

Informe final del proyecto, grupo 2

Laboratorio de Ingeniería de Software, 2016



**Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza**

Adrian Moreno Jimeno - 631537
Javier Murillo Magdalena - 650448
Marcos Ruiz García - 648045
Jorge Berdún Udina - 591454

Índice

1 Introducción (pág. 2)

- 1.1 Descripción del proyecto (pág. 2)
- 1.2 Equipo de desarrollo (pág. 2)
- 1.3 Estructura del documento (pág. 3)

2 Producto (pág. 4)

- 2.1 Primer sprint (pág. 4)
 - 2.1.1 *Planificación de lanzamientos y objetivos* (pág. 4)
 - 2.1.2 *Análisis de riesgos* (pág. 5)
 - 2.1.3 *Requisitos* (pág. 6)
 - 2.1.4 *Bocetos GUI* (pág. 7)
 - 2.1.5 *Estado actual* (pág. 7)
 - 2.1.6 *Modelo de dominio* (pág. 9)
 - 2.1.7 *Vista de Módulos* (pág. 10)
 - 2.1.8 *Vista de Componentes y Conectores* (pág. 11)
 - 2.1.9 *Diagrama de Despliegue* (pág. 12)
 - 2.1.10 *Estrategia y herramientas de pruebas* (pág. 13)
 - 2.1.11 *Otras medidas usadas para mejorar la calidad del producto* (pág. 13)
- 2.2 Segundo sprint (pág. 13)
 - 2.2.1 *Planificación de lanzamientos y objetivos* (pág. 13)
 - 2.2.2 *Análisis de riesgos* (pág. 13)
 - 2.2.3 *Requisitos* (pág. 14)
 - 2.2.4 *Estado actual* (pág. 15)
 - 2.2.5 *Arquitectura del sistema* (pág. 19)
 - 2.2.6 *Diagrama de módulos* (pág. 19)
 - 2.2.7 *Vista de Componentes y Conectores* (pág. 24)
 - 2.2.8 *Diagrama de Despliegue* (pág. 26)
 - 2.2.9 *Esquema de base de datos* (pág. 27)
 - 2.2.10 *Estrategia y herramientas de pruebas* (pág. 28)

3 Proceso (pág. 29)

- 3.1 Primer sprint (pág. 29)
 - 3.1.1 *Estrategia de control de versiones* (pág. 29)
 - 3.1.2 *Esfuerzos por persona y actividad* (pág. 29)
 - 3.1.3 *Estrategia de mejora de procesos* (pág. 29)
 - 3.1.4 *Herramientas utilizadas* (pág. 29)
- 3.2 Segundo sprint (pág. 30)
 - 3.2.1 *Estrategia de control de versiones* (pág. 30)

4 Conclusiones (pág. 31)

- 4.1 Primer sprint (pág. 31)
 - 4.1.1 *Cumplimiento de objetivos de evaluación* (pág. 31)
- 4.2 Segundo sprint (pág. 33)
 - 4.2.1 *Cumplimiento de objetivos de evaluación* (pág. 33)

5 ANEXOS (pág. 35)

- 5.1 Manual de Usuario (pág. 35)
- 5.2 API server ofrecida (pág. 36)

1 Introducción

1.1 Descripción del proyecto

EINApIs es un sistema de consulta de información geoposicionada del campus universitario de la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza.

Con EINApIs se trata de acercar el concepto de smart-campus a los usuarios, se incluyen planos del campus y de sus edificios (plantas, aulas, laboratorios, cafeterías, comedores...), y servicios disponibles tales como el de máquinas expendededoras. Asimismo, se incluyen aspectos como horarios de apertura y de actividad docente

EINApIs contará además, gracias a una serie de sensores, de información extra de los diferentes espacios. Como por ejemplo, poder consultar el stock de las máquinas expendededoras, o la ocupación de los mismos.

Se ha optado por crear una aplicación orientada al día a día de los estudiantes y usuarios habituales del campus, ya que facilita el que alumnos recién llegados al campus se familiaricen antes, aparte de mostrar información que siempre resulta útil para alumnos veteranos y personal del centro.

El usuario podrá consultar toda esta información navegando a través de los mapas dispuestos, siendo capaz de seleccionar en ellos el espacio del cual desee obtener información.

1.2 Equipo de desarrollo

El equipo de desarrollo se encuentra formado por cuatro estudiantes de la Universidad de Zaragoza del Grado de Ingeniería Informática y especializados en Ingeniería de Software:

| | |
|---------------------------------|----------------------------------|
| Adrian Moreno Jimeno | Scrum master/ Desarrollador |
| Javier Murillo Magdalena | Dueño de producto/ Desarrollador |
| Jorge Berdún Udina | Desarrollador |
| Marcos Ruiz García | Desarrollador |

1.3 Estructura del documento

A continuación pasamos a detallar la estructura de esta documentación a partir de esta sección de Introducción.

Nótese que en las secciones que hagan referencia al segundo sprint solo se explicarán los cambios o novedades que haya respecto al primer sprint, por lo que si una sección sólo aparece en el primer sprint es porque esta sección no presenta ninguna novedad que merezca la pena mencionar en el segundo. Toda nueva sección o apartado que encontraremos en el segundo sprint y no este en el primero es debido a una nueva implementación desarrollada en el segundo sprint.

Además todas las secciones se dividirán entre información relativa al primer y segundo sprint.

- **2 Producto:** Información relacionada con nuestro proyecto, su diseño, arquitectura, requisitos y distintas vistas que ayudan a comprender las funcionalidades de nuestra aplicación en su totalidad
 - Planificación de lanzamientos y objetivos
 - Análisis de riesgos: Posibles riesgos a la hora de desarrollar nuestro producto.
 - Requisitos: Requisitos funcionales de nuestra aplicación
 - Bocetos GUI
 - Estado actual (Funcionalidad implementada, Interfaz de usuario Usabilidad, Problemas conocidos)
 - Modelo de dominio
 - Vista de Módulos
 - Vista de Componentes y Conectores
 - Diagrama de Despliegue
 - Estrategia y herramientas de pruebas
 - Otras medidas usadas para mejorar la calidad del producto
- **3 Proceso:** Estrategias y metodologías seguidas para desarrollar cada uno de los dos sprints.
 - Estrategia de control de versiones.
 - Esfuerzos por persona y actividad.
 - Estrategia de mejora de procesos.
- **4 Conclusiones:** Opiniones e ideas finales sobre el proyecto, así como cantidad de objetivos alcanzados y evaluación de los mismo.
- **5 ANEXOS:** Información extra relacionada con nuestro proyecto.
 - Manual de usuario: Breve manual de usuario sobre de manejo de nuestra aplicación
 - API server ofrecida con las operaciones disponibles y ejemplos de su invocación.

2 Producto

2.1 Primer sprint

2.1.1 Planificación de lanzamientos y objetivos

A continuación se presenta una lista con los hitos propuestos para el primer sprint:

- 15-02-2016. Lanzamiento de proyecto
- 29-02-2016. Primer diseño arquitectural y GUI
- 14-02-2016. Presentación de la primera iteración.
- 15-02-2016. Entrega del informe de la primera iteración.

Los **objetivos** a cumplir por el equipo de desarrollo son los siguientes:

| |
|--|
| <ul style="list-style-type: none">• El código y la documentación del proyecto se alojan en GitHub. Se trabaja de forma habitual contra Git |
| <ul style="list-style-type: none">• Compilación y gestión de dependencias (p.ej. descarga de bibliotecas externas) están basada en scripts (Gradle, Maven...) |
| <ul style="list-style-type: none">• Se llevará un control de esfuerzos con las horas dedicadas por persona. Se trabaja un número de horas en el entorno de lo requerido para la asignatura (90 horas dedicadas al “trabajo en grupo” por persona). Se entregará un resumen cada dos semanas |
| <ul style="list-style-type: none">• La aplicación cumple adecuadamente con sus requisitos |
| <ul style="list-style-type: none">• La documentación arquitectural es la adecuada al momento del proyecto, refleja fielmente el sistema, e incluye al menos tres vistas: módulos, componentes-y-conectores, y despliegue del sistema |
| <ul style="list-style-type: none">• La arquitectura del sistema es por capas. |
| <ul style="list-style-type: none">• Se usan adecuadamente estos conceptos de diseño dirigido por el dominio: entidades, objetos valor, agregados, factorías y repositorios• Se usan adecuadamente en el código y se reflejan adecuadamente en la documentación arquitectural |
| <ul style="list-style-type: none">• Se ha puesto en marcha y se usa un servicio de mapas tipo WMS con los edificios disponibles del campus Río Ebro. Los mapas de este servicio se superponen en el cliente sobre otro servicio externo (p.ej. Open Street Map) que proporcione un mapa de la zona |
| <ul style="list-style-type: none">• Cobertura de tests automáticos de al menos el 25% del código (unitarios y/o de integración) |

- La documentación arquitectural incluye una discusión adecuada sobre razones arquitecturales
- El modelo de dominio utiliza adecuadamente estos conceptos de diseño (dirigido por el dominio): servicios, paquetes, interfaces reveladoras, aserciones, funciones libres de efectos secundarios
- Se usan adecuadamente en el código y su uso se refleja adecuadamente en la documentación
- El estilo cartográfico de los edificios en el servicio de tipo WMS refleja el tipo de uso de cada espacio (por ejemplo, los laboratorios de un color, los despachos de otro etc.)
- El modelo de dominio incluye alguna restricción o especificación correctamente implementada, y ésta se utiliza en alguna funcionalidad de la aplicación
- La aplicación permite hacer algún tipo de consulta que podemos clasificar como “Análisis SIG” y esto se documenta adecuadamente, haciendo referencia a conceptos vistos en teoría
- El servicio de mapas WMS se ha teselado, y se usa así desde el cliente

2.1.2 Análisis de riesgos

Se presentan dos riesgos potenciales en el desarrollo de este proyecto: difícil elección de tecnología, y nuevos conceptos de diseño.

El equipo no tiene experiencia previa en un dominio como son la geolocalización y los modelos geográficos, por lo que la decisión de con qué tecnología se va a trabajar es compleja. Asimismo, el equipo no ha trabajado nunca con herramientas asociadas a este campo, como QGIS, PGAdmin, GeoKettle, GeoServer, o servicios WMS.

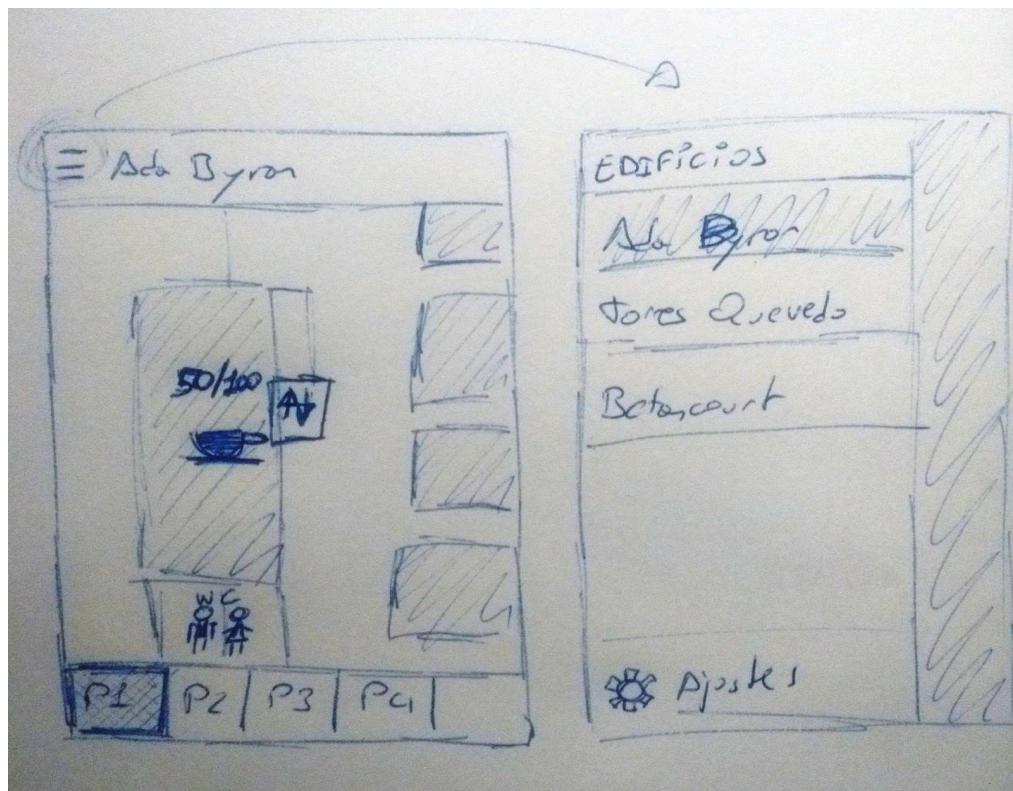
Por otra parte, no se tiene conocimiento previo sobre la teoría de Diseño Dirigido por el Dominio, por lo que la rápida asimilación de conceptos y técnicas para el desarrollo es clave para el cumplimiento de los objetivos propuestos en el apartado anterior.

Para la mitigación de estos riesgos, se ha propuesto una reunión en la que se discutirá y decidirá finalmente la tecnología a utilizar, buscando información sobre proyectos ya realizados sobre dicha tecnología. Además, el equipo trabajará de forma conjunta ejercicios en los que se incluyan aspectos relacionados sobre el diseño dirigido por el dominio.

2.1.3 Requisitos

| | |
|-------------|---|
| RF1 | El sistema mostrará el mapa del campus universitario EINA. |
| RF2 | El sistema deberá permitir al usuario desplazarse por el mapa horizontalmente. |
| RF3 | El sistema deberá permitir al usuario cambiar de planta mediante un control superpuesto. |
| RF4 | El sistema deberá permitir al usuario seleccionar el edificio del que quiere mostrar el mapa mediante un menú. |
| RF5 | El sistema se conectará a un servicio propio de WMS para gestionar el mapa del campus |
| RF6 | El sistema se conectará a un servicio externo para obtener el mapa de la zona |
| RF7 | La aplicación contará con una Base de datos o repositorio con todos los datos de las localizaciones y otras informaciones que se consideren necesarias para el correcto funcionamiento de la misma |
| RF8 | En el mapa de la aplicación se distinguen con un símbolo único las cafeterías, máquinas expendedoras, laboratorios, salas de estudio y bibliotecas de la planta y edificio seleccionados para su visualización. |
| RF9 | El sistema deberá permitir consultar información sobre las cafeterías del campus, incluyendo aforo máximo y ocupación actual. |
| RF10 | El sistema deberá permitir consultar información sobre las máquinas expendedoras del campus, incluyendo cola de personas, stock de los productos, y operabilidad. |
| RF11 | El sistema deberá permitir consultar información sobre los laboratorios del campus, incluyendo aforo máximo, ocupación actual y si se encuentra abierto o no. |
| RF12 | El sistema deberá permitir consultar información sobre las salas de estudio del campus, incluyendo aforo máximo, ocupación actual y si se encuentra abierto o no. |
| RF13 | La aplicación deberá permitir la creación de incidencias |
| RF14 | La aplicación deberá permitir enviar un email con la incidencia. |
| RF15 | Una incidencia consta de un título, una foto, una localización y una descripción |
| RF16 | La aplicación deberá permitir la consulta de las asignaturas impartidas en cada aula o laboratorio |
| RF17 | La aplicación deberá permitir visualizar una ruta desde dos localizaciones indicadas por el usuario |

2.1.4 Bocetos GUI



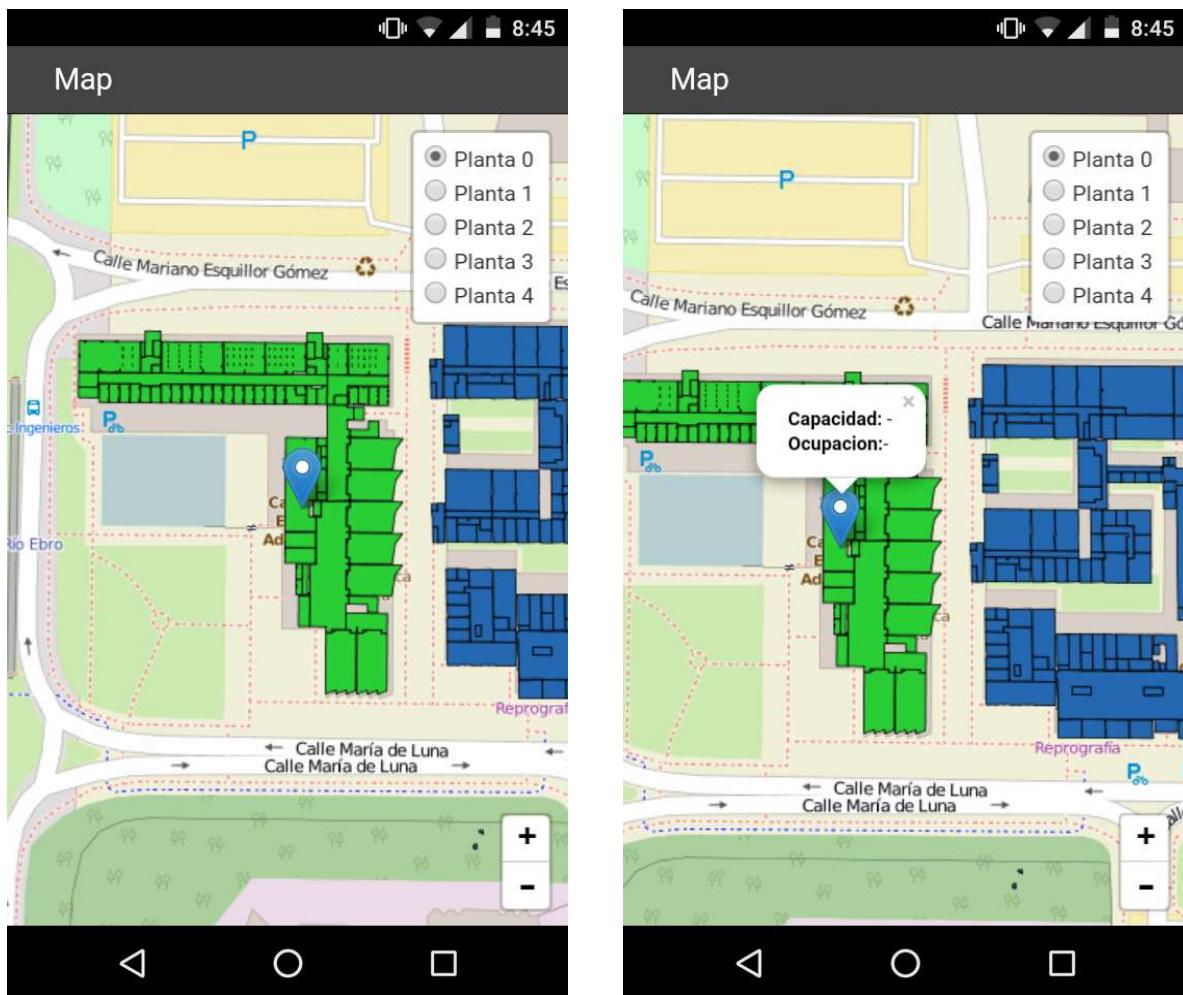
2.1.5 Estado actual

A. Funcionalidad implementada

Al final de esta primera iteración, la funcionalidad implementada es la siguiente:

- El sistema muestra un mapa de la zona haciendo uso de los servicios de OpenStreetMap.
- El sistema permite al usuario navegar a través del mismo.
- El sistema hace uso de servicios WMS propios para superponer los mapas de los edificios del campus.
- El sistema permite al usuario visualizar la plantada deseada de los edificios.
- Base de Datos con información de prueba sobre las cafeterías.

B. Interfaz de usuario



C. Usabilidad

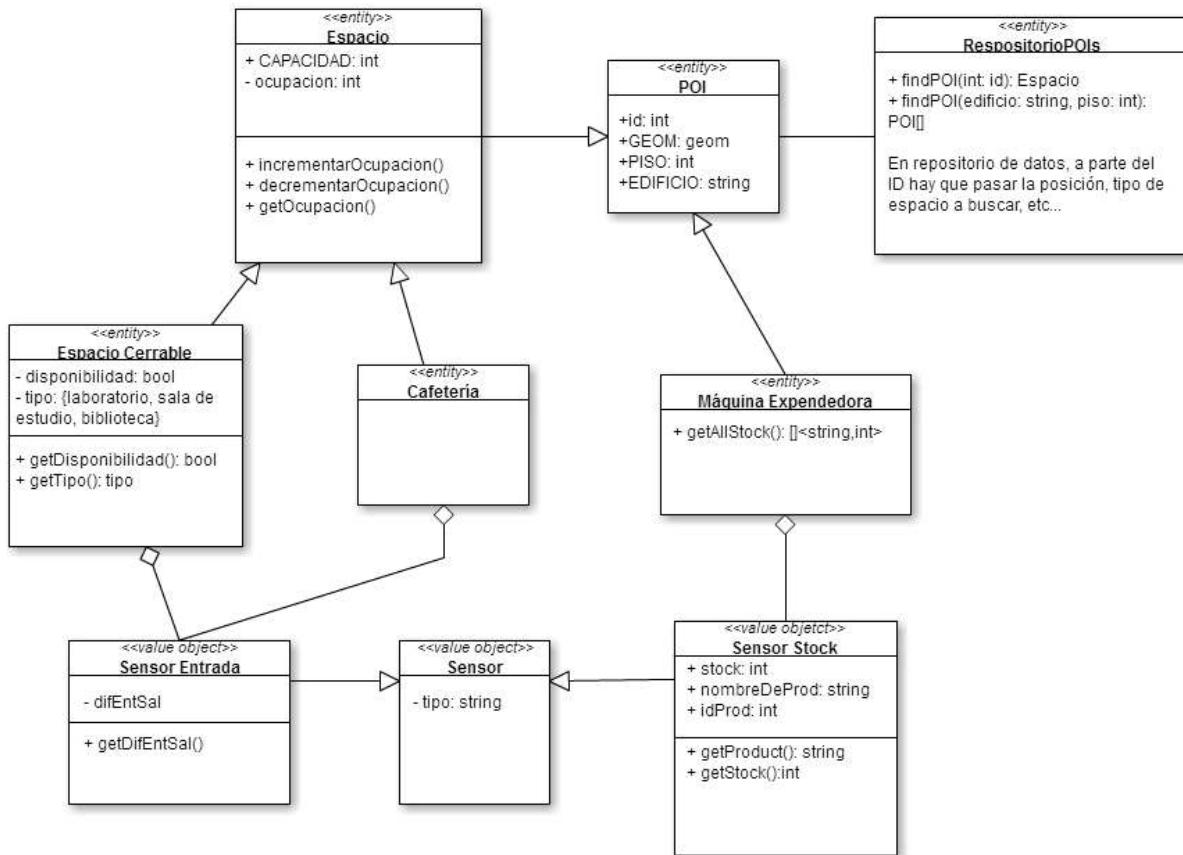
Se ha de destacar que, para la simplicidad de la interfaz de usuario, no se han dividido las plantas por edificio, sino que un cambio de planta afecta a los tres edificios del campus. Asimismo, la aplicación permite hacer zoom sobre la zona deseada.

D. Problemas conocidos

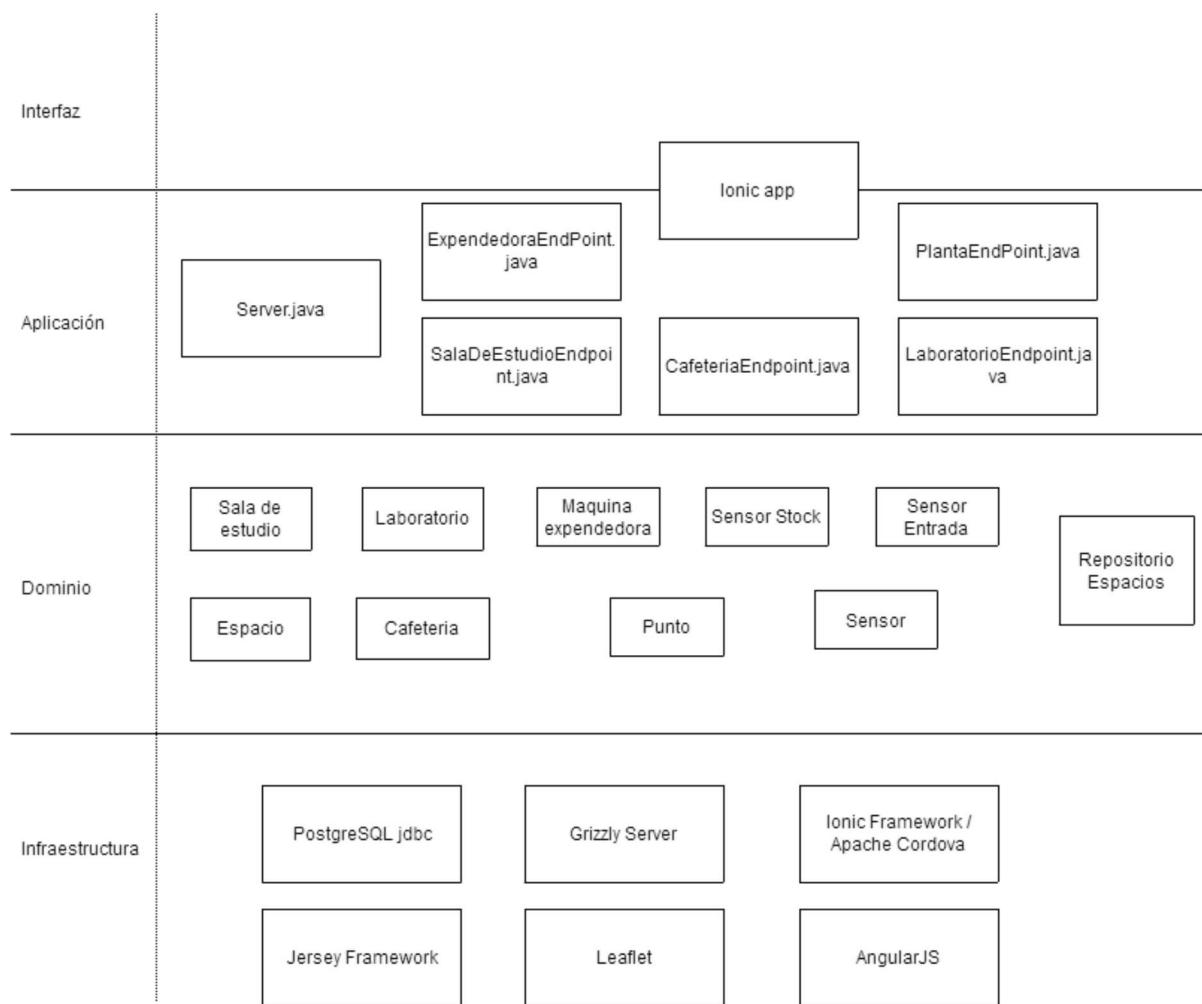
Se presenta una lista con los problemas conocidos hasta ahora:

- Botones de zoom no se comportan correctamente. El equipo de desarrollo plantea eliminarlos, dejando la opción de *pinch-to-zoom* para ampliar las zonas.
- El servicio de WMS no sirve datos a partir de cierto nivel de zoom. Actualmente, se ha solucionado mediante una capa de datos vectoriales. La solución definitiva pasa por revisar la configuración de GeoServer, para asegurar que los mapas se sirven a la escala requerida.

2.1.6 Modelo de dominio

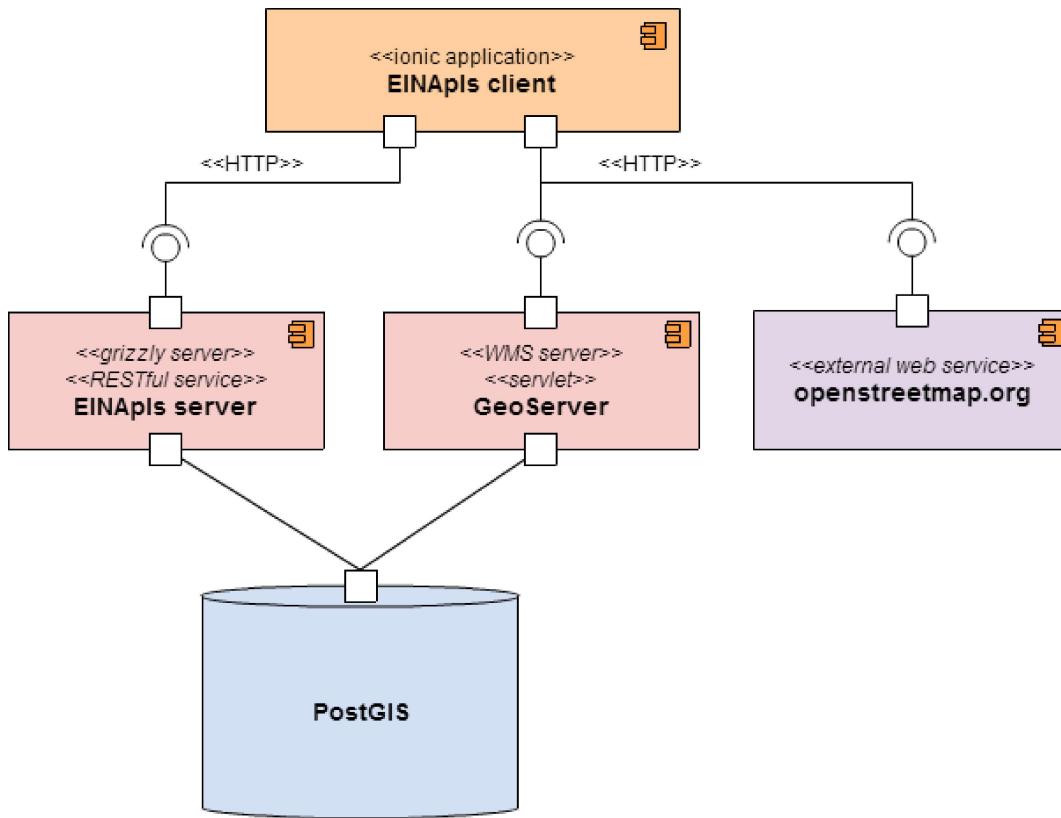


2.1.7 Vista de Módulos



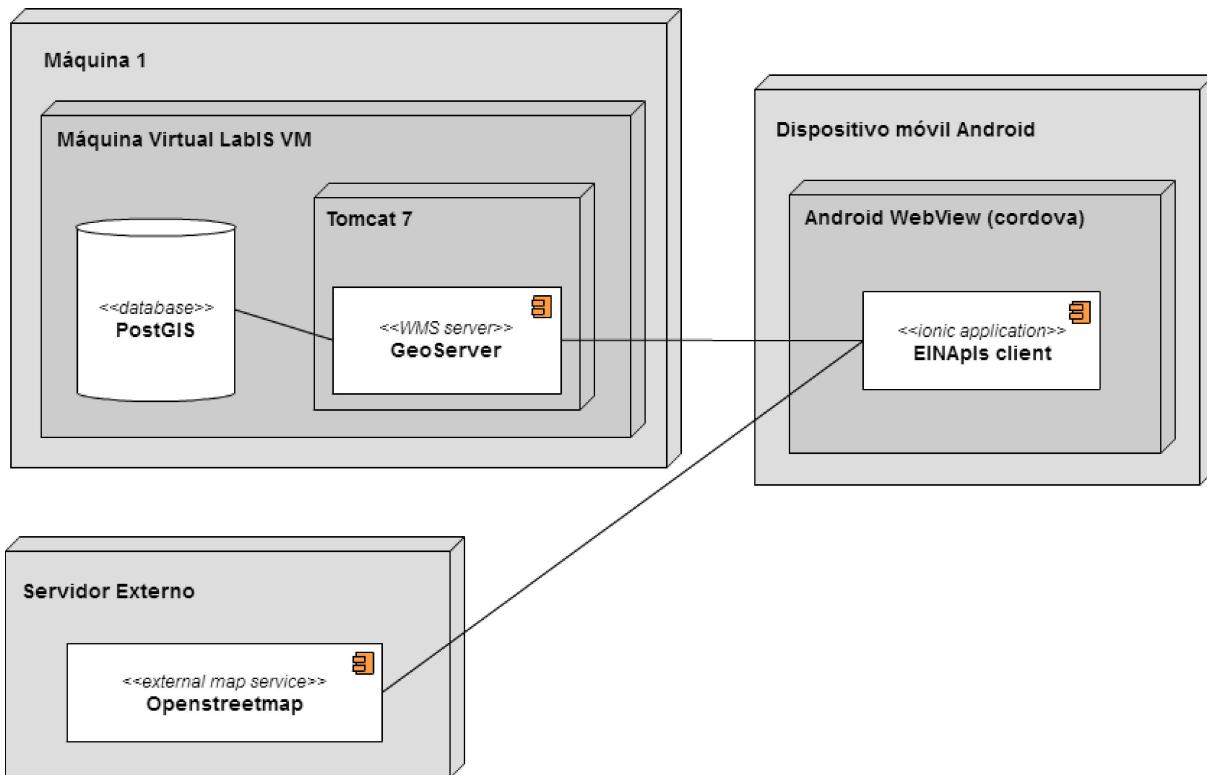
Actualmente, la aplicación del cliente no está bien estructurada. Su arquitectura no está bien definida en componentes, por lo que es difícil reflejar qué partes de la aplicación se encuentran en la capa de aplicación y qué partes se encuentran en la capa de interfaz. En la siguiente iteración se solucionará este problema, estructurando la aplicación correctamente mediante los componentes de AngularJS.

2.1.8 Vista de Componentes y Conectores



- El componente EINApis client posee un puerto para interactuar con servicios de mapas y otro para obtener la información de los sensores mediante una API REST.
- El componente GeoServer ofrece una interfaz la cual da servicio a las peticiones correspondientes de las diferentes plantas del campus.
- El componente externo openstreetmap.org ofrece una interfaz para la inclusión del mapa general de la zona mediante tiles.
- El componente EINApis server ofrece una interfaz de tipo REST sobre HTTP. Este componente tiene un puerto para conectarse a la base de datos de PostGIS, donde se almacenan los datos.
- En la base de datos PostGIS se almacenan los datos espaciales correspondientes a los mapas de nuestro sistema, tanto de los diferentes espacios del campus, como de los puntos de interés.

2.1.9 Diagrama de Despliegue



Se diferencian cuatro nodos en el diagrama de despliegue: Máquina 1, Máquina 2, Servidor Externo, y Dispositivo Android.

La máquina 1 contiene la máquina virtual sobre la cual se despliega la base de datos PostGIS (contenedora de los datos espaciales) y el servidor Tomcat 7, el cual contiene el componente GeoServer encargado de dar servicio de mapas WMS a nuestro sistema. Esta máquina virtual se basa en la máquina virtual proporcionada para realizar las prácticas. De esta forma, nos evitamos tener que instalar e integrar estos componentes desde cero, además de poder trabajar con QGIS directamente en esta misma máquina.

Las ventajas de tener estos componentes encapsulados en una máquina virtual es que pueden portarse y replicarse conservando todas las configuraciones. Como alternativa a este sistema se planteó desplegar el sistema en un entorno de cloud gratuito, pero se descartó porque esta opción era más sencilla. También se planteó hacer un despliegue en local, pero se descartó esta opción debido al coste que hubiera supuesto replicar la configuración en los equipos de todos los desarrolladores.

El servidor externo contiene el componente correspondiente a Open Street Map para la inclusión de mapas generales de la zona. Para este sprint se ha omitido el servidor de EINAPIs porque no se ha llegado a poner en funcionamiento (se ha incluido en el diagrama de despliegue del segundo sprint)

Finalmente, la aplicación cliente se ejecuta sobre el dispositivo Android, el cual contiene una WebView sobre la que se ejecutará la aplicación cliente. Esta WebView viene implementada por el framework cordova, sobre el cual está construida la aplicación del cliente.

Se ha elegido este framework porque el equipo tenía más experiencia trabajando con Angular que con Android. Además, Ionic permite compilar a distintas plataformas, entre ellas la web, lo que permite hacer el desarrollo mucho más ágil.

2.1.10 Estrategia y herramientas de pruebas

En esta iteración se ha implementado la infraestructura de pruebas para el servidor mediante jUnit y gradle. Sin embargo, puesto que se trata de la primera iteración y debido a la falta de tiempo, no se han realizado pruebas automáticas.

2.1.11 Otras medidas usadas para mejorar la calidad del producto

En el lado del servidor, se ha hecho uso de Gradle para la automatización de la construcción del código y gestión de dependencias, facilitando las tareas de compilación, pruebas, y despliegue. En el lado del cliente, se ha hecho uso del framework ionic para la gestión de dependencias, compilación y despliegue. Este framework actúa como envoltorio de varias herramientas situadas por debajo para estos mismos propósitos, tales como npm, gulp, bower y cordova.

El despliegue de la aplicación cliente se ha hecho mediante Ionic View, que permite desplegar las aplicaciones de ionic y ejecutarlas en un dispositivo móvil sin necesidad de instalarlas como paquetes independientes. Esta funcionalidad nos permite probar la aplicación en un dispositivo real de forma rápida y cómoda, pudiendo desplegar la última versión a todos los dispositivos de prueba mediante un sólo comando.

2.2 Segundo sprint

2.2.1 Planificación de lanzamientos y objetivos

A continuación se presenta una lista con los hitos alcanzados:

- 18-04-2016. Reunión de seguimiento del proyecto
- 26-05-2016. Presentación del resultado final
- 28-05-2016. Entrega del informe final.

En el segundo sprint se aspiran a cumplir los siguientes **objetivos**, aparte de los ya nombrados en el sprint 1:

- La arquitectura del sistema es hexagonal
 - Se implementa así y se refleja adecuadamente en la documentación

2.2.2 Análisis de riesgos

Para el desarrollo de este segundo sprint, el equipo ha tratado de solucionar los riesgos del anterior sprint. En este segundo sprint, se han detectado dos riesgos: poca experiencia con las herramientas GIS y falta de tiempo por parte del equipo para implementar funcionalidades.

Aunque las herramientas de trabajo ya no son desconocidas para el equipo, este sigue teniendo poca experiencia trabajando con ellas, por lo que pueden surgir problemas que supongan un alto coste en horas para el equipo. Es por esto que debe documentarse todo el proceso de trabajo con estas herramientas al máximo, para poder replicar los procedimientos exitosos y para evitar cometer el mismo error varias veces.

Además, el equipo se encuentra realizando otros trabajos para otras asignaturas, todos ellos con fechas de entrega muy próximas. Esto puede implicar que la planificación del proyecto se descuadre y al final el equipo no disponga de tiempo suficiente para completar el proyecto. Para mitigar este riesgo, es necesario que los miembros del equipo se planifiquen adecuadamente, tanto como equipo como individualmente.

2.2.3 Requisitos

| | |
|-------------|---|
| RF1 | El sistema mostrará el mapa del campus universitario EINA. |
| RF2 | El sistema deberá permitir al usuario desplazarse por el mapa horizontalmente. |
| RF3 | El sistema deberá permitir al usuario cambiar de planta mediante un control superpuesto. |
| RF4 | El sistema deberá permitir al usuario seleccionar el edificio del que quiere mostrar el mapa mediante un menú. |
| RF5 | El sistema se conectará a un servicio propio de WMS para gestionar el mapa del campus |
| RF6 | El sistema se conectará a un servicio externo WMS para obtener el mapa de la zona |
| RF7 | La aplicación contará con una Base de datos o repositorio con todos los datos de las localizaciones y otras informaciones que se consideren necesarias para el correcto funcionamiento de la misma |
| RF8 | En el mapa de la aplicación se distinguen con un símbolo único las cafeterías, máquinas expendedoras, laboratorios, salas de estudio y bibliotecas de la planta y edificio seleccionados para su visualización. |
| RF9 | El sistema deberá permitir consultar información sobre las cafeterías del campus, incluyendo aforo máximo y ocupación actual. |
| RF10 | El sistema deberá permitir consultar información sobre las máquinas expendedoras del campus, incluyendo cola de personas, stock de los productos, y operabilidad. |
| RF11 | El sistema deberá permitir consultar información sobre los laboratorios del campus, incluyendo aforo máximo, ocupación actual y si se encuentra abierto o no. |
| RF12 | El sistema deberá permitir consultar información sobre las salas de estudio/bibliotecas del campus, incluyendo aforo máximo, ocupación actual y si se encuentra abierto o no. |

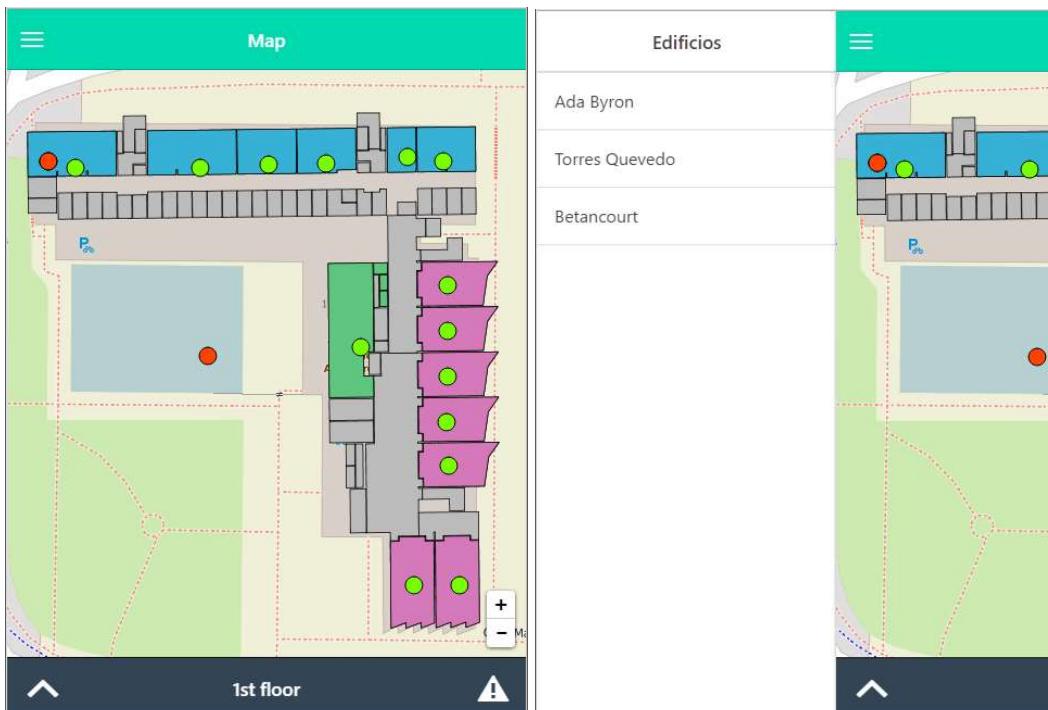
| | |
|-------------|--|
| RF13 | La aplicación deberá permitir la creación de incidencias |
| RF14 | Una incidencia consta de un título, un estado [abierta aceptada cerrada], una foto, una localización y una descripción |
| RF15 | Los Administradores podrán consultar todas las incidencias en una lista. |
| RF16 | Solo el Administrador podrá modificar el estado de una incidencia |
| RF17 | Los Usuarios solo podrán visualizar en el mapa aquellas incidencias cuyo estado sea el de “aceptada”. |
| RF18 | La aplicación deberá permitir la consulta de las asignaturas impartidas en cada aula o laboratorio |
| RF19 | La aplicación deberá permitir visualizar una ruta desde dos localizaciones indicadas por el usuario |

2.2.4 Estado actual

A Funcionalidad implementada

Al final de esta segunda iteración, se han implementado todos los requisitos que se habían propuesto, a excepción de los datos de los horarios, los cuales han sido simulados en vez de trabajar directamente con un servicio externo y no han sido implementados finalmente en la interfaz.

B. Interfaz de usuario

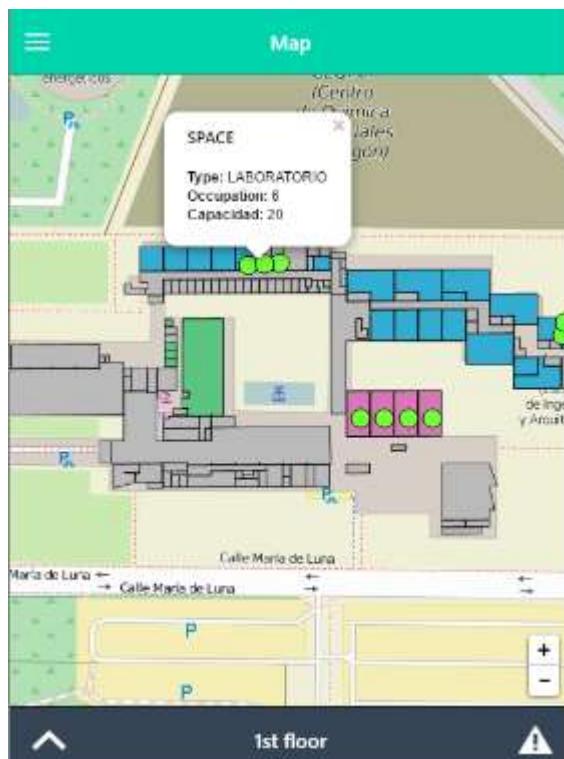




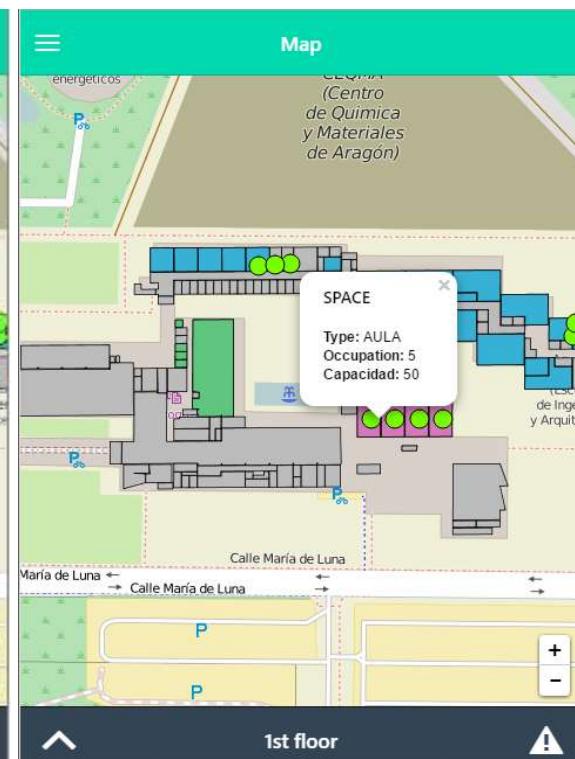
Planta principal del edificio Torres Quevedo



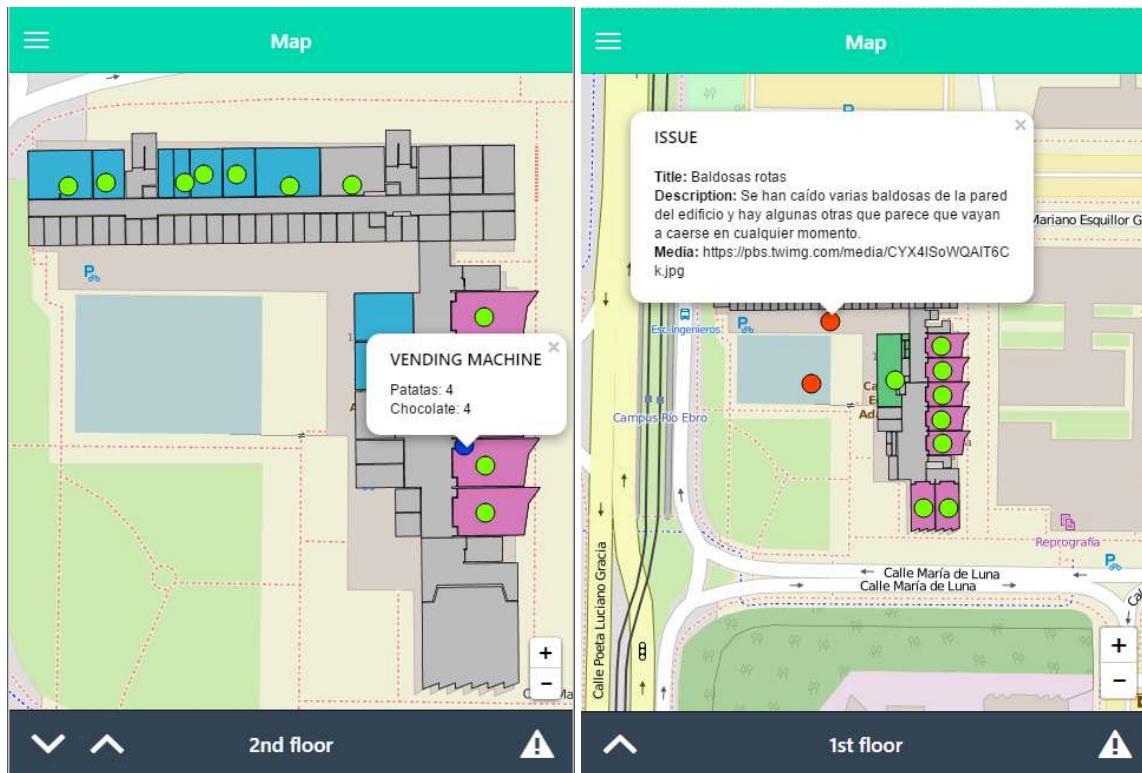
Planta primera del edificio Torres Quevedo



Información sobre un laboratorio



Información sobre un aula

*Información sobre una máquina expendedora**Información sobre una incidencia*

The image consists of two side-by-side screenshots of a mobile application. Both screens have a green header bar with the word 'Map' in white text.

Left Screenshot: This shows a detailed floor plan of a building. Several green circular markers are placed on the map. At the bottom, there is a large orange button with the text 'Create issue' in white.

Right Screenshot: This shows a modal dialog titled 'Create issue'. It has a checkmark icon on the left and an 'X' icon on the right. The form fields are as follows:

- Title:** (empty input field)
- Description:** (placeholder text: 'Write a description for the issue')
- Media:** (empty input field)

*Modo de creación de incidencia**Menú de creación de incidencia*

C. Usabilidad

La aplicación tiene un menú lateral para cambiar de un edificio a otro limitando la navegación por el mapa a dicho edificio. Con la barra inferior se puede tanto cambiar de mapa como añadir incidencias de una manera sencilla. Se han solucionado los problemas con los controles del mapa.

Se han coloreado las distintas zonas de interés en el mapa en función de si se trata de aulas, cafeterías, bibliotecas, etc. Hay tres tipos de puntos con los que se puede interactuar en el mapa: espacios (verdes), máquinas expendedoras (azules) e incidencias (rojos).

Para crear incidencias se selecciona el modo de crear incidencias (botón inferior a la derecha) y se toca en el mapa dónde se quiere colocar la incidencia. En ese momento se despliega un formulario para que el usuario pueda introducir sus datos y guardar la incidencia.

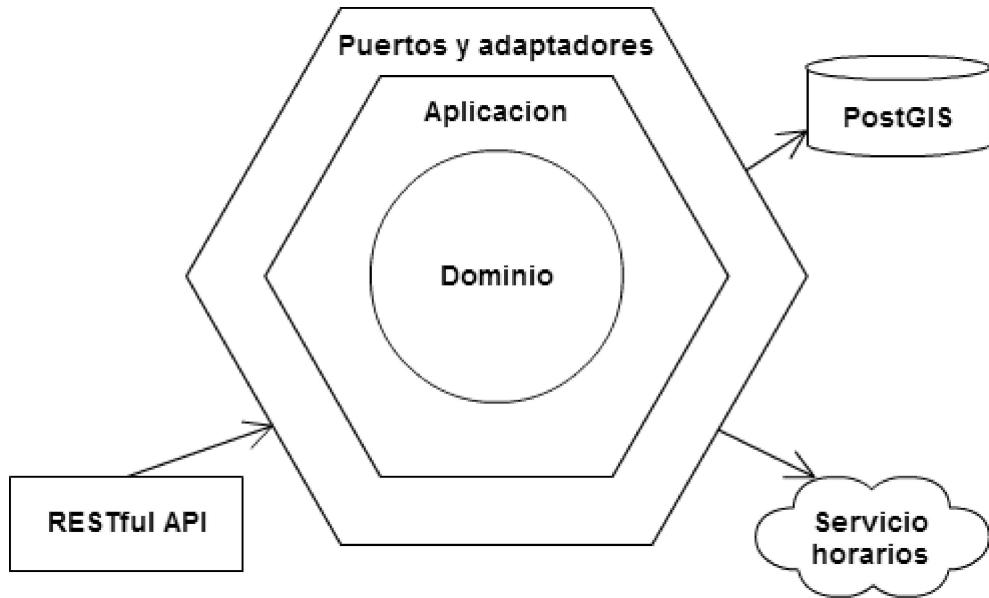
D. Problemas conocidos

Cuando se selecciona un punto situado muy cerca del margen del mapa y aparece el popup con la información adicional, este no se muestra en su totalidad porque se sale de las coordenadas establecidas como límite del mapa. En el caso de los puntos situados en la parte superior, además, el control para cerrar el popup queda fuera del alcance del usuario.

Los sensores a veces lanzan excepciones. Se cree que, por la numerosa cantidad de sensores que hay (uno por espacio) si muchos realizan peticiones a la vez sobre la base de datos está rechaza algunas peticiones.

Al realizar la transformación de coordenadas con los mapas proporcionados, QGIS elimina ciertos elementos como por ejemplo algunos pasillos.

2.2.5 Arquitectura del sistema

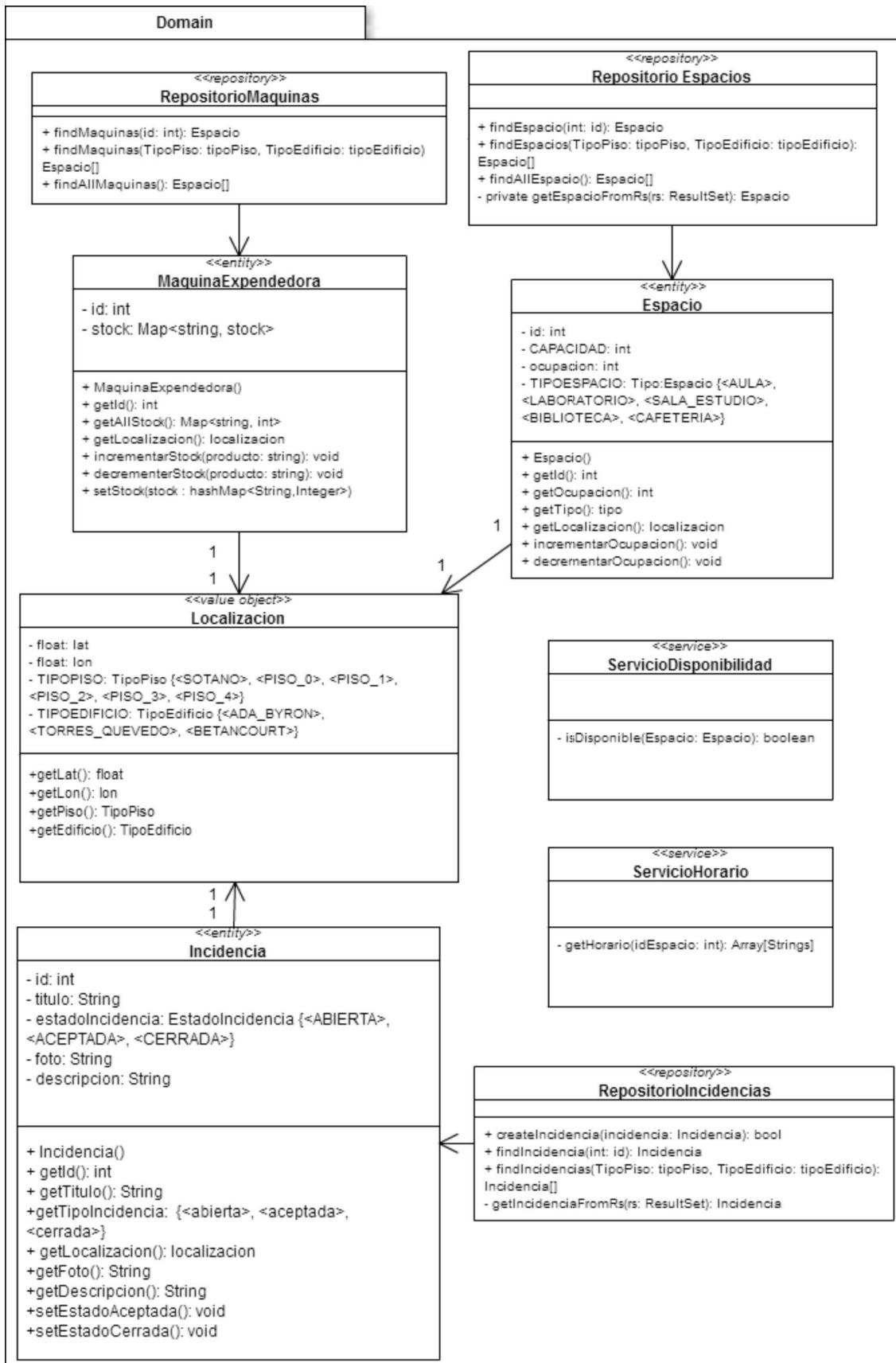


Se ha optado por seguir una arquitectura hexagonal, reemplazando así el anterior modelo de capas. En la arquitectura hexagonal, todos los detalles dependientes de implementación se concentran en el anillo exterior, quedando tanto entradas como salidas al mismo nivel. Esto nos aporta la ventaja de que el modelo de dominio queda completamente aislado de las implementaciones, que pueden modificarse, añadirse o eliminarse sin necesidad de modificar el dominio.

2.2.6 Diagrama de módulos

El anillo exterior de la arquitectura hexagonal se corresponde con el paquete de puertos y adaptadores, mientras que en el interior del “hexágono” nos encontramos con los paquetes de aplicación y dominio.

A. Modelo de dominio



A diferencia del sprint anterior, en este sprint se ha optado por mantener el diagrama de dominio sencillo, eliminando herencias y asociaciones. En lugar de hacer que todos los elementos que tuvieran una localización descendieran de un objeto POI, se ha creado un objeto valor Localizacion y se ha añadido a cada una de las entidades que necesitan estar localizadas.

Se valoró integrar el servicio Disponibilidad en la propia entidad Espacio, pero finalmente se decidió mantenerlo como servicio, porque de esta forma, permite implementar reglas más complejas para el cálculo de la disponibilidad que no sólo dependan de los propios espacios (horarios de examen, bajas de profesores, etc.).

Igualmente, el servicio Horario también se ha desacoplado de las entidades Espacio porque es necesario obtener información externa para poder calcular los horarios. También se planteó crear un objeto valor para representar los horarios, pero finalmente descartamos esta opción porque el horario no se utiliza dentro del dominio, sino fuera de este, por lo que, en todo caso, lo único que hubiera sido necesario implementar es un Data Transfer Object.

También se valoró incluir una entidad Planta o Planta de edificio que agrupara todas las entidades de una misma planta o edificio, pero finalmente se descartó debido a que introducía complejidad adicional al dominio sin ofrecer una ventaja significativa.

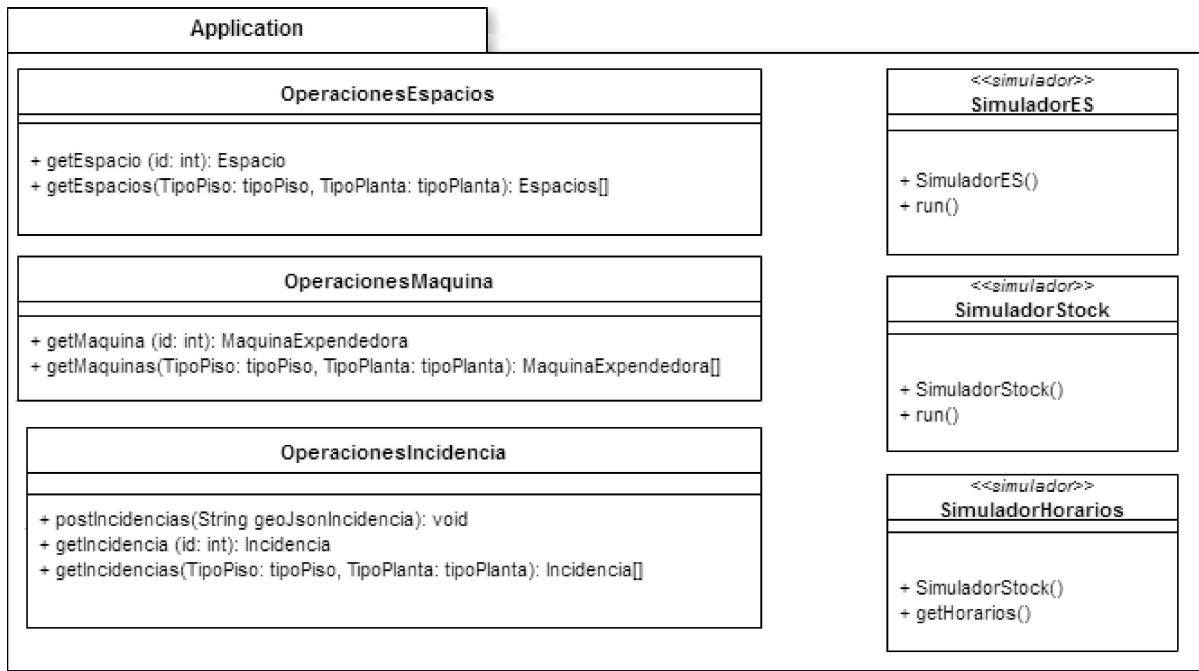
Finalmente, se decidió eliminar por completo los sensores del dominio, ya que estaban siendo utilizados meramente como interfaces y sólo añadían complejidad innecesaria al dominio. Se ha optado por que los simuladores interactúen directamente con los objetos del dominio desde la capa de aplicación.

Las incidencias cuentan con 3 estados:

- **ABIERTA:** Estado inicial. La incidencia acaba de ser enviada al sistema por un Usuario y aún no ha sido consultada por ningún Administrador.
- **ACEPTADA:** La incidencia ha sido aceptada como válida por un Administrador, y será visible en el mapa para el resto de usuarios hasta que algún Administrador la marque como Cerrada.
- **CERRADA:** Fase final de una incidencia que indique ya ha sido solucionada o que no se debe de mostrar en el mapa.

Cuando un usuario sube una incidencia, los administradores pueden consultarla y decidir si se trata de una incidencia válida y de interés público, de ser así la marcarán como “aceptada” y se mostrará al resto de usuarios de la aplicación. Por contra si no es una incidencia válida, será marcada como “cerrada”. Dada una incidencia en estado de “aceptada”, un administrador la marcará como “cerrada” una vez que se haya solventado.

B. Aplicación

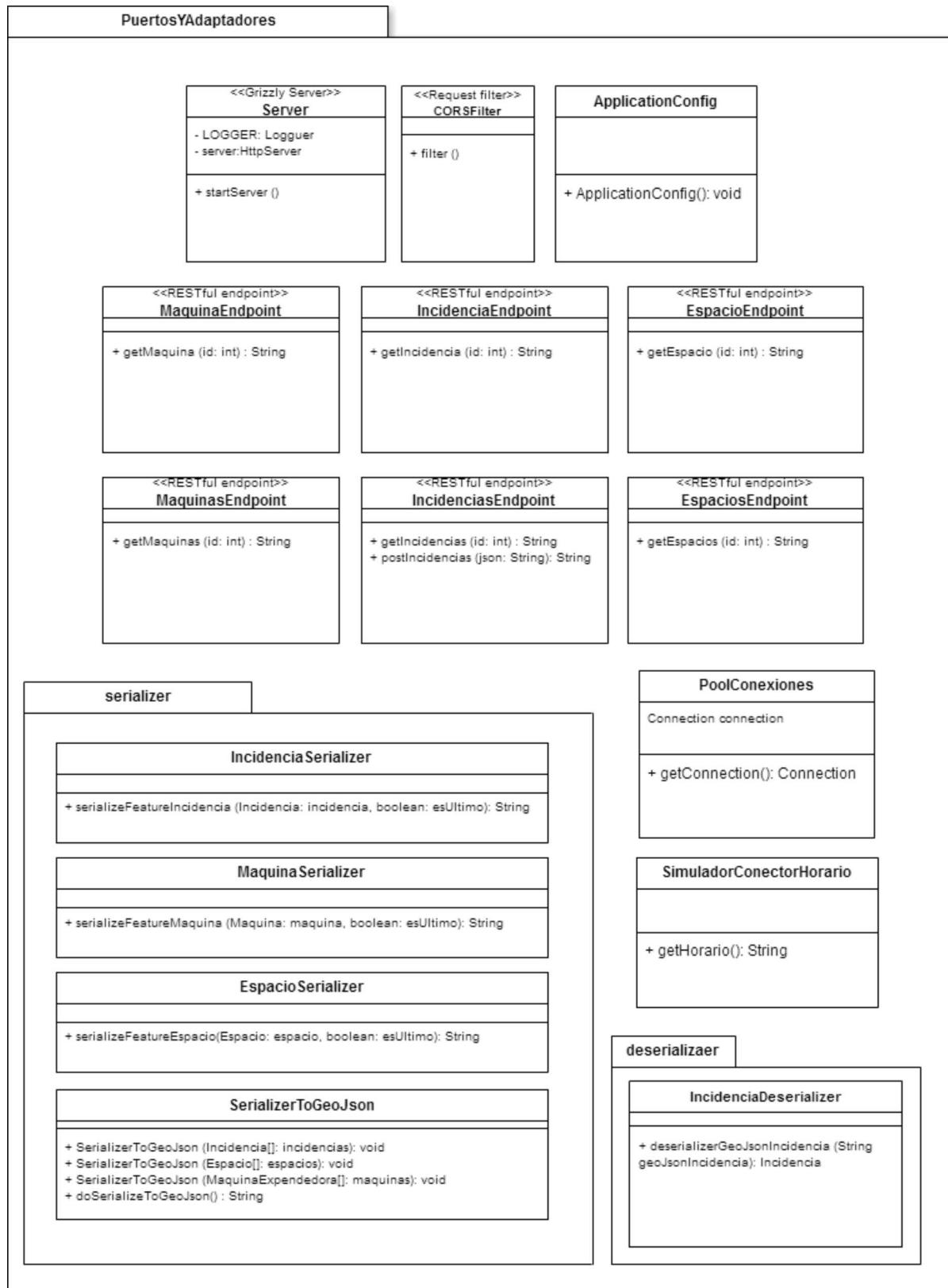


En el paquete de aplicación se encuentran todas las operaciones mediante las cuales se coordinarán los elementos del dominio. Estas operaciones serán llamadas desde el anillo exterior, cuando se reciban llamadas a la API REST (o a cualquier otro puerto de entrada que esté implementado, en este caso sólo está la API REST).

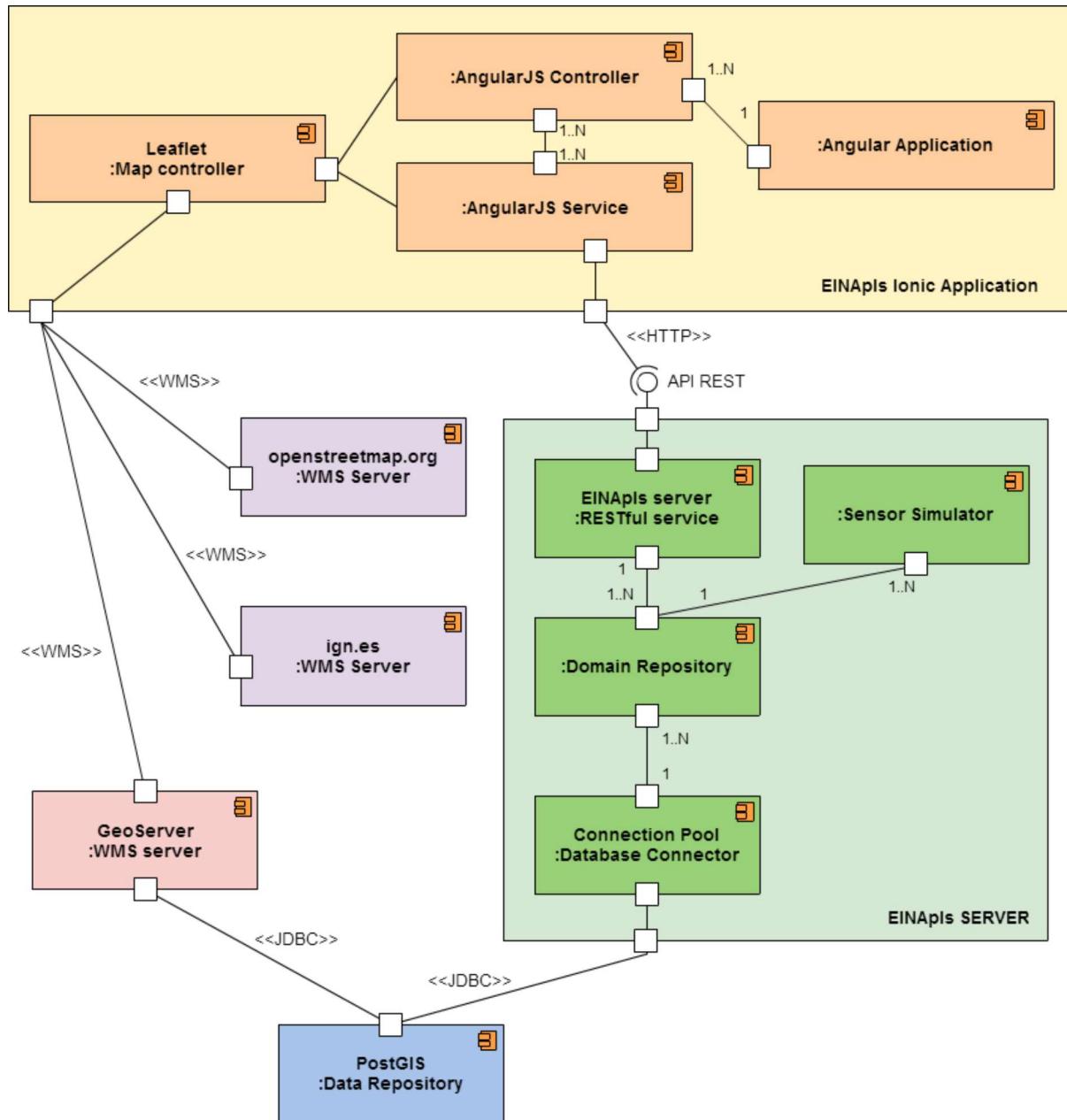
A parte de estas operaciones, también encontramos los simuladores de entrada/salida y de consumo de las máquinas. Como ya se ha comentado anteriormente, se eliminaron los sensores del dominio porque añadían complejidad excesiva sin añadir expresividad al dominio, por lo que se movieron a la capa de aplicación. En este caso, lo más correcto hubiera sido implementar en la capa de aplicación las operaciones que realizarían los supuestos sensores y, en la capa de puertos y adaptadores, colocar las correspondientes implementaciones (simuladores, sensores reales que se comunican mediante algún tipo de API o de mensajería, etc.), pero por falta de tiempo para esto se decidió implementar los simuladores directamente en la capa de aplicación, ya que interactúan directamente con la capa de dominio.

C. Puertos y adaptadores

En este paquete encontramos todos los detalles de implementación, desde la implementación del adaptador que ofrece la interfaz REST (el servidor grizzly que hace uso de jax-rs) hasta la conexión con la base de datos o la serialización de elementos a GeoJSON. Las ventajas de la arquitectura hexagonal es que, para modificar una tecnología, sólo necesitamos modificar esta capa. Por ejemplo, si quisiéramos ofrecer una API mediante SOAP, sólo necesitaríamos añadir un adaptador capaz de escuchar y responder peticiones SOAP (probablemente autogenerado mediante una especificación) y un serializador a XML. De la misma forma, podríamos ofrecer una interfaz web basada en plantillas (JSP), cambiar la base de datos, etc.



2.2.7 Vista de Componentes y Conectores



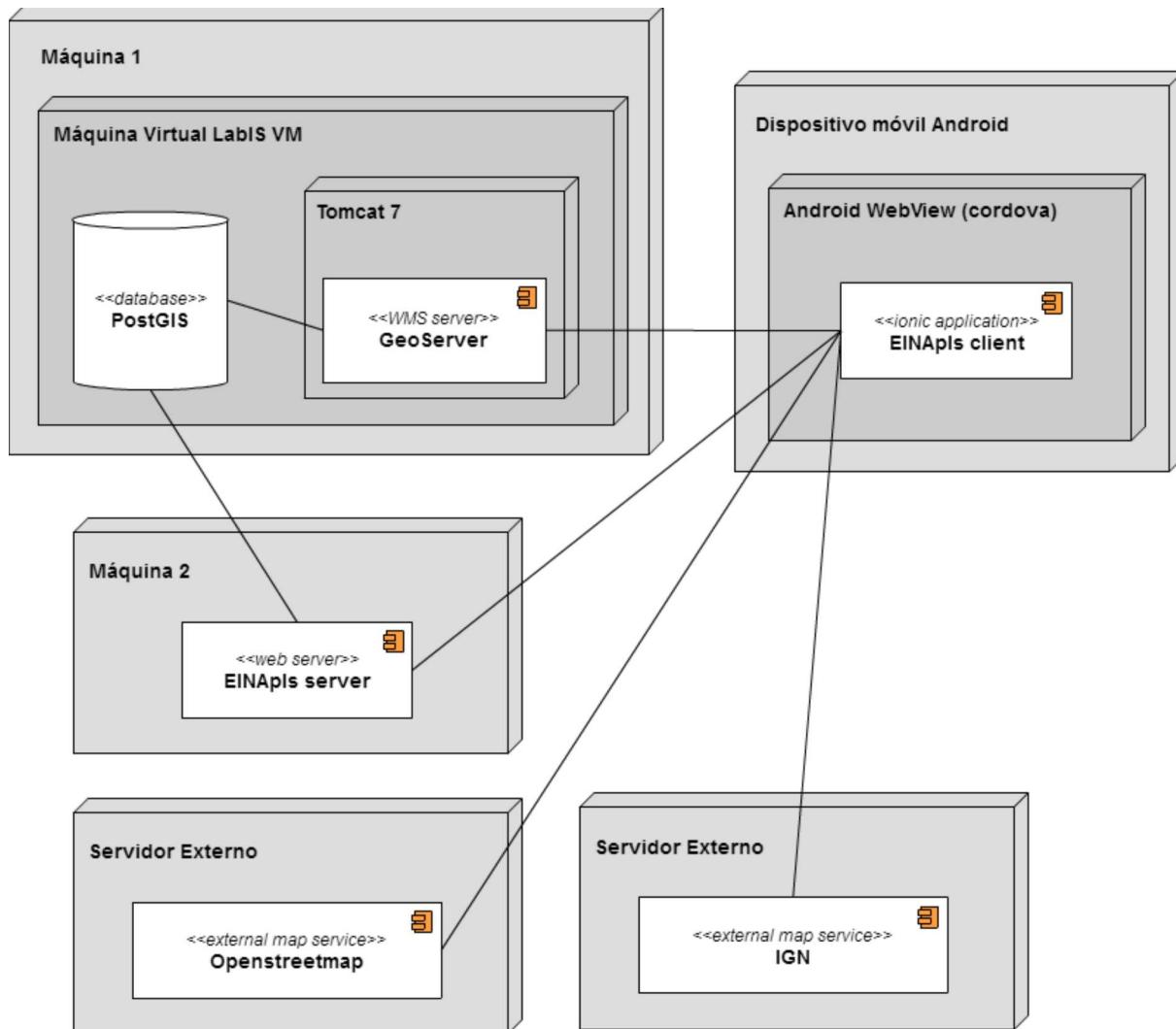
En la parte del cliente, se ha encapsulado todo el acceso a la API REST en los servicios de Angular, que son llamados desde los controladores. Esto es así para respetar el patrón MVC que propone angular. Las llamadas a la librería de Leaflet, que es la que proporciona el controlador para el mapa y las funciones para encapsular el acceso a los servidores WMS, se hacen tanto desde servicios como desde controladores de Angular. Esto es porque parte de la librería se comporta como controlador (nos ofrece herramientas para crear controles e interactuar con el mapa) y parte funciona como modelo (nos permite crear colecciones de datos, obtener información de servicios WMS, etc).

Una alternativa podría haber sido crear un servicio que encapsulara toda la interacción con la API. Esto nos proporcionaría mucha flexibilidad a la hora de cambiar de biblioteca de manejo de mapas (p. ej., usando la biblioteca de Google Maps), pero añadiría muchísima complejidad que, teniendo en cuenta el tiempo del que dispone el grupo y que no hay intención de cambiar de proveedor de mapas a corto plazo, no merece la pena.

En el lado del servidor, se puede observar que los simuladores corren independientemente del servidor REST. Esto se ha hecho para simplificar el desarrollo, pero otra alternativa podría haber sido crear una interfaz (mediante REST o mediante un protocolo de paso de mensajes, como por ejemplo RabbitMQ o ActiveMQ) para acceder al servidor de forma externa y que un simulador, ejecutándose en otra máquina, se conectara contra esa API y realizará las modificaciones necesarias.

Tanto los simuladores como el servidor RESTful se conectan a los repositorios del dominio, de donde obtienen los datos con los que luego trabajarán. Estos datos se obtienen de la base de datos mediante una conexión, que la dispensará el pool de conexiones. Estos repositorios devolverán entidades y objetos valor, que luego podrán pasárseles a los distintos servicios y serializadores.

2.2.8 Diagrama de Despliegue



Se ha expandido el diagrama de despliegue con respecto al primer sprint para reflejar el estado actual de la aplicación.

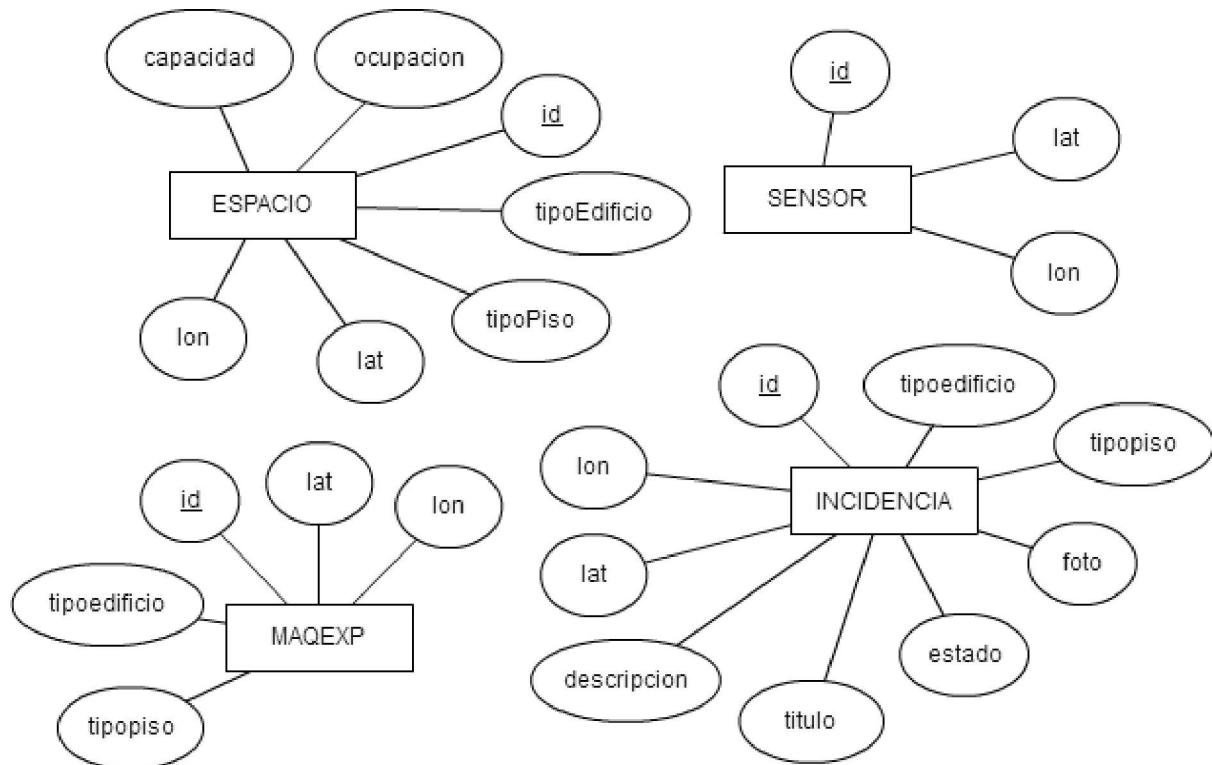
La máquina virtual sobre la que se despliegan PostGIS y GeoServer ha sido modificada, añadiendo un disco duro adicional en el que se guarda una nueva instancia de la base de datos. En este segundo sprint, como alternativa a este sistema se planteó hacer uso de docker, que hubiera sido más eficiente, pero se encontraron problemas a la hora de acceder a los contenedores de docker desde el exterior de la máquina anfitrión, por lo que no hubiera sido posible desacoplar el servidor EINApis server. Este problema puede solucionarse, pero hubiera requerido invertir demasiadas horas.

La máquina 2 contiene el servidor de nuestra aplicación, que es el que realmente encapsula el dominio y la aplicación. Este servidor ofrece una interfaz REST para acceder al dominio, y se conecta a la base de datos PostGIS de la máquina virtual. Este servidor no conserva estado, por lo que puede ser replicado en distintas máquinas (el estado se guarda en la base de datos). La limitación a la hora de replicar viene dada por las conexiones que

puede soportar la base de datos. Alternativamente, se planteó integrar el servidor en la misma máquina virtual de prácticas, bajo tomcat, pero se optó por esta alternativa porque ofrece más flexibilidad a la hora de desplegar este servidor fuera de la máquina virtual.

A parte del servidor de openstreetmap, se ha añadido el servidor de ign como proveedor de WMS externo. Esto es porque, aunque el mapa de openstreetmap es muy detallado, tiene el nivel de zoom limitado hasta 19. Se ha optado por utilizar el servicio de ign para cargar el entorno en niveles de zoom mayores, aunque no muestre tantos detalles como openstreetmap. La aplicación cliente sigue funcionando igual que en el primer sprint.

2.2.9 Esquema de base de datos



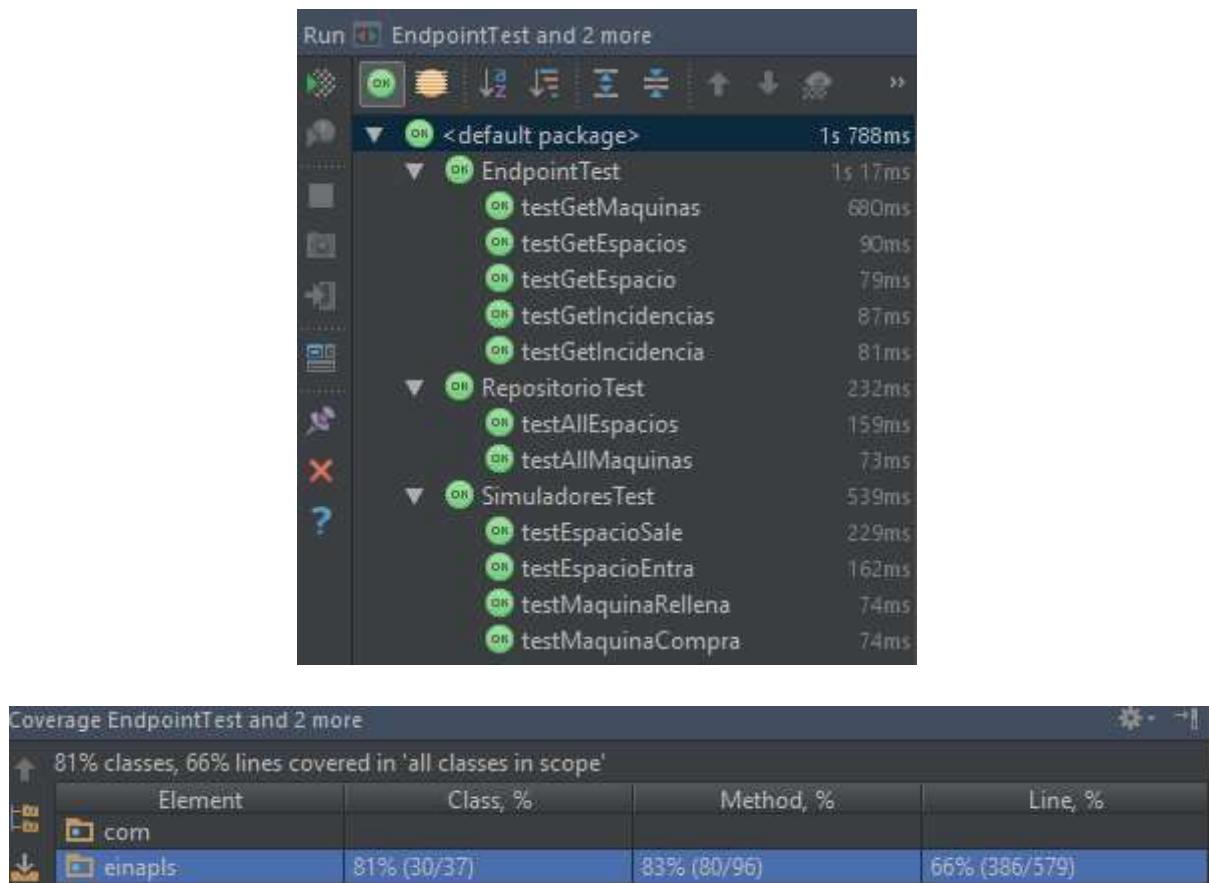
Restricciones:

tipoedificio solo puede ser: ADA_BYRON, BETANCOURT, TORRES_QUEVEDO

tipopiso solo puede ser: SOTANO, PISO_0, PISO_1, PISO_2, PISO_3, PISO_4
CAFETERIA, LAB, SESTUDIO, BIBLIOTECA, AULA

Queríamos tener una base de datos lo con las menos relaciones posibles para simplificar el manejo de esta debido a nuestra inexperiencia, por ello cada uno de las tablas son independientes.

2.2.10 Estrategia y herramientas de pruebas



En esta iteración se ha implementado la infraestructura de pruebas para el servidor mediante jUnit llegando a una cobertura de código del **66%** en el back-end para un total de 11 test. Para medir la cobertura se ha utilizado IntelliJ IDEA, que provee de herramientas para pasar los tests automáticamente y medir esta cobertura. Sin embargo, los tests (sin medida de cobertura) pueden pasarse independientemente del IDE con el que se trabaje mediante gradle por la línea de comandos, con el comando 'gradle build'.

La clases EndpointTest, RepositorioTest y SimuladoresTest tienen como objetivo comprobar el correcto funcionamiento de los diferentes *endpoints* del sistema, repositorios (máquinas, y espacios), y simuladores (presencia, y stock) respectivamente.

3 Proceso

3.1 Primer sprint

3.1.1 Estrategia de control de versiones

Se ha creado una organización en la plataforma de desarrollo colaborativo GitHub. Se cuenta con dos repositorios: EINApIs-backend y EINApIs-ionic-client. Esto es, por un lado el cliente, y, por otro, el servidor. El equipo de desarrollo trabaja sobre una misma rama.

3.1.2 Esfuerzos por persona y actividad

Los esfuerzos por persona y correspondientes a las actividades realizadas se encuentran entregados en la plataforma Moodle¹.

3.1.3 Estrategia de mejora de procesos

Se ha seguido la metodología SCRUM para la aplicación de buenas prácticas colaborativas, y así obtener el mejor resultado posible dentro del contexto de proyecto. Para este caso, Adrián Moreno ejerce de scrum master y Javier Murillo ejerce de product owner. El resto del equipo ejercerán de desarrolladores.

En SCRUM se realizan entregas parciales las cuales son priorizadas por el beneficio que aportan al receptor del proyecto. Se ha creído conveniente hacer uso de este proceso ya que es ideal para entornos complejos donde se necesitan obtener resultados pronto.

El proyecto se ejecuta en bloques, en este caso, iteraciones de un mes natural. Al final de las cuales se realiza un ejercicio de demostración y retrospectiva para analizar los resultados y realizar las adaptaciones necesarias de manera objetiva, así como analizar la metodología de trabajo, eliminando los obstáculos identificados.

3.1.4 Herramientas utilizadas

A través de una hoja Excel se ha llevado la contabilidad de horas trabajadas por cada miembro, así como la gestión del tablero SCRUM donde se define el estado actual de la iteración, y las tareas por hacer, así como las tareas en desarrollo y finalizadas.

Para la comunicación con el equipo se ha utilizado la plataforma Telegram. Los documentos relacionados con la gestión del proyecto se han mantenido en un directorio compartido en Google Drive y el código, así como la documentación, en GitHub, como ya se ha comentado.

¹ Laboratorio de Ingeniería de Software (2015/16).
<https://moodle2.unizar.es/add/course/view.php?id=12539>

3.2 Segundo sprint

3.2.1 Estrategia de control de versiones

Con respecto al código, se ha seguido la misma estrategia que se seguía en el primer sprint. Además, cada dos semanas aproximadamente se ha realizado un backup de la base de datos, el cual se guarda en Google Drive ya que PostGIS tiene la opción de realizar dicho backup de manera sencilla.

4 Conclusiones

4.1 Primer sprint

El proyecto EINApIs ha planteado diferentes retos al equipo de desarrollo. No se contaba con experiencia previa en el dominio que se plantea, EINApIs es el primer sistema desarrollado que hace uso de conceptos relacionados con modelos geográficos.

Esto supone que parte del tiempo disponible de los miembros del equipo ha de ser destinado al aprendizaje sobre las herramientas asociadas a este campo, tales como QGIS, PGAdmin, o GeoKettle. Este desconocimiento ha dificultado también la elección de tecnología a usar en la parte del cliente.

Se ha perdido demasiado tiempo probando distintas alternativas, lo que ha hecho que al final no se pudieran completar todos los objetivos del sprint. Esto ha reducido el valor aportado en esta iteración del proceso, por lo que es importante aprender del error y tratar de evitarlo en futuras iteraciones o proyectos.

No obstante, la elección de un proceso de trabajo como SCRUM ha mitigado parte de estos riesgos, ya que las reuniones frecuentes y los análisis al final de cada iteración han permitido detectar determinados problemas a tiempo, lo que nos permite poner remedio de forma ágil a estos problemas.

Finalmente, cabe destacar que el proceso de diseño del modelo de dominio ha permitido adquirir una mayor comprensión del problema, facilitando así el diseño arquitectural del sistema.

4.1.1 Cumplimiento de objetivos de evaluación

Se presenta a continuación la lista de objetivos evaluables, junto a su grado de cumplimiento. Resaltados en verde se encuentran los objetivos actualmente cumplidos. Resaltados en rojo, los objetivos todavía por cumplir.

| Objetivo | % Hecho |
|--|---------|
| <ul style="list-style-type: none">• El código y la documentación del proyecto se alojan en GitHub. Se trabaja de forma habitual contra Git• Respecto a la documentación se puede usar la wiki y/o guardar documentos en el formato que sea en GitHub. El caso es que la documentación, igual que el código, se mantendrá de manera habitual en GitHub | 100 |
| <ul style="list-style-type: none">• Compilación y gestión de dependencias (p.ej. descarga de bibliotecas externas) están basada en scripts (Gradle, Maven...) | 100 |
| <ul style="list-style-type: none">• Se llevará un control de esfuerzos con las horas dedicadas por persona. Se trabaja un número de horas en el entorno de lo requerido | 100 |

| | | |
|--|---|--|
| | para la asignatura (90 horas dedicadas al “trabajo en grupo” por persona). Se entregará un resumen cada dos semanas | |
| • La aplicación cumple adecuadamente con sus requisitos | 20 | |
| • La documentación arquitectural es la adecuada al momento del proyecto, refleja fielmente el sistema, e incluye al menos tres vistas: módulos, componentes-y-conectores, y despliegue del sistema | 100 | |
| • Se usan adecuadamente estos conceptos de diseño dirigido por el dominio: entidades, objetos valor, agregados, factorías y repositorios • Se usan adecuadamente en el código y se reflejan adecuadamente en la documentación arquitectural | 100 | |
| • Se ha puesto en marcha y se usa un servicio de mapas tipo WMS con los edificios disponibles del campus Río Ebro. Los mapas de este servicio se superponen en el cliente sobre otro servicio externo (p.ej. Open Street Map) que proporcione un mapa de la zona | 100 | |
| • Cobertura de tests automáticos de al menos el 25% del código (unitarios y/o de integración) | 0 | |
| • La documentación arquitectural incluye una discusión adecuada sobre razones arquitecturales | 0 | |
| • El modelo de dominio utiliza adecuadamente estos conceptos de diseño (dirigido por el dominio): servicios, paquetes, interfaces reveladoras, aserciones, funciones libres de efectos secundarios • Se usan adecuadamente en el código y su uso se refleja adecuadamente en la documentación | 0 | |
| • El estilo cartográfico de los edificios en el servicio de tipo WMS refleja el tipo de uso de cada espacio (por ejemplo, los laboratorios de un color, los despachos de otro etc.) | 0 | |
| • El modelo de dominio incluye alguna restricción o especificación correctamente implementada, y ésta se utiliza en alguna funcionalidad de la aplicación | 0 | |
| • La arquitectura del sistema es hexagonal • Se implementa así y se refleja adecuadamente en la documentación | 0 | |
| • La aplicación permite hacer algún tipo de consulta que podamos clasificar como “Análisis SIG” y esto se documenta adecuadamente, haciendo referencia a conceptos vistos en teoría | 0 | |
| • El servicio de mapas WMS se ha teselado, y se usa así desde el cliente | 0 | |

4.2 Segundo sprint

Tras la finalización del segundo sprint de manera satisfactoria de los requisitos que se plantearon a principio del mismo se ha conseguido como resultado una aplicación la cual maneja mapas con una arquitectura la cual nos permite un mantenimiento y mejora futuros mucho más cómodos y efectivos gracias a la arquitectura hexagonal implementada.

La modificación y adaptación de los mapas a la base de datos para poder mostrarlos en OpenStreetMap con la codificación WMS 84 Pseudo Mercator ha llevado más tiempo del que se hubiera deseado ya que se trata de un trabajo monótono y repetitivo, además de que QGIS al transformar las coordenadas se come algunos polígonos lo que no se ha conseguido solucionar. Dicho trabajo, una vez se sabe como hacer, hay que repetirlo una vez por cada planta de cada edificio y si algo está mal hay que volver a empezar prácticamente de 0 en la mayoría de los casos.

La población de los espacios en la base de datos, aunque a priori puede parecer una tarea sencilla, también se ha comido mucho tiempo ya que hay que conseguir los puntos céntricos de cada espacio parseando los datos de la salida de la base de datos.

La implementación del back-end ha sufrido numerosos cambios a lo largo del desarrollo, aunque en general esta no ha dado problemas a parte de los típicos que se suelen dar en estos casos. La conversión de objetos GeoJSON a objetos Java se ha hecho de manera manual y los sensores han sido sencillos de implementar.

La implementación del front-end no ha ocasionado demasiados problemas, ya que Angular permite construir un patrón MVC bien definido que facilita la separación de código. Sí se detectaron algunas incompatibilidades entre Ionic framework y la librería para trabajar con mapas Leaflet, pero una vez localizados y solucionados, el funcionamiento no da problemas. Es muy interesante plantearse el uso de tecnologías como Ionic Framework, ya que con una misma codebase se pueden construir aplicaciones para distintas plataformas con muy poco esfuerzo. Sin embargo, también tienen sus pegas, ya que cuando se necesita acceso a una tecnología específica de la plataforma (por ejemplo, la cámara de fotos), se añade una capa de abstracción que supone un nivel más de complejidad y dificultad.

4.2.1 Cumplimiento de objetivos de evaluación

| Objetivo | % Hecho |
|---|---------|
| <ul style="list-style-type: none"> El código y la documentación del proyecto se alojan en GitHub. Se trabaja de forma habitual contra Git Respecto a la documentación se puede usar la wiki y/o guardar documentos en el formato que sea en GitHub. El caso es que la documentación, igual que el código, se mantendrá de manera habitual en GitHub | 100 |
| <ul style="list-style-type: none"> Compilación y gestión de dependencias (p.ej. descarga de bibliotecas externas) están basada en scripts (Gradle, Maven...) | 100 |

| | |
|--|-----|
| <ul style="list-style-type: none"> Se llevará un control de esfuerzos con las horas dedicadas por persona. Se trabaja un número de horas en el entorno de lo requerido para la asignatura (90 horas dedicadas al “trabajo en grupo” por persona). Se entregará un resumen cada dos semanas | 100 |
| <ul style="list-style-type: none"> La aplicación cumple adecuadamente con sus requisitos | 85 |
| <ul style="list-style-type: none"> La documentación arquitectural es la adecuada al momento del proyecto, refleja fielmente el sistema, e incluye al menos tres vistas: módulos, componentes-y-conectores, y despliegue del sistema | 100 |
| <ul style="list-style-type: none"> Su usan adecuadamente estos conceptos de diseño dirigido por el dominio: entidades, objetos valor, agregados, factorías y repositorios Se usan adecuadamente en el código y se reflejan adecuadamente en la documentación arquitectural | 90 |
| <ul style="list-style-type: none"> Se ha puesto en marcha y se usa un servicio de mapas tipo WMS con los edificios disponibles del campus Río Ebro. Los mapas de este servicio se superponen en el cliente sobre otro servicio externo. | 100 |
| <ul style="list-style-type: none"> Cobertura de tests automáticos de al menos el 25% del código (unitarios y/o de integración) | 100 |
| <ul style="list-style-type: none"> La documentación arquitectural incluye una discusión adecuada sobre razones arquitecturales | 100 |
| <ul style="list-style-type: none"> El modelo de dominio utiliza adecuadamente estos conceptos de diseño (dirigido por el dominio): servicios, paquetes, interfaces reveladoras, aserciones, funciones libres de efectos secundarios Se usan adecuadamente en el código y su uso se refleja adecuadamente en la documentación | |
| <ul style="list-style-type: none"> El estilo cartográfico de los edificios en el servicio de tipo WMS refleja el tipo de uso de cada espacio (por ejemplo, los laboratorios de un color, los despachos de otro etc.) | 100 |
| <ul style="list-style-type: none"> El modelo de dominio incluye alguna restricción o especificación correctamente implementada, y ésta se utiliza en alguna funcionalidad de la aplicación | |
| <ul style="list-style-type: none"> La arquitectura del sistema es hexagonal Se implementa así y se refleja adecuadamente en la documentación | 100 |
| <ul style="list-style-type: none"> La aplicación permite hacer algún tipo de consulta que podamos clasificar como “Análisis SIG” y esto se documenta adecuadamente, haciendo referencia a conceptos vistos en teoría | |
| <ul style="list-style-type: none"> El servicio de mapas WMS se ha teselado, y se usa así desde el cliente | 0 |

5 ANEXOS

5.1 Manual de Usuario

Al seleccionar la opción de 'Selección de edificio' (1) se mostrará una lista desplegable a la izquierda con los edificios del campus.

Puede hacerse clic en cualquier de los puntos (2) para obtener información al respecto. Los puntos verdes representan espacios, los puntos rojos incidencias.

Aparecerá un desplegable en la parte inferior de la pantalla al hacer clic en 'Selección de piso' (3), para navegar entre las diferentes plantas de los edificios.

Al seleccionar 'Creación de incidencia' (4) se presentará al usuario un formulario con los datos a llenar para el envío de la misma.

El usuario tiene disponible la herramienta zoom (5) para aumentarlo o disminuirlo.



5.2 API server ofrecida

| | |
|-----------------------------|--|
| Petición | GET /places/{tipoEdificio}/{tipoPiso} |
| Respuesta | Una lista en geojson de espacios cuyo edificio es {tipoEdificio} y piso es {tipoPiso} |
| Ejemplo de respuesta | <pre>{ "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [-0.8884047, 41.683758] }, "properties": { "id": 88, "ocupacion": 12, "capacidad": 50, "tipoEspacio": "AULA", "tipoPiso": "PISO_1", "tipoEdificio": "ADA_BYRON" } }, { "type": "Feature", "geometry": { "type": "Point", "coordinates": [-0.88840294, 41.68359] }, "properties": { "id": 86, "ocupacion": 17, "capacidad": 50, "tipoEspacio": "AULA", "tipoPiso": "PISO_1", "tipoEdificio": "ADA_BYRON" } }] }</pre> |

| | |
|-----------------------------|--|
| Petición | GET /places/{id} |
| Respuesta | Un objeto en geojson que representa un espacio |
| Ejemplo de respuesta | <pre>{ "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [-0.8888103, 41.683987] }, "properties": { "id": 164, "ocupacion": 99, "capacidad": 20, "tipoEspacio": "LABORATORIO", "tipoPiso": "PISO_2", "tipoEdificio": "ADA_BYRON" } }] }</pre> |

| | |
|-----------------------------|--|
| Petición | GET /machines/{tipoEdificio}/{tipoPiso} |
| Respuesta | Una lista en geojson de maquinas expendedoras cuyo edificio es {tipoEdificio} y piso es {tipoPiso} |
| Ejemplo de respuesta | <pre>{ "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [-0.888449, 41.68354] }, "properties": { "stock": { "Patatas": 3, "Chocolate": 4 }, "tipoPiso": "PISO_1", "tipoEdificio": "ADA_BYRON" } }] }</pre> |

| | |
|-----------------------------|---|
| Petición | GET /machines/{id} |
| Respuesta | Un objeto en geojson que representa una máquina expendedora con una lista que representa su stock |
| Ejemplo de respuesta | <pre>{ "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [-0.888449, 41.68354] }, "properties": { "id": "1", "stock": { "Patatas": 3, "Chocolate": 4 }, "tipoPiso": "PISO_1", "tipoEdificio": "ADA_BYRON" } }] }</pre> |

| | |
|-----------------------------|---|
| Petición | GET /issues/{tipoEdificio}/{tipoPiso} |
| Respuesta | Una lista en geojson de incidencias cuyo edificio es {tipoEdificio} y piso es {tipoPiso} |
| Ejemplo de respuesta | <pre>{ "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [2, 2] }, "properties": { "id": "1", "titulo": "titulo", "estadoIncidencia": "ABIERTA", "foto": "fdsfd", "descripcion": "des", "tipoPiso": "PISO_0", "tipoEdificio": "ADA_BYRON" } }] }</pre> |

| | |
|-----------------------------|---|
| Petición | GET /issues/{id} |
| Respuesta | Un objeto en geojson que representa una incidencia |
| Ejemplo de respuesta | <pre>{ "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [2, 2] }, "properties": { "id": "1", "titulo": "titulo", "estadoIncidencia": "ABIERTA", "foto": "fdsfd", "descripcion": "des", "tipoPiso": "PISO_0", "tipoEdificio": "ADA_BYRON" } }] }</pre> |

| | |
|----------------------------|---|
| Petición | POST /issues |
| Respuesta | Introduce una incidencia en el servidor y devuelve un GeoJson con su info. |
| Ejemplo de petición | <pre>{ "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [123.321, 123.321] }, "properties": { "titulo": "Un titulo", "estadoIncidencia": "ABIERTA", "foto": "una url", "descripcion": "Una descripcion", "tipoPiso": "SOTANO", "tipoEdificio": "ADA_BYRON" } }] }</pre> |

| | |
|-----------------------------|---|
| Ejemplo de respuesta | <pre>{ "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [123.321, 123.321] }, "properties": { "titulo": "Un titulo", "estadoIncidencia": "ABIERTA", "foto": "una url", "descripcion": "Una descripcion", "tipoPiso": "SOTANO", "tipoEdificio": "ADA_BYRON" } }] }</pre> |
|-----------------------------|---|

| Diccionario | |
|--------------------|--|
| {tipoEdificio} | ADA_BYRON TORRES_QUEVEDO BETANCOURT |
| {tipoPiso} | SOTANO PISO_0 PISO_1 PISO_2 PISO_3 PISO_4 |
| espacio | puede representar un: AULA, LABORATORIO, BIBLIOTECA, SALA_DE_ESTUDIO |