En el nodo Servidor encontramos cuatro componentes, Geoserver y Servicio REST, incluido en el componente Tomcat, que son los que establecen conexión con el nodo cliente, y a su vez se conectan con el nodo BD Espacial fotos por medio de JDBC y Directorio fotos que es accedido por HTTP por la aplicación cliente.

A nivel de componente tenemos:

- **Ionic.** Es el framework encargado de los estilos gráficos y de realizar la aplicación multiplataforma así como crear aplicaciones nativas de manera sencilla.

- **Leaflet.** Es el framework encargado de pintar el mapa con los datos. Se conecta con Google Maps para poner de base el mapa de Google y a partir de ahí, insertar los edificios y los marcadores.

- **Controller.js.** Es la clase que se encargan de la vista del usuario y de su interacción con Ionic, además de recoger los eventos del usuario. Ha sido detallada en el anexo.

- **Services.js.** Es la clase que se encarga de comunicarse con Leaflet y pasarle los datos que serán pintados en el mapa. Todas sus operaciones han sido detalladas en el anexo.

- **Tomcat.** Es el contendor de aplicaciones, donde se ejecuta la aplicación desarrollada, dentro están los componentes Geoserver y servicios REST.

- **Geoserver.** Se encarga de proporcionar los planos de los edificios georeferenciados para ser mostrados en Leaflet así como información de los planos en formato GeoJSON.

- **Servicios REST.** El componente se encarga de proporcionar información a la aplicación realizando consultas a la Base de Datos. Esta información incluye información de los edificios, de las estancias o la información para el formulario de búsqueda. Este componente incorpora dos puertos que son BusquedasRestController y EstanciasRestController y que se pueden ver con más detalle en el anexo.

- **BD Espacial.** En este componente se guardan los planos de los edificios georeferenciados, así como información sobre los planos y más información extra que tiene la UTC.

- **Directorio fotos.** Contiene las fotos que luego son mostradas junto a las estancias.

A modo de mejorar la aplicación podría optarse por separar las Bases de Datos en otro nodo, añadiendo fiabilidad a la aplicación. Esto no se ha hecho para el trabajo debido a la restricción de disponer de una única máquina servidor.

3.2 Mapa de navegación

El mapa de navegación muestra las diferentes pantallas de la aplicación de manera sencilla y que pueda guiar al diseño final.

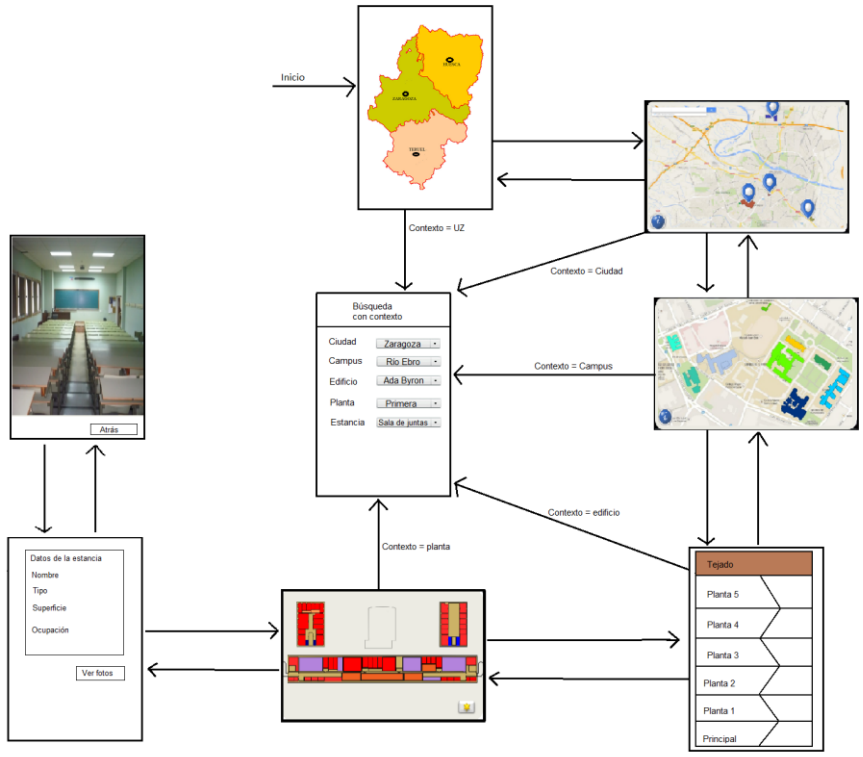

Figura 3. Mapa de navegación.

El mapa de navegación fue creado a partir de las ideas que tenía el cliente sobre la aplicación que quería. Se ha intentado seguir lo máximo posible el mapa en cuanto a estructura y navegación, sin embargo se ha simplificado alguna pantalla como puede ser la de seleccionar planta que está integrada en el propio mapa de Google.

La aplicación comienza con un mapa de Aragón, en el que puedes elegir ciudad para visitar en el plano. También puedes desde el mapa de Aragón, ir al buscador.

En el mapa de la ciudad elegida, se mostrarán los edificios resaltados y con marcadores, en los que pulsando en el marcador mostrarán información del edificio seleccionado, pudiendo ir al buscador también. Haciendo eso, podremos escoger la planta sobre la que queremos ver el plano. Una vez escogida visualizaremos el plano del edificio con las diferentes estancias diferenciadas.

En esta pantalla tenemos la opción de agregar una planta o estancia a favoritos (botón inferior derecha) y también acceder al buscador. Seleccionando una estancia podremos ver la información más relevante de ésta, como es, datos de la estancia, nombre tipo, superficie y ocupación. Además incluye un botón para ver las fotos de dicha estancia si es que estuvieran disponibles. Por último tenemos la pantalla con la foto y botón para volver atrás.

4. Implementación

La aplicación se ha desarrollado creando tres prototipos de manera incremental discutidos con el cliente, con el propósito de mejorar la aplicación.

El primer prototipo fue creado para realizar el esqueleto de la aplicación con las diversas opciones que iba a tener la aplicación y de esta manera aprender cómo funcionaba Ionic y AngularJS además de probar el componente Leaflet y aprender cómo funcionaba y se integraba con la aplicación.

El segundo prototipo sirvió para conectar Leaflet con Geoserver y mostrar los datos en el mapa. En este segundo prototipo surgieron multitud de problemas, debido a errores en los datos recibidos. Una vez corregidos, se podía visualizar los edificios en el mapa de manera sencilla. Durante el intervalo de los problemas con los datos del cliente, se desarrolló el formulario de búsqueda, con un servidor muy sencillo.

En el tercer prototipo se creó el servidor REST para ser conectado por la aplicación cliente y poder obtener información acerca de los edificios proporcionados. Después de realizar la conexión se mezclaron los datos de Geoserver y de los edificios para mostrar marcadores donde los edificios mostrando información como el nombre del edificio, dirección y número de plantas y cuál se visualizaría en el futuro. Además se mejoró la conexión del formulario de búsqueda y ya se rellenaba el formulario con los datos de la Base de Datos.

Se han utilizado las siguientes herramientas y entornos de trabajo:

- Entornos de trabajo: WebStorm, Spring Tool Suite y pgAdmin III.
- Ofimática: Microsoft Office.
- Control de versiones: Github y Git Bash.
- Diagramas UML: StarUML
- Otros: Notepad++

4.1 Cliente

Como se ha comentado previamente, para implementar el cliente se han utilizado tecnologías web, como HTML CSS y JavaScript. Para abstraer el componente multiplataforma se ha utilizado el framework Ionic que por debajo trabaja con AngularJS, un framework de JavaScript desarrollado por Google.

Todas estas tecnologías se encuentran en cualquier navegador moderno de sobremesa o de dispositivos móviles, por lo que no habrá ningún problema en acceder a la aplicación.

También se ha trabajado con Leaflet que es una librería JavaScript para la creación de mapas interactivos. Esta parte se ha llevado la mayor parte del desarrollo del Cliente. Por defecto Leaflet no trabaja con Google Maps, por lo que hay que utilizar un plugin desarrollado por Pavel Shramov (Nº 6 de la Bibliografía). Con el plugin utilizado, ya se podía utilizar los mapas de Google para visualizar los edificios, aunque no incluye algunas de las funcionalidades de Google Maps, como Street View. Aunque esto podría ser interesante para la aplicación, no es un requisito para la aplicación por lo que no ha sido un problema. En las próximas versiones de Leaflet aseguran que se podrá trabajar con Google de manera nativa, sin embargo, a la hora de desarrollar el trabajo no está disponible dicha versión.

Aparte de las librerías y framework utilizados, también se ha utilizado jQuery para facilitar ciertas tareas, como la conexión del cliente con el servicio REST del servidor, o para utilizar el servicio WFS de Geoserver.

4.2 Servidor

El servidor consta de dos componentes, Geoserver y los servicios REST. Geoserver se utilizar para suministrar datos al cliente de los edificios de la universidad y de sus coordenadas para insertar los marcadores. Se utilizan dos servicios para ello, WMS (Web Map Service) para mostrar el mapa y WFS (Web Feature Service) para obtener los datos de los edificios en formato GeoJSON y saber dónde están georeferenciados.

Para proveer los datos, Geoserver se conecta con la Base de Datos con extensión PostGIS, y devuelve los datos aplicándole transformaciones, como cambios en el sistema de coordenadas, aplicación de estilos.

El componente REST, está desarrollado en Java 1.8. Para realizar el servicio REST se ha utilizado Spring Boot, que abstrae mucha de la complejidad y posibilita crear servicios REST con poco código fuente. Spring Boot está integrado con Gradle, que sirve como repositorio de librerías y de configuración del servidor. Con un solo comando, Gradle te descarga todas las librerías necesarias para el proyecto, además de crearte un fichero WAR entre otras cosas.

El servidor contiene dos partes importantes, el código Cliente conservando la arquitectura de los ficheros WAR y los ficheros propiamente del servidor. En los ficheros del servidor, se pueden identificar 4 carpetas. La primera es la que inicia el servicio, que contiene el método main de la aplicación. La carpeta de base de datos, que gestiona la conexión con la base de datos y devuelve un objeto Connection para poder ser usado en los servicios REST. Después está la carpeta de dominio, que son clases DAO de tablas de la base de datos. Esto es útil para después pasar los atributos de la clase a JSON usando la librería GSON de Google. Al ser objetos los transforma a JSON y los envía al cliente de manera muy sencilla. Por último tenemos la carpeta de los servicios REST, en el que están los métodos que reciben las peticiones REST y devuelven los datos.

Al haber utilizado Spring Boot, el mapeo de los servicios REST con los métodos de la carpeta correspondiente se realiza mediante etiquetas. Se tiene una etiqueta principal que es:

```
@RequestMapping("/busquedas")
```

En el que luego los métodos añaden la etiqueta:

```
@RequestMapping(
                    value = "/codigoespacios",
                    method = RequestMethod.GET,
                    produces = "application/json")
```

Por lo que para utilizar dicho servicio REST sólo hay que componer la URL, http:dirip:8080/búsquedas/codigoespacios.

El resto de operaciones en el anexo I.

4.3 Bases de datos

Para la Base de Datos, se ha utilizado PostgreSQL, ya que era necesario trabajar con geometrías y esto es fácil hacerlo con el plugin PostGIS de PostgreSQL. Tanto PostgreSQL como PostGIS son open source, por lo que eran la mejor opción a la hora de desarrollar el proyecto.

En la Base de Datos se han almacenado dos tipos de datos, por un lado, se han almacenado los ficheros SHP que corresponden a los planos de cada edificio de la Universidad de Zaragoza que tiene la UTC. Para insertar los planos en la Base de Datos, primero hay que realizar una transformación a los ficheros SHP. Para ello he utilizado GeoKettle, otra herramienta open source para transformar ficheros SHP a este caso, ficheros SQL. Por cada plano se ha creado una tabla en la Base de Datos con la siguiente nomenclatura *codigocampus_numeroEdificio_planta,* como por ejemplo, el Edificio Cervantes se denomina *csf_1106_00* para la primera planta. Esto se ha realizado así ya que cuando Geoserver accede a la Base de Datos, crea una capa por cada tabla que exista en la Base de Datos, por ello se ha decidido crear una tabla por planta de cada edificio.

Los otros tipos de datos que se almacenan en la Base de Datos son datos que contienen información de los edificios, como nombre, calle, superficie, altura. Estos datos nos los proporcionaron en formato Access y tuve que exportarlos a PostgreSQL para poder disponer de todos los datos en una misma base de datos.

También tenemos fotos de las estancias que nos permiten mostrar al usuario, que están guardadas en el servidor, pero que en una tabla de la Base de Datos (exportada a través de Access) están las direcciones guardadas, así como la estancia a la que se refiere. Estas fotos están reducidas en tamaño para facilitar mostrarlas al usuario y sobre todo si utilizan la aplicación desde un dispositivo móvil.

Durante el desarrollo de la aplicación me di cuenta de que había algunos problemas con los datos de las estancias que nos habían proporcionado. Cada edificio utiliza una notación distinta para sus estancias, por lo que hay muchos edificios que no tienen numerados los despachos ni las aulas, por lo que es más difícil utilizar el buscado o usar el plano para buscar una estancia determinada. Estos problemas fueron remitidos a la UTC para que nos dieran una solución, que a fecha de la memoria no fue resuelta.

4.4 Pruebas

Como pruebas, se han realizado pruebas en múltiples dispositivos, navegadores y sistemas operativos. Las principales pruebas se han llevado a cabo en un portátil con Windows 8 como sistema operativo y de navegador tanto Chrome como Firefox. Como segundo dispositivo más usando en pruebas ha sido un ordenador de sobremesa con Windows 7 y Chrome como navegador.

También han sido importantes las pruebas en un dispositivo con iOS 8, un iPad, con el navegador Safari y en un móvil Android 5.0 con navegador Chrome y nativo de Android. Todos estos dispositivos de distinto tamaño de pantalla para ver cómo se adapta la aplicación a distintos tamaños.

Por último, también se ha utilizado aunque de manera menos importante un iPhone con iOS 8 y Safari y un Smartphone con Windows Phone, pudiendo probar la aplicación en los tres sistemas operativos móviles dominantes del mercado. Y por último se han realizado alguna prueba en Linux en un portátil con Debian.

En todas las pruebas la visualización y el comportamiento de la aplicación era el adecuado, sin embargo se encontraron problemas con el navegador Firefox, tanto en versión móvil como de sobremesa, debido al plugin de Leaflet para poder visualizar el mapa de Google Maps.
Por ahora el problema no ha sido subsanado y como medio de contención, se avisa con un mensaje de alerta a cualquier usuario con navegador Firefox diciendo que su navegador no está soportado por la aplicación.

También se detectó que la búsqueda avanzada de la aplicación es bastante lenta, por lo que en un futuro se ha pensado añadir una caché para reducir el tiempo de espera.

4.5 Máquina virtual

También hay que destacar en la implementación, la máquina virtual donde reside la aplicación, que se corresponde con el servidor en el diagrama de despliegue. Dicha máquina reside en el Centro de Cálculo, en el Campus San Francisco y ha sido cedida a la UTC debido a las negociaciones que ha habido entre ambos departamentos.

En la máquina virtual hay instaladas una serie de aplicaciones para facilitar el uso de la máquina o el despliegue de aplicaciones. Por un lado está instalado webmin, para poder realizar labores de administración de manera sencilla y desde el navegador.

También está instalado Tomcat, en la versión 7, para poder desplegar aplicaciones (en mi caso, ficheros WAR) y que se pueda ejecutar la aplicación en el navegador. Una de las aplicaciones desplegadas y que son esenciales para el proyecto es Geoserver, para mostrar los planos que residen en la Base de Datos, el cual también necesita usuario y contraseña debido a que contiene información sobre los planos.

Por último, se encuentra la Base de Datos PostgreSQL y su plugin PostGIS, donde están almacenados los planos de los edificios así como información adicional sobre los edificios.

5. Gestión del proyecto

En esta sección se encuentra la documentación de la gestión del proyecto. Ésta se compone de la documentación del modelo del proceso elegido, la gestión de la configuración, la planificación y el coste total del proyecto.

El modelo del proceso escogido ha sido iterativo e incremental, en el cual se han creado tres prototipos que han sido mostrados al cliente. Con la valoración del cliente se han añadido mejoras en la aplicación y creada nueva funcionalidad.

5.1 Gestión de la configuración

Para la gestión de la configuración he utilizado diversas herramientas para ayudar todo lo posible a la realización del proyecto y añadir consistencia a los documentos y ficheros creados.

Para almacenar toda la información relativa a documentos, como diagramas y actas de reuniones se ha utilizado Dropbox, debido a su facilidad de uso, disponibilidad en la nube y poder recuperar versiones anteriores de archivos.

Sin embargo Dropbox no es muy útil para el código fuente desarrollado, por ello se ha utilizado Git, en concreto Github, al principio con una cuenta gratuita (ya que el proyecto es open source y no se necesita más), sin embargo al ser estudiante de la Universidad de Zaragoza pedí una cuenta gratuita de desarrollador que me fue concedida, aunque no hubo muchos cambios en cuanto al desarrollo.

Al trabajar en solitario, solo se ha desarrollado en una rama todo el proyecto, y no ha habido ningún problema de incompatibilidad en los datos. Para subir el código a Github he utilizado la herramienta que ellos mismos proporcionan, Git Bash, que además de trabajar fácilmente con repositorios Git, añade la capacidad de utilizar comandos Linux en Windows. Otra de las cosas positivas de trabajar con Github, es la sincronización de los datos con los entornos de trabajo utilizados.

En cuanto a versiones de las tecnologías empleadas se ha utilizado:

Nombre de la tecnología	Versión
Cliente	
Leaflet	0.7.3*
Ionic	1.0.1
Angular	1.3.13
jQuery	2.1.3
Servidor	
Java	1.8
Gradle	2.3
Spring-Boot	4.1.2
Tomcat	7.0.52
PostGIS	
PostgreSQL	9.3
PostGIS	2.1

Tabla 3. Tecnologías empleadas.

*Primero se utilizó la versión 0.7.3 pero a principios de Septiembre sacaron la versión 0.7.5 que resolvía problemas y que es la que ha sido utilizada para finalizar el proyecto.

5.2 Coste total

La metodología de control del tiempo invertido en el proyecto ha sido utilizando unas plantillas Excel que utilizan los trabajadores del IAAA, en el que se mide la productividad por horas y se incluye un código numérico para cierta tarea. Para realizar el TFG sólo existía un código único para mi trabajo, en el que iban incluidas todas las horas que le he dedicado. Como esto no es preciso (no se tiene en cuenta las diferentes horas dedicadas a diferentes partes del proyecto) he llevado paralelamente otra plantilla en el que he ido detallando las horas dedicadas a cada parte del trabajo.

Para mostrar los datos, los he agrupado en 5 grupos: Aplicación cliente, Servidor, Base de datos y Memoria (que incluye toda la documentación creada) y otros que incluye reuniones y trabajo no incluido en el resto de grupos.

Actualizado 20/09/2015

Tarea	**Horas**
Cliente	154
Servidor	54
Base de Datos	11
Memoria	63
Otros	23
Total	**305**

Tabla 4. Coste desglosado en horas.

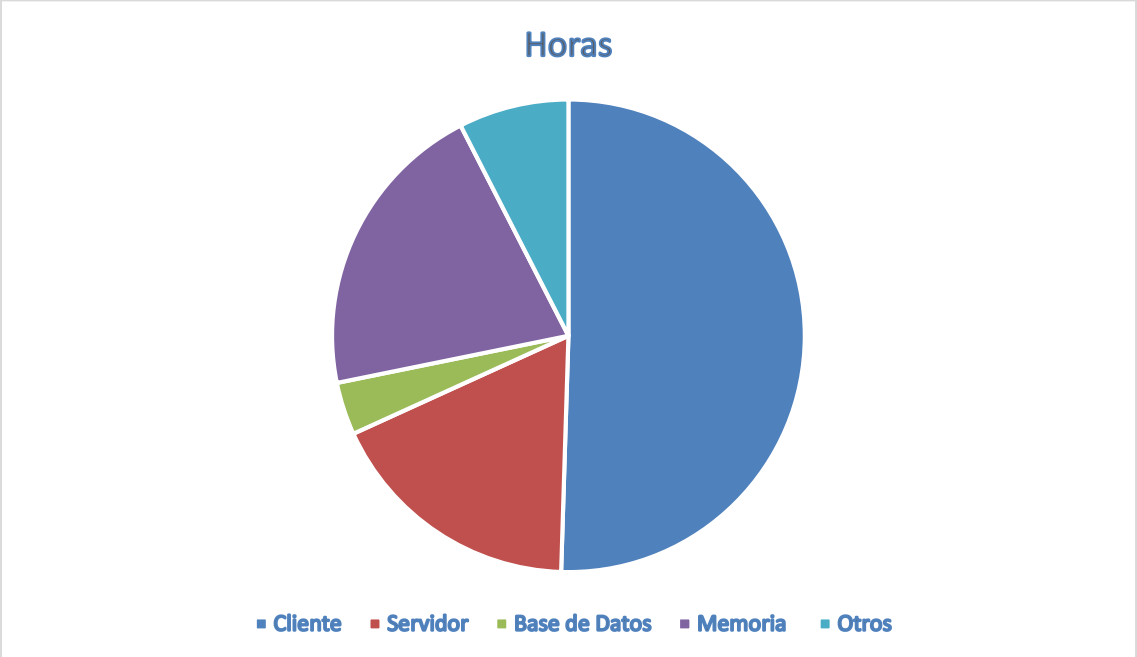

Figura 4. Gráfico reparto de tareas.

Como se puede observar, lo que más tiempo ha llevado ha sido desarrollar el Cliente, ya que es lo que más requisitos contemplaban. Además, es la parte donde menos conocimiento disponía y por tanto, más tiempo he necesitado dedicar para auto aprender y resolver problemas.

Después lo que más tiempo me ha llevado se encuentra la memoria, que ha sido revisada varias veces hasta encontrarse en la disposición actual. Como es el entregable y en lo que se basa todo el proyecto, se le ha dedicado bastante tiempo para pulirlo al máximo

En tercer lugar, ha sido del Servidor, que es el tiempo dedicado a Geoserver y al servidor REST. En ambos tenía ciertos conocimientos, necesitando ampliarlos más para realizar el proyecto.

En cuanto a la BD, no ha llevado mucho tiempo ni problemas, lo que más tiempo llevó fue la exportación de datos de Access a PostgreSQL.

5.3 Planificación

A continuación se muestra la planificación del proyecto representado en un diagrama de Gantt

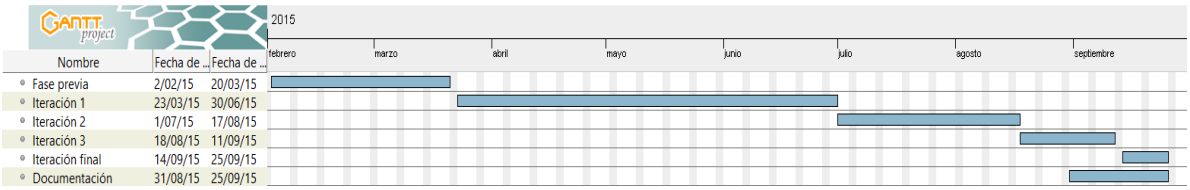

Figura 5. Diagrama de Gantt.

En el diagrama podemos comprobar que existen 6 fases, siendo el inicio el 2 de febrero y acabando el día de la entrega de esta memoria. La primera fase, la he llamado fase previa, que incluye recogida de requisitos, primeras reuniones y preparación del proyecto.

A continuación tenemos los prototipos en las 3 iteraciones comentadas previamente. Como se puede observar, la primera iteración tiene una duración más larga que los otros dos por dos razones. Primera, porque al ser una tecnología nueva se necesita cierto aprendizaje, y segundo porque fue realizado durante el último curso académico, por lo que se le dedicó menos tiempo.

La segunda iteración empieza después de la reunión con la UTC y mostrarle el primer prototipo, dura casi todo el verano, donde se corrigieron errores y se añadió más funcionalidad.

La tercera iteración es el más corto, ya que había menos cosas nuevas que enseñar y que depurar. El prototipo 2 y 3 no se presenta a la UTC, solamente a mi tutor Rubén, ya que los clientes de la UTC no están disponibles por dichas fechas.

Después del tercer prototipo está la iteración final, en el que se ha retocado la aplicación y haciendo cambios sobretodo estéticos.

Por último está la documentación. Quise empezar cuanto antes con ella para poder revisarla tranquilamente y que no hubiera problemas graves con ella.

La duración total del proyecto han sido 7 meses y medio, sin embargo los 4 primeros se le dedicó menos tiempo debido a que al mismo tiempo tenía el octavo cuatrimestre, habiéndolo acordado así con la UTC y con Rubén.

5.4 Gestión de riesgos

A continuación se encuentra la tabla de análisis de riesgos que se preparó al inicio del proyecto, habiéndose presentado alguno, como el de la falta de datos del cliente.

Riesgo	Impacto	Plan de mitigación
Desconocimiento de algunos componentes.	Medio	Desarrollo temprano por medio de prototipos.
Falta de datos por parte del cliente.	Medio	Trabajar con los datos que tengamos y recordarles que cedan todos los datos.
Perdida de datos (código y memorias).	Alto	Usar herramientas online como Dropbox y Github para tenerlo siempre accesible.
No llegar a tiempo a la entrega.	Bajo	Entregar en otra fecha de entrega.

Tabla 5. Análisis de riesgos.

6. Conclusión

A modo de conclusiones, hay que destacar varios aspectos. El primero, que al trabajar con un cliente de verdad, cambia bastante la forma de trabajo, ya que hay que entenderlo que la otra persona quiere, así como ser capaz de realizar cambios en la aplicación. También se aprende a tratar con los problemas como la falta de datos con los que poder trabajar y a tener paciencia con el cliente. Sin embargo, me ha servido como experiencia y para mejorar en la programación, así como en tener una disciplina de trabajo.

Por otra parte, al trabajar con tecnologías Web, cambia el modo de trabajar que se ha aprendido en la Universidad. La primera cosa a tener en cuenta es que tienes que realizar tu aplicación una vez, pero tiene que funcionar en cualquier dispositivo, por lo que requiere trabajar más que si no fuese compatible, aunque la recompensa es mayor. Otra de las particularidades de trabajar con las tecnologías Web es la depuración. No se parece en nada a lo utilizado hasta ahora (Java o C++), ya que la información que te aporta la consola es escasa y cuesta mucho más poder depurar cosas. Para intentar mitigar esto he utilizado el programa WebStorm con su plugin para Chrome en el que ayuda a depurar. Por otra parte, al ser una tecnología con cada vez más uso, se encuentran muchas guías y mucha ayuda por Internet. Además, en mi caso con Leaflet, existen muchos plugins que añaden funcionalidad de manera sencilla a tu aplicación, realizados por otras personas.

Sobre el servicio REST, el poder realizarlo con Spring Boot me he dado cuenta de la importancia de las etiquetas en Java, en cómo como con pocas líneas de código puedes implementar un servicio REST que antes costarían muchas más líneas. Además, la integración con Gradle me parece acertada para futuros proyectos, el poder centralizar las librerías que se utilizan para la aplicación y que con un solo comando cualquier otra persona del proyecto se pueda descargar las mismas librerías. Además ayuda con la configuración (en mi caso, con la compatibilidad de Tomcat 7) así como ejecutar test.

Con Geoserver me he dado cuenta de la importancia de la información geométrica hoy en día, y que son datos difíciles de tratar y que requieren especial cuidado, pero que son muy potentes a la hora de exponerlos al usuario.

Por último hay una serie de cosas que no se han podido añadir a la aplicación por falta del tiempo pero que hubieran sido interesantes para el usuario, como puede ser una identificación de la aplicación para que los trabajadores de la UTC puedan utilizar la aplicación en mayor profundidad, debido a la sensibilidad de algunos datos. Otra cosa que me hubiera gustado añadir es una serie de notificaciones con las noticias de la Universidad. Durante el desarrollo del TFG actualizaron el framework Ionic para añadir notificaciones nativas para la aplicación, sin embargo no he podido ponerlo en práctica. También me gustaría añadir la realización de backups de manera periódica de la máquina virtual.

7. Bibliografía

1. Ionic [*22/09/2015]* http://ionicframework.com/
2. Leaflet [*22/09/2015*] http://leafletjs.com/
3. Angular [*22/09/2015*] https://angularjs.org/
4. Spring Boot [*22/09/2015*] http://projects.spring.io/spring-boot/
5. Soporte multi lenguaje [*22/09/2015*] http://blog.novanet.no/creating-multilingual-support-using-angularjs/
6. Google Maps en Leaflet [*22/09/2015*] https://github.com/shramov/leaflet-plugins
7. Crear secciones de imágenes [*22/09/2015*] http://enralados.com/como-crear-mapas-de-imagenes.html
8. JSON y Geoserver [*22/09/2015*] http://www.gaiaresources.com.au/json-with-geoserver-and-leaflet
9. Github [*22/09/2015*] https://github.com/JorgeGaruz/UZapp
10. Webmin [*22/09/2015*] http://www.webmin.com/
11. Tomcat [*22/09/2015]* https://tomcat.apache.org/tomcat-7.0-doc/index.html

Anexo I: Diseño detallado

En esta sección se van a detallar los módulos más importantes de la aplicación, que han sido presentados en el diagrama de despliegue. Primero se presentan los servicios REST del servidor. Después se detallarán los módulos del cliente Controller.js y Services.js, los dos módulos más importantes de la aplicación del Cliente. Controller.js controla la interacción con el usuario y Services.js se comunica con Leaflet y crea todos los mapas. Existen otras clases que no se detallan como app.js o Factory.js pero no tienen tanta complejidad como estas dos y son menos interesantes.

Por último se incluye un Diagrama de Secuencia en el que se muestra la interacción del usuario con la aplicación cuando el usuario escoge una planta desde el mapa y después selecciona una estancia.

1. Interfaz de los servicios REST

La documentación de la interfaz de los servicios REST está basada en dos clases, que son las encargadas de exponer al público dichos servicios. Las clases son BusquedasRestController y EstanciasRestController. Las rutas indicadas son relativas a la ruta donde esté alojada la aplicación.

1.1 BusquedasRestController

Los servicios REST incluidas en esta clase, se utilizan principalmente para el formulario de búsqueda, además de para buscar información de los edificios que son mostrados en el mapa. Todos los servicios tienen en común la ruta */búsquedas/* que será concatenado con la ruta del método en concreto.

- **codigoEspacios()**

Este servicio sirve para conseguir todos los códigos de los espacios de la Universidad, para ser mostrados en la búsqueda avanzada. La ruta es */codigoespacios*, realizando una petición GET y devuelve el resultado en un JSON

Si hay un error devuelve un código HTTP 500.

Ejemplo de petición al servicio:

Petición:

1. **Request URL:**

   ```
   http://dirIP/busquedas/codigoespacios
   ```

2. **Request Method:**

   ```
   GET
   ```

3. **Status Code:**

   ```
   200 OK
   ```

Respuesta:

1. **Content-Length:**

   434480

2. **Content-Type:**

   application/json;charset=UTF-8

3. **Date:**

   Tue, 22 Sep 2015 15:34:54 GMT

JSON devuelto (solo una parte):

[{"ID_espacio":"CHU.1068.01.010"},{"ID_espacio":"CHU.1068.01.020"},{"ID_espacio":"CHU.1068.01.030"},{"ID_espacio":"CHU.1068.01.040"},{"ID_espacio":"CHU.1068.01.060"},{"ID_espacio":"CHU.1068.01.070"},{"ID_espacio":"CHU.1068.01.090"},{"ID_espacio":"CHU.1068.01.100"},{"ID_espacio":"CHU.1068.01.120"},{"ID_espacio":"CHU.1068.01.130"},{"ID_espacio":"CHU.1068.01.140"},{"ID_espacio":"CHU.1068.01.150"},

- **Campus(String ciudad)**

Este servicio se utiliza para devolver los distintos campus que existen en la ciudad *ciudad,* siendo un valor del 1 al 3. Se utiliza para filtrar en la búsqueda. La ruta es */campus*, realizando una petición GET y devolviendo el resultado en un JSON

Si hay un error devuelve un código HTTP 500.

Ejemplo de petición al servicio:

1. **Request URL:**

   http://dirIP/busquedas/campus?ciudad=1

2. **Request Method:**

   GET

3. **Status Code:**

   200 OK

Respuesta:

1. **Content-Length:**

   170

2. **Content-Type:**

   application/json;charset=UTF-8

3. **Date:**

   Tue, 22 Sep 2015 15:39:34 GMT

JSON devuelto:

[{"ID":3,"campus":"Campus Miguel Servet"},{"ID":4,"campus":"Campus Paraninfo-Empresariales"},{"ID":2,"campus":"Campus Rio Ebro"},{"ID":1,"campus":"Campus San Francisco"}]

- **Edificio(String campus)**

Este servicio sirve para mostrar en el formulario de búsqueda todos los edificios del campus *campus*, siendo un valor del 1 al 6. La ruta es */edificio*, realizando una petición GET y devolviendo el resultado en un JSON.

Si hay un error devuelve un código HTTP 500.

Ejemplo de petición al servicio:

Petición:

1. **Request URL:**

   http://dirIP/busquedas/edificio?campus=4

2. **Request Method:**

   GET

3. **Status Code:**

   200 OK

Respuesta:

1. **Content-Length:**

   4726

2. **Content-Type:**

   application/json;charset=UTF-8

3. **Date:**

   Tue, 22 Sep 2015 10:25:23 GMT

JSON devuelto:

[{"ID_Edificio":"CPE.1034.","edificio":"AULA MAGNA -F. CIENCIAS ECONOMICAS Y EMPRESARIALES","direccion":"C/Dr. Cerrada, 1","plantas":["00","01","02"]},{"ID_Edificio":"CPE.1107.","edificio":"BIBLIOTECA DE ECONOMICAS Y EMPRESARIALES","direccion":"C/Dr. Cerrada, 1","plantas":["00","01","0A","1A","S1"]},{"ID_Edificio":"CPE.1033.","edificio":"FACULTAD DE CIENCIAS ECONOMICAS Y EMPRESARIALES","direccion":"C/Dr. Cerrada, 1","plantas":["00","01","02","03","04","05","06","0A","1A","2A","3A","S1","S2"]},{"ID_Edificio":"CPE.1032.","edificio":"PARANINFO","direccion":"Plaza Paraíso, 4","plantas":["00","01","02","03","04","SS"]}]

- **infoEdificio(String edificio)**

Este servicio sirve para mostrar la información de un edificio, dando su id con el parámetro *edificio* siendo un código juntando el nombre del campus (CSF, por ejemplo), el número de edificio (1106 para el Edificio Cervantes) y el número de planta. La información es el nombre del edificio, dirección y número de plantas. La ruta es */infoedificio*, realizando una petición GET y devolviendo un JSON.

Si hay un error devuelve un código HTTP 500.

Ejemplo de petición al servicio:

Petición:

1. **Request URL:**

   ```
   http://dirIP/busquedas/infoedificio?edificio=CSF.1106.
   ```

2. **Request Method:**

   ```
   GET
   ```

3. **Status Code:**

   ```
   200 OK
   ```

Respuesta:

1. **Content-Length:**

   ```
   126
   ```

2. **Content-Type:**

   ```
   application/json;charset=UTF-8
   ```

3. **Date:**

   ```
   Tue, 22 Sep 2015 15:43:21 GMT
   ```

JSON devuelto:


```
[{"ID_Edificio":"CSF.1106.","edificio":"EDIFICIO CERVANTES","direccion":"C/Corona de Aragón,
42","plantas":["00","01","02"]}]
```


1.2 EstanciasRestController

Los servicios REST incluidos en esta clase, son utilizados en la visualización de planos, cuando se quiere saber información general de la estancia (nombre) o cuando se quiere saber más información de la estancia. Todos los servicios tienen en común la ruta */estancias/* que será concatenado con la ruta del método en concreto.

- **getEstancia(String estancia)**

Este servicio se utiliza cuando el usuario ha seleccionado una estancia, pasando su código de espacio como atributo *estancia,* para formar el parámetro hay que juntar el nombre del campus (CSF, por ejemplo), el número de edificio (1106 para el Edificio Cervantes) y el número de planta y añadirle el código único de la estancia (020 para el vestíbulo de la planta 0).

Devuelve toda la información necesaria al usuario relativo a una estancia, nombre, tipo de uso, superficie. La ruta es */getEstancia*, realizando una petición GET y devolviendo un JSON.

Si hay un error devuelve un código HTTP500.

Ejemplo de petición al servicio:

Petición:

1. **Request URL:**

   ```
   http://dirIP/estancias/getEstancia?estancia=CSF.1110.00.020
   ```

1. **Request Method:**

   ```
   GET
   ```

2. **Status Code:**

   ```
   200 OK
   ```

Respuesta:

1. **Content-Length:**

   ```
   99
   ```

2. **Content-Type:**

   ```
   application/json;charset=UTF-8
   ```

3. **Date:**

   ```
   Tue, 22 Sep 2015 15:46:17 GMT
   ```

JSON devuelto:

{"ID_espacio":"CSF.1110.00.020","ID_centro":"VESTIVULO","tipo_uso":"PASILLO","superficie":"454.30"}

- **getAllEstancias(String espacio)**

Este servicio se utiliza en el buscador para mostrarle al usuario todas las estancias que existen dado un edificio y una planta, que se juntan en el parámetro *espacio*. Para formar el parámetro hay que juntar el nombre del campus (CSF, por ejemplo), el número de edificio (1106 para el Edificio Cervantes). Devuelve todos los códigos de espacios y nombres de las estancias de dicho edificio y planta. La ruta es */getAllEstancia* realizando una petición GET y devolviendo un JSON.

Si hay un error devuelve un código HTTP500.

Ejemplo de petición al servicio:

Petición:

1. **Request URL:**

   ```
   http://dirIP/estancias/getAllEstancias?estancia=CRE.1200.00
   ```

2. **Request Method:**

   GET

3. **Status Code:**

   200 OK

Respuesta:

1. **Content-Length:**

   4448

2. **Content-Type:**

   application/json;charset=UTF-8

3. **Date:**

   Tue, 22 Sep 2015 15:42:55 GMT

JSON devuelto (parte):


```
[{"ID_espacio":"CRE.1200.00.150","ID_centro":"ALMACEN"},{"ID_espacio":"CRE.1200.00.201","ID_centro":"ALMACEN"},{"ID_espacio":"CRE.1200.00.025","ID_centro":"ASCENSORES},{"ID_espacio":"CRE.1200.00.050","ID_centro":"AULA_A.02"},{"ID_espacio":"CRE.1200.00.070","ID_centro":"AULA_A.04"},{"ID_espacio":"CRE.1200.00.090","ID_centro":"AULA_A.06"},{"ID_espacio":"CRE.1200.00.100","ID_centro":"AULA_A.07"},{"ID_espacio":"CRE.1200.00.350","ID_centro":"BAÑOS"},{"ID_espacio":"CRE.1200.00.360","ID_centro":"BAÑOS"},{"ID_espacio":"CRE.1200.00.610","ID_centro":"BAÑOS"},{"ID_espacio":"CRE.1200.00.620","ID_centro":"BAÑOS"},{"ID_espacio":"CRE.1200.00.600","ID_centro":"CAFETERÍA"},{"ID_espacio":"CRE.1200.00.590","ID_centro":"COCINA"},{"ID_espacio":"CRE.1200.00.630","ID_centro":"CONSERJERIA"},{"ID_espacio":"CRE.1200.00.650","ID_centro":"CUADRO
ELECTRICO"},{"ID_espacio":"CRE.1200.00.530","ID_centro":"CUADRO
ELÉCTRICO"},{"ID_espacio":"CRE.1200.00.160","ID_centro":"DESPACHO"},{"ID_espacio":"CRE.1200.00.170","ID_centro":"DESPACHO"},{"ID_espacio":"CRE.1200.00.175","ID_centro":"DESPACHO"},{"ID_espacio":"CRE.1200.00.490","ID_centro":"DESPACHO
0.05"},{"ID_espacio":"CRE.1200.00.480","ID_centro":"DESPACHO
0.06"},{"ID_espacio":"CRE.1200.00.470","ID_centro":"DESPACHO
0.07"},{"ID_espacio":"CRE.1200.00.460","ID_centro":"DESPACHO 0.08"},
```


2. Controller.js

Este módulo es uno de los más importantes en el lado del cliente, ya que se encarga de las interacciones del usuario, así como los cambios de pantalla y los elementos visuales. Este módulo es debido al uso de AngularJS, ya que en ella se puede crear un controlador por cada pantalla y se decidió crearlos todos en la misma clase, que es controller.js. A continuación se muestran los distintos controladores que existen.

- **AppCtrl.** Este controlador se encarga de la interacción en la pantalla principal, controla el mapa interactivo que el usuario debe pulsar para visitar una ciudad determinada. También es el controlador por defecto, por lo que si una vista no tiene controlador se le asigna este.

- **MapCtrl.** Este controlador se encarga de la vista del mapa con Google Maps a través de Leaflet. Para ello invoca a Services.js y registra los eventos que suceden en el mapa, como seleccionar una planta para mostrar su plano.

- **PlanCtrl.** Este controlador se encarga de la vista del plano del edificio seleccionado. También invoca a Services.js para crear el plano y registra los eventos que suceden en el plano, como seleccionar una estancia para mostrar su información.

- **TranslationCtrl.** Este controlador se encarga de la traducción de la aplicación en la pantalla de ajustes. Se encarga tanto de inicializar la aplicación con un idioma (por defecto el español) como controlar la traducción de la aplicación por el idioma escogido por el usuario.

- **SearchCtrl.** Este controlador se encarga de la pantalla de búsquedas, y se encarga de controlar el comportamiento de cada desplegable de la pantalla, así como del botón de búsqueda. Cada desplegable tiene una función específica dentro del controlador que invoca una factoría de AngularJS que es la encargada de la conexión con el servicio REST del servidor.

- **EstanciaCtrl.** Este controlador se encarga de la pantalla con la información de la estancia seleccionada. Es uno de los controladores más complejos, ya que además de rellenar los datos de la estancia, en él comprobamos si dicha estancia tiene fotos disponibles y cuántas, y esta información se guarda en un objeto AngularJS y si tiene fotos disponibles cambia a la pantalla de fotos, si no, muestra un mensaje de alerta.

- **PhotoCtrl.** Este controlador se encarga de mostrar las fotos al usuario y crear un visor de imágenes sencillo, en el que poder pasar de foto con unos botones. La única complejidad de este controlador son los límites de las fotos y deshabilitar los botones siguiente o anterior si no se puede ir a la siguiente foto o a la anterior.

3. Services.js

Este módulo se encarga de toda la comunicación con el framework Leaflet para poder visualizar mapas y planos. Es el módulo donde más he trabajado y donde más cambios he hecho, ya sea por mejor rendimiento, por encontrar mejores alternativas o porque ha sido la clase donde más problemas he tenido, ya que Leaflet era una tecnología nueva para mí. Uno de los problemas más graves fue que los datos que me proporcionó la UTC no estaban bien georeferenciados y se necesitó bastante tiempo para descubrirlo porque pensaba que era problema de Leaflet. A continuación voy a describir las funciones que nos podemos encontrar en esta clase.

- **crearMapa($scope,miFactoria,opción,GetInfoService).** Es sin duda la función más completa de todo el fichero y que más cambios ha conllevado. En esta función se crea el mapa a partir del mapa de Google Maps y se le añaden los plugins de Geolocalización y búsqueda de marcadores. Después se añaden las capas de los edificios de la ciudad escogida a través de una llamada al servicio WMS de

Geoserver y se procede a invocar la función *añadirMarcadores* que será descrita a continuación. Una vez terminado devolvemos el mapa al controlador para que pueda hacer después cambios de ciudad en el mapa. Los parámetros de entrada son $scope, objeto de AngularJS que puedes almacenar cualquier tipo de información y comparte la información entre los diferentes controladores, miFactoria, que contiene la información de las 3 ciudades, como la longitud y latitud a la que se encuentran y que se muestra en el mapa según el parámetro opción que es elegido por el usuario en la pantalla inicial o en el menú. Por último tenemos GetInfoService que se encarga de la conexión con los servicios REST para poder añadir los marcadores con la información del edificio.

- **añadirMarcadores($scope,index,GetInfoService).** Esta función, que es invocada por la anterior, añade marcadores en la posición de los edificios que ha añadido *crearMapa*.Lo primero que hace es una llamada al servicio WFS de Geoserver que contiene las coordenadas de los edificios (ya que por WFS no se puede obtener las coordenadas para añadir los marcadores en el punto exacto) y luego se contacta con los servicios REST para obtener la información del edificio que estamos añadiendo y se crea un marcador de Leaflet con toda esta información. De los parámetros de entrada, el único que cambia es index, que proporciona la información sobre el edificio del cual tenemos que pedir información a Geoserver y al servicio REST.

- **localizarHuesca, localizarZaragoza y localizarTeruel ($scope, miFactoria).** Estas tres funciones se utilizan para cambiar el punto de vista del mapa, si es que ya estaba creado, si no se crea uno nuevo. Es invocada por el controlador cuando el usuario pulsa sobre una ciudad y el mapa realiza una animación cambiando las coordenadas.

- **crearPlano($scope, GetInfoService).** Esta es la otra función más importante, que se encarga de crear el mapa con el plano seleccionado en el mapa. Se encarga de realizar una petición al servicio WFS de Geoserver para traer el plano del servidor y después se añade a Leaflet y se le asocia la función onEachFeature cuando se pulsa sobre cualquier parte del plano y que será comentada a continuación.

- **onEachFeature(feature,layer) y whenClicked(e).** Estas dos funciones van relacionadas. whenClicked es invocada por Leaflet cuando el usuario pulsa en cualquiera de las estancias del plano mostrado, ya que así ha sido dispuesto en onEachFeature. Cuando se invoca whenClicked se obtiene el ID de la estancia seleccionada por medio del atributo e, y se invoca al servicio REST por medio de GetInfoService, para mostrar al usuario el nombre de la estancia y si quiere obtener más información sobre la misma. Al principio existía otra alternativa, que era cargar toda la información de las estancias antes de mostrárselo al usuario, sin embargo, esto mermaba el rendimiento de la Base de Datos y se escogió esta solución con mejor rendimiento.

4. Diagrama de secuencia de la interacción del usuario con el mapa

El diagrama de secuencia retrata la relación de los componentes cuando el usuario elige la planta de un edificio en el mapa de Google Maps hasta que elige la información de una estancia en otro plano. Este diagrama representa casi todos los componentes existentes en la aplicación.

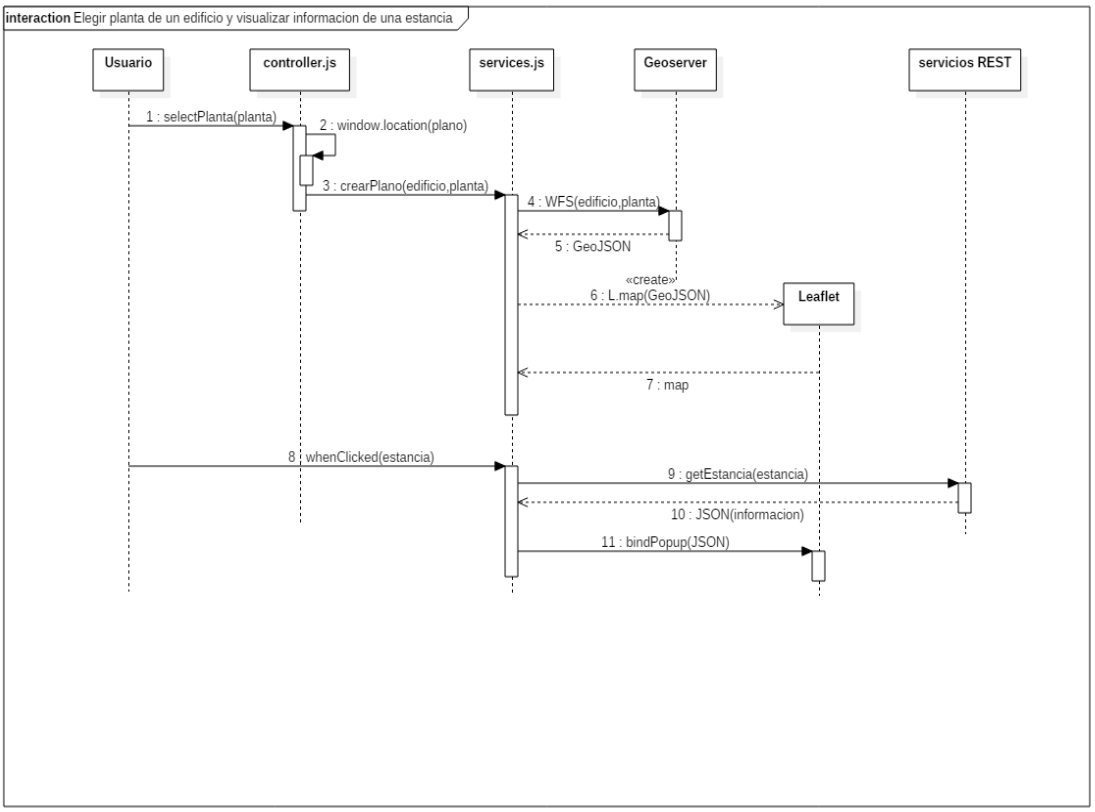

Figura 6. Diagrama de secuencia.

En el diagrama de secuencia se puede ver la interacción del usuario con el sistema. Empieza cuando un usuario selecciona un edificio del mapa de Google Maps, en el que hay resaltados diversos edificios. Después de abrir el marcador del edificio, que muestra información del edificio y un desplegable, el usuario selecciona una planta del edificio que quiere ver. Una vez realizado esto, se llama a la función asociada de la clase controller.js que guarda los datos y cambia la pantalla que visualiza y esto hace cambiar el controlador asociado también.

Una vez cambiado el controlador, el nuevo controlador invoca al método *crearPlano (edificio, planta*) que se ha visto anteriormente (estos datos se guardan en el $scope de AngularJS) de la clase services.js.

Una vez en la clase services.js, se llama al servicio WFS de Geoserver, pasándole la información del edificio y planta para que nos devuelva el plano de la planta de ese edificio. El formato de vuelta será un GeoJSON, en el que además de contener la información de las estancias, nos las devuelve georeferenciadas por lo que luego es más fácil trabajar con ellas.

A continuación se crea el mapa de Leaflet vacío y se le añade los datos del plano en formato GeoJSON. En la creación, se le asocia una función para cuando el usuario haga click, salte a dicha función y se pueda averigurar el ID de la estancia que ha hecho click, todo esto mediante Leaflet.

Después de crear el plano, el usuario elige una estancia sobre la que quiere saber información. Éste evento llama a la función asociada que hemos creado y que nos da el ID de la estancia. Con este ID llamamos al servicio REST para averiguar la información de dicha estancia, como el nombre que tiene, el tipo de estancia, la superficie o la ocupación.

Una vez recogida la información del servidor en formato JSON, se crea un popup de Leaflet con la información y que se muestra al usuario, dándole la posibilidad de visualizar más información y de ver fotos si es que están disponibles para dicha estancia.

Anexo II: Manual de instalación

1. Servidor

Para tener el código con todo el proyecto, hay que descargarlo de GitHub, en la siguiente dirección: https://github.com/JorgeGaruz/UZapp. Para descargar el código existen dos opciones en la parte derecha de la página, Clone in Desktop y Download ZIP. La primera requiere el programa de GitHub, por lo que se recomienda la segunda opción. Después de descargar el fichero hay que descomprimirlo con cualquier programa compatible con ficheros ZIP. Una vez descomprimido el fichero aparecerá una nueva carpeta *UZapp-master* y dentro contiene todo el código fuente de la aplicación. En la carpeta server está todo el código del servidor, que también incluye el código del cliente para ser convertido en fichero WAR para un servidor de aplicaciones. También hay otra carpeta que se denomina web que es el código del cliente de manera más accesible.

La instalación se puede realizar de dos maneras, una a través del fichero WAR y otra ejecutando las clases Java, ya que en Spring Boot una de las clases contiene el main que inicializa el servidor.

La manera más sencilla es disponer de un servidor de aplicaciones y desplegar el fichero WAR. Sin embargo hay que modificar tres ficheros para que esto funcione bien. En la carpeta */src/main/webapp/www/js/controller.js* hay que modificar las líneas 218 y 219, modificando la variable URI_busquedas y URI_estancias por la dirección donde vaya a ser desplegada el servidor.

También hay que modificar la clase que se conecta con la Base de Datos, en caso que esté en otra dirección que no sea la predefinida. Esto se realiza modificando el fichero *ConnectionManager.java* que está en la carpeta */src/main/java/com/uzapp/bd.* En concreto hay que modificar la línea 38 y añadir la direción en la que se encuentre la base de datos, si es que se ha cambiado la dirección.

Por ultimo hay que modificar el fichero *bd.txt* que se encuentra en */src/main/resources* y cambiar el nombre y contraseña del usuario de la Base de Datos.

Para crear el fichero WAR lo más sencillo es utilizar Gradle. En la carpeta server y con los cambios realizados, por medio de la consola de comandos escribimos *Gradle build* y realizará una serie de operaciones y creará el fichero WAR dentro de la carpeta warFiles.

Después de realizar esto, se sube el fichero WAR a tomcat (o se ejecuta el código con el método main) y ya estará el servidor en funcionamiento. Para comprobar que todo se encuentra correctamente, solo se necesita visitar la dirección *http:direccionIP:8080/mapa/www* en cualquier navegador.

2. Cliente

Para el cliente, después de instalar el servidor sólo hace falta visitar la dirección *http:direccionIP:8080/mapa/* en cualquier navegador compatible. También será posible en Android instalar la aplicación por medio del .apk. Para realizar esto, tenemos que tener el servidor corriendo en una máquina separada y modificar el fichero *web/www/js/controller.js* las líneas 218 y 219, modificando la variable URI_busquedas y URI_estancias por la dirección donde vaya a ser desplegada el servidor. Hecho esto, por línea de comandos dentro de la carpeta www escribimos: *cordova build --release android* ó *ionic build android*, ambas deberían funcionar. Entonces el apk lo encontraremos en la carpeta

platforms/android/build/outputs/apk y ya podremos instalarlo en un móvil con Android. Se puede hacer lo mismo para crear una aplicación para iOS, sin embargo, para probar la aplicación en un dispositivo iOS necesitaría la aplicación estar subida al App Store.

3. Base de Datos

Para la Base de Datos, se recomienda utilizar PostgreSQL, debido al plugin PostGIS para trabajar con datos georeferenciados. Una vez instalado ambas cosas, hay que crear una Base de Datos y añadir la extensión postGIS. Hecho esto, se crea un tablespace y una cuenta de usuario si fuese necesario. Por último se crea un Schema y se importan en él los datos de la aplicación. Una vez hecho esto, ya estaría funcionando la Base de Datos y solo habría que modificar la seguridad, modificando los ficheros de configuración para que sólo permita direcciones de una determinada dirección.

Para pasar los datos de Access a PostgreSQL en Windows se sigue el siguiente proceso:

1. Instalar PostgreSQL ODBC Driver(aquí http://www.postgresql.org/ftp/odbc/versions/msi/)
2. En herramientas administrativas(Panel de control), seleccionar Orígenes de datos ODBC
3. Añadir un nuevo origen de datos de usuario de nombre PostgreSQL Unicode y rellenar con los datos de la BD.
4. En el fichero .mdb(Access), seleccionar la tabla a exportar y escoger Bases de Datos ODBC
5. Decidimos el nombre de la tabla y en seleccionar origen de datos, escogemos la pestaña de Origen de datos de equipo. Seleccionamos el origen de datos creado en el paso nº3.
6. Por último seleccionamos desde PGAdmin la tabla recien creada para modificar la clave primaria, ya que en la exportación no se conserva.

Anexo III: Manual de usuario

Este es el manual de usuario de la Aplicación móvil para la consulta y geolocalización de los espacios de los edificios de la Universidad de Zaragoza a partir de los datos de la Unidad Técnica de Construcciones y Energía. En esta aplicación, el usuario puede visualizar los edificios que pertenecen a la Universidad de Zaragoza en las tres provincias de Aragón en el mapa de Google Maps. Además de visualizar los edificios, puede acceder a los planos de los edificios, seleccionado una planta concreta. En la visualización de los planos se puede obtener información de cualquiera de las estancias que aparecen en el plano pulsando el botón de *Más información*. Dentro de la información de la estancia se visualiza una información completa de la estancia, además de mostrar el botón de fotos, el cual si es pulsado, muestra si la estancia tiene foto para mostrar o no. En caso de ser afirmativo te lleva a otra pantalla y muestra un visor de fotografías.

Además de esto, la aplicación también tiene opciones de búsqueda y favoritos, además de ajustes en el que se puede cambiar el idioma de la aplicación entre otras cosas.

Las imágenes de este manual o el proceso que en ellas se ve pueden variar con alguna actualización de la aplicación.

1. Inicio

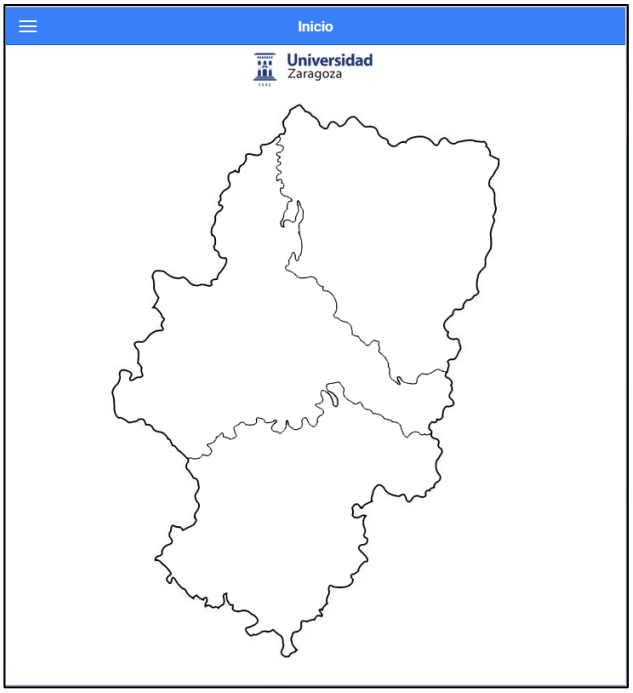

Figura 7. Mapa de inicio.

Esta es la pantalla de inicio, la primera pantalla que se ve cuando ejecutamos la aplicación. En ella muestra el título de la pantalla, el símbolo de menú, el logo de la universidad y un mapa de Aragón para seleccionar cuál de las tres provincias se desea visitar. Para acceder al mapa de la ciudad solamente hay que pulsar sobre la provincia que se desea.

En el menú de la aplicación, que es accesible desde cualquier ventana de la aplicación, ofrece las siguientes opciones:

Figura 8. Menú.

Inicio: Muestra la pantalla principal de la aplicación, con el mapa de Aragón.

Búsqueda: Pantalla en el que aparece un formulario de búsqueda para buscar una estancia concreta, utilizando filtros.

Búsqueda avanzada: Pantalla útil para los trabajadores de la UTC, ya que muestra todos los códigos de espacios existentes en la Base de Datos y muestra el seleccionado.

Favoritos: Pantalla donde aparecen las estancias marcadas como favoritos del usuario.

Ajustes: Pantalla donde se configuran diferentes ajustes de la aplicación, como el idioma.

Zaragoza: Opción para mostrar el mapa de Zaragoza sin necesidad de pulsar en el mapa de Aragón.

Huesca: Opción para mostrar el mapa de Huesca sin necesidad de pulsar en el mapa de Aragón.

Teruel: Opción para mostrar el mapa de Teruel sin necesidad de pulsar en el mapa de Aragón.

2. Mapa

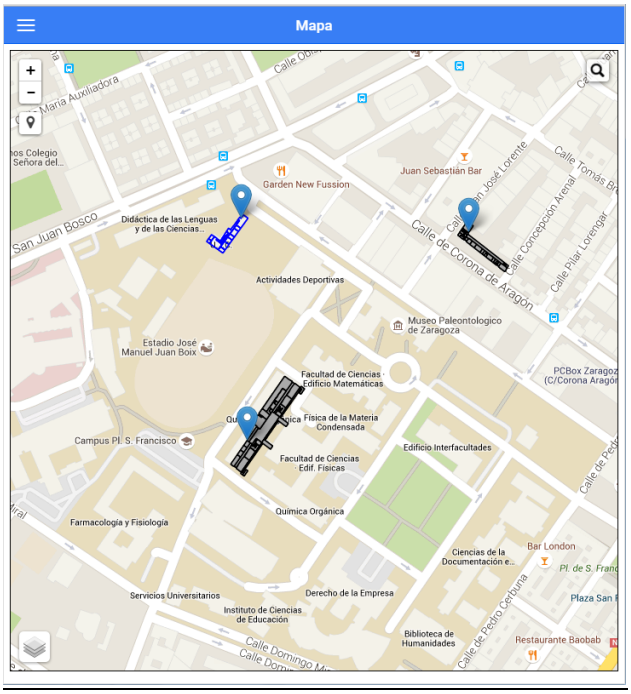

Figura 9. Visualización del mapa.

En la pantalla de Mapa, está la parte importante de la aplicación. En ella se ve el mapa de Google Maps pero con edificios resaltados y con marcadores sobre los edificios. Se puede navegar libremente por el mapa y muestra información de las calles y locales o cualquier punto de interés sobre la zona. Pasando el cursor por encima de los marcadores muestra el nombre del edificio sin necesidad de pulsar sobre él.

En el mapa se visualizan diferentes elementos que sirven de complemento al usuario y que son mostrados a continuación

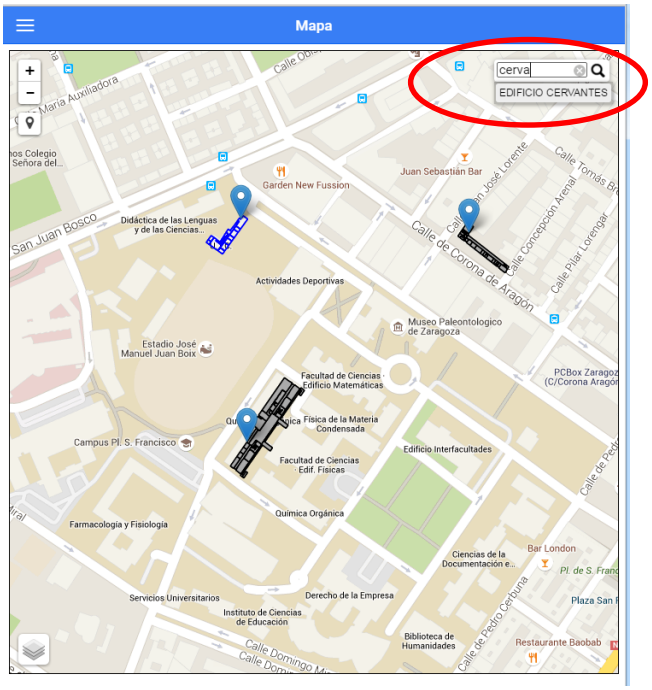

Figura 10. Búsqueda de edificios.

En la parte superior derecha existe un buscador predictivo, en el que se puede escribir el nombre del edificio que se quiere buscar. Si encuentra algún resultado que concuerda con lo introducido, lo sugiere.

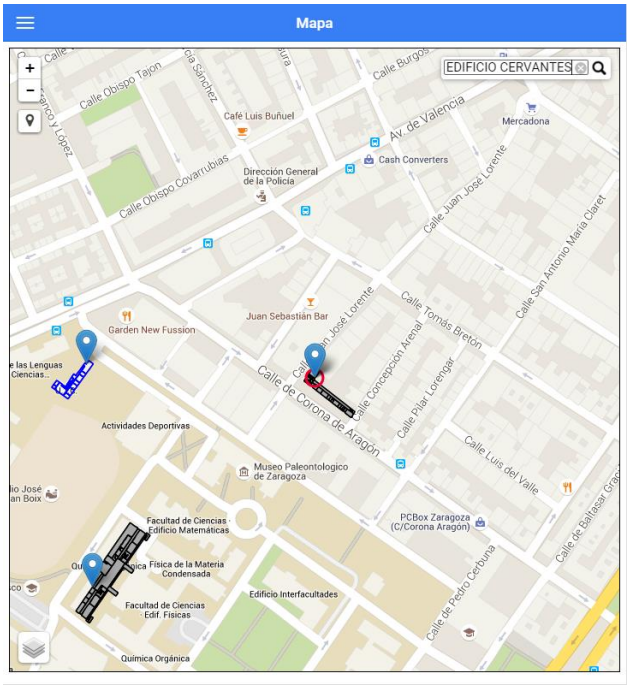

Figura 11. Resultado de una búsqueda en el mapa

El resultado de una búsqueda se muestra con un círculo rojo rodeando al marcador y además redirige el mapa hacia dicha posición.

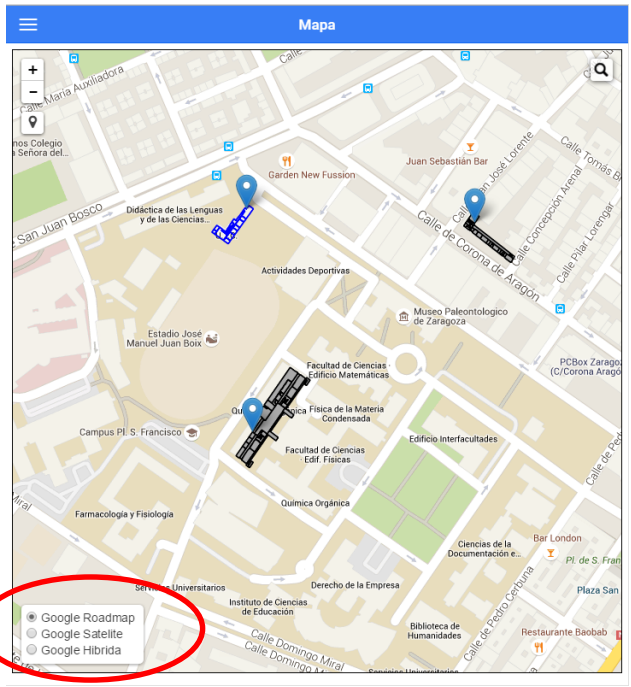

Figura 12. Selección filtros del mapa.

En la parte inferior izquierda encontramos con un filtro para poder mostrar la capa de Google Maps que queramos. Podemos elegir el Google Roadmap, que es la capa por defecto, Satelite que muestra la imagen por satélite e Híbrida que muestra imágenes por satélite con el nombre de las calles.

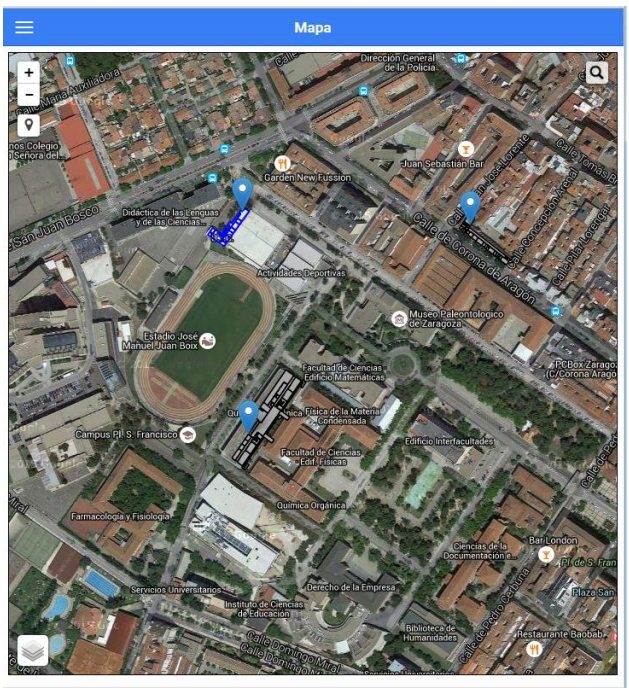

Figura 13. Filtro satélite.

Esta es la vista Híbrida, que incluye imagen por satélite y el nombre de calles y establecimientos. Como se puede observar, se siguen mostrando los edificios resaltados así como los marcadores, pero éstos son menos visibles.

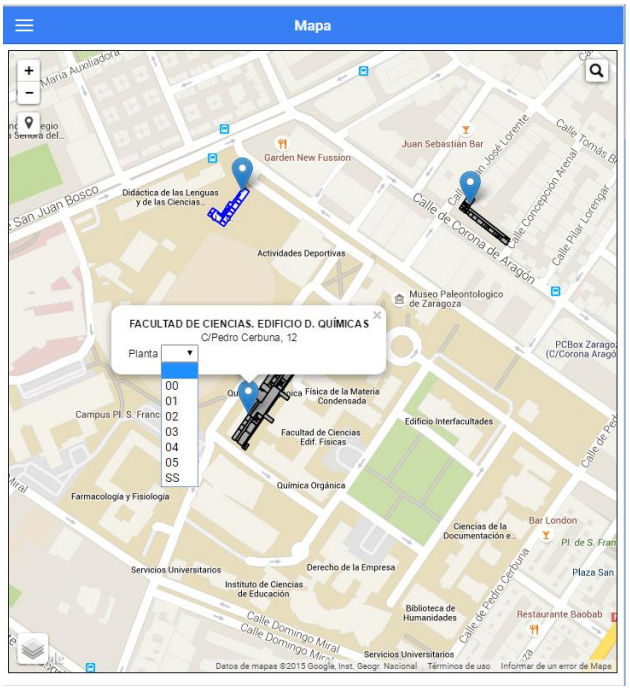

Figura 14. Información de un edificio

Cuando se selecciona un marcador, éste se abre y muestra información del edificio relacionado, como el nombre del edificio, la dirección y el número de plantas que tiene.

Para seleccionar el número de plantas hay que pulsar en el desplegable y seleccionar la planta correspondiente, que después te llevará al plano.

3. Plano

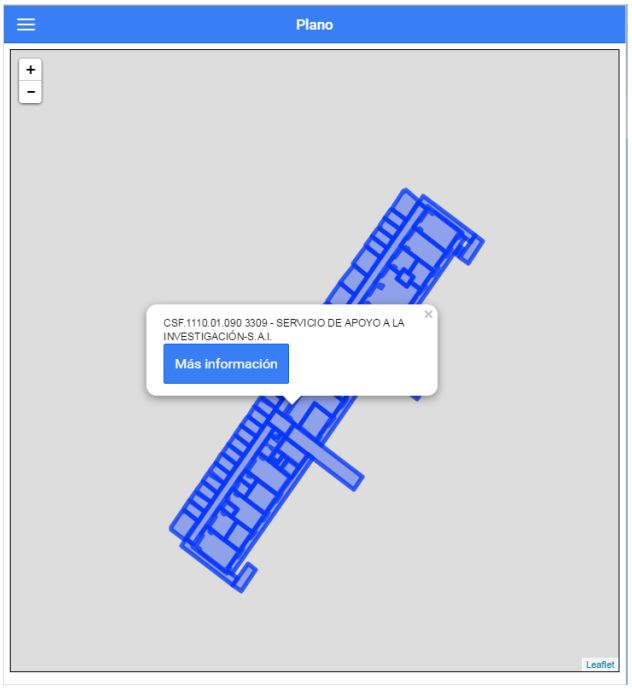

Figura 15. Visualización de un plano

Una vez seleccionada la planta, se visualiza en otro visor de mapas el plano de la planta y el edificio seleccionado, en el que se visualizan las diferentes estancias. Cuando se pulsa una estancia se abre otro marcador mostrando la información de dicha estancia, como el código de espacio asociado y el nombre de la estancia.

También se muestra un botón para saber más información de la estancia y poder ver fotos si hubiese disponible para dicha estancia.

4. Estancia

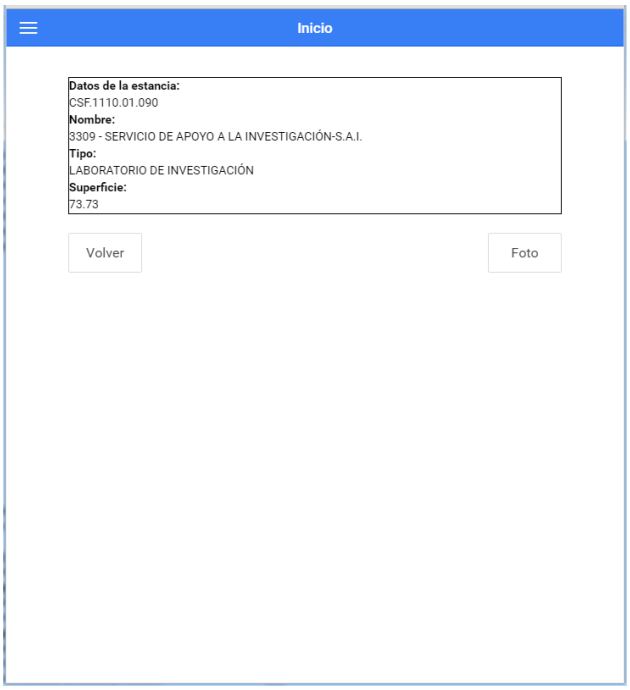

Figura 16. Información de una estancia.

Una vez seleccionada una estancia en el plano, o realizando el formulario de búsqueda, aparece la información de la estancia. Esta información incluye los datos de la estancia (código de espacio de la UTC), el nombre que tiene dicha estancia, el tipo de estancia y la superficie. Ésta información puede variar de una versión a otra.

También aparecen dos botones: Volver, que vuelve al plano y Foto, que busca si en dicha estancia se encuentra disponible una foto o no.

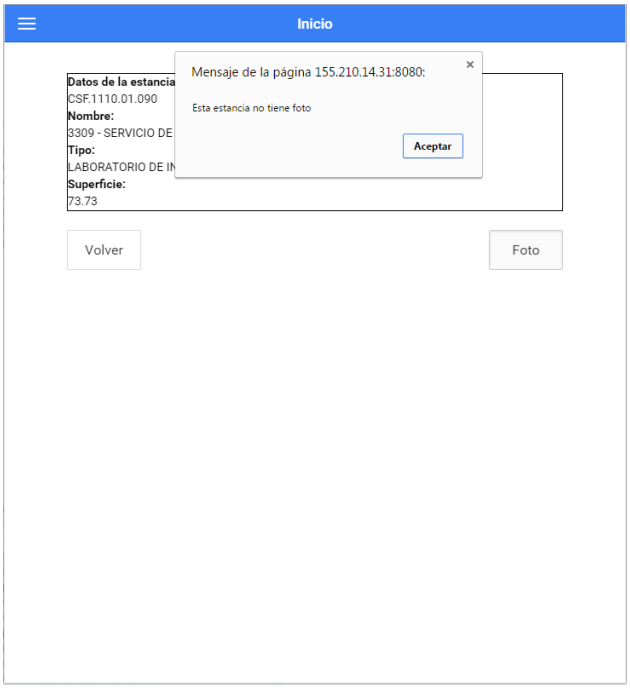

Figura 17. Foto no encontrada.

En este caso, la estancia asociada no tiene foto disponible.

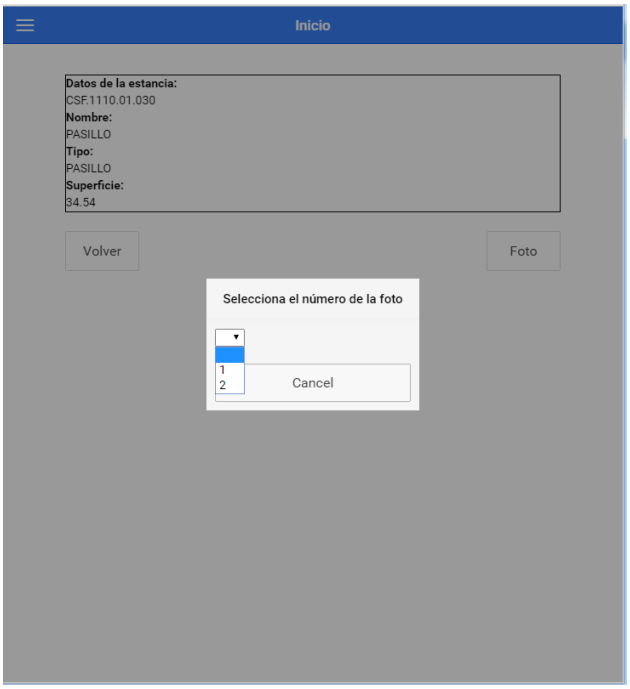

Figura 18. Selección número de fotos encontradas.

En este caso, sí que existe foto asociada a la estancia. En este caso, existen 2 fotos de la estancia, por lo que se tiene la opción de escoger cualquiera de las dos para mostrar.

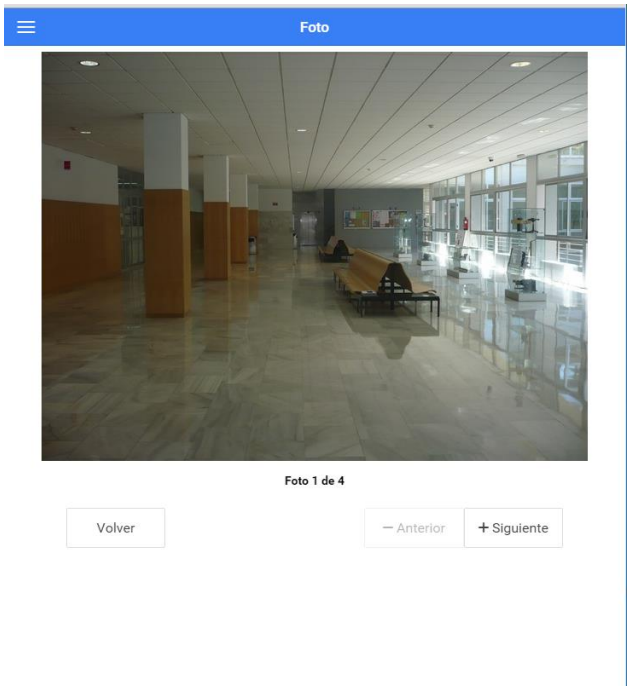

Figura 19. Foto de una estancia.

Foto asociada de la estancia. En esta pantalla se puede escoger las distintas fotos disponibles de la estancia. Además se muestra de manera escrita la foto en la que te encuentras.

5. Otras opciones.

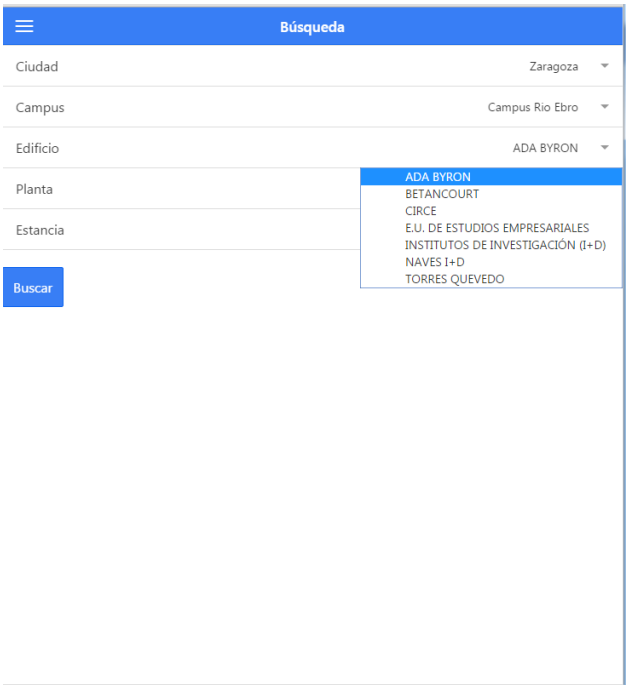

Figura 20. Formulario de búsqueda.

En esta pantalla tenemos el formulario de búsqueda, para encontrar una estancia. Existen distintos desplegables que van añadiendo filtros a la búsqueda. Una vez completados todos los filtros, al pulsar el botón Buscar te lleva a la información de la estancia que se ha buscado.

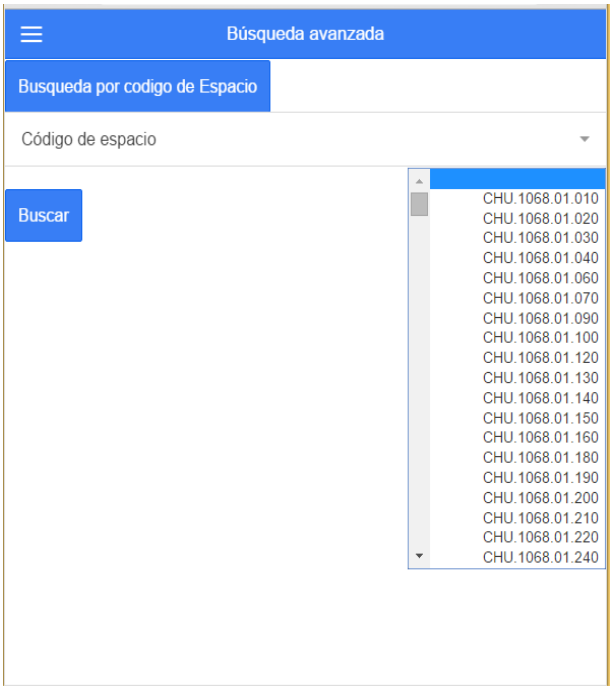

Figura 21. Formulario de búsqueda avanzada.

Esta pantalla corresponde a la búsqueda avanzada, sólo es útil para los trabajadores de la UTC. Pulsando el botón “Búsqueda por código de Espacio” aparece un desplegable donde aparecen todos los códigos de Espacio de la Base de Datos. Este proceso tarda bastante, por lo que hay que tener paciencia. Una vez seleccionado el código de espacio, se pulsa el botón de buscar y te llevará a la estancia asociada.

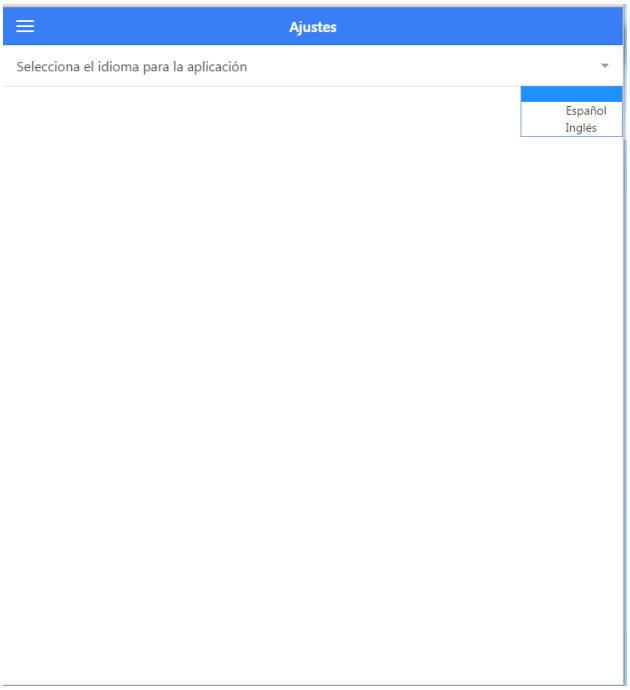

Figura 22. Ajustes de la aplicación.

En esta pantalla tenemos los ajustes de la aplicación. En esta pantalla es dónde se puede cambiar el idioma de la aplicación, pudiendo seleccionar Inglés y Español, pero esto se puede actualizar en un futuro a más idiomas.

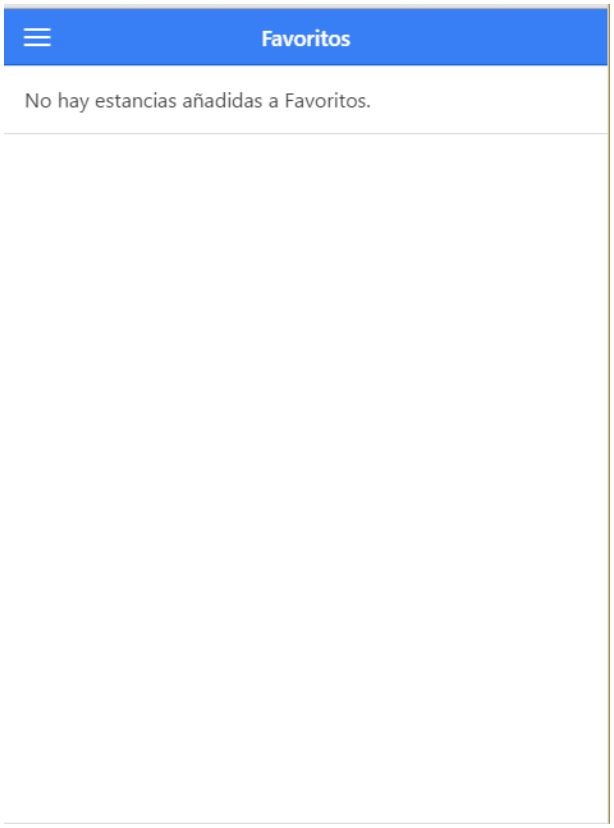

Figura 23. Favoritos

Pantalla de estancias favoritas, en la imagen no se ha añadido ninguna estancia.

Índice de figuras

Índice de tablas