Universitat Oberta de Catalunya

uoc.edu

Aplicación de los patrones CQRS y Event/Sourcing en el desarrollo de un portal inmobiliario

Francisco Jesús Ropero Morales
Grado en Ingeniería Informática
Desarrollo Web

Gregorio Robles Martínez
Santi Caballe Llobet

06/2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Aplicación de los patrones CQRS y Event/Sourcing en el desarrollo de un portal inmobiliario*
Nombre del autor:	*Francisco Jesús Ropero Morales*
Nombre del consultor/a:	*Gregorio Robles Martínez*
Nombre del PRA:	*Santi Caballe Llobet*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Desarrollo Web*
Idioma del trabajo:	*Castellano*
Palabras clave	*JAVA EE, CQRS, Spring*

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

Con este proyecto se pretende poner en práctica las competencias y habilidades adquiridas durante el grado, así como explorar nuevas tecnologías y conocimientos en el ámbito del desarrollo web.

Partiendo de esta premisa, se ha optado por un proyecto que tiene como objetivo la aplicación de los patrones de diseño CQRS mediante el desarrollo de una aplicación de publicación y gestión de anuncios inmobiliarios.

Se ha seguido una metodología clásica en cascada cuyo producto final es una aplicación con la que poder probar las ventajas del desarrollo basado en eventos.

Como apoyo gráfico se complementa con un cliente web desarrollado en Java con el *framework Vaadin*.

Abstract (in English, 250 words or less):

The aim of this project is to put into practice the competences and skills acquired during the degree, as well as to explore new technologies and knowledge in the field of web development.

Based on this premise, we have chosen a project that aims to apply the CQRS design patterns through the development of an application for the publication and management of real estate advertisements.

A classic waterfall methodology has been followed, the final product of which is an application with which to test the advantages of event-based development.

As graphic support, it is complemented with a web client developed in Java with the *Vaadin* framework.

A Marta, mi apoyo físico y moral durante este largo camino, quién ha sufrido el lado más amargo de esta etapa. Ella es merecedora de este éxito tanto como yo.

A mis padres y hermanas, por su cariño y confianza, por ser fuente de motivación en todo momento.

A mis amigos, la familia que se escoge, porque habéis conseguido que la distancia no se note, que sepa que siempre tengo ayuda, consuelo, risas y alegrías. Sin vosotros no hubiese merecido tanto la pena.

"Si caminas solo, irás más rápido;
si caminas acompañado, llegarás más lejos"

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Siempre me he considerado una persona inquieta con unas ganas de aprender constantes, es por ello por lo que, aun habiendo adquirido una gran amplitud de conocimientos a lo largo de la titulación, siempre me quedaba con las ganas de aprender y aplicar esas nuevas tecnologías que iba descubriendo en lecturas esporádicas, pero que por falta de tiempo nunca terminaba de afrontar.

Algunas de esas tecnologías son el patrón de programación *Event Sourcing* [1], la arquitectura dirigida por eventos [2] y el patrón de diseño CQRS [3].

Teniendo en cuenta que el objetivo de este trabajo debe ser el poner en práctica las competencias y habilidades adquiridas durante nuestra formación, pienso que afrontando el reto de aprender y aplicar una tecnología desconocida es también una cualidad que debe cumplir un graduado en ingeniería informática.

Como producto en el que aplicar este paradigma se ha optado por el desarrollo de un portal inmobiliario en el que procesar y gestionar los eventos derivados de la creación y modificación de anuncios de inmuebles.

Se ha optado por este escenario por permitir, en su complejidad, mostrar algunas de las ventajas del patrón mencionado bajo una temática concreta que además es de uso común.

1.2. Objetivos del Trabajo

Desarrollar una aplicación que permita poner en práctica las ventajas de los patrones de diseño *Event/Sourcing* y *CQRS*.

1.2.1. Sub-objetivos

- Implementar una estructura desacoplada que permita el tratamiento, almacenamiento y gestión de eventos.
- Desarrollar un cliente web independiente reactivo a los eventos producidos.
- Construir una API REST que permita el alta, modificación y consulta de inmuebles.
- Elaborar un microservicio que permita la notificación mediante correo electrónico de cambios en los inmuebles.
- Proporcionar una API que permita la consultar la evolución del precio de un inmueble en el tiempo.

1.3. Enfoque y método seguido

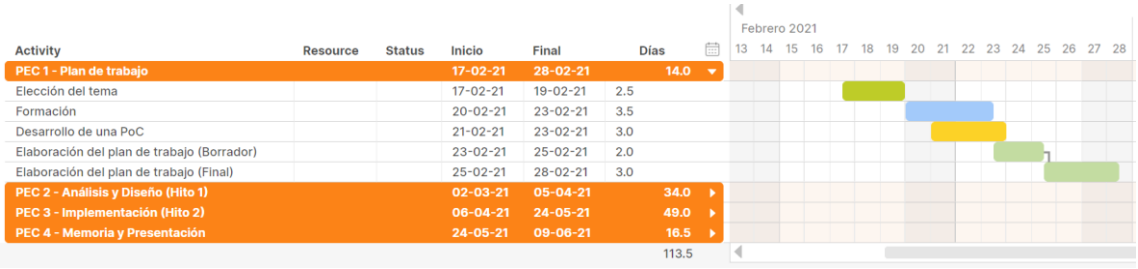
Dado que, desde el principio, el objetivo principal era el aprendizaje e interiorización de una nueva tecnología más que la de desarrollar un producto en sí, se ha optado por el desarrollo de un producto nuevo.

Teniendo en cuenta que el alcance y definición del producto final estaban bastante claros, se ha optado por seguir una metodología de planificación clásica como es el desarrollo en cascada.

1.4. Planificación del Trabajo

Para la gestión temporal de las diferentes fases de diseño se ha usado la herramienta web Tom's Planner [4] que proporciona una interfaz simple e intuitiva con la que crear tu propio diagrama de Gantt.

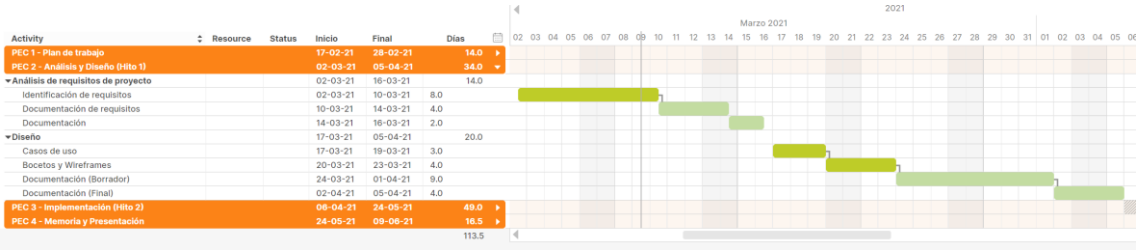
En la PEC 1, se ha concretado el plan de trabajo, que incluye la elección de tema, un tiempo de autoformación, desarrollo de una prueba de concepto, así como la planificación temporal de las siguientes fases. La planificación correspondiente a esta fase puede verse en la *Ilustración 1 - Planificación PEC1.*



Activity	Resource	Status	Inicio	Final	Días
PEC 1 - Plan de trabajo			17-02-21	28-02-21	14.0
Elección del tema			17-02-21	19-02-21	2.5
Formación			20-02-21	23-02-21	3.5
Desarrollo de una PoC			21-02-21	23-02-21	3.0
Elaboración del plan de trabajo (Borrador)			23-02-21	25-02-21	2.0
Elaboración del plan de trabajo (Final)			25-02-21	28-02-21	3.0
PEC 2 - Análisis y Diseño (Hito 1)			02-03-21	05-04-21	34.0
PEC 3 - Implementación (Hito 2)			06-04-21	24-05-21	49.0
PEC 4 - Memoria y Presentación			24-05-21	09-06-21	16.5
					113.5

Ilustración 1 - Planificación PEC1

En la PEC 2, se ha realizado un análisis del proyecto y el diseño de la solución a implementar, definiendo los requisitos, casos de uso y bocetos del producto final, tal y como se muestra en la *Ilustración 2 - Planificación PEC2*.



Activity	Resource	Status	Inicio	Final	Días
PEC 1 - Plan de trabajo			17-02-21	28-02-21	14.0
PEC 2 - Análisis y Diseño (Hito 1)			02-03-21	05-04-21	34.0
Análisis de requisitos de proyecto			02-03-21	16-03-21	14.0
Identificación de requisitos			02-03-21	10-03-21	8.0
Documentación de requisitos			10-03-21	14-03-21	4.0
Documentación			14-03-21	16-03-21	2.0
Diseño			17-03-21	05-04-21	20.0
Casos de uso			17-03-21	19-03-21	3.0
Bocetos y Wireframes			20-03-21	23-03-21	4.0
Documentación (Borrador)			24-03-21	01-04-21	9.0
Documentación (Final)			02-04-21	05-04-21	4.0
PEC 3 - Implementación (Hito 2)			06-04-21	24-05-21	49.0
PEC 4 - Memoria y Presentación			24-05-21	09-06-21	16.5
					113.5

Ilustración 2 - Planificación PEC2

En la PEC 3 se ha realizado la implementación y testing de la aplicación, comenzando por la infraestructura y *core,* proyecciones y finalizando con el desarrollo de la interfaz web. Esta planificación se detalla en la *Ilustración 3 - Planificación PEC3.*

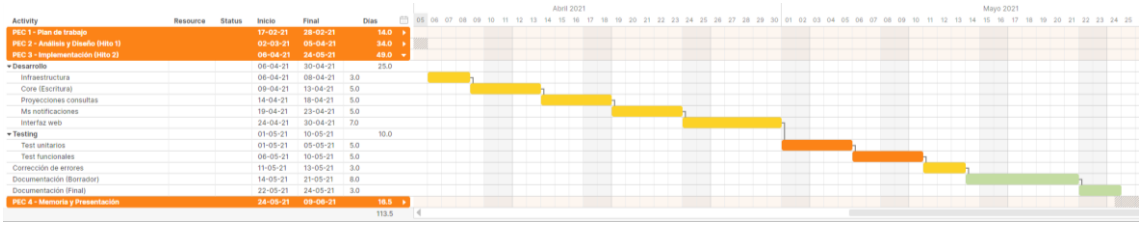


Ilustración 3 - Planificación PEC3

La última entrega, la PEC 4, queda destinada a la elaboración de la memoria, presentación y video del proyecto, tal y como se muestra en la *Ilustración 4 - Planificación PEC4.*

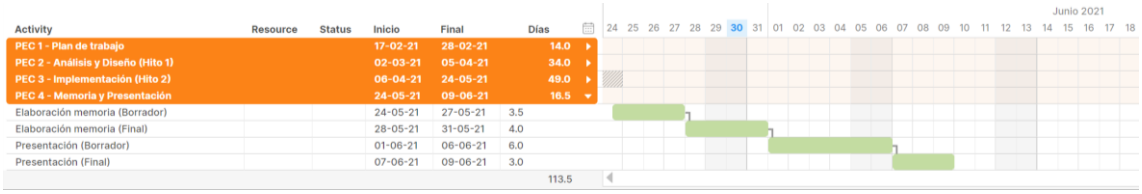


Ilustración 4 - Planificación PEC4

Para el control de cada tarea de desarrollo o pruebas se ha usado la aplicación Trello [5] en su versión web. En la *Ilustración 5 - Tablero de Trello* se puede ver un tablero de Trello usado para el control de las tareas.

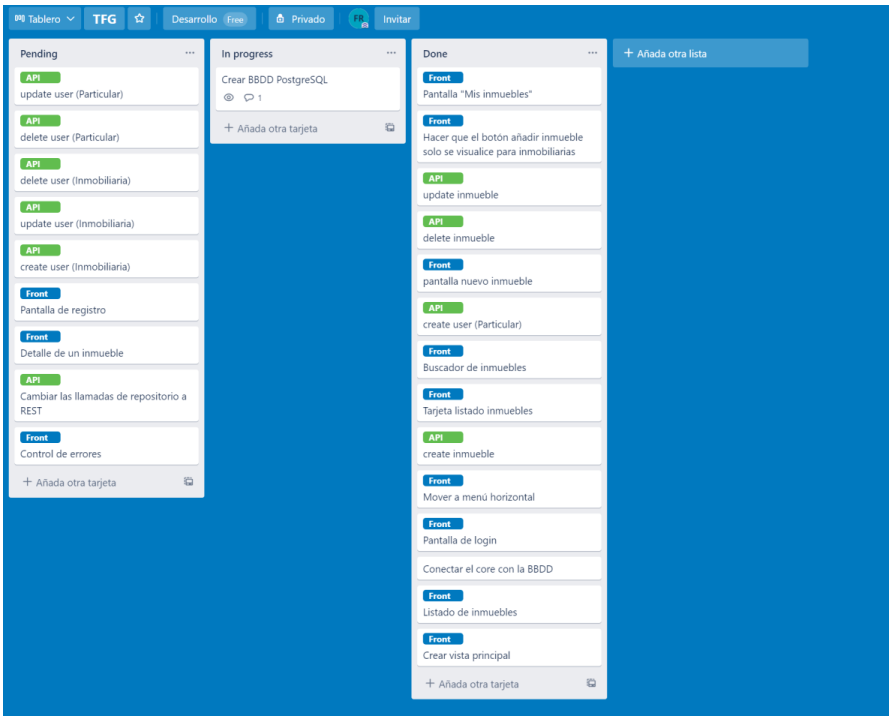


Ilustración 5 - Tablero de Trello

1.5. Breve sumario de productos obtenidos

Del trabajo realizado, se obtienen los siguientes productos:

- Repositorio con el código fuente desarrollado [6].
- Aplicación *Inmo* con las funcionalidades y características descritas.

1.6. Breve descripción de los otros capítulos de la memoria

- **Capítulo 2:** Evaluación de los posibles riesgos en la planificación y planes de mitigación.

- **Capítulos 3 y 4:** Análisis y diseño del proyecto, en estos capítulos se recogen los requisitos, casos de uso y se diseña técnica y visualmente el producto que se pretende desarrollar.

- **Capítulo 5:** En este capítulo se nombra y justifica las diferentes tecnologías usadas en la implementación del trabajo.

- **Capítulo 6:** Detalle de la implementación de las diferentes capas de la aplicación. En este capítulo se profundiza en la parte más técnica del desarrollo.

- **Capítulo 7:** Pruebas realizadas y resultados obtenidos.

- **Capítulo 8:** Muestra gráfica del producto final.

- **Capítulos 9 – 12:** Final de la memoria donde se recogen las conclusiones, bibliografía, glosario y anexos.

2. Riesgos

Ante el desarrollo de un proyecto de cierta complejidad y duración determinada es necesario identificar con antelación aquellos riesgos que pueden poner en peligro el cumplimiento de la planificación inicial o la viabilidad del producto para poder disponer de un plan de contingencia ante estos eventos.

Los posibles riesgos a tener en cuenta son:

- Infravalorar el tiempo de dedicación estimado para cada tarea debido a la falta de experiencia.
- La falta de conocimiento sobre la tecnología a usar.
- Componentes externos que puedan afectar a la dedicación (trabajo, familia, otras asignaturas).

Acciones para mitigar estos riesgos:

La formación y análisis en las siguientes fases del proyecto deben de proporcionar una idea más clara del coste de desarrollo de cada parte de la aplicación.

Por otro lado, la planificación previa realizada en esta entrega ha tenido en cuenta el tiempo diario disponible, por lo que, siempre que se cumpla la planificación se debería de poder realizar todo el trabajo a tiempo.

3. Análisis y requisitos

3.1. Roles

Se identifican tres roles diferentes:

- **Invitado:** usuario que no requiere de registro.
- **Usuario registrado:** particular que está registrado y puede iniciar sesión mediante su dirección de correo y contraseña.
- **Inmobiliarias:** empresas que inician sesión mediante una dirección de correo y contraseña y disponen de funcionalidades añadidas respecto a los usuarios particulares.

3.2. Requisitos

Los principales requisitos funcionales se han detallado en la *Tabla 1 - Requisitos funcionales.*

Tabla 1 - Requisitos funcionales

Requisitos funcionales	
RF – 001	Como invitado quiero poder consultar el catálogo de inmuebles publicados.
RF – 002	Como usuario registrado quiero poder suscribirme a un inmueble y recibir notificaciones cuando su precio cambie.
RF – 003	Como inmobiliaria quiero crear, modificar y eliminar mis inmuebles.
RF – 004	Como usuario registrado quiero poder ver la evolución del precio de un inmueble.
RF – 005	Como usuario registrado quiero solicitar información sobre un inmueble publicado.

3.3. Casos de uso

En este apartado se detallan los casos de uso identificados, relacionados con el rol o roles afectados y con el requisito funcional en cuestión.

Tabla 2 - CU-RF01

CU – RF01	Consultar inmuebles
Stakeholders	Usuario no registrado, Usuario registrado
Precondiciones	N/A
Escenario principal	1. El usuario accede a la aplicación. 2. Se muestra la pantalla principal con el buscador y filtros. 3. El usuario introduce los criterios de búsqueda. 4. El usuario pulsa el botón “buscar”. 5. El sistema muestra en la misma pantalla el listado de inmuebles con los criterios establecidos.
Escenario alternativo	5.a.1 No existen ningún inmueble que cumplan los criterios introducidos. 5.a.2 El sistema muestra el mensaje “No hay resultados”

Tabla 3 - CU-RF02

CU – RF02	Suscripción / Desuscripción a inmuebles
Stakeholders	Usuario registrado
Precondiciones	El usuario se encuentra logueado en el sistema y el usuario ha ejecutado los pasos del escenario principal del **CU – RF001**
Escenario principal	1. El usuario se encuentra ante el listado de inmuebles tras realizar una búsqueda. 2. El usuario selecciona un inmueble. 3. Se muestra la pantalla de detalle del inmueble seleccionado. 4. El usuario no se encuentra suscrito al inmueble. 5. El usuario activa la opción “Notificar cambios” 6. El sistema muestra un mensaje de confirmación “Se ha suscrito a este inmueble”. 7. El sistema enviará un correo electrónico al usuario cada vez que se modifique el anuncio.
Escenario alternativo	4.a.1 El usuario ya se encuentra suscrito al inmueble. 4.a.2 El usuario desactiva la opción “Notificar cambios”.

	4.a.3 El sistema muestra un mensaje de confirmación “No recibirá notificaciones de este inmueble”.

Tabla 4 - CU-RF03-01

CU – RF03 – 01	Crear un inmueble
Stakeholders	Inmobiliaria
Precondiciones	El usuario de tipo Inmobiliaria se encuentra logueado en el sistema.
Escenario principal	1. El usuario hace clic en el menú de inmobiliaria > Publicar un anuncio. 2. Se muestra la pantalla de alta de un inmueble con el formulario en blanco. 3. El usuario introduce los datos del inmueble. 4. El usuario pulsa el botón guardar. 5. El sistema muestra un mensaje de confirmación “Cambios guardados correctamente”.
Escenario alternativo	4.a.1 El usuario pulsa el botón cancelar. 4.a.2 El sistema muestra la pantalla anterior. 3.b.1 El usuario introduce datos inválidos o insuficientes. 3.b.2 El sistema muestra un mensaje de error.

Tabla 5 - CU-RF03-02

CU – RF03 – 02	Modificar un inmueble
Stakeholders	Inmobiliaria
Precondiciones	El usuario de tipo Inmobiliaria se encuentra logueado en el sistema y ha publicado algún inmueble.
Escenario principal	1. El usuario hace clic en el menú de inmobiliaria > Mis anuncios. 2. Se muestra el listado de inmuebles publicados por el usuario. 3. El usuario hace clic en el botón editar del inmueble seleccionad 4. Se muestra la pantalla de modificación de un inmueble con el formulario precargado con los datos del inmueble seleccionado. 5. El usuario modifica los datos del inmueble. 6. El usuario pulsa el botón guardar. 7. El sistema muestra un mensaje de confirmación “Cambios guardados correctamente”.
Escenario alternativo	6.a.1 El usuario pulsa el botón cancelar. 6.a.2 El sistema muestra la pantalla anterior.

	5.b.1 El usuario introduce datos inválidos o insuficientes. 5.b.2 El sistema muestra un mensaje de error.

Tabla 6 - CU-RF03-03

CU – RF03 – 03	Eliminar un inmueble
Stakeholders	Inmobiliaria
Precondiciones	El usuario de tipo Inmobiliaria se encuentra logueado en el sistema y ha publicado algún inmueble.
Escenario principal	1. El usuario hace clic en el menú de inmobiliaria > Mis anuncios. 2. Se muestra el listado de inmuebles publicados por el usuario. 3. El usuario hace clic en el botón editar del inmueble seleccionado. 4. Se muestra un mensaje con el texto “Esta acción no puede deshacerse. ¿Desea continuar?” y los botones aceptar y cancelar. 5. El usuario pulsa el botón aceptar. 6. El sistema muestra un mensaje de confirmación “Cambios guardados correctamente”.
Escenario alternativo	5.a.1 El usuario pulsa el botón cancelar. 5.a.2 Se cierra el mensaje de confirmación y se mantiene en la pantalla de “mis anuncios”.

Tabla 7 - CU-RF04

CU – RF04	Consultar el histórico de precios de un inmueble
Stakeholders	Usuario registrado
Precondiciones	El usuario se encuentra logueado en el sistema y el usuario ha ejecutado los pasos del escenario principal del **CU – RF001**
Escenario principal	1. El usuario se encuentra ante el listado de inmuebles tras realizar una búsqueda. 2. El usuario selecciona un inmueble. 3. Se muestra la pantalla de detalle del inmueble seleccionado. 4. El usuario pulsa el botón “Evolución del precio” 5. El sistema muestra un modal con la gráfica del histórico del precio.
Escenario alternativo	N/A

Tabla 8 - CU-RF05

CU – RF05	Solicitar información sobre un inmueble
Stakeholders	Usuario registrado
Precondiciones	El usuario se encuentra logueado en el sistema y el usuario ha ejecutado los pasos del escenario principal del **CU – RF001**
Escenario principal	1. El usuario se encuentra ante el listado de inmuebles tras realizar una búsqueda. 2. El usuario selecciona un inmueble. 3. Se muestra la pantalla de detalle del inmueble seleccionado. 4. El usuario se desplaza al final de la pantalla y rellena el cuadro de texto. 5. El usuario pulsa el botón enviar. 6. El sistema envía el mensaje al anunciante por correo electrónico con los datos del usuario.
Escenario alternativo	4.a.1 El usuario no introduce ningún texto. 4.a.2 El usuario pulsa el botón enviar. 4.a.2 El sistema muestra un mensaje de error.

4. Diseño

4.1. Arquitectura global

El sistema está formado por tres componentes diferenciados: el sistema de escritura, el sistema de lectura y el cliente web. Estas capas se muestran en la *Ilustración 6 - Arquitectura global.*

Desde el cliente web o desde la API del *Command Service* se lanzan los comandos contra los agregados del sistema (usuarios, inmuebles, etc.), estos comandos provocan los eventos que quedan registrados en el *event store* que proporciona el servidor de *Axon*. En este punto ocurren dos acciones: se actualiza el estado actual del sistema en una base de datos desnormalizada y a su vez el evento es capturado por el sistema de lectura para saber que el sistema ha cambiado y procede a actualizar la vista.

Para el cliente web se ha elegido usar la plataforma de desarrollo *Vaadin*, la cual permite simplificar mucho el desarrollo de esta pieza aportando características importantes como la de ofrecer cambios *push* en el catálogo cuando se lance un evento de creación o borrado.

Por otro lado, se desarrollan *APIs* de lectura y escritura documentadas mediante *Swagger*.

Por último, la base de datos de lectura, *View Database*, será una base de datos desnormalizada implementada sobre el SGBD PostgreSQL.

Para simplificar el desarrollo, todo el proyecto estará desarrollado en un único proyecto *Spring Boot* que facilita el despliegue y uso de este.

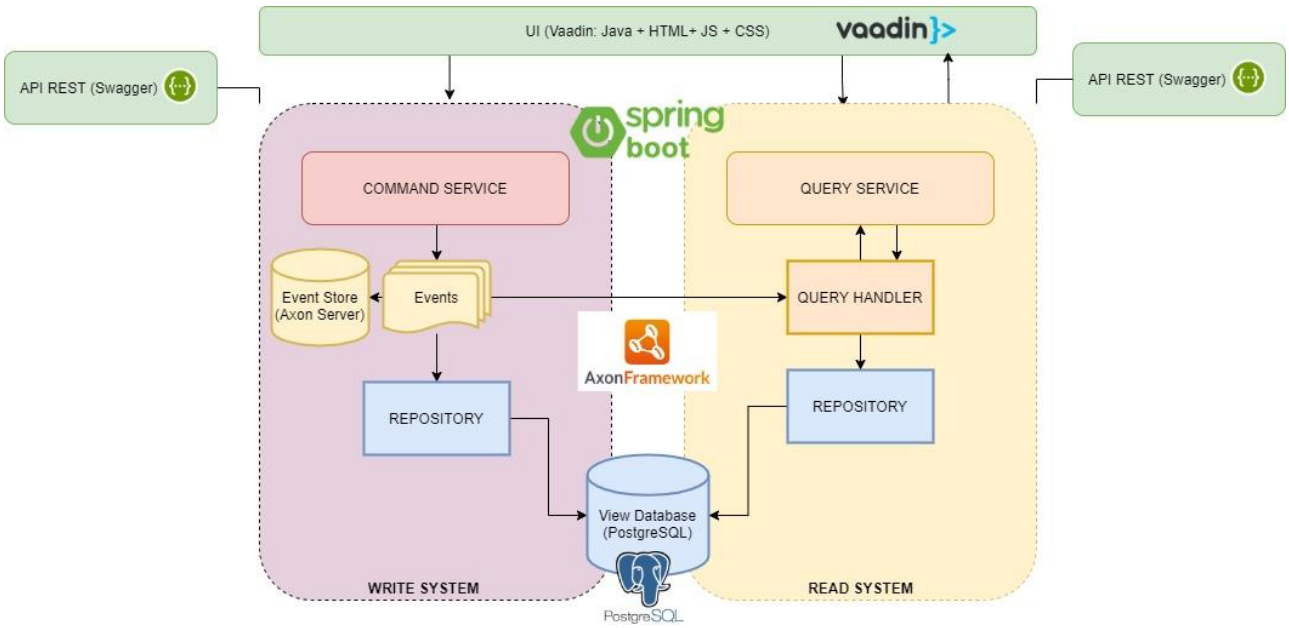


Ilustración 6 - Arquitectura global

4.2. Punto de vista de la información

En la *Ilustración 7 - Diagrama desde el punto de vista de la información* se describen las clases del sistema analizado, los atributos, tipos de datos y relaciones entre sí.

En el sistema se muestran los dos tipos de usuarios registrados (particular e inmobiliaria) que se han detallado en el apartado de la memoria dedicado a los roles y requisitos funcionales.

La entidad principal de nuestra aplicación es el inmueble, estos inmuebles solo pueden ser publicados por inmobiliarias y pueden o no tener imágenes asociadas.

Con las clases identificadas se puede modelar y cumplir con todos los casos de usos definidos.

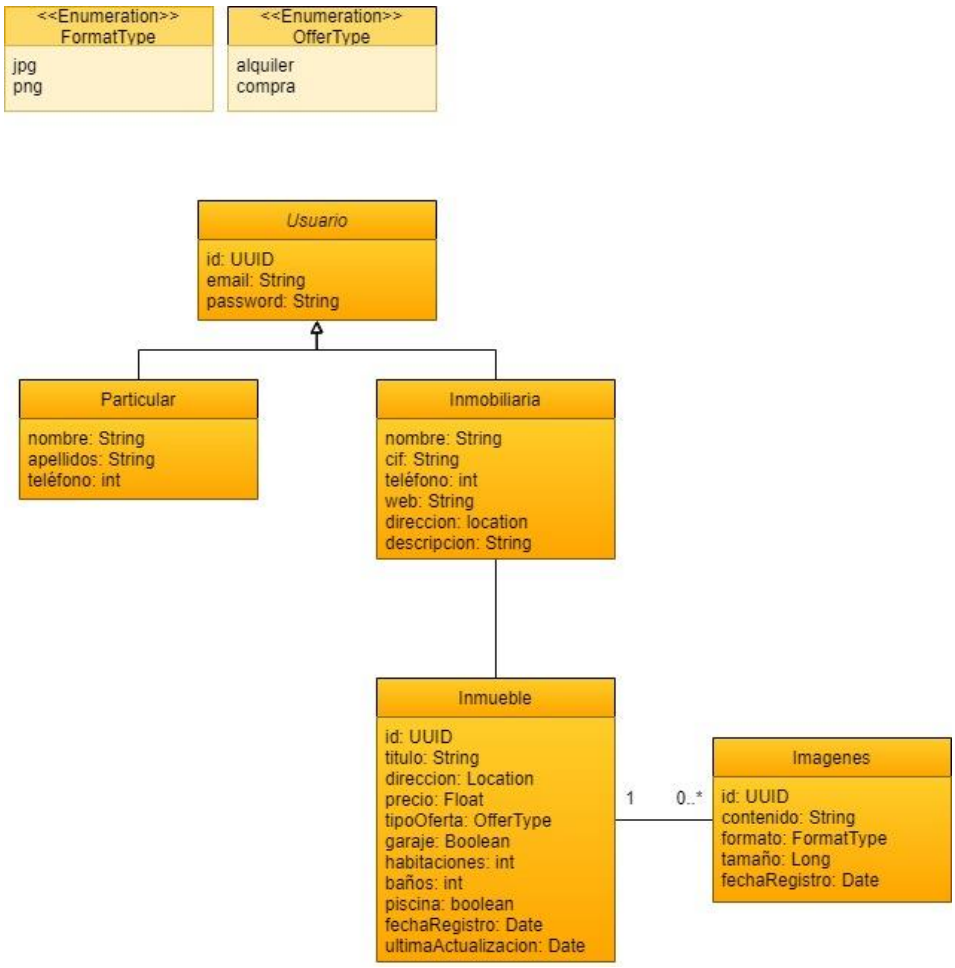


Ilustración 7 - Diagrama desde el punto de vista de la información

4.3. Punto de vista de la computación

En este diagrama de la *Ilustración 8 - Diagrama desde el punto de vista de la computación* se muestra un breve esquema con los diferentes componentes que intervienen en el flujo habitual de la aplicación, cada uno de estos objetos se detallarán en sucesivos capítulos.

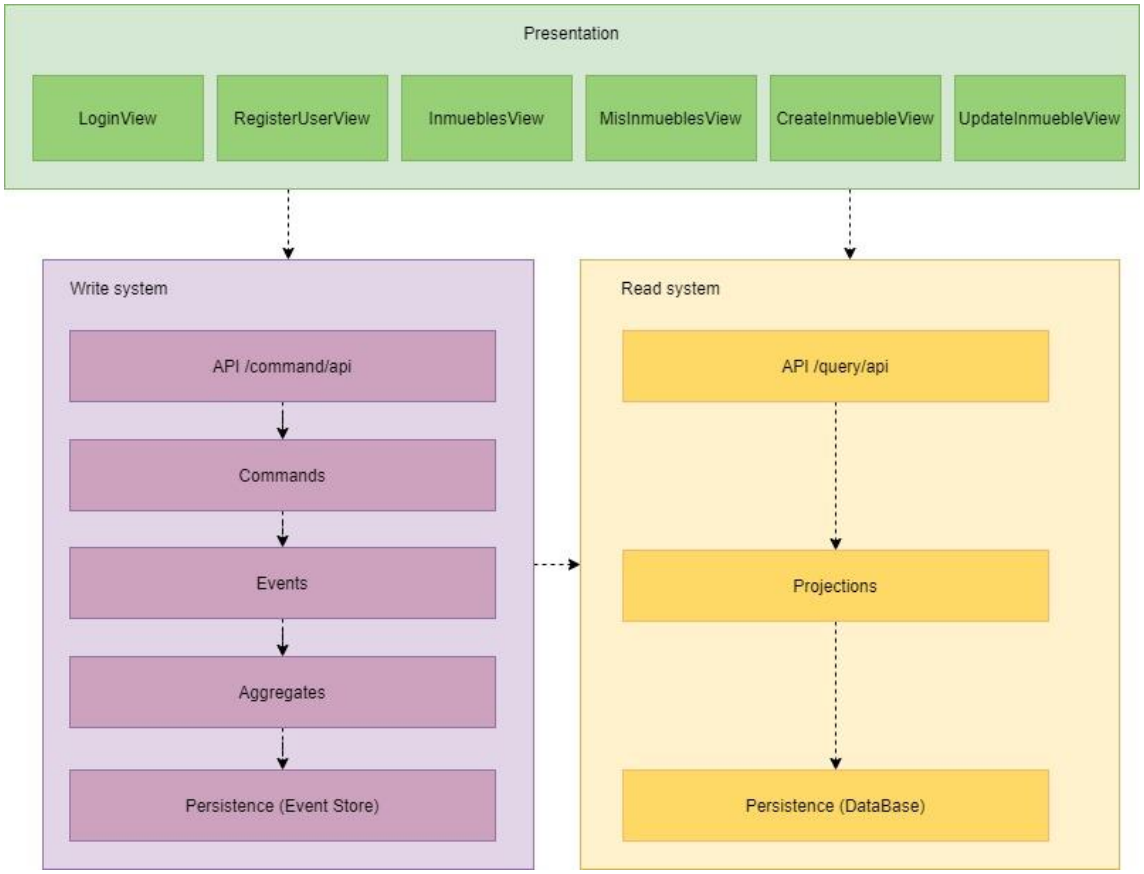


Ilustración 8 - Diagrama desde el punto de vista de la computación

4.4. Wireframes

En este apartado se muestran los bocetos a bajo nivel elaborados para describir de forma aproximada la distribución de elementos en las diferentes vistas del cliente web y las funcionalidades asociadas.

Se pretende que el cliente web sea un mero apoyo al conjunto del sistema ya que se proporcionan las APIs tanto de escritura como de consulta con la idea de desacoplar el core y clientes.

4.4.1. Pantalla principal

Esta es la pantalla principal desde la que se podrá buscar inmuebles por dirección y filtros de tipo (alquiler o compra), superficie, número de habitaciones, número de baños, precio…

La pantalla se divide en tres zonas, desde la parte superior se acceden a las opciones generales como información del usuario, menú para publicar un

anuncio, login, logout… A continuación, se observa la zona del buscador y filtros y por último el listado de resultados de la búsqueda.

Los resultados se muestran en pequeñas “tarjetas” con la información más relevante de cada inmueble. Desde cada tarjeta se puede acceder a la pantalla de detalle que se muestra en la *Ilustración 9 - Wireframe Pantalla principal*.

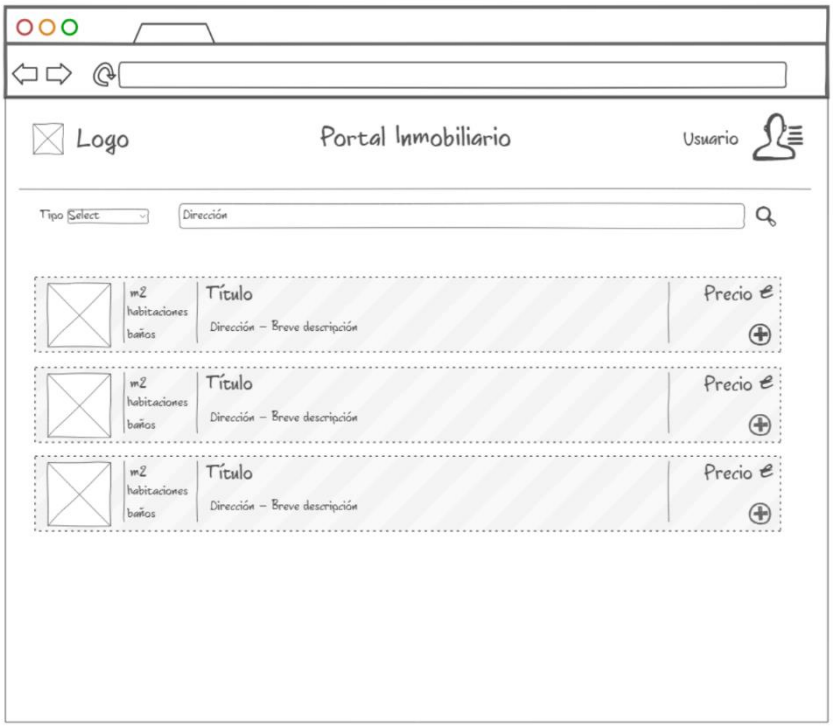


Ilustración 9 - Wireframe Pantalla principal

4.4.2. Detalle del anuncio

En esta pantalla vemos el detalle del anuncio compuesto por un *slide* de las fotografías del inmueble, toda la información disponible como título, descripción superficie, precio, etc. y los datos del anunciante.

Aprovechando la capacidad de la gestión y almacenamiento de eventos mostraremos la evolución del precio del inmueble en un gráfico independiente.

En el pie del detalle del inmueble se muestra un cuadro de texto mediante el cual los usuarios pueden solicitar información adicional al anunciante. Estos mensajes se enviarán mediante correo electrónico a la dirección de email del anunciante.

Se puede observar este boceto en las ilustraciones *Ilustración 10 - Wireframe Detalle del anuncio 01 e Ilustración 11 - Wireframe Detalle del anuncio 02.*

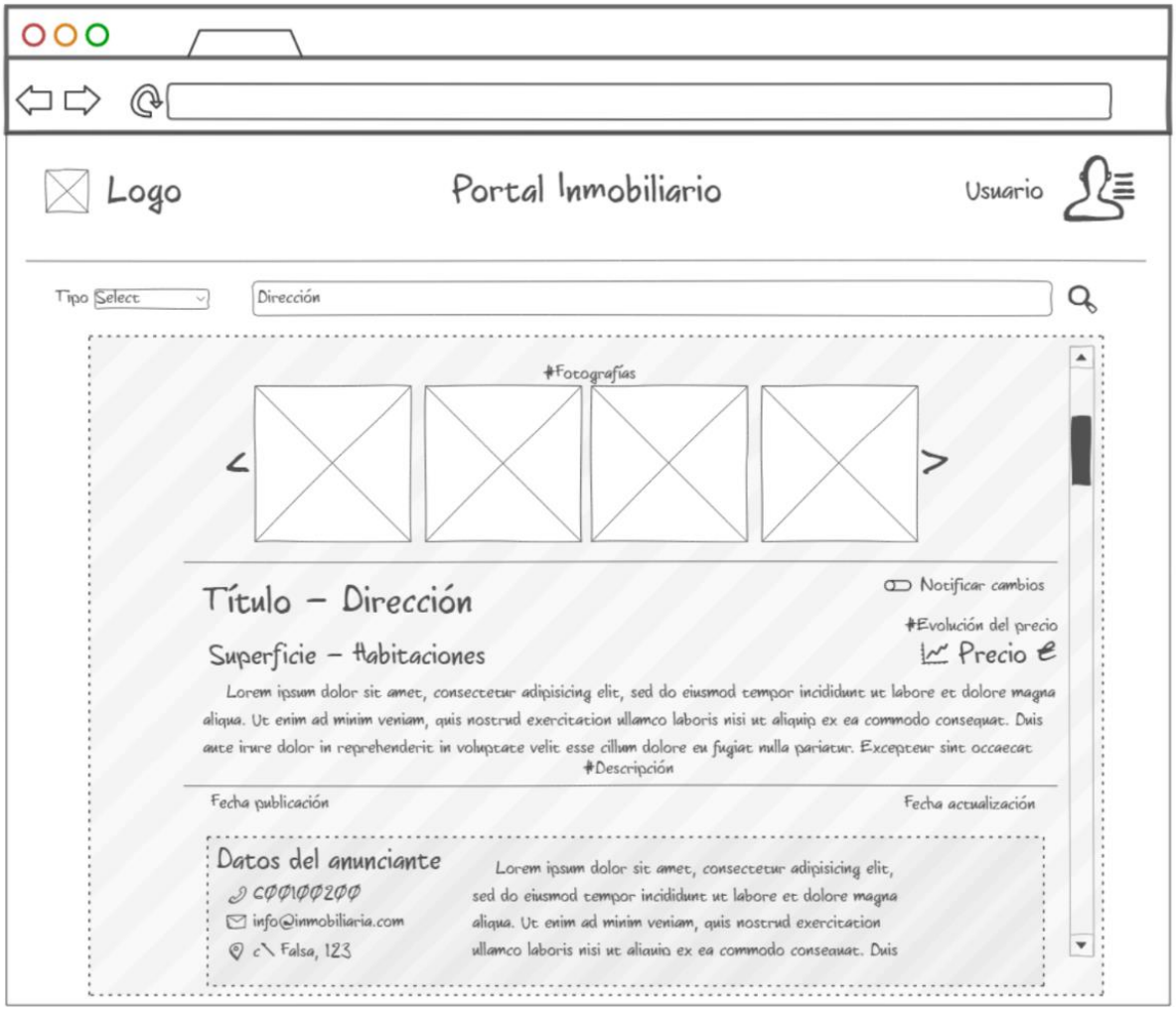


Ilustración 10 - Wireframe Detalle del anuncio 01

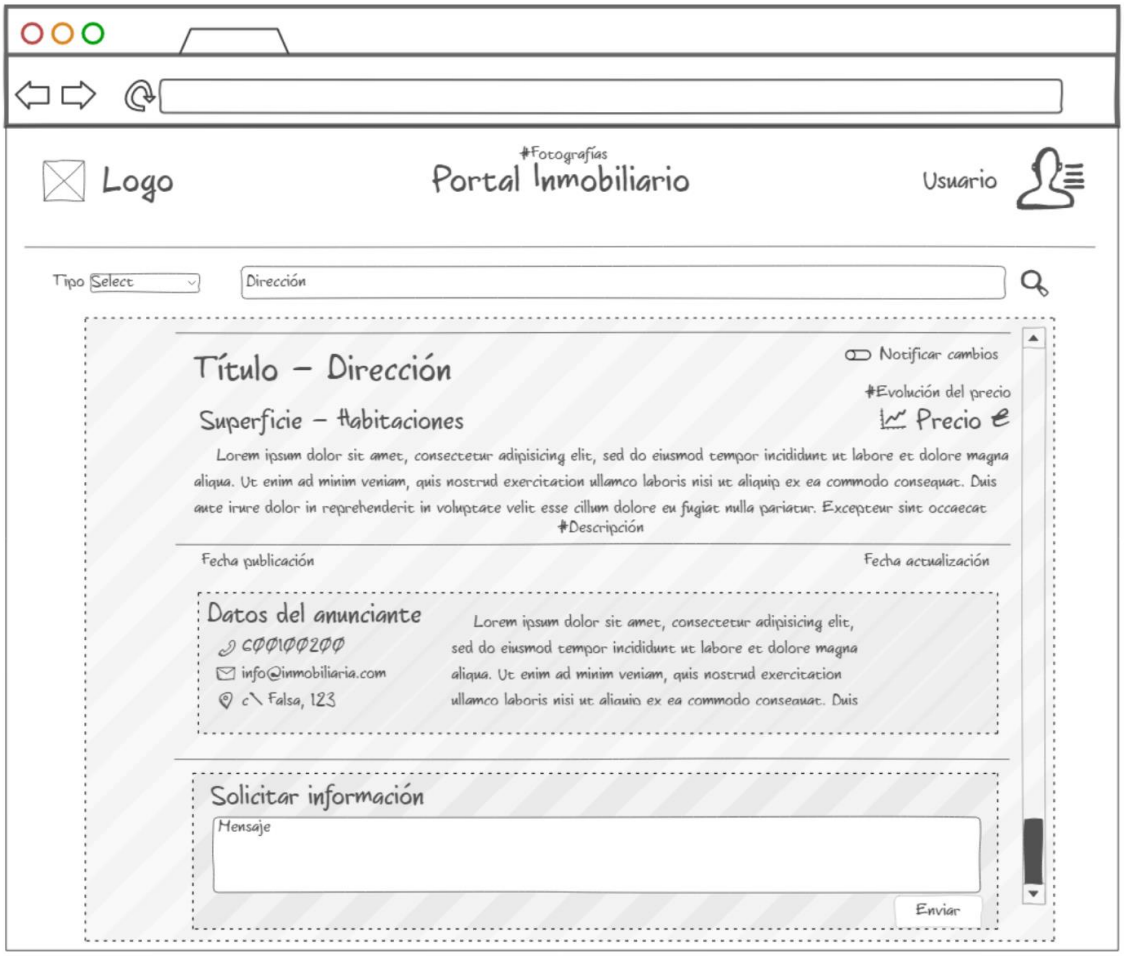


Ilustración 11 - Wireframe Detalle del anuncio 02

4.4.3. Menú Inmobiliaria

En la pantalla principal, si el usuario de tipo inmobiliaria ha iniciado sesión, haciendo clic en el icono de usuario se desplegará un menú con las opciones “Publicar un anuncio”, “Mis anuncios” y “Salir”.

La opción “Publicar un anuncio” lleva al formulario de alta de un inmueble, mientras que “Mis anuncios” mostrará un listado de los inmuebles registrados por el usuario desde el cual podrá editar o eliminar dicho inmueble.

Estas pantallas se tratarán con detalle en puntos posteriores.

Esta distribución se muestra en la *Ilustración 12 - Wireframe Menú inmobiliaria*.

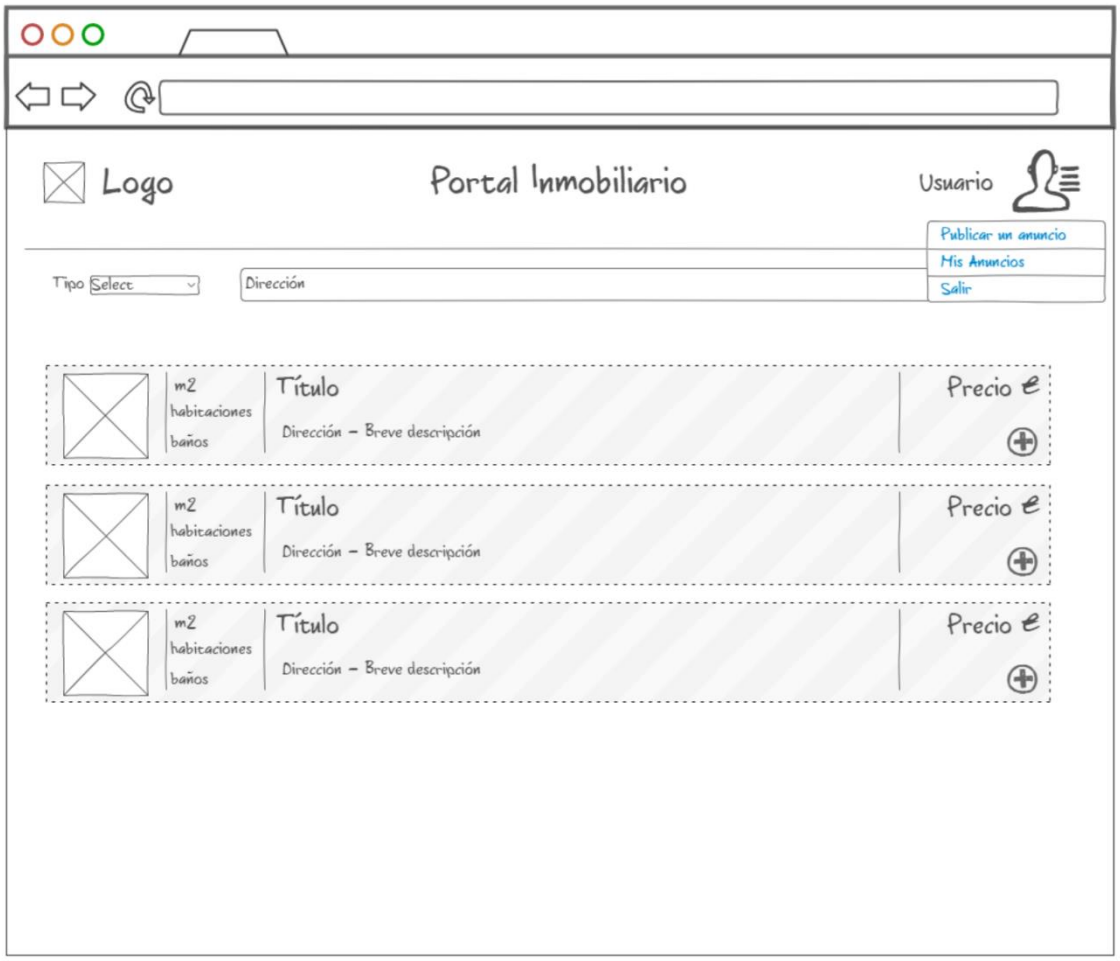


Ilustración 12 - Wireframe Menú inmobiliaria

4.4.4. Listado “Mis anuncios”.

Desde esta pantalla una inmobiliaria podrá visualizar, editar y eliminar los inmuebles que ha publicado previamente.

Cada ítem está compuesto por una “tarjeta” con la información resumida del anuncio tal y como aparece en el apartado Pantalla principal*4.4.1 Pantalla* principal

En cada tarjeta se muestra un botón para editar, que llevará a la pantalla 4.4.5 Formulario de alta/modificación de un inmueble. precargando el formulario con la información del inmueble seleccionado; y un botón para eliminar el anuncio, el cual solicitará confirmación previa.

El boceto de esta pantalla se muestra en la *Ilustración 13 - Wireframe Listado “Mis anuncios”.*

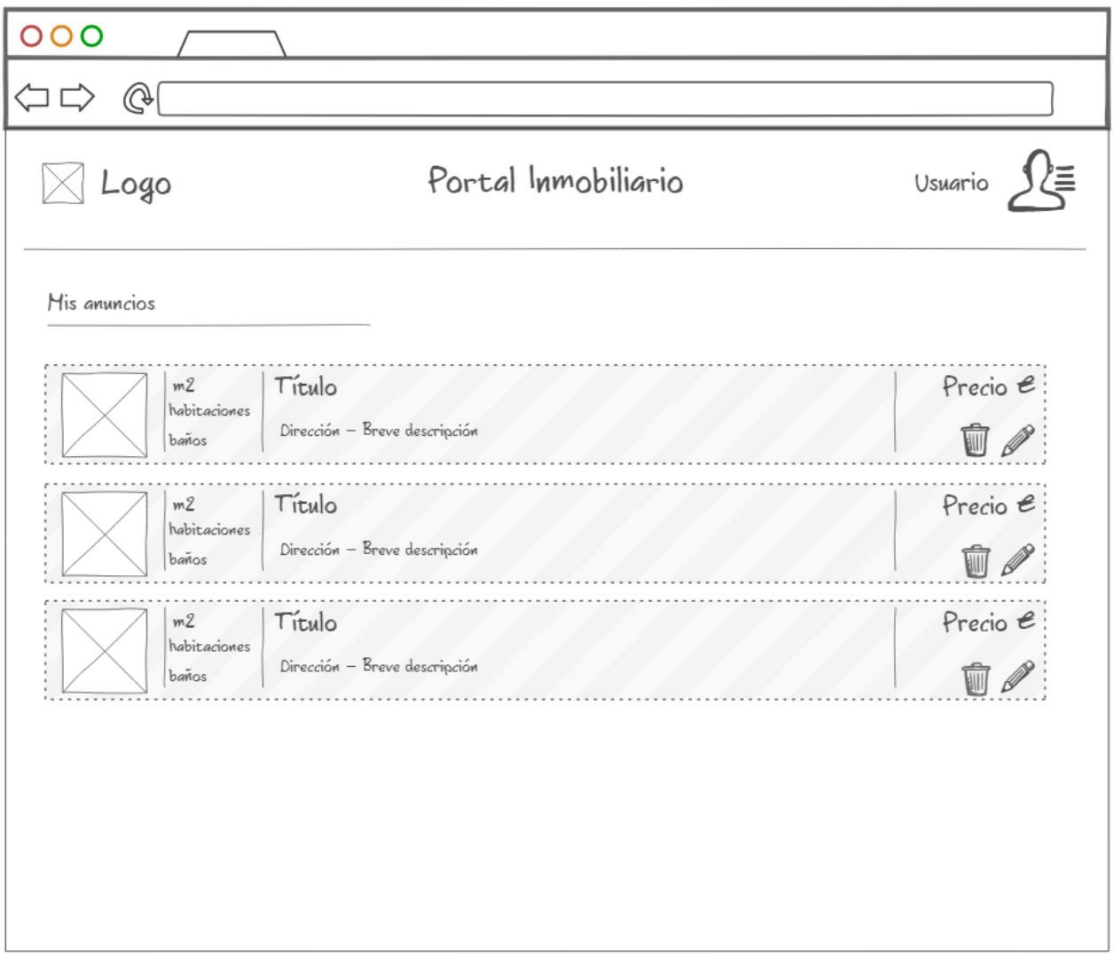


Ilustración 13 - Wireframe Listado “Mis anuncios”

4.4.5. Formulario de alta/modificación de un inmueble.

En la *Ilustración 14 - Wireframe Formulario de alta/modificación de un inmueble* se puede ver un boceto del formulario mediante el cual una inmobiliaria puede crear o editar un inmueble.

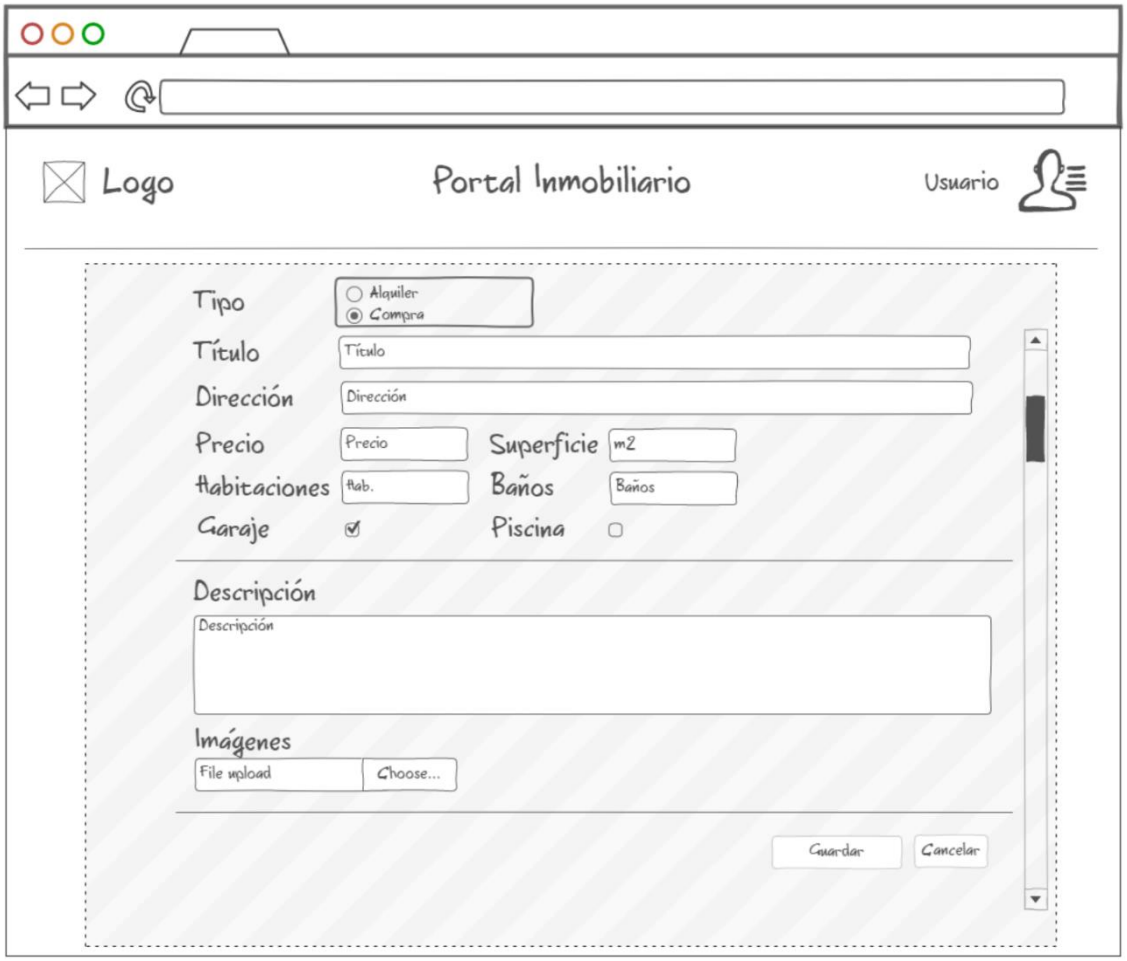


Ilustración 14 - Wireframe Formulario de alta/modificación de un inmueble

5. Tecnologías

En este capítulo se detallan las diferentes tecnologías usadas, así como la justificación de su elección.

- **JAVA:** la elección de este lenguaje viene dada por el conocimiento previo del que dispongo y la cantidad de documentación y ayuda disponible. Se ha usado la versión de JDK 11.

- **Spring Boot:** es, posiblemente, la herramienta más usada actualmente para simplificar la configuración y arranque de un proyecto Spring. Desde su inicializador podemos seleccionar las dependencias deseadas y generar un proyecto base desde el cuál comenzar a desarrollar nuestra aplicación.

- **Spring Framework:** Spring es un Framework para JAVA que nos simplifica la tarea de desarrollar una aplicación JAVA aplicando los patrones MVC, Singleton (mediante la inyección de dependencias), POJO, etc. También facilita el acceso a datos o la realización de test unitarios.

- **Axon IQ Framework y Axon Server:** Axon IQ [7] es un framework JAVA de código abierto y totalmente compatible con Spring que se basa en la aplicación del diseño dirigido por dominio (DDD), CQRS y el patrón de arquitectura Event sourcing, su servidor, también open source, nos proporciona una event store mediante la cual podemos gestionar y consultar los eventos almacenados en nuestro sistema.

- **Swagger:** es un conjunto de herramientas de código abierto que mediante el uso de unas sencillas anotaciones nos permite documentar de forma estructurada e interactiva nuestras APIs REST.

- **Vaadin**[8]**:** es una plataforma open source para el desarrollo de aplicaciones web sobre java que incluye una serie de componentes web, herramientas y framework.

- **PostgreSQL:** es un sistema gestor de bases de datos *(SGBD)* relacional de código abierto, orientado a objetos.

6. Implementación

Siguiendo con la planificación inicial se ha optado por comenzar por la infraestructura y core de la aplicación, dejando en última instancia el desarrollo del cliente web.

Esta fase es la que más esfuerzo a requerido, entre otros motivos, por la falta de conocimiento previo en la materia, pero a su vez ha hecho que sea un proceso bastante enriquecedor.

En los siguientes apartados se expondrán las técnicas y herramientas usados en la implementación del proyecto.

6.1. Organización

Se ha hecho uso de Trello para el control del *backlog* de tareas, tal y como se ha detallado en el punto 1.4 Planificación del Trabajo.

Como repositorio de software y sistema de control de versiones he creado un repositorio privado en Github [9] y como cliente Git he usado la herramienta GitKraken [10] que proporciona una interfaz gráfica muy cuidada para ejecutar todas las órdenes necesarias para mantener sincronizado el contenido local con el repositorio. En la *Ilustración 15 - Listado de commits en Github* podemos ver una el listado de commits desde la interfaz web de Github, mientras que en la *Ilustración 16 - Listado de commits en GitKraken* podemos ver el mismo listado desde la aplicación de escritorio GitKraken.

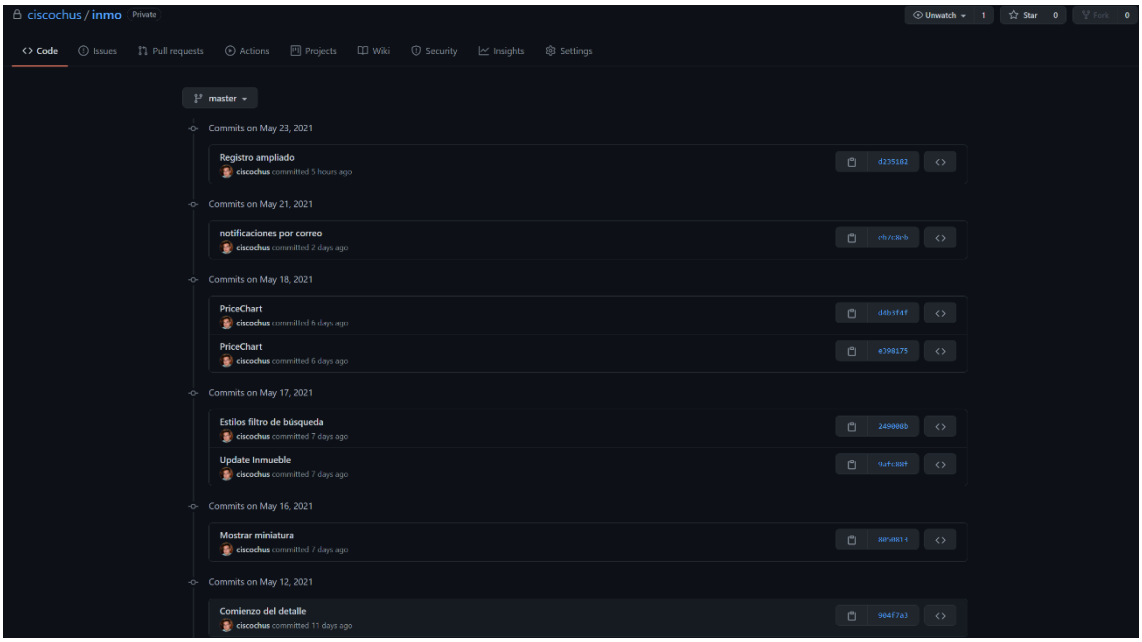


Ilustración 15 - Listado de commits en Github

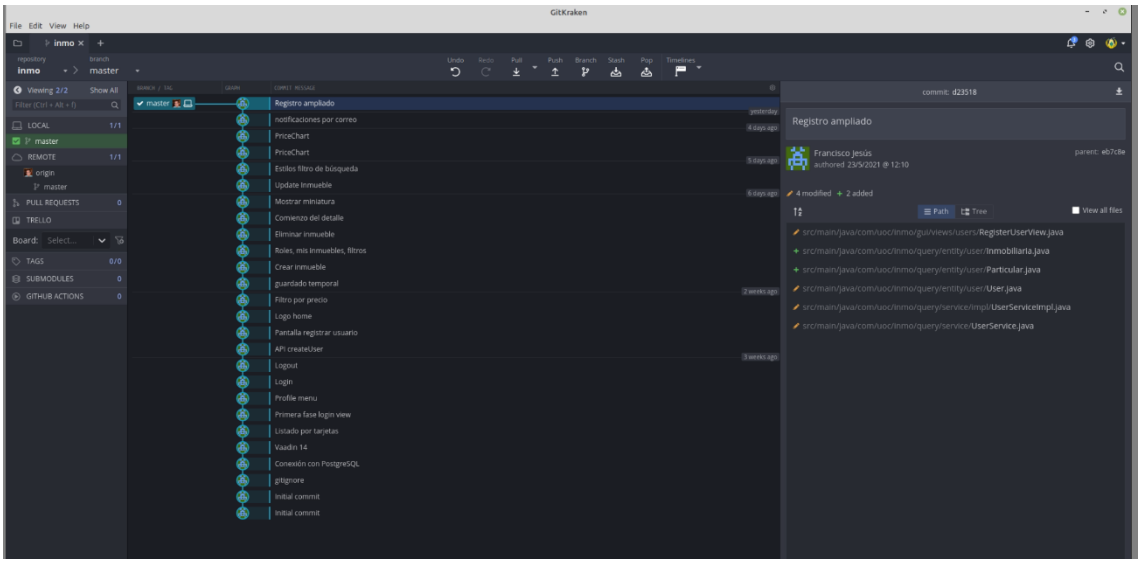


Ilustración 16 - Listado de commits en GitKraken

6.2. Entorno de desarrollo

Como IDE se ha escogido *Visual Studio Code*, editor de código *open source* que gracias a la gran cantidad de complementos permite una adaptación personal y una gran ayuda en la producción de código. En la *Ilustración 17 - Visual Studio Code* se puede observar la interfaz gráfica de *Visual Studio Code.*

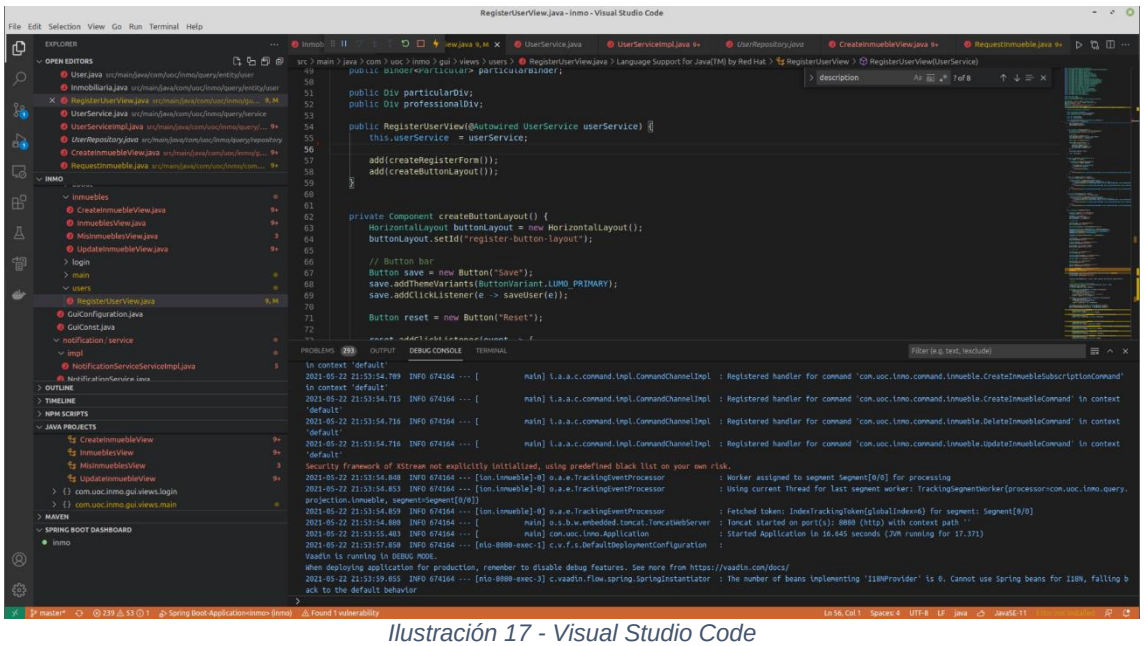


Ilustración 17 - Visual Studio Code

A nivel de BBDD se hace uso de *PgAdmin* para administrar la base de datos creada en PostgreSQL. Podemos ver una captura de *PgAdmin* en la *Ilustración 18 - PgAdmin.*

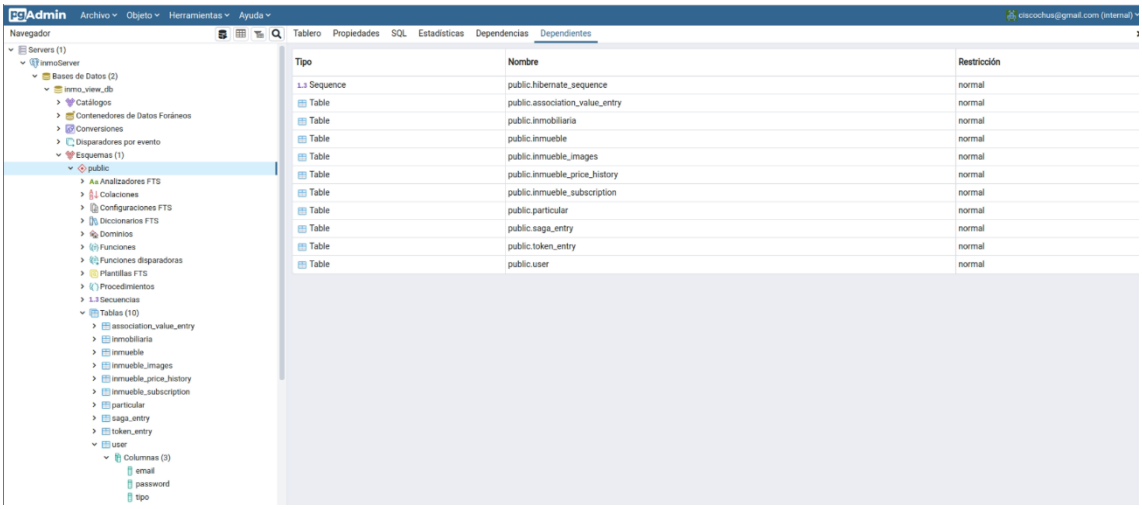


Ilustración 18 - PgAdmin

6.3. Implementación del sistema de escritura

Tal y como se explica en el apartado diseño de la arquitectura el sistema se divide en tres subsistemas: lectura, escritura y cliente web, en este punto se aborda la implementación del subsistema de escritura.

El subsistema de escritura se encarga de procesar los commands del patrón CQRS, en nuestro proyecto, estos commands tratan y procesan los eventos relacionados con la creación, actualización y borrado de inmuebles, los datos relacionados con la creación y gestión de usuarios queda fuera del tratamiento de estos eventos y se procesan directamente en el sistema de lectura.

6.3.1. API

Se implementan cinco *endpoints* bajo el *mapping route* **/api/command** para lanzar los comandos:

- CreateInmuebleCommand
- UpdateInmuebleCommand
- DeleteInmuebleCommand
- CreateInmuebleSubscriptionCommand
- DeleteInmuebleSubscriptionCommand

Estos comandos se detallarán en la sección correspondiente. Tanto estos endpoints como los del subsistema de consulta están documentados con swagger, tal y como se puede ver en la *Ilustración 19 - Swagger Command-API.*

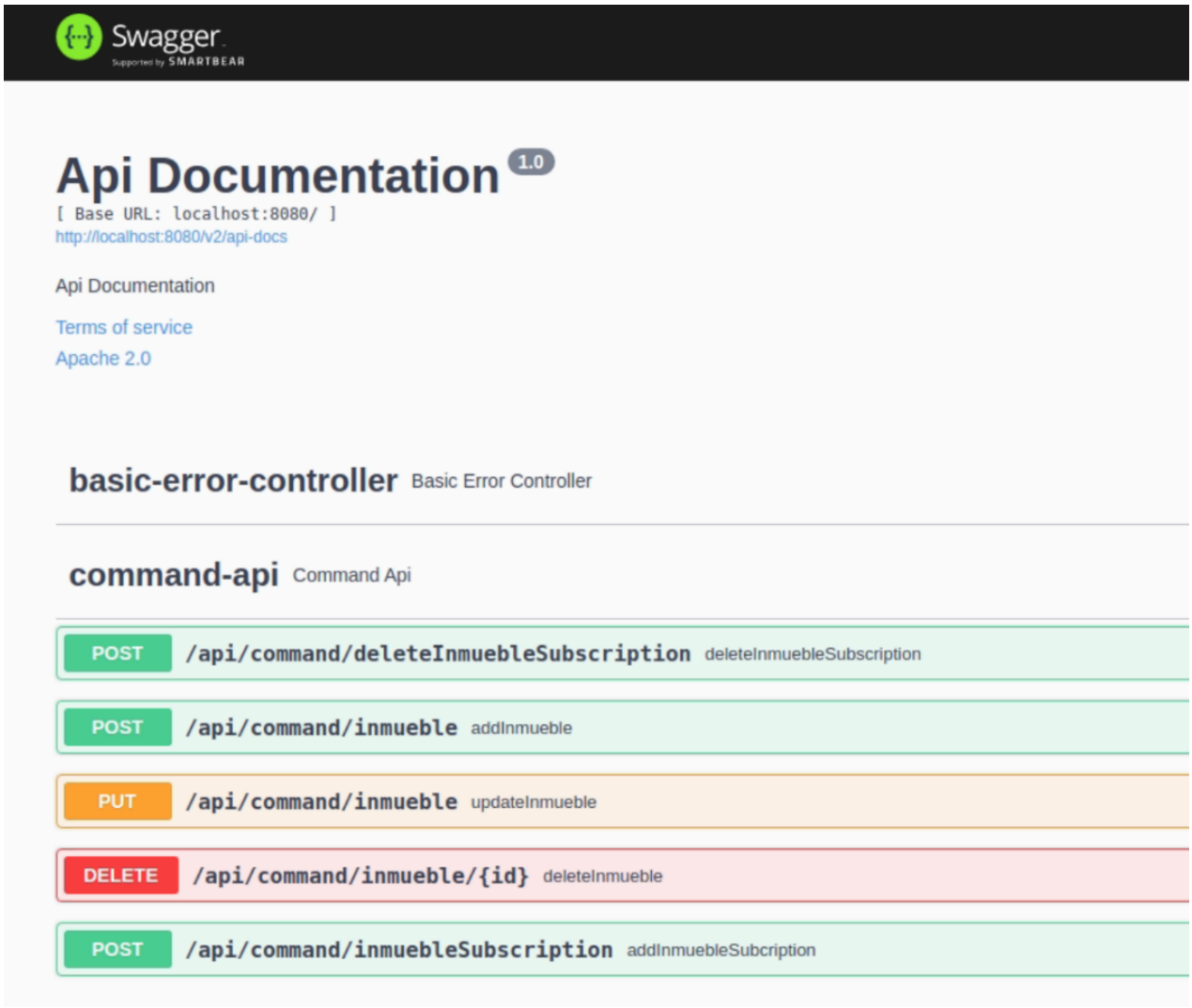


Ilustración 19 - Swagger Command-API

6.3.2. Commands y eventos

Los comandos o commands son objetos que representan las tareas o acciones básicas de nuestro sistema. En nuestro caso, solamente trabajamos con comandos relacionados con la entidad Inmueble.

Cuando se lanza un comando, este es capturado por un CommandHandler, el cual a su vez lanza el evento o los eventos derivados de esta acción.

Como ya se ha comentado, en nuestro sistema trabajamos con cinco eventos diferentes que separaremos en dos grupos, los asociados al aggregate InmuebleAggregate y los asociados al aggregate InmuebleSubscriptionAggregate, el concepto de agregados se detalla en el siguiente apartado.

Parte del código relacionado con la implementación de estos comandos se puede observar en las ilustraciones: *Ilustración 20 - Comandos de InmuebleAgregate* e *Ilustración 21 - Comandos de InmuebleSubscriptionAggregate.*

- Comandos de InmuebleAgregate:
  - CreateInmuebleCommand
  - UpdateInmuebleCommand
  - DeleteInmuebleCommand

```
@CommandHandler
public InmuebleAggregate(CreateInmuebleCommand command){
    log.entry(command);
    AggregateLifecycle.apply(new InmuebleCreatedEvent(command));
}

@CommandHandler
public void handle(UpdateInmuebleCommand command){
    log.entry(command);
    AggregateLifecycle.apply(new InmuebleUpdatedEvent(command));
}

@CommandHandler
public void handle(DeleteInmuebleCommand command){
    log.entry(command);
    AggregateLifecycle.apply(new InmuebleDeletedEvent(command.id));
}
```

Ilustración 20 - Comandos de InmuebleAgregate

- Comandos de InmuebleSubscriptionAggregate:
    - CreateInmuebleSubscriptionCommand
    - DeleteInmuebleSubscriptionCommand

```
@CommandHandler
public InmuebleSubscriptionAggregate(CreateInmuebleSubscriptionCommand command){
    log.entry(command);
    AggregateLifecycle.apply(new InmuebleSubscriptionCreatedEvent(command));
}

@CommandHandler
public void handle(DeleteInmuebleSubscriptionCommand command){
    log.entry(command);
    AggregateLifecycle.apply(new InmuebleSubscriptionDeletedEvent(command));
}
```

Ilustración 21 - Comandos de InmuebleSubscriptionAggregate

Estos comandos lanzan sus correspondientes eventos, estos eventos son objetos que, como su propio nombre indica, codifican la información del evento producido por la acción que acaba de ocurrir. Es decir, el comando CreateInmuebleCommand lanza el evento InmuebleCreatedEvent.

Este evento contiene toda la información del comando que lo crea y a diferencia de los comandos puede ser “capturado” en diferentes puntos de la aplicación simultáneamente.

En nuestro caso cada evento es capturado en dos puntos diferentes del sistema, en el subsistema de escritura, los eventos son capturados en el propio agregado por los EventSourcingHandlers los cuales se encargan de persistir dichos eventos en el EventStore mientras que, en el subsistema de lectura, en la clase InmuebleRepositoryProjection se capturan mediante un EventHandler con el objetivo de mantener la última versión del catálogo de inmuebles actualizada en la base de datos de lectura.

En las ilustraciones *Ilustración 22 - EventSourcingHandlers de Inmueble* e *Ilustración 23 - EventSourcingHandlers de InmuebleSubscription* se muestra los fragmentos de código encargados de capturar en primera instancia los eventos creados por los commands.

```java
@EventSourcingHandler
public void on(InmuebleCreatedEvent event){
    log.entry(event);
    id = event.id;
    title = event.title;
    address = event.address;

    price = event.price;
    area = event.area;
    type = event.type;

    garage = event.garage;
    pool = event.pool;

    rooms = event.rooms;
    baths = event.baths;

    description = event.description;

    email = event.email;

    images = event.images;

    log.trace("new state of aggregate: {}", this);
}
```

Ilustración 22 - EventSourcingHandlers de Inmueble

```java
@EventSourcingHandler
public void on(InmuebleSubscriptionCreatedEvent event){
    log.entry(event);
    id = event.id;
    inmuebleId = event.inmuebleId;
    email = event.email;

    log.trace("new state of aggregate: {}", this);
}

@EventSourcingHandler
public void on(InmuebleSubscriptionDeletedEvent event){
    log.entry(event);
    id = event.id;
    AggregateLifecycle.markDeleted();
    log.trace("new state of aggregate: {}", this);
}
```

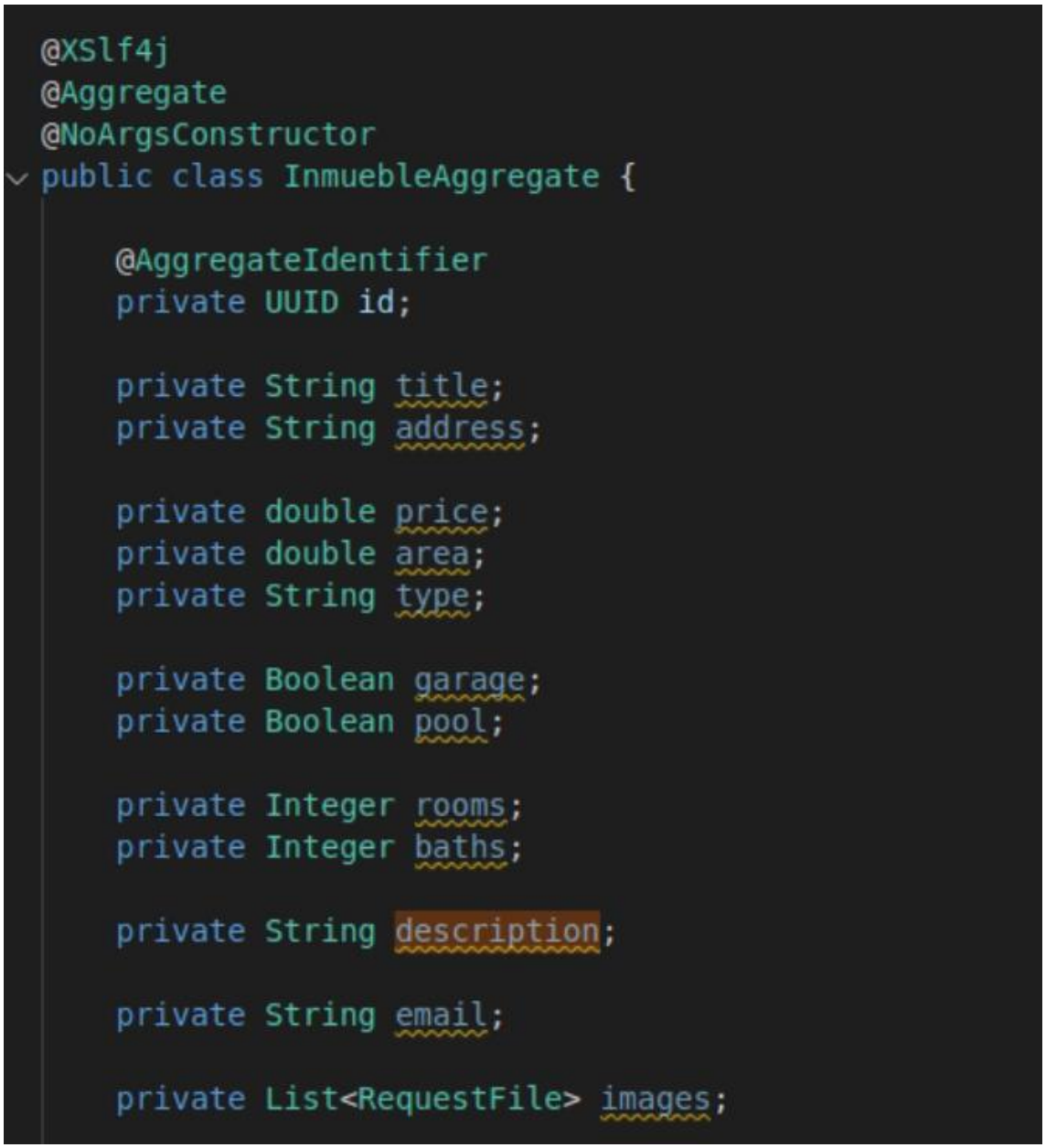
Ilustración 23 - EventSourcingHandlers de InmuebleSubscription

6.3.3. Aggregates

Los agregados o *aggregates* son entidades que representan el estado de la información ante cada evento que las modifique. Es decir, cada evento que provoca un cambio sobre un agregado crea una especie de *snapshot* del estado del agregado en ese momento, por lo que podemos ver todos los eventos en orden cronológico que han modificado el objeto desde su creación hasta el momento actual.

En nuestro caso trabajamos con dos tipos de agregados diferentes, *InmuebleAggregate* que codifica la información de cada inmueble e *InmuebleSubscriptionAggregate* que organiza la información de las subscripciones a los diferentes inmuebles.

En las ilustraciones *Ilustración 24 - InmuebleAggregate e Ilustración 25 – InmuebleSubscriptionAggregate* se muestran unos fragmentos de código de la implementación de los *aggregates* mencionados.



```java
@XSlf4j
@Aggregate
@NoArgsConstructor
public class InmuebleAggregate {

    @AggregateIdentifier
    private UUID id;

    private String title;
    private String address;

    private double price;
    private double area;
    private String type;

    private Boolean garage;
    private Boolean pool;

    private Integer rooms;
    private Integer baths;

    private String description;

    private String email;

    private List<RequestFile> images;
```

Ilustración 24 - InmuebleAggregate

```java
@XSlf4j
@Aggregate
@NoArgsConstructor
public class InmuebleSubscriptionAggregate {

    @AggregateIdentifier
    private UUID id;

    private UUID inmuebleId;
    private String email;
```

Ilustración 25 – InmuebleSubscriptionAggregate

Gracias al servidor de Axon, el cual hace las veces de coordinador de eventos y de almacén de los mismo, podemos visualizar gráficamente todos los eventos lanzados.

En la *Ilustración 26 - Dashboard Axon server* podemos ver el dashboard de Axon server mostrando los comandos detectados por el sistema.

En la *Ilustración 27 - Detalle de un evento* se observa el listado y detalle de los eventos lanzados, así como del agregado asociado a dicho evento.

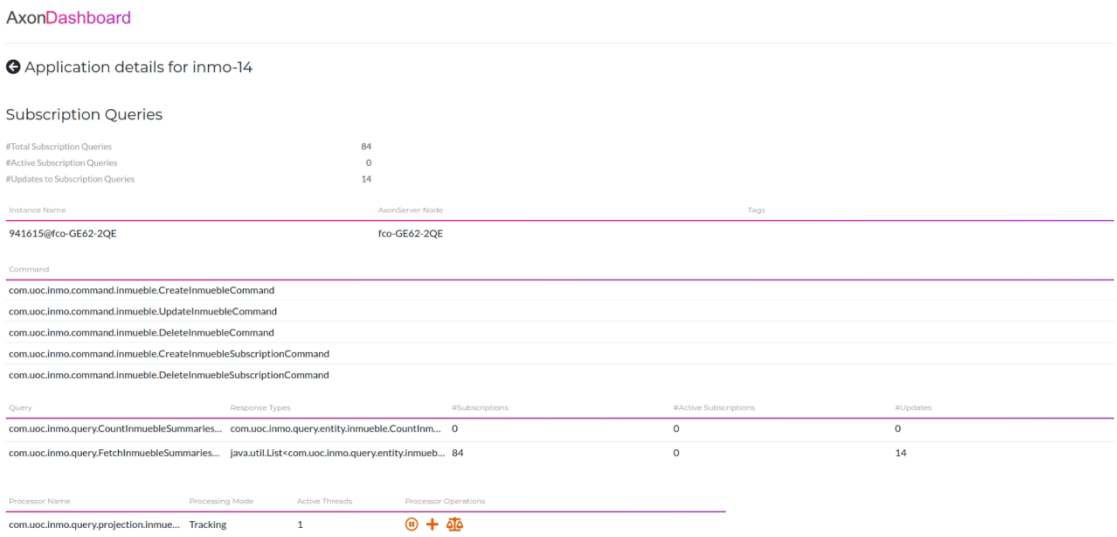


Ilustración 26 - Dashboard Axon server

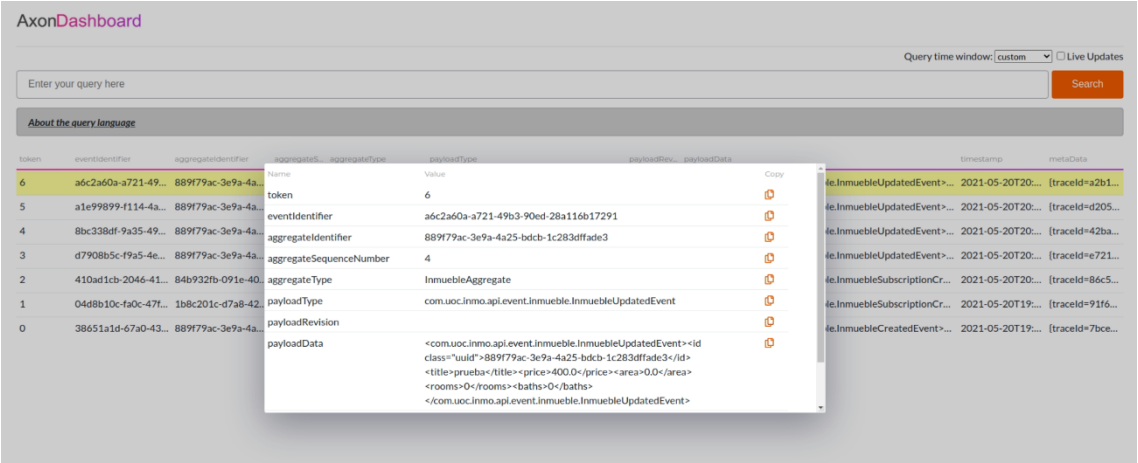


Ilustración 27 - Detalle de un evento

6.4. Implementación del sistema de lectura

Este subsistema tiene una tarea doble, capturar los eventos lanzados por los commands en la capa de escritura para mantener actualizada la base de datos y exponer la información para que pueda ser consumida por terceros.

Adicionalmente, también “notifica” de cambios en el catálogo al cliente web para que actualice la vista correspondiente ante un cambio en el listado de inmuebles.

6.4.1. API

Al igual que ocurre en el subsistema de escritura, hemos expuesto una serie se endpoints mediante una API REST bajo el route mapping /api/query con la finalidad de poder consultar los datos del sistema, así como crear usuarios, que tal y como hemos comentado en el apartado dedicado a la escritura y solamente por simplificar la complejidad del proyecto hemos decidido no tratarlos como agregados y limitarnos solamente a procesar los eventos dedicados al catálogo de inmuebles.

En la *Ilustración 28 - Swagger Query-API* puede observarse una captura de swagger donde se muestran los endpoints de la capa de lectura.

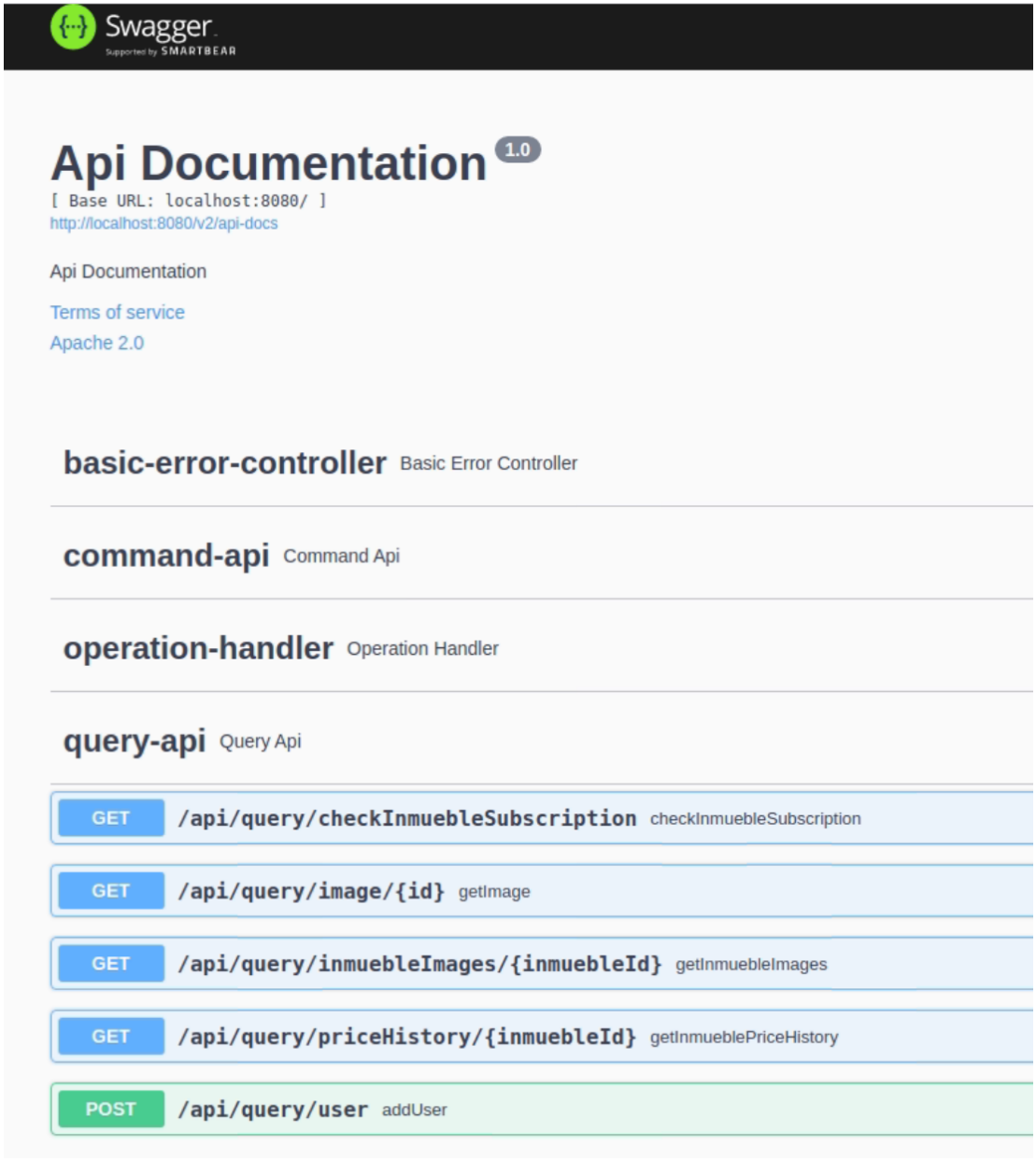


Ilustración 28 - Swagger Query-API

6.4.2. Proyecciones

En el sistema de lectura se ha implementado la clase *InmuebleRepositoryProjection* cuya finalidad es la de capturar los eventos lanzados por el sistema de escritura y persistir, haciendo uso de JPA, las entidades afectadas en la base de datos de lectura. Esto permite que la base de datos siempre se encuentre actualizada.

En la *Ilustración 29 - EventHandler* podemos ver la implementación del *EventHandler* encargado de capturar los eventos de tipo *InmuebleCreatedEvent* y persistirlos en la base de datos.

Cuando se crea un nuevo inmueble o se actualiza alguno existente se emite una *Query* que a través del *gateway* del propio servidor de *Axon* comunica a los subscriptores del cambio. Uno de estos subscriptores se encuentra en la vista de listado de inmuebles codificada en el subsistema del cliente web.

```
@EventHandler
public void on(InmuebleCreatedEvent event){
    InmuebleSummary entity = new InmuebleSummary();

    entity.setId(event.getId());

    entity.setTitle(event.getTitle());
    entity.setAddress(event.getAddress());
    entity.setPrice(event.getPrice());
    entity.setArea(event.getArea());
    entity.setGarage(event.getGarage());
    entity.setPool(event.getPool());
    entity.setRooms(event.getRooms());
    entity.setBaths(event.getBaths());
    entity.setType(event.getType());
    entity.setDescription(event.getDescription());

    entity.setEmail(event.getEmail());

    entity.setCreated(new Date());
    entity.setUpdated(new Date());

    entity.setImages(convertToInmuebleImagesList(event.getImages(), entity));

    inmuebleSummaryRepository.save(entity);

    queryUpdateEmitter.emit(CountInmuebleSummariesQuery.class,
        query -> event.getId().toString().startsWith(""),
        new CountChangedUpdate());

    queryUpdateEmitter.emit(FetchInmuebleSummariesQuery.class,
        query -> event.getId().toString().startsWith(""),
        entity);

    log.trace("new inmueble projection saved: {}", entity);
}
```

Ilustración 29 - EventHandler

6.5. Implementación del cliente web

La idea inicial de este componente era la de crear un cliente web simple con el que poder probar las diferentes funcionalidades del sistema.

Se ha optado por una interfaz de diseño minimalista y adaptable a diferentes tipos de dispositivos.

Se ha hecho uso del framework de java Vaadin por su compatibilidad con el ecosistema Spring, aunque inicialmente debido a la falta de documentación se descartó el uso de las últimas versiones de Vaadin, tras un largo proceso de investigación y aprendizaje se ha podido usar las últimas versiones de todos los componentes elegidos.

El tratamiento de sesiones de usuarios se ha implementado mediante Spring-security (*Ilustración 30 - Configuración Spring Security*).

```
@EnableWebSecurity
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;

    @Autowired
    public SecurityConfiguration(UserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /**
     * Registers our UserDetailsService and the password encoder to be used on login attempts.
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        super.configure(auth);
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .requestCache().requestCache(new CustomRequestCache())

            // Restrict access to our application.
            .and().authorizeRequests().antMatchers("/ui/signup", "/ui/inmuebles", "/ui/inmuebles/*", "/swagger-ui/*", "/api/*").permitAll()

            // Allow all flow internal requests.
            .requestMatchers(SecurityUtils::isFrameworkInternalRequest).permitAll()

            // Allow all requests by logged in users.
            .anyRequest().hasAnyAuthority(Role.getAllRoles())
            .and().formLogin()
            .loginPage(GuiConst.LOGIN_URL).permitAll()
            .loginProcessingUrl(GuiConst.LOGIN_PROCESSING_URL)
```

Ilustración 30 - Configuración Spring Security

Se han implementado las pantallas:

- Login
- Registro de usuario
- Listado de inmuebles
- Mis inmuebles
- Creación de un inmueble
- Detalle de inmueble

Y el componente histórico de precios. Se puede ver un detalle del cliente en la *Ilustración 31 - Detalle de un inmueble*.

Ilustración 31 - Detalle de un inmueble

Se muestran con más detalle las diferentes pantallas en el capítulo *8. Resultados.*

7. Pruebas

En este capítulo se detallan las pruebas funcionales realizadas en base a los requisitos planteados en el capítulo 3.2 Requisitos y casos de uso detallados en el capítulo 3.3 Casos de uso.

Tabla 9 - Prueba 01

Prueba 01	
CU – RF01	Consultar inmuebles
Escenario	Principal
Descripción	El usuario accede a la aplicación. Se muestra la pantalla principal con el buscador y filtros. El usuario introduce los criterios de búsqueda. El usuario pulsa el botón “buscar”. El usuario introduce unos criterios de búsqueda para los que existen inmuebles que los cumplen.
Comportamiento esperado	El sistema muestra en la misma pantalla el listado de inmuebles con los criterios establecidos.
Resultado	OK

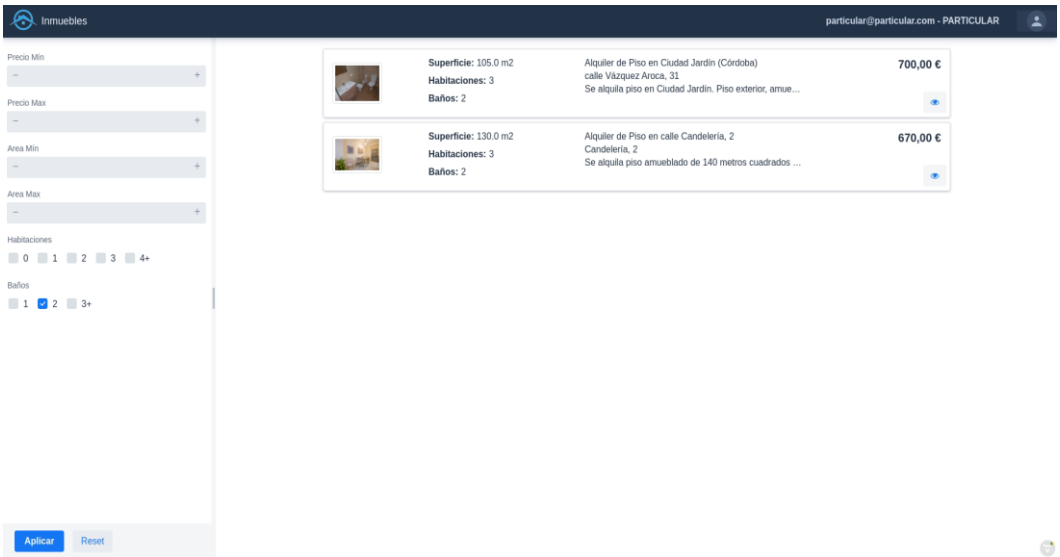


Ilustración 32- Resultado Prueba 01

Tabla 10 - Prueba 02

Prueba 02	
CU – RF01	Consultar inmuebles
Escenario	Alternativo
Descripción	El usuario accede a la aplicación. Se muestra la pantalla principal con el buscador y filtros. El usuario introduce los criterios de búsqueda. El usuario pulsa el botón “buscar”. El usuario introduce unos criterios de búsqueda para los que no existen inmuebles que los cumplen.
Comportamiento esperado	El sistema muestra el mensaje “No hay resultados”.
Resultado	OK

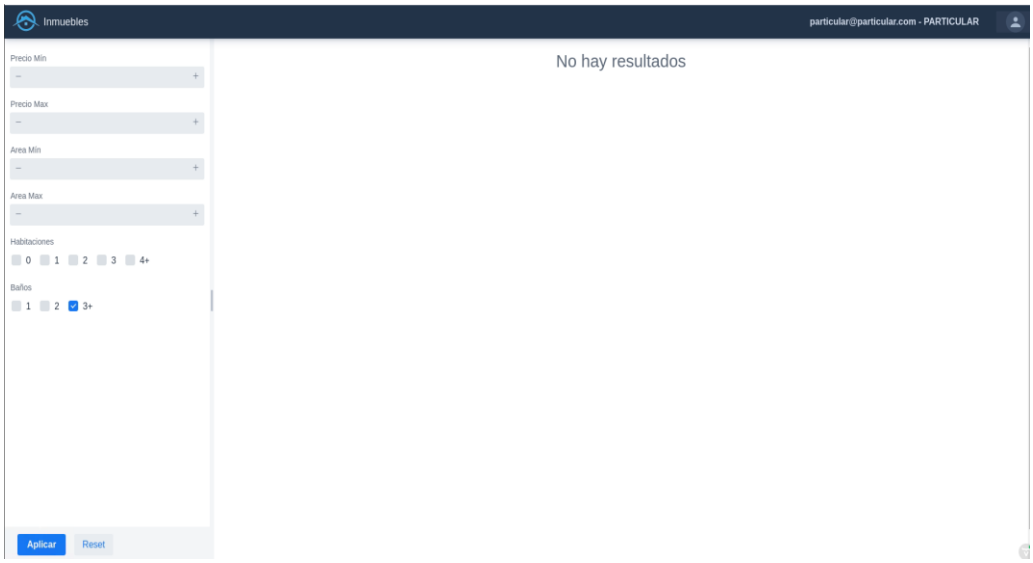


Ilustración 33 - Resultado Prueba 02

Tabla 11 - Prueba 03

Prueba 03	
CU – RF02	Suscripción / Desuscripción a inmuebles
Escenario	Principal
Descripción	El usuario ha iniciado sesión y abre el detalle de un inmueble al que no se encuentra subscrito. El usuario activa la opción “Notificar cambios”.
Comportamiento esperado	El sistema muestra un mensaje de confirmación “Se ha suscrito a este inmueble”.
Resultado	OK

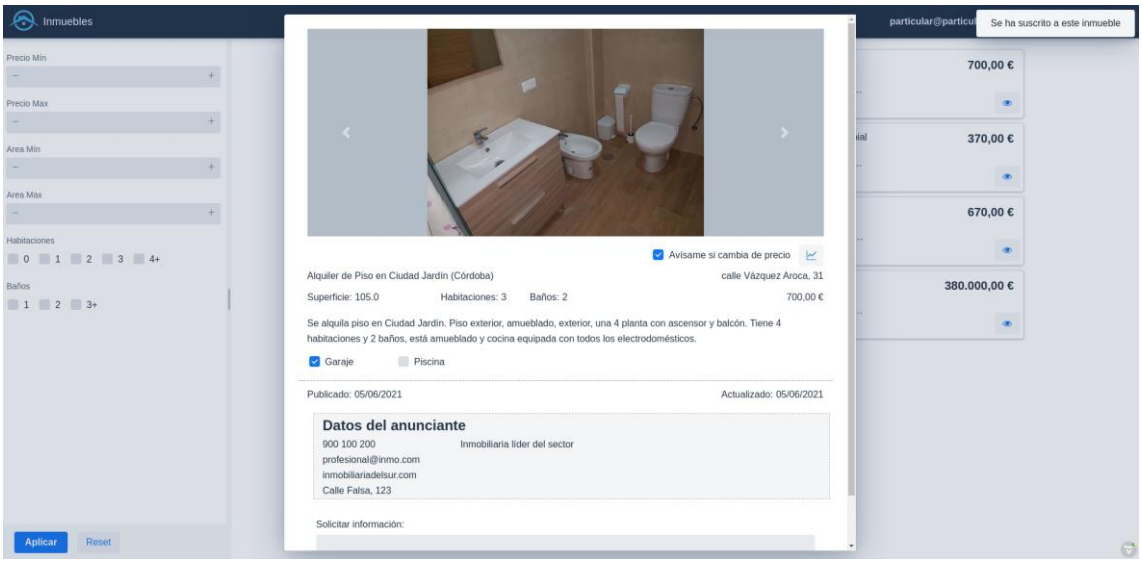


Ilustración 34 - Resultado Prueba 03

Tabla 12 - Prueba 04

Prueba 04	
CU – RF02	Suscripción / Desuscripción a inmuebles
Escenario	Alternativo
Descripción	El usuario ha iniciado sesión y abre el detalle de un inmueble al que se encuentra subscrito. El usuario desactiva la opción “Notificar cambios”.
Comportamiento esperado	El sistema muestra un mensaje de confirmación “No recibirá notificaciones de este inmueble”.
Resultado	OK

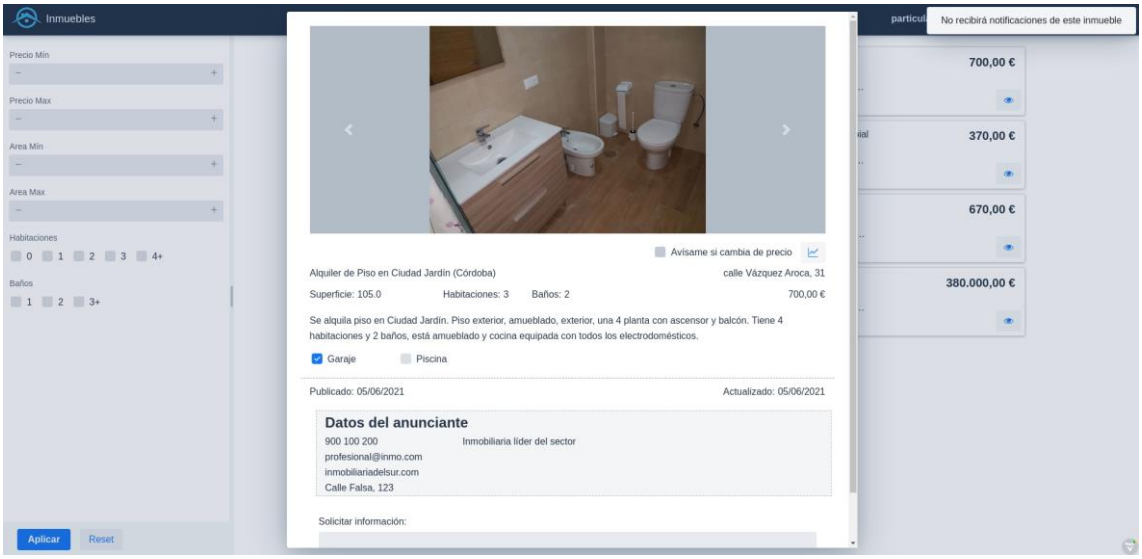


Ilustración 35 - Resultado Prueba 04

Tabla 13 - Prueba 05

Prueba 05	
CU – RF03 – 01	Crear un inmueble
Escenario	Principal
Descripción	El usuario de tipo *Inmobiliaria* ha iniciado sesión, accede a la pantalla de alta de un inmueble, introduce datos correctos y pulsa el botón “Guardar”
Comportamiento esperado	El sistema muestra un mensaje de confirmación “Cambios guardados correctamente”.
Resultado	OK

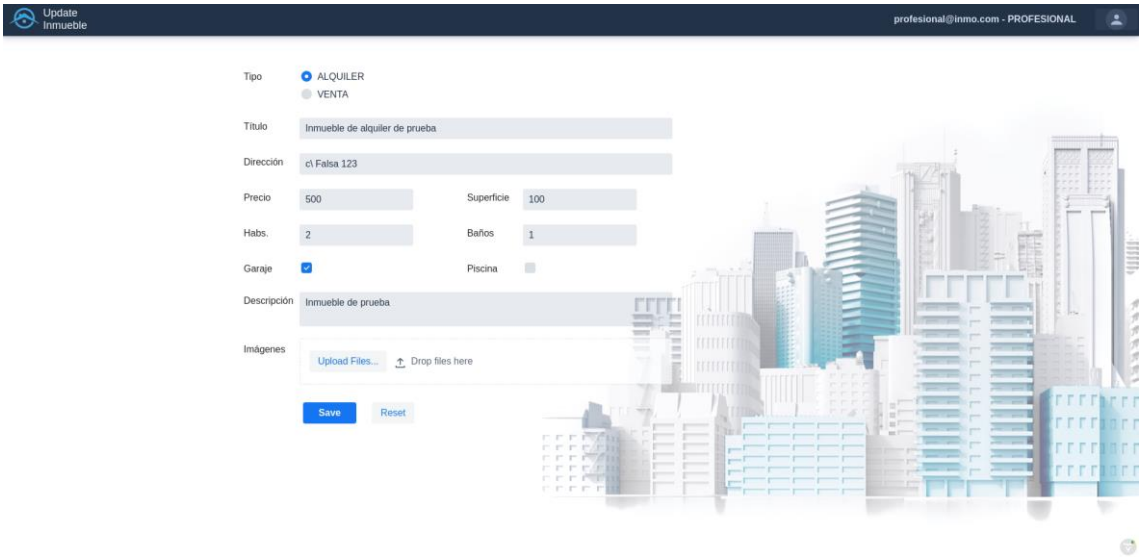


Ilustración 36 - Resultado Prueba 05 - 01

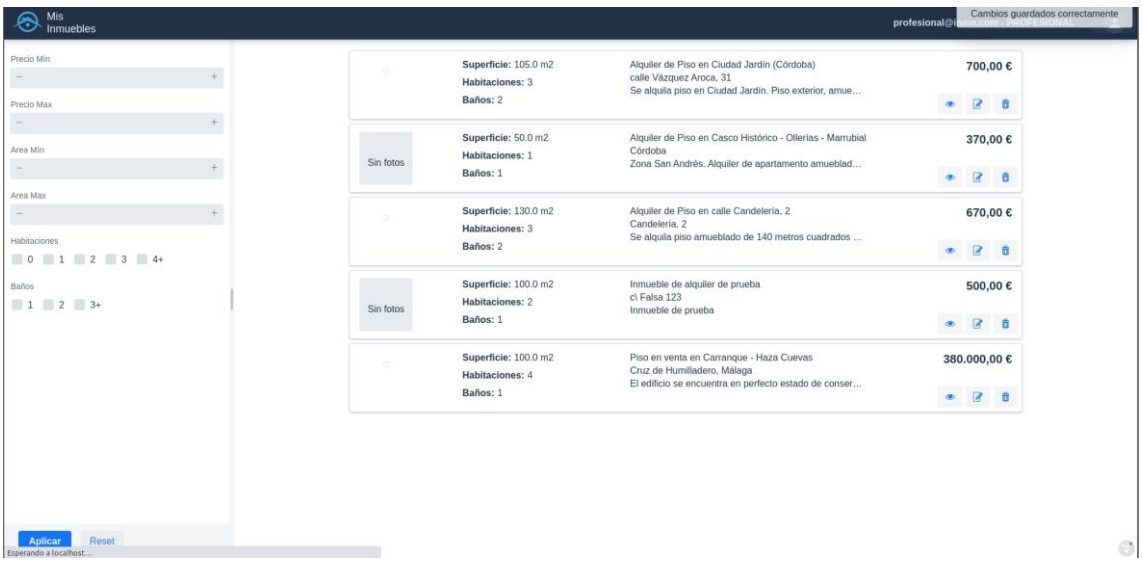


Ilustración 37 - Resultado Prueba 05 – 02

Tabla 14 - Prueba 06

Prueba 06	
CU – RF03 – 02	Modificar un inmueble
Escenario	Principal
Descripción	El usuario de tipo *Inmobiliaria* ha iniciado sesión, accede a la pantalla de *Mis inmuebles,* hace clic en el botón editar un inmueble, modifica alguno de los valores previamente introducidos y pulsa el botón Guardar.
Comportamiento esperado	El sistema muestra un mensaje de confirmación “Cambios guardados correctamente”.
Resultado	OK

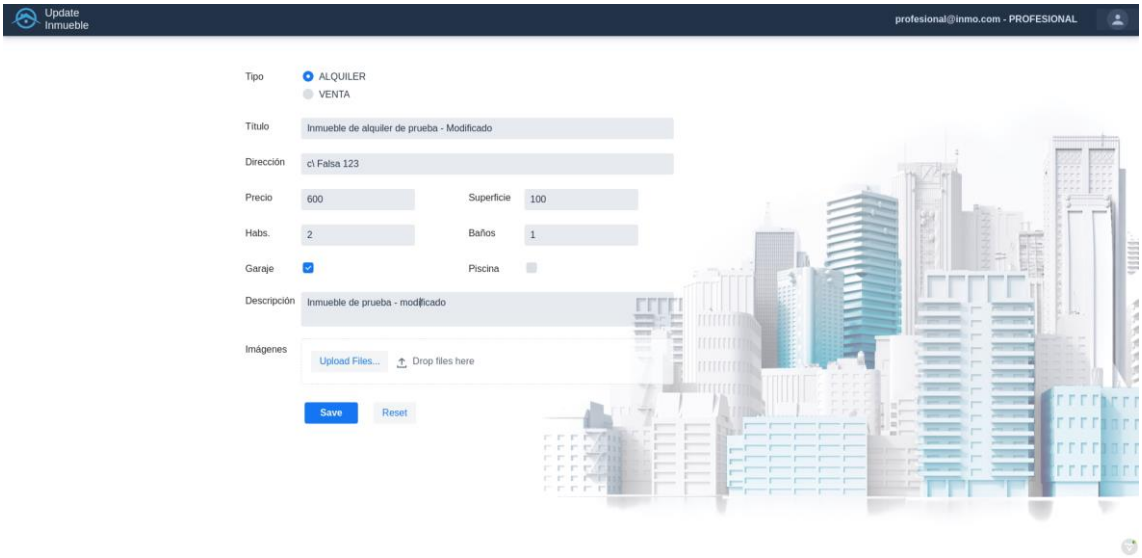


Ilustración 38 - Resultado Prueba 06 - 01

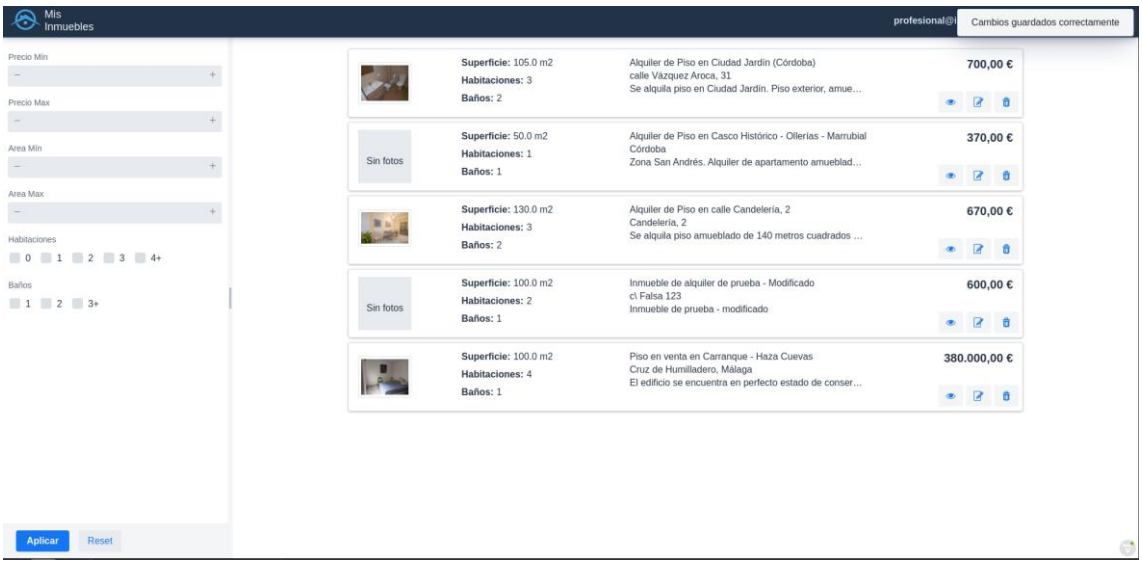


Ilustración 39 - Resultado Prueba 06 - 02

Tabla 15 - Prueba 07

Prueba 07	
CU – RF03 – 03	Eliminar un inmueble
Escenario	Principal
Descripción	El usuario de tipo *Inmobiliaria* ha iniciado sesión, accede a la pantalla de *Mis inmuebles,* hace clic en el botón eliminar un inmueble. Se muestra un mensaje con el texto “Esta acción no puede deshacerse. ¿Desea continuar?” y los botones aceptar y cancelar. El usuario pulsa el botón aceptar
Comportamiento esperado	El sistema muestra un mensaje de confirmación “Cambios guardados correctamente”.
Resultado	OK

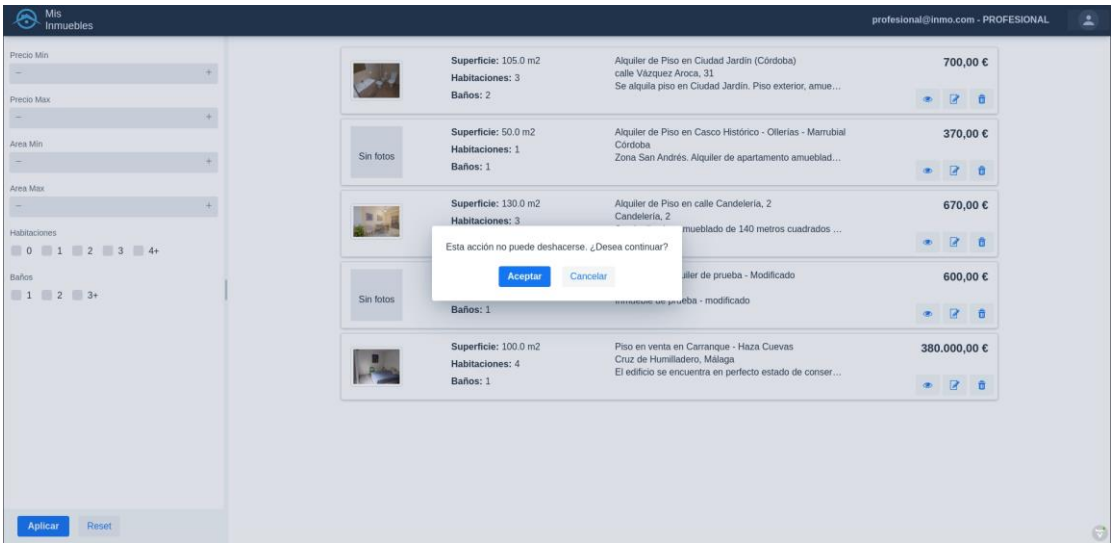


Ilustración 40 - Resultado Prueba 07 – 01

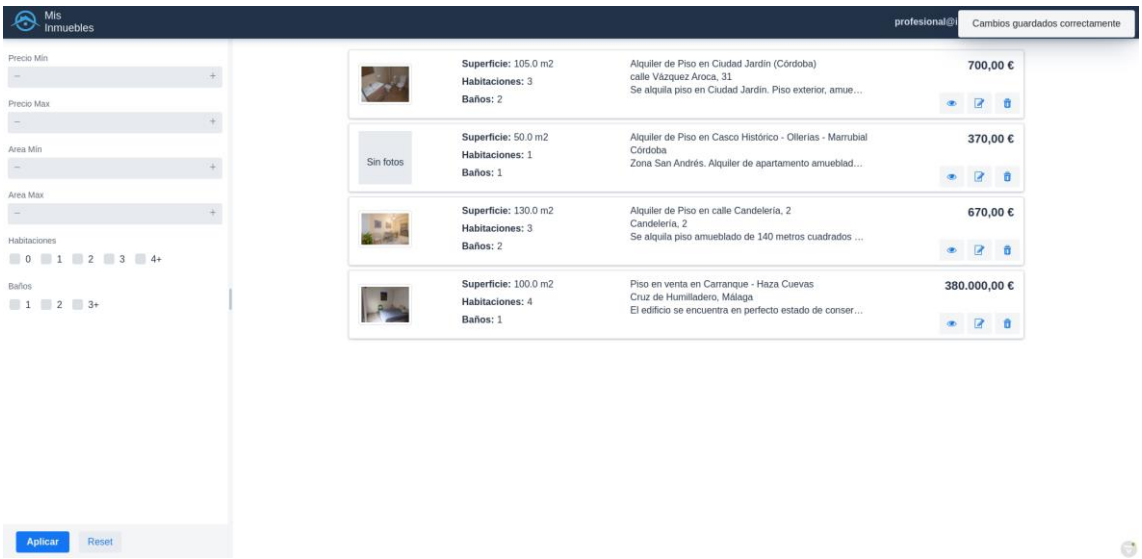


Ilustración 41 - Resultado Prueba 07 – 02

8. Resultados

En este capítulo se pretende mostrar mediante una serie de capturas, complementarias a las mostradas en capítulos anteriores, el resultado del desarrollo.

Comenzamos mostrando la pantalla de login y de registro de usuarios en las ilustraciones *Ilustración 42- Pantalla de login e Ilustración 43 - Pantalla de registro.*

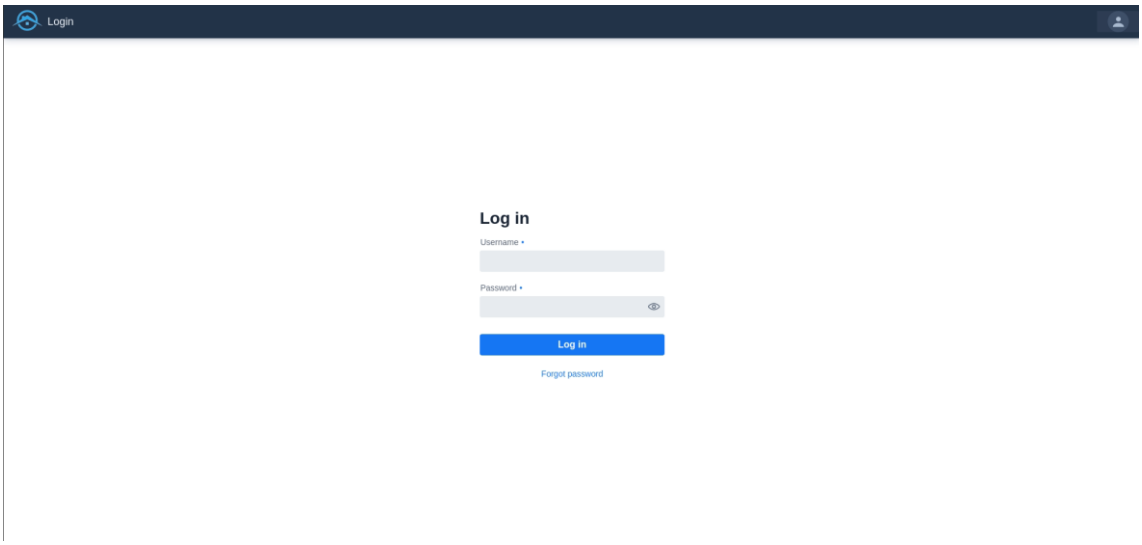


Ilustración 42- Pantalla de login

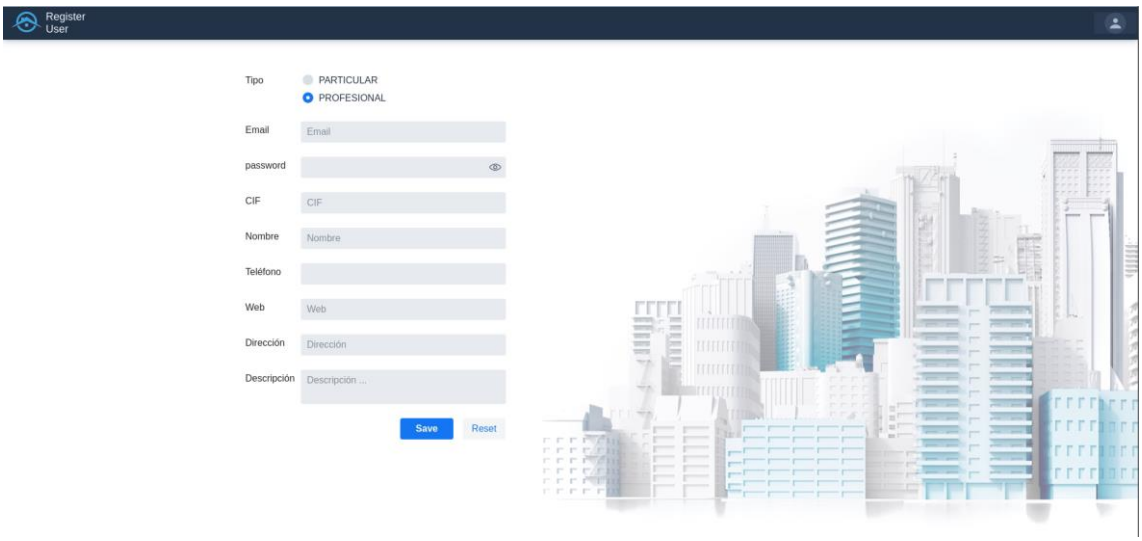


Ilustración 43 - Pantalla de registro

Continuamos mostrando la pantalla de listado de inmuebles en la *Ilustración 44 - Listado de inmuebles.*

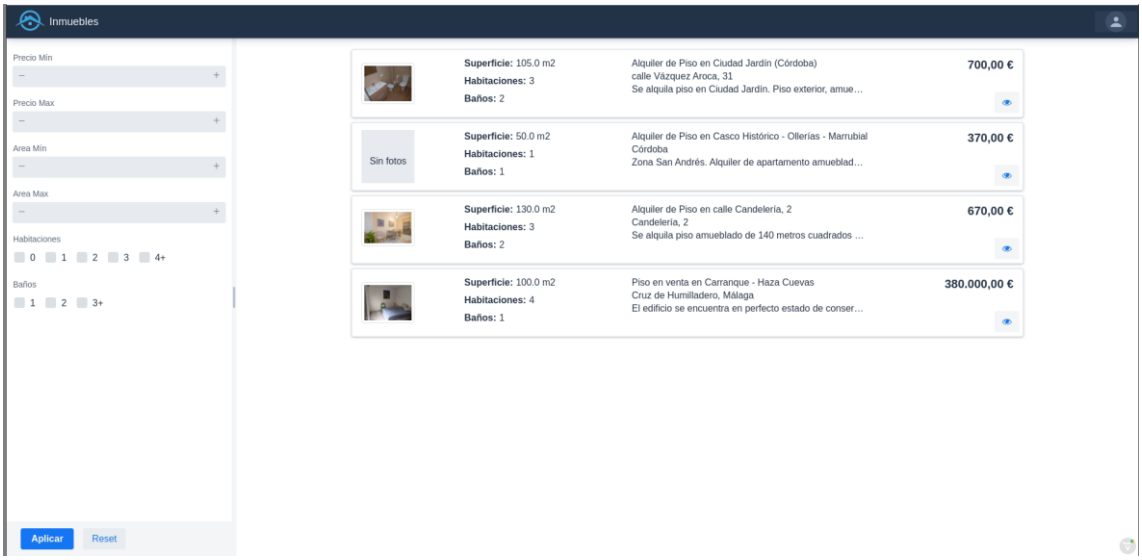


Ilustración 44 - Listado de inmuebles

En las ilustraciones *Ilustración 45 - Detalle de un inmueble 01* e *Ilustración 46 - Detalle de un inmueble 02* se muestra el detalle de un inmueble.

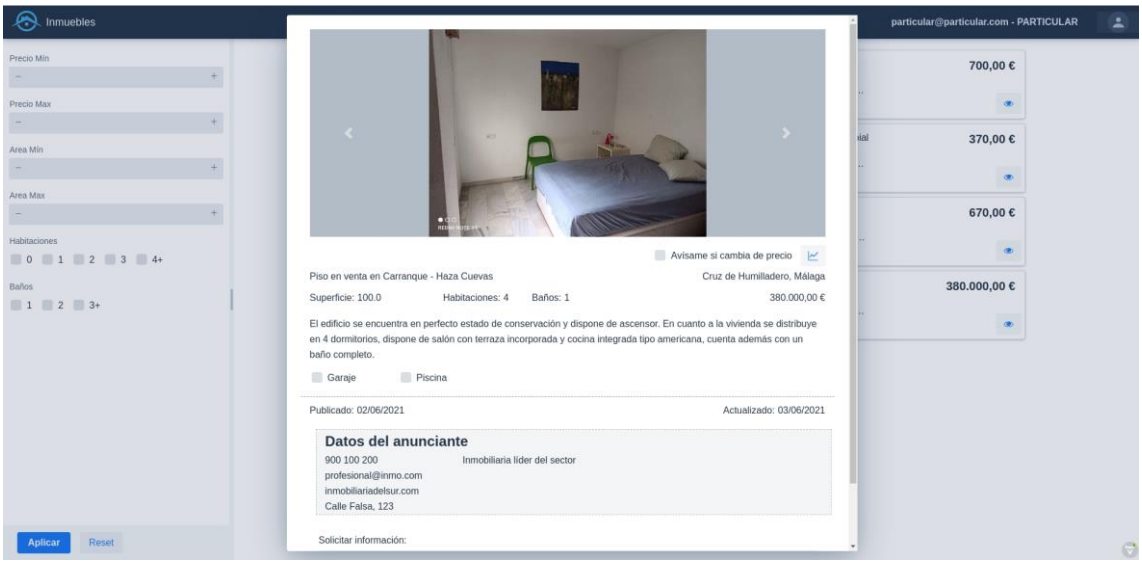


Ilustración 45 - Detalle de un inmueble 01

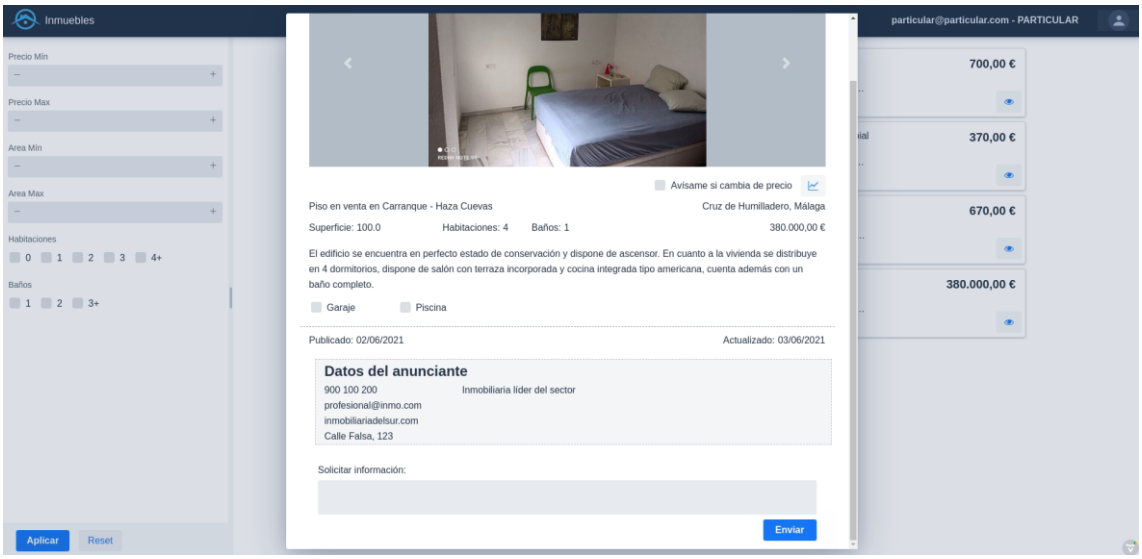


Ilustración 46 - Detalle de un inmueble 02

Si el inmueble ha sufrido variaciones en el precio, desde el detalle del inmueble podemos ver un componente desde el cual visualizar dicha variación. En la *Ilustración 47 - Gráfico de evolución del precio.*

Ilustración 47 - Gráfico de evolución del precio

Si iniciamos sesión con un usuario profesional (inmobiliaria), podemos ver el listado de sus publicaciones, *Ilustración 48- Pantalla "Mis inmuebles"* y crear un inmueble, *Ilustración 49 - Pantalla de creación de un inmueble e Ilustración 50 - Pantalla de creación de un inmueble 02.*

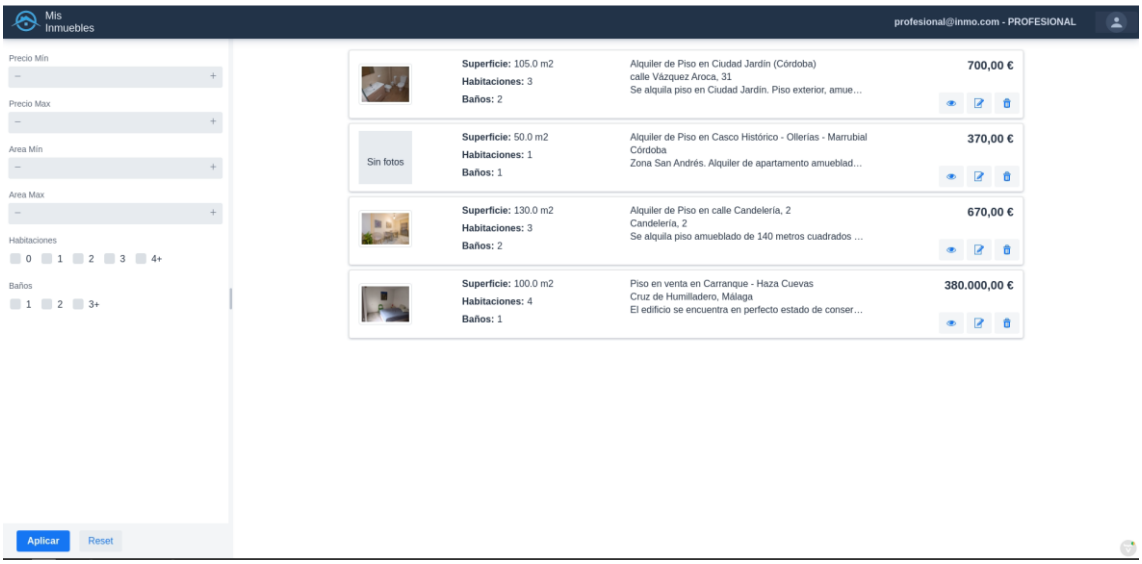


Ilustración 48- Pantalla "Mis inmuebles"

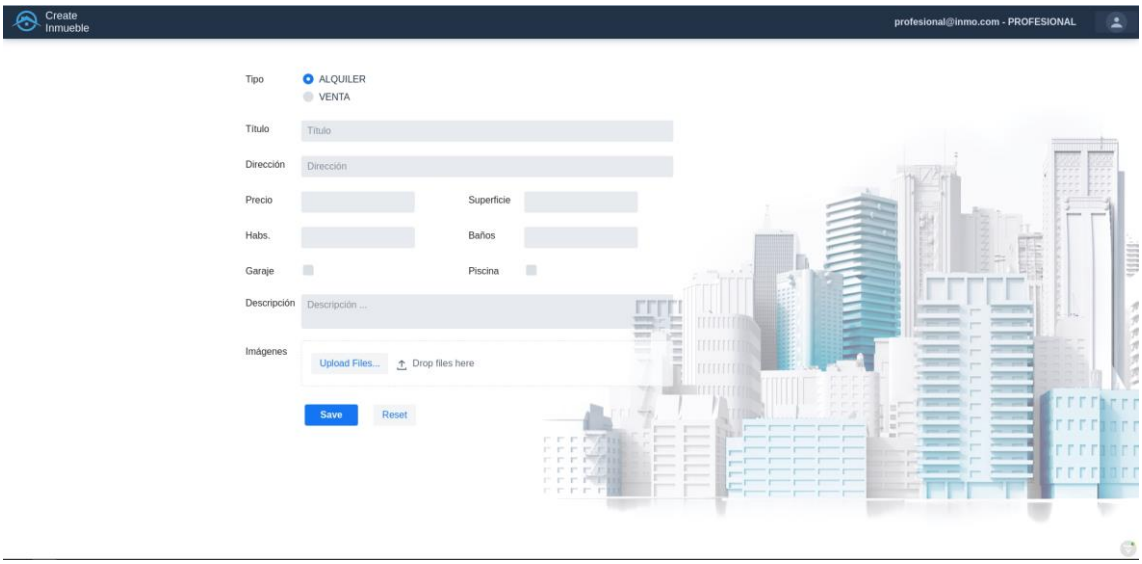


Ilustración 49 - Pantalla de creación de un inmueble 01

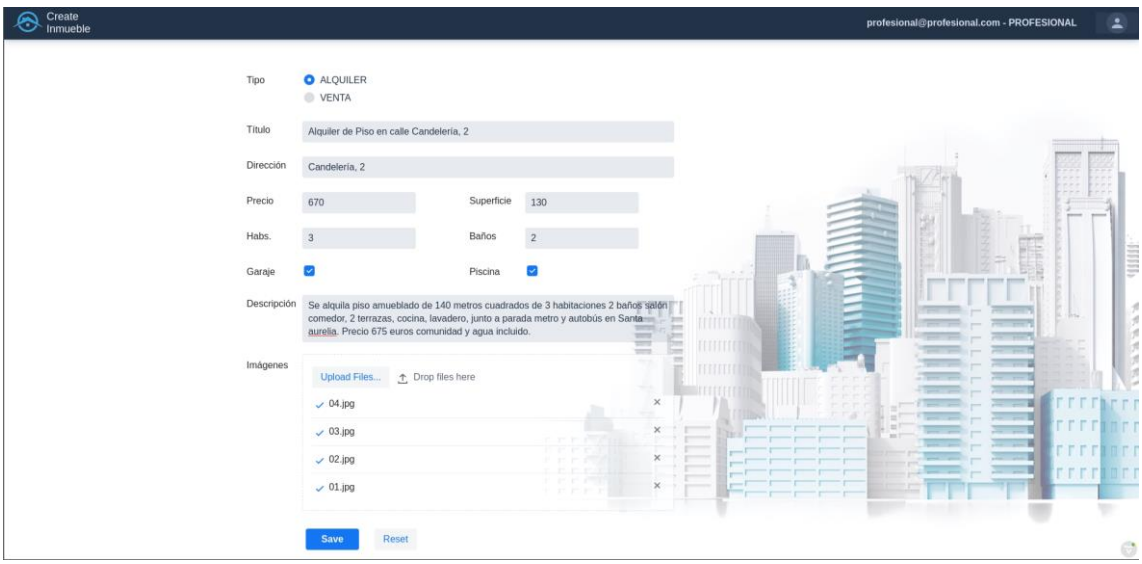


Ilustración 50 - Pantalla de creación de un inmueble 02

9. Conclusiones

Tras la finalización del proyecto, puedo decir que los objetivos planteados al comienzo de este se han cumplido de forma satisfactoria tanto el objetivo principal de elaborar una aplicación que hiciese uso de los patrones de diseño *Event/Sourcing* y *CQRS* como los objetivos secundarios propios del gestor inmobiliario.

Una reflexión que obtengo en este punto es que tengo la sensación de solo haber explorado un pequeño porcentaje de las opciones que ofrece esta tecnología, por otro lado, no se puede pasar por alto el incremento de la complejidad de desarrollo que implica.

En una siguiente evolución de este proyecto sería interesante tratar de desarrollar lo aquí expuesto en una arquitectura completamente distribuida separando la capa de lectura y escritura en microservicios totalmente independientes.

En cuanto a la planificación, es cierto que ha habido que hacer adaptaciones sobre la misma, no ya por problemas en el desarrollo si no por eventos ajenos a la asignatura que han mermado la capacidad disponible en ciertos momentos, nada que no se haya podido adecuar y controlar gracias al seguimiento del tutor.

Personalmente me siento muy satisfecho tanto con el trabajo realizado como con el tema escogido.

10. Glosario

***API**:* son las siglas en inglés de *Application Programming Interfaces*, es decir, Interfaz de programación de aplicaciones. Es una serie de protocolos y estándares que permiten la comunicación entre dos aplicaciones.

Aggregate: entidades que representan el estado de la información ante cada evento que las modifique.

Command: *objetos que representan las tareas o acciones básicas de nuestro sistema.*

Endpoint: cada uno de los puntos de entrada de nuestras APIs.

Event: se trata de cada uno de los eventos lanzados y/o capturados por nuestra aplicación.

Framework: se trata de un conjunto de guías, reglas y conceptos o herramientas sobre el que resolver un problema concreto, en nuestro caso, Spring es un *framework* de Java sobre el que nos hemos basado para desarrollar parte del proyecto.

IDE: *Integrated Development Environment*, en español Entorno integrado de desarrollo. Se trata de un software que proporciona una serie de herramientas para que el desarrollador pueda codificar la aplicación.

11. Bibliografía

[1] “Event Sourcing.” https://martinfowler.com/eaaDev/EventSourcing.html (accessed May 29, 2021).

[2] “Event-driven architecture - Wikipedia.” https://en.wikipedia.org/wiki/Event-driven_architecture (accessed May 29, 2021).

[3] “CQRS.” https://martinfowler.com/bliki/CQRS.html (accessed May 29, 2021).

[4] “Planificador de proyectos | Diagrama de Gantt Online | Tom’s Planner.” https://www.tomsplanner.es/ (accessed May 29, 2021).

[5] “Trello.” https://trello.com/ (accessed May 29, 2021).

[6] “ciscochus/inmo.” https://github.com/ciscochus/inmo (accessed May 30, 2021).

[7] “Axon Framework.” https://axoniq.io/product-overview/axon-framework (accessed Jun. 06, 2021).

[8] “Vaadin - An open platform for building web apps in Java.” https://vaadin.com/ (accessed Jun. 06, 2021).

[9] “GitHub.” https://github.com/ (accessed Jun. 06, 2021).

[10] “Free Git GUI for Windows, Mac, Linux | GitKraken.” https://www.gitkraken.com/ (accessed Jun. 06, 2021).

12. Anexos

El contenido de la entrega:

1. Código fuente.
2. Memoria en formato PDF.
3. Presentación.
4. Vídeo de la presentación.