Laboratorio Proyecto Software

Informe del Proyecto

Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Adrián Martínez Pérez - 576307

1. INTRODUCCIÓN

En éste informe se recoge toda la información del producto y el proceso elaborado aplicando diseño dirigido por el dominio y prácticas sobre el manejo de información geográfica. La aplicación construida trata sobre la creación de incidencias para el mantenimiento del Campus Río Ebro de la Universidad de Zaragoza, además de ofrecer un mapa interactivo con el que se pueda obtener información sobre cualquier espacio existente.

Éste proyecto ha sido creado por Adrián Martínez Pérez, con NIP 576307, estudiante del mismo Campus.

En el informe se recogen los siguientes apartados:

- Introducción: breve inicio con el resumen de lo que se va a ver.
- Producto: espacio con la información sobre el producto final.
- Proceso: descripción de las herramientas utilizadas y dificultades encontradas a lo largo del proceso de construcción.
- Conclusiones: resumen final.

2. PRODUCTO

2.1. PLANIFICACIÓN

El proyecto empezó a planearse al día 30 de Noviembre del 2018, y empezó a construirse el día 31.

La planificación incluye las primeras decisiones iniciales y de diseño, como los requisitos de la aplicación y las herramientas a utilizar en el proceso, y se dispuso de 2 semanas para su construcción completa.

2.2. REQUISITOS

Los requisitos se dividen en dos categorías: requisitos globales, los cuales se cumplen para todo el mundo, y requisitos de administración, usados por los administradores de la aplicación.

El formato de los requisitos son historias de usuario, que se enumeran a continuación:

- Globales:

- Como usuario quiero ver el mapa para obtener información sobre el Campus y las incidencias.
- Como usuario quiero crear incidencias para que se solucionen.
- Como usuario quiero consultar que incidencias hay ya aceptadas o completadas recientemente.
- Como usuario quiero poder consultar noticias de carácter general y de interés.

-Administración

- Como administrador quiero identificarme para poder usar las funciones de administrador.

- Como administrador quiero ver el mapa con las incidencias creadas, aceptadas, canceladas recientemente y completadas recientemente marcadas en él.
- Como administrador quiero cambiar el estado de las incidencias para producir su resolución.
- Como administrador quiero crear y eliminar noticias de interés general sobre el mantenimiento del campus.

2.3. ARQUITECTURA

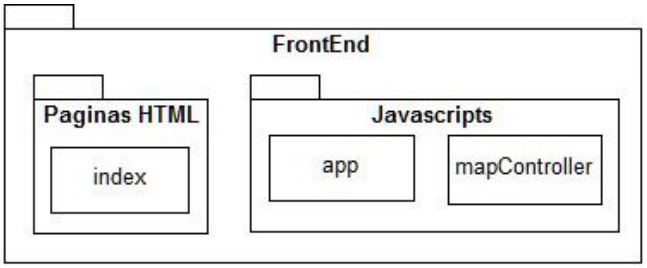
2.3.1. Modelo de datos

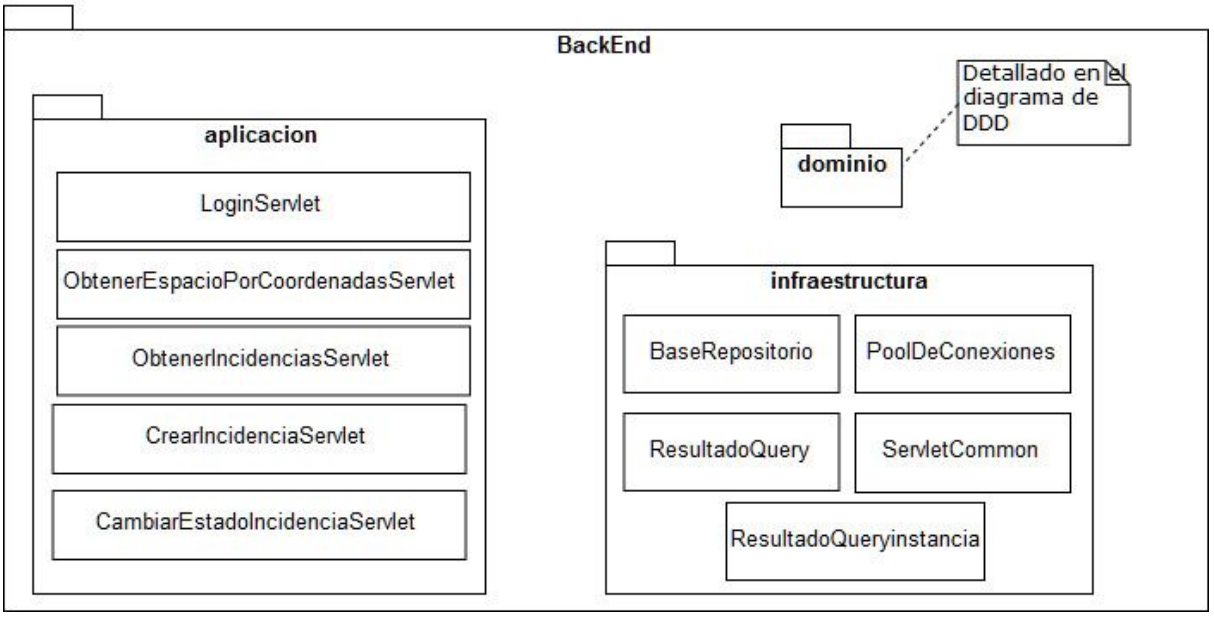
El modelo de datos se ha hecho lo más sencillo posible, sin restricciones innecesarias al menos en su paso por la BD. Se puede apreciar un modelo totalmente relacional, con 3 tablas con sus correspondientes campos y el tipo de dato que se puede guardar en ellos.

Todas las tablas tienen un identificador universal, generado automáticamente con el algoritmo generador de UUIDv4, guardado como una cadena de 36 caracteres.

La tabla de espacios contiene la geometría del propio espacio, para así poder hacer consultas espaciales.

2.3.2. Vista de módulos





Se ha utilizado el modelo por capas descrito en la asignatura: la capa cliente (Frontend) hace llamadas a la capa de aplicación que utiliza las capas de dominio e infraestructura para proveer de funcionalidad a la aplicación.

Cada capa superior utiliza estrictamente las inferiores.

- Módulos:

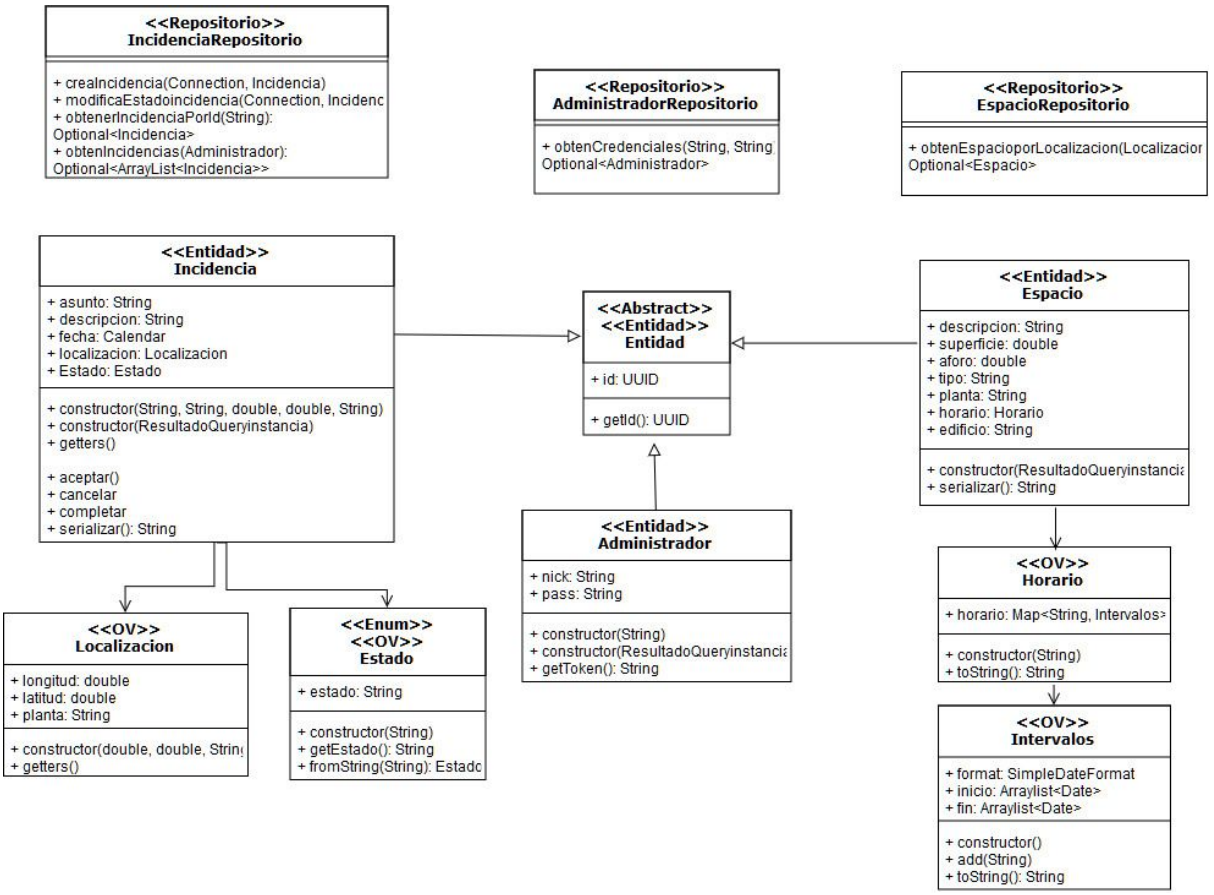
- Páginas HTML
  - index.html: página principal de la aplicación.
- JavaScript
  - app.js: módulo inicial de la aplicación.
  - mapController.js: controlador del mapa
- aplicacion
  - LoginServlet.java: funcionalidad del login de administradores.
  - ObtenerEspacioporCoordenadasServlet.java: funcionalidad de obtención de la información de un espacio según su ubacación.
  - ObtenerIncidenciasServler.java: funcionalidad de obtención de la información de las incidencias necesarias.
  - CrearIncidenciaServlet.java: funcionalidad de creación de incidencias.

- CambiarEstadoIncidenciaServlet.java: funcionalidad de modificación del estado de una incidencia.

- infraestructura
  - PoolDeConexiones.java: obtiene conexiones con la BD.
  - BaseRepositorio.java: ejecuta código SQL sobre la BD.
  - ResultadoQuery: contenedor de datos entre el dominio y la infraestructura. Como un ResultSet pero sin la estructura de la BD.
  - ResultadoQueryInstancia: cada fila de la BD, convirtiendo los datos a cadenas de caracteres.
  - ServletCommon.java: herramientas comunes para los servlets.

2.3.2.1. Dominio

Se ha procurado en lo posible que el dominio de problema esté aislado en su propio lenguaje y con sus propias reglas, tal como promulga el DDD.

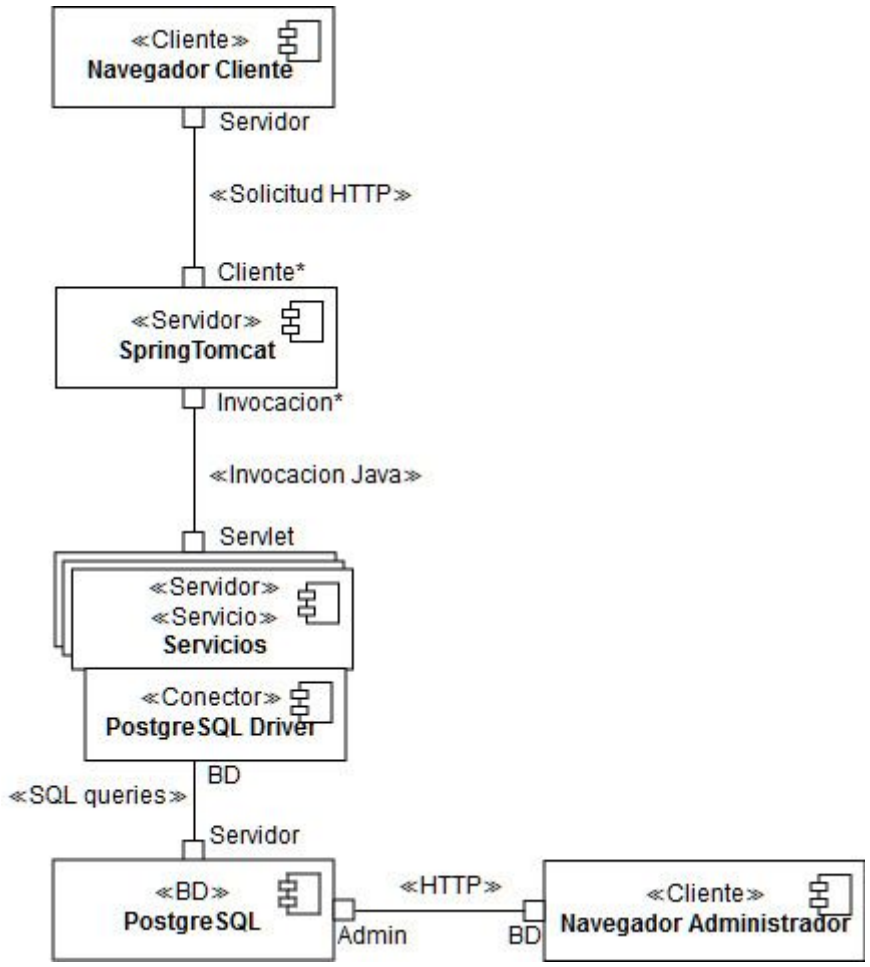


Se puede comprobar que existen 3 clases repositorio, una por cada agregado: espacios, incidencias y administradores. Nótese que todos los objetos hablan el idioma de dominio y de Java, utilizando los objetos de la infraestructura para la obtención de los datos necesarios.

Los agregados son muy sencillos, así que dejamos que las factorías sean los propios constructores.

Finalmente, es la capa de aplicación la que debe dar uso de el dominio, y manejar los errores y el funcionamiento general de la aplicación.

2.3.3. Vista de componentes y conectores



- Elementos
  - Navegador Cliente: puede haber múltiples instancias al mismo tiempo.
  - Gretty Tomcat: contenedor de servlets que enruta la petición HTTP hacia los controladores de las aplicaciones bedel y/o geoserver correspondientes, configurados mediante el fichero WEB-INF/web.xml de la aplicación web. Gradle compila antes la aplicación en un fichero war, que es el que se despliega.
    - Cliente:
      -Protocolo: HTTP 1.1.
      -Puerto: 8080.
  - Servicios: cumplen cada uno de los requisitos de la aplicación. Llamados desde el contenedor Tomcat.
    - Servlet:
      -Contexto: JVM, invocación a métodos.

  - PostgreSQL:
    - Servidor BD: solo los servicios de la aplicación bedel acceden a la BD. Se usa un pool de conexiones basado en la tecnología de HikariCP que

devuelve una conexión en la que se envían las consultas SQL. Se usa el driver oficial de PostgreSQL JDBC, con el que HikariCP se autoconfigura.
-Protocolo: TCP/IP.
-Puerto: 5432

- □ Admin: instancia del programa pgAdmin4 que se conecta a la BD para su administración (implementacion externa).
  -Protocolo: HTTP 1.1.
  -Puerto: aleatorio, por encima del 50000.
  -Usuario: postgres.
  -Password: 1234.

- Navegador Administrador: puede haber múltiples instancias al mismo tiempo.

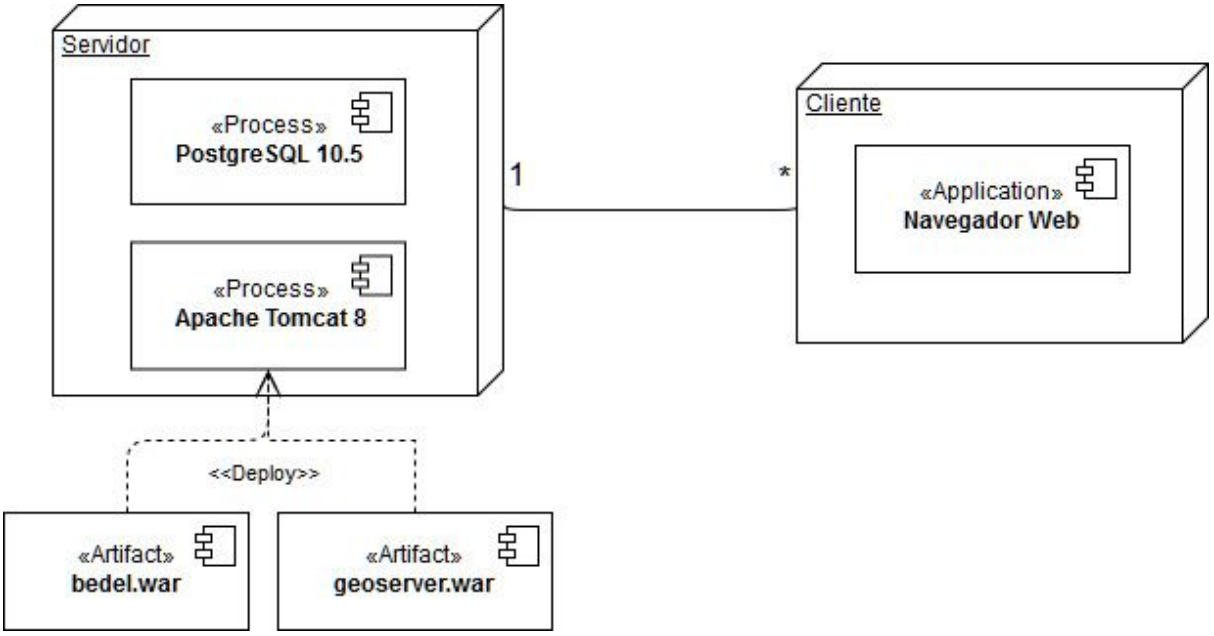
- API

El orden de los parámetros que van en el body es irrelevante. Los tipos de datos están escritos en términos de Java y JSON. Los datos, inicialmente en formato String, se convierten al formato del dominio, y si no se puede da error 500.

- GET /login.
  head: application/json
  body: {nick: String, pass: String}
  Success return: 200 OK {token: {nick: String, pass: String, id: String}}
  Error return: 404 NOT_FOUND {“error”}
  Failure: 500 INTERNAL_SERVER_ERROR {“error”}
- GET /obtenerEspacioPorCoordenadas.
  head: application/json
  body: {lon: double, lat: double, planta: String}
  Success return: 200 OK {descripcion: String, superficie: double, aforo: double, tipo: String, planta: String, horario: String, edificio: String, id:String} OR {exterior: boolean}
  Error and Failure: 500 INTERNAL_SERVER_ERROR {“error”}
- GET /obtenerIncidencias.
  head: application/json
  body: {token: {nick: String, pass: String, id: String}}
  Success return: 200 OK {incidencias: [{asunto: String, descripcion: String, fecha: String, lon: double, lat: double, planta: String, estado: String, id: String}]}
  Error and Failure: 500 INTERNAL_SERVER_ERROR {“error”}
- POST /crearIncidencia.
  head: application/json
  body: {asunto: String, descripcion: String, lon: double, lat: double, planta: String}
  Success return: 200 OK {“Incidencia creada”}
  Error and Failure: 500 INTERNAL_SERVER_ERROR {“error”}
- PUT /cambiarEstadoIncidencia.
  head: application/json
  body: {token: {nick: String, pass: String, id: String}, nuevoEstado: String}

Success return: 200 OK {“Estado de la incidencia modificado”}
Error and Failure: 500 INTERNAL_SERVER_ERROR {“error”}

2.3.4. Vista de despliegue



Se ha conseguido desplegar las dos aplicaciones necesarias en el mismo Apache Tomcat, que atienden peticiones en el mismo puerto pero distinta URL. El despliegue se realiza muy fácilmente gracias a Gradle y el plugin Gretty, que permite la creación de *farms*, scripts que permiten lanzar varias aplicaciones web a la vez.

En el mismo servidor se encuentra la BD PostgreSQL.

2.4. ESTADO ACTUAL

La aplicación se encuentra en un estado incompleto. Sólo se ha implementado el backend, dejando al frontend en un estado en el que se puede comprobar el mapa y usarlo, pero no está conectado en manera alguna con el backend. El mapa, construido con el API de OpenLayers, incluye el servicio WMS del geoserver modificado para ofrecer los mapas de la EINA categorizados y georreferenciados con el sistema de coordenadas EPSG:25830, así como un selector de alturas.

En cuanto al backend, las funcionalidades implementadas son funcionales y están testeadas. Faltan las funcionalidades de crear noticias, eliminar noticias y obtener espacios según un filtro.

Los tests realizados son tests de integración, y no unitarios, por lo que la cobertura de código es del 0%.

3. PROCESO

3.1. CONTROL DE VERSIONES

El repositorio, con nombre *Bedel*, se encuentra alojado en GitHub, bajo una organización denominada UNIZAR-30249-2018-BEDEL. En él se encuentra la aplicación Bedel, la aplicación geoserver en formato .war, el dump de la BD y la creación de la tabla de espacio en formato SQL, la documentación del proyecto y los archivos necesarios para desplegar la aplicación.

3.2. ESFUERZOS

Al trabajar en el proyecto en solitario sólo se han anotado las horas usadas por día. La hoja de cálculo se puede encontrar en el repositorio de Github.

	Horas Trabajadas
31/08/2018	5:35:00
01/09/2018	2:10:00
02/09/2018	-
03/09/2018	3:30:00
04/09/2018	4:30:00
05/09/2018	2:50:00
06/09/2018	3:05:00
07/09/2018	-
08/09/2018	-
09/09/2018	-
10/09/2018	3:10:00
11/09/2018	5:35:00
12/09/2018	9:20:00
13/09/2018	10:20:00
14/09/2018	1:00:00
Total Horas	51:05:00

3.3. HERRAMIENTAS UTILIZADAS

- Software
  - Geoserver 2.13.2 en formato war como base.
  - QGIS 3.2.2 para introducir y categorizar los shapefiles en Geoserver.
  - GeoKettle 2.5 para introducir las categorías en los shapefiles y para crear la tabla espacio en la BD.
  - IntellIJ IDEA 2018.2.3 como entorno de desarrollo, con plugin para trabajar con Git.
  - PostgreSQL 10.5 como BD.
  - pgAdmin 4 como herramienta de manejo de la BD.
- Dependencias Frontend
  - AngularJS 1.7.2 como framework de desarrollo.
  - UI-bootstrap 3.0.4 para la visualización de la aplicación.
  - Openlayers 5.2 para la visualización del mapa, con mapa base de OpenStreetMaps.
  - ol-layerswitcher 3 para añadir un control de alturas al mapa.
  - Proj4js 2.5 para trabajar con el sistema de coordenadas EPSG:25830.
- Dependencias Backend
  - Gradle 4.10 con plugin Gretty 2 para gestión de dependencias y despliegue.
  - HikariCP 3.2 como pool de conexiones con la BD.
  - net.sf.json-lib 2.4-jdk15 para el manejo del formato JSON.
  - JDBC PostgreSQL 42.2.5 como driver JDBC.

- javax.servlet-api 3.1 para la construcción de la capa de aplicación.
- JUnit 4.12 para los tests.
- Mockito 2+ para el mock de las peticiones y respuestas de los servlets.
- slf4j Logger para logging de tests y depuración.

- Lenguajes
  - Java 8.
  - JavaScript.
  - HTML.
  - Groovy para gestión de dependencias.
  - JSON para paso de datos entre cliente y servidor.
- Software de gestión
  - Drive.

3.4. PROBLEMAS ENCONTRADOS

- La aplicación geoserver.war no guardaba los cambios realizados, por lo que una instalación limpia no contendría los mapas de la EINA.
  Solución: compilar un nuevo geoserver.war con los datos correctos.
- Geoserver daba errores de compatibilidad con Gretty.
  Solución: cambio de contenedor de aplicaciones a Tomcat 8.
- La librería Leaflet no hacía bien la reproyección al sistema de coordenadas correcto, y seguía trabajando con el sistema por defecto a pesar de cambiarlo.
  Solución: cambio de librería a OpenLayers.
- Al cruzar los datos Access con los shapefiles se cruzan menos datos que espacios.
  Solución: ninguna, al hacer un INNER JOIN no se incluyen los espacios sin datos.
- La integración de AngularJS, UIBootstrap y OpenLayers en el frontend me hizo perder mucho tiempo.
  Solucion: ninguna.
- Se ha intentado realizar una medición de la cobertura de código de los tests de integración realizados, pero no aparece ninguna.
  Solucion: ninguna.

4. CONCLUSIONES

4.1. RESUMEN

La aplicación está sin terminar, pero el backend contiene la mayoría de la funcionalidad, y la más importante. Los tests de integración ayudan un poco a ver ésta funcionalidad. El DDD ha estado presente en el proceso desde el principio, optando por una implementación basada en servlets para hacer más fácil el uso de ésta técnica, y eso se nota en el aislamiento del dominio de la BD y su modelo.

La aplicación de los SIG también ha sido completa, llegando a proyectar en pantalla los mapas y poder jugar un poco con ellos, teniendo especial cuidado tanto en los sistemas de coordenadas y las reproyecciones usadas como en la visualización de los mismos mapas.

4.2. EVALUACION DEL PROYECTO

- Suficiente

- El código y la documentación del proyecto se alojan en GitHub. Se trabaja de forma habitual contra Git.
    - Proyecto en GitHub: https://github.com/UNIZAR-30249-2018-BEDEL/Bedel.
- Compilación y gestión de dependencias basadas en scripts (Gradle, Maven, npm...).
    - Gradle y su wrapper usado.
- 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 (unas 90 horas por persona). Se entregará un resumen al mes.
    - Se ha llevado un control de horas por día, llegando a poco más de lo requerido.
- La aplicación cumple sus requisitos.
    - Faltan requisitos y gran parte de la GUI.
- La documentación arquitectural es la adecuada al momento del proyecto, refleja fielmente el sistema e incluye al menos una vista de módulos, otra de componentes-y-conectores, y otra de despliegue.
    - La documentación arquitectural es fiel, pero no adecuada ni verbosa. Falta la vista de CyC.
- La arquitectura del sistema es por capas.
    - La arquitectura es por capas (interfaz, aplicacion, dominio, infraestructura).
- Se usan adecuadamente estos conceptos de diseño dirigido por el dominio: entidades, objetos valor, agregados, factorías y repositorios.
    - Si, se ha pensado bien el dominio.
- La aplicación permite hacer modificaciones de datos concurrentes
    - Si, se elimina el auto-commit, y se realizan commits al finalizar la transacción y rollbacks al dar fallo en algún momento, siempre y cuando hayan involucradas escrituras en la BD.
- 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.

- Si, se despliega Geoserver como servicio WMS y se usa OpenLayers como servicio de mapas.