Universitat Oberta de Catalunya

uoc.edu

Workspaces

Elena Barranco García
Grado de Ingeniería Informática
Java EE

Albert Grau Perisé
Santi Caballé Llobet

05/01/2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Workspaces*
Nombre del autor:	*Elena Barranco García*
Nombre del consultor/a:	*Albert Grau Perisé*
Nombre del PRA:	*Santi Caballé Llobet*
Fecha de entrega:	01/2022
Titulación:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Java EE*
Idioma del trabajo:	*Castellano*
Palabras clave	*Java, Spring, Clean Architecture*

Resumen del Trabajo

Este proyecto tiene como finalidad la creación de una plataforma que permita dar de alta espacios de trabajo diversos y colectivos. Estos espacios podrán ser alquilados por los usuarios y supondrán un lugar de intercambio de ideas y experiencias.

La idea de Workspaces aparece en el contexto de la pandemia por COVID-19, debido a que muchas empresas y organizaciones tuvieron que adoptar el teletrabajo como medida principal para seguir ofreciendo sus servicios a pesar de las restricciones.

Para el desarrollo de la aplicación se ha optado por seguir una metodología en cascada, en la que se han establecido el alcance y los requisitos al comienzo de la planificación. Las etapas del proyecto han coincidido con las entregas propuestas por la asignatura.

Se ha decidido utilizar una arquitectura limpia como base del desarrollo de la aplicación Java, con el objetivo de facilitar la expansión de sus funcionalidades y de abordar futuras implementaciones.

Los productos obtenidos han sido una API REST escrita en Java con el *framework* Spring y una interfaz de usuario desarrollada en React. La integración de ambas aplicaciones ha dado como resultado un prototipo del portal Workspaces, que permite la publicación, búsqueda, filtrado y alquiler de espacios de trabajo, así como la administración del sistema.

La conclusión principal es que la elaboración de este trabajo ha supuesto participar en todas las etapas del ciclo de desarrollo de un proyecto de software y que se han logrado superar los retos propuestos, a pesar de las dificultades encontradas durante el proceso.

Abstract

This project aims to create a platform to publish and share unique workspaces. These spaces can be rented by users and will be a place for the exchange of ideas and experiences.

The idea of Workspaces took shape in the context of the COVID-19 pandemic, due to the fact that many companies and organizations had to adopt remote working as the main measure to continue offering their services despite the restrictions.

Having established at the beginning of the planning the scope and requirements, the most appropriate methodology to opt for would be the waterfall model. The phases of the project coincided with the deliverables proposed by the subject.

It has been decided to use a clean architecture as the start point for the development of the Java application, in order to facilitate the expansion of its functionalities and to address future implementations.

The obtained products are a REST API written in Java built on Spring framework and a user interface developed in React. The integration of both applications results in a prototype of the Workspaces portal, which allows the publication, search, filtering, and rental of workspaces, as well as the system administration.

In conclusion, the completion of this work has involved participating in all the stages of the development cycle of a software Project. Furthermore, the proposed challenges have been overcome, despite the difficulties encountered during the process.

Índice

Tabla de ilustraciones

1. Introducción

1.1. Contexto y justificación del Trabajo

Los acontecimientos ocurridos a lo largo del año 2020 han alterado, de alguna manera u otra, la vida de muchas personas, en el plano individual y colectivo. Durante este tiempo hemos asistido a una transformación en los diferentes ámbitos de la sociedad, en la forma de relacionarnos entre nosotros y con el medio.

Nuestra relación con el trabajo también se ha visto afectada, siendo la aplicación del teletrabajo una medida adquirida por numerosas organizaciones y empresas para continuar ofreciendo sus servicios durante el estado de alarma. Según los datos ofrecidos por Observatorio Nacional de Tecnología y Sociedad, el porcentaje de personas que teletrabajaban habitualmente en España se multiplicó por tres en el segundo trimestre de 2020, alcanzando el 16,2%[1].

Con la relajación de las restricciones, algunas empresas han optado por abrir sus oficinas para permitir progresivamente la vuelta a la modalidad presencial, ya sea de manera parcial o total. En cambio, otras, han optado por establecer el teletrabajo de forma permanente y han actualizado sus acuerdos para adaptarse a esta nueva situación.

Aunque el porcentaje de población que trabaja en remoto de forma habitual ha ido disminuyendo, en el segundo trimestre de 2021 seguía siendo del 9,4%, valor que representa el doble del que existía antes del confinamiento. A esto hay que añadir el porcentaje de teletrabajo ocasional, que ha ido en incremento durante este periodo y que se está estabilizando sobre el 5,3%[2].

Debido a la falta de espacios adecuados para el teletrabajo y, en ocasiones, al distanciamiento social provocado por este, algunas personas buscan la posibilidad de desempeñar sus tareas profesionales en entornos que ofrezcan los servicios apropiados y donde puedan encontrarse con otras personas.

1.2. Objetivos y alcance del Trabajo

El objetivo principal de este proyecto es crear un portal que permita dar de alta y alquilar espacios de trabajo compartidos y ofrecer un lugar de intercambio para las personas trabajadoras. Más allá de centrarse en proveer un listado de

[1] Observatorio Nacional de Tecnología y Sociedad. (septiembre de 2021). *Flash datos de teletrabajo en el segundo en el segundo trimestre de 2021*.

[2] Ídem

espacios *coworking* tradicionales, la idea es visibilizar y reutilizar lugares alternativos como oficinas, cafeterías o cualquier otra estancia que ofrezca unos servicios mínimos de calidad para realizar actividades laborales.

Como objetivos secundarios, con esta propuesta se espera:

- Fomentar el reciclado de espacios, gracias a su adaptación a fines más demandados.
- Impulsar la comunicación y el intercambio de ideas entre personas trabajadoras de ámbitos multidisciplinares.

Este proyecto se desarrolla dentro del marco de un Trabajo Final de Grado, por lo que también forman parte de él los siguientes objetivos:

- Aprendizaje de Java EE y *frameworks* relacionados.
- Exploración de nuevas tecnologías.
- Redacción de la memoria final como resultado del trabajo.

1.3. Enfoque y método seguido

Este trabajo pretende desarrollar las bases de un producto nuevo. Para ajustarnos a las fechas de entrega se ha decidido acotar el alcance y establecer unos requisitos concretos que permitan abordar las funcionalidades más relevantes en cuanto a arquitectura y lógica de negocio. Es por ello por lo que se ha tenido que renunciar a diversas características que podrían ser interesantes y que procuraremos presentar en el apartado de “Trabajo futuro”.

Esta definición acotada de los requisitos y el establecimiento de unas fechas fijas nos permiten adoptar la metodología de trabajo tradicional o en cascada. Esta metodología sigue un proceso lineal en el que las tareas se realizan de forma secuencial, de manera que una nueva etapa se inicia solo cuando se ha finalizado la anterior[3].

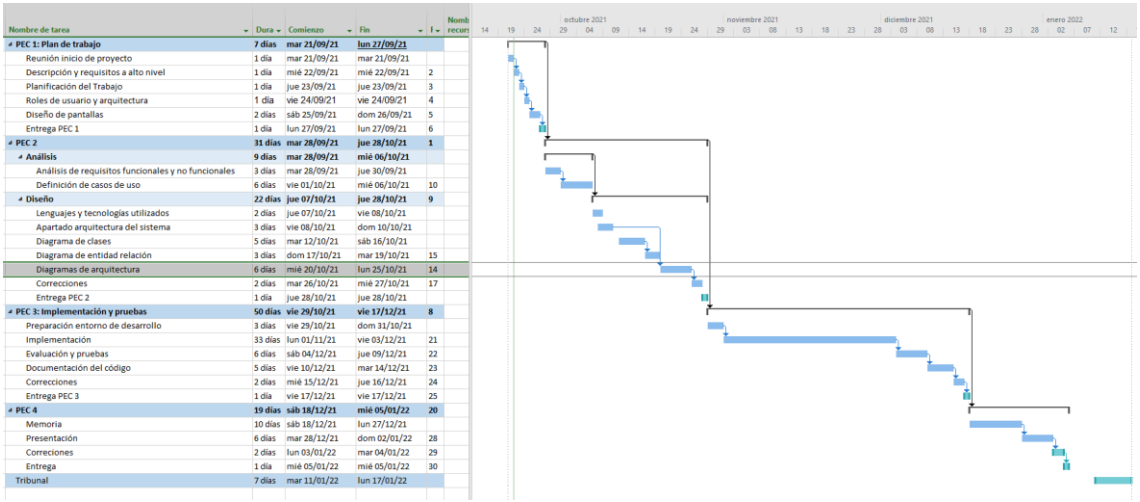
1.4. Planificación del Trabajo

La realización del trabajo se producirá en cuatro fases, que coinciden con las entregas propuestas por la asignatura. Seguidamente se describen las tareas que componen cada una de ellas:

[3] González González, F., & Calero Castañeda, S. L. (2019). *Comparación de las metodologías cascada y ágil para el aumento de la productividad en el desarrollo de software* (Doctoral dissertation, Universidad Santiago de Cali).

- PEC 1: se trata de la entrega inicial, en la que debemos redactar una presentación de nuestra propuesta. El documento debe incluir una descripción del proyecto (contexto y objetivos), un listado de requisitos a alto nivel, la planificación del trabajo acompañada de un diagrama de Gantt (especificando fechas y subtareas), la definición de roles de usuario, la arquitectura y las pantallas principales del sistema.
- PEC 2: incluirá la parte de la memoria correspondiente a los requisitos (funcionales y no funcionales), el análisis y el diseño. Se incluirá una descripción detallada de la arquitectura y los diagramas de software correspondientes. También se comenzará con la formación en las tecnologías que se vayan a utilizar, si fuera preciso.
- PEC 3: implementación de la aplicación en J2EE y pruebas.
- PEC 4: redacción de la memoria y preparación de la presentación.

Para complementar esta información se ha añadido un diagrama de Gantt especificando las fechas previstas para las entregas y tareas a realizar.



Nombre de tarea	Dura	Comienzo	Fin	Nomb recurs
PEC 1: Plan de trabajo	7 días	mar 21/09/21	lun 27/09/21	
Reunión inicio de proyecto	1 día	mar 21/09/21	mar 21/09/21	
Descripción y requisitos a alto nivel	1 día	mié 22/09/21	mié 22/09/21	2
Planificación del Trabajo	1 día	jue 23/09/21	jue 23/09/21	3
Roles de usuario y arquitectura	1 día	vie 24/09/21	vie 24/09/21	4
Diseño de pantallas	2 días	sáb 25/09/21	dom 26/09/21	5
Entrega PEC 1	1 día	lun 27/09/21	lun 27/09/21	6
PEC 2	31 días	mar 28/09/21	jue 28/10/21	1
Análisis	9 días	mar 28/09/21	mié 06/10/21	
Análisis de requisitos funcionales y no funcionales	3 días	mar 28/09/21	jue 30/09/21	
Definición de casos de uso	6 días	vie 01/10/21	mié 06/10/21	10
Diseño	22 días	jue 07/10/21	jue 28/10/21	9
Lenguajes y tecnologías utilizados	2 días	jue 07/10/21	vie 08/10/21	
Apartado arquitectura del sistema	3 días	vie 08/10/21	dom 10/10/21	
Diagrama de clases	5 días	mar 12/10/21	sáb 16/10/21	
Diagrama de entidad relación	3 días	dom 17/10/21	mar 19/10/21	15
Diagramas de arquitectura	6 días	mié 20/10/21	lun 25/10/21	14
Correcciones	2 días	mar 26/10/21	mié 27/10/21	17
Entrega PEC 2	1 día	jue 28/10/21	jue 28/10/21	
PEC 3: Implementación y pruebas	50 días	vie 29/10/21	vie 17/12/21	8
Preparación entorno de desarrollo	3 días	vie 29/10/21	dom 31/10/21	
Implementación	33 días	lun 01/11/21	vie 03/12/21	21
Evaluación y pruebas	6 días	sáb 04/12/21	jue 09/12/21	22
Documentación del código	5 días	vie 10/12/21	mar 14/12/21	23
Correcciones	2 días	mié 15/12/21	jue 16/12/21	24
Entrega PEC 3	1 día	vie 17/12/21	vie 17/12/21	25
PEC 4	19 días	sáb 18/12/21	mié 05/01/22	20
Memoria	10 días	sáb 18/12/21	lun 27/12/21	
Presentación	6 días	mar 28/12/21	dom 02/01/22	28
Correciones	2 días	lun 03/01/22	mar 04/01/22	29
Entrega	1 día	mié 05/01/22	mié 05/01/22	30
Tribunal	7 días	mar 11/01/22	lun 17/01/22	

Ilustración 1. Diagrama de Gantt

1.5. Breve sumario de productos obtenidos

Tras la realización de este trabajo se obtendrán los siguientes productos:

- Código fuente del portal para buscar espacios de trabajo, compuesto por:
  - Aplicación Java.
  - Aplicación React.
- Memoria del TFG:
  - Planificación del trabajo.

    - Análisis y diseño del sistema.
    - Implementación.
    - Evaluación y pruebas.
    - Conclusiones y trabajo futuro.
- Instrucciones de despliegue de la aplicación, manual de usuario y resultados de los test (en documentos aparte).
- Archivo multimedia para la presentación del TFG.

1.6. Breve descripción de los otros capítulos de la memoria

Además de la presente introducción, la memoria constará de los siguientes capítulos:

- Análisis: se recogerán las necesidades del sistema, identificando los roles de los usuarios que interaccionarán con este, los requisitos funcionales y no funcionales y los casos de uso.
- Diseño: se desarrollará la solución propuesta a las necesidades recogidas en el análisis. Describiremos la arquitectura elegida y las decisiones de diseño, acompañándolas de diagramas de entidad-relación, de clases y de arquitectura.
- Implementación: se explicarán los detalles más relevantes sobre la implementación de la aplicación Java a nivel de código, además de las pantallas y decisiones más significativas de la aplicación React.
- Evaluación y pruebas: describiremos las pruebas que realizadas y evaluaremos los resultados obtenidos.
- Conclusión y trabajo futuro: expondremos las conclusiones extraídas del trabajo y posibilidades de ampliación con nuevas características.
- Glosario.
- Bibliografía.

2. Análisis

2.1. Roles de usuario

Se han identificado cuatro roles de usuario que interactuarán directamente con el portal:

- Administrador: encargado de gestionar el contenido del portal y de asegurar su correcto funcionamiento. Tendrá acceso a un menú de administración para operar con las entidades más importantes del sistema.
- Cliente: usuario registrado con permisos para realizar búsquedas de espacios de trabajo y reservar un asiento en ellos. También podrá visualizar y modificar su perfil, además de consultar el historial de reservas, desde el que tendrá acceso a la visualización, modificación y cancelación de estas.
- Host o anfitrión: usuario registrado con permiso para publicar y gestionar un espacio de trabajo. Deberá aportar información de contacto adicional y aceptar los términos y condiciones antes de publicar el primer espacio. Tendrá acceso al historial de reservas realizadas en sus espacios y podrá modificarlas y cancelarlas.
- Usuario no registrado: usuario con permisos para realizar búsquedas y filtrar resultados, pero no para efectuar una reserva hasta que se haya registrado.

2.2. Requisitos

En este apartado se expondrán los requisitos del sistema, que son las condiciones o capacidades que debe poseer el sistema para satisfacer las especificaciones que se han descrito. Distinguiremos entre requisitos funcionales y no funcionales, estos últimos orientados a las cualidades y restricciones del sistema[4].

2.2.1. Requisitos no funcionales

Se han clasificado los requisitos no funcionales en tres tipos diferentes:

[4] Molina Hernández, Y., Granda Dihigo, A., & Velázquez Cintra, A. (2019). *Los requisitos no funcionales de software. Una estrategia para su desarrollo en el Centro de Informática Médica*. Revista Cubana de Ciencias Informáticas, 13(2), 77-90.

- Rendimiento: el sistema no debe tardar más de dos segundos en realizar las búsquedas y renderizar los resultados por pantalla.
- Seguridad: aunque el servicio API REST que ofrecerá la aplicación solo se pretende utilizar de manera interna, se requerirá un token para poder efectuar las peticiones a esta. Además, se establece que el tipo de datos con el que se comunicará será JSON. Por otro lado, las contraseñas del sistema de guardarán encriptadas en la base de datos.
- Disponibilidad: La plataforma estará operativa, en un primer momento, para navegadores Chrome, Firefox y Safari. Queda fuera del alcance la adaptación a dispositivos móviles.

2.2.2. Requisitos funcionales

A continuación, se muestra un listado con los requisitos funcionales, incluyendo su código, nombre y descripción:

- R01 - Registro de usuarios: las personas que utilicen el portal podrán registrarse como anfitrión o como cliente. Esto le permitirá al primero publicar un espacio de trabajo y al segundo realizar una reserva.
    - En el caso de los anfitriones, será imprescindible que proporcionen el número del DNI y el domicilio fiscal, que será utilizada por el servicio externo de pagos con motivos tributarios.
- R02 - Perfil de usuario: los usuarios registrados podrán identificarse en el sistema y acceder a una pantalla para visualizar y modificar sus datos personales.
    - Los usuarios identificados podrán salir del sistema y permanecer sin identificar.
- R03 - Publicación de un espacio de trabajo: la plataforma permitirá a un anfitrión publicar un espacio de trabajo, que será visible para ser alquilado por los clientes. Para que un usuario pueda realizar una publicación se deben haber aceptado los términos y condiciones del portal, asegurando que se cumplen unos estándares de calidad para el trabajo. Estos requerimientos se resumen en:
    - Disponer de mobiliario adecuado, incluyendo sillas y mesas de trabajo.
    - Contar con una conexión de internet estable, iluminación apropiada y enchufes.
    - Ofrecer las medidas adecuadas de seguridad e higiene, así como acceso baños.

- R04 - Modificación de un espacio de trabajo: los datos de un espacio creado podrán ser modificados posteriormente por el anfitrión. También se podrá eliminar si no hay reservas activas.
- R05 - Detalles de un espacio de trabajo: los usuarios (tanto anfitriones como clientes) podrán visualizar los detalles de un espacio, para ver el precio, la descripción y las imágenes adjuntas.
- R06 - Listar los espacios de un anfitrión: los anfitriones podrán acceder a un listado de espacios que hayan publicado.
- R07 - Realizar una búsqueda: los usuarios (registrados o no) podrán realizar una búsqueda de espacios introduciendo una ubicación y un rango de fechas, obteniendo un listado de espacios disponibles:
    - Una vez obtenidos los resultados se podrán aplicar diversos filtros, como el tipo de espacio (coworking, oficina, cafetería, etc.), precio o servicios ofrecidos (pantallas, servicio de restaurante, terraza, jardín, etc.).
- R08 - Realizar la reserva de un espacio: los usuarios registrados podrán realizar la reserva de una plaza en el espacio seleccionado en periodos de tiempo que van desde un día hasta un mes.
- R09 - Visualizar historial de reservas: los usuarios podrán consultar un listado de reservas, cuyo contenido variará según el rol que desempeñen en el portal. En el caso del anfitrión, contará con una lista de las reservas que hayan tenido lugar en sus espacios, mostrando los estados (pendiente, finalizada o cancelada). Los clientes tendrán a su disposición el historial de espacios que hayan reservado, incluyendo los mismos estados.
- R10 - Cancelación de una reserva: los clientes podrán cancelar una reserva y obtener un reembolso completo siempre que la cancelación se realice con, al menos, 24 horas de antelación.
- R11 - Administración: el sistema deberá permitir al administrador la gestión de servicios y usuarios, incluyendo operaciones de creación, visualización, modificación y eliminación. Además, posibilitará la visualización del listado de espacios de trabajo y reservas.

2.3. Casos de uso

En este apartado se describen los casos de uso identificados en el sistema. En primer lugar, presentaremos una tabla con el título de cada caso, actores involucrados y requisito que satisface.

2.3.1. Tabla de casos de uso

ID	Caso de uso	Actores	Requisito
UC01	Registrar un usuario	Usuario no registrado	R01
UC02	Login de usuario	Administrador, anfitrión, cliente	R02
UC03	Logout	Administrador, anfitrión, cliente	R02
UC04	Visualizar el perfil de usuario	Anfitrión, cliente	R02
UC05	Modificar los datos personales	Anfitrión, cliente	R02
UC06	Publicar un espacio de trabajo	Anfitrión	R03
UC07	Modificar los datos de un espacio	Anfitrión	R04
UC08	Consultar los detalles de un espacio	Administrador, anfitrión, cliente	R05
UC09	Listar los espacios de un anfitrión	Anfitrión	R06
UC10	Eliminar un espacio	Anfitrión	R04
UC11	Realizar una búsqueda	Cliente, usuario no registrado	R07
UC12	Aplicar filtros a la búsqueda	Cliente, usuario no registrado	R07
UC13	Realizar la reserva de un espacio	Cliente	R08
UC14	Visualizar el historial de reservas	Anfitrión, cliente	R09
UC15	Cancelar una reserva	Cliente	R10
UC16	Listar espacios de trabajo	Administrador	R11
UC17	Listar reservas	Administrador	R11
UC18	Gestionar usuarios	Administrador	R11
UC19	Gestionar servicios	Administrador	R11

2.3.2. Descripción de casos de uso

A partir de la tabla definida anteriormente, detallaremos cada caso de uso, incluyendo una descripción, detalles sobre el flujo y las condiciones. Para una mejor lectura y comprensión de las tablas se ha decidido agruparlas en cuatro bloques.

El primer bloque, relacionado con la gestión de perfiles, recoge los casos de uso cuyo objetivo es el registro, identificación y consulta de datos de usuario. El segundo engloba los casos de uso relacionados con la gestión de espacios. El tercero reúne aquellos involucrados en el proceso de reserva de un espacio. Por último, en el bloque de administración se han incluido los relacionados con la gestión del portal.

Esta clasificación es una mera convención, por lo que asumimos que podría haberse seguido otra manera de clasificar los casos, ya que en muchas ocasiones intervienen diversos tipos de entidades principales.

2.3.2.1. Casos de uso relacionados con la gestión de perfiles

UC01	Registrar un usuario
Descripción	Un usuario no registrado podrá darse de alta en el sistema introduciendo sus datos personales.
Actores	Usuario no registrado.
Pre-condición	El usuario debe tener una cuenta de correo electrónico operativa y no haberse registrado previamente en el sistema con ella.
Flujo principal	1. El usuario pulsa el botón de “Sign-up”. 2. El sistema muestra una pantalla con un formulario. 3. El usuario introduce sus datos personales. 4. El usuario acepta los términos y condiciones de uso. 5. El usuario pulsa el botón de “Create Account”. 6. El sistema valida los datos introducidos y crea un nuevo usuario en el sistema. 7. El sistema muestra un mensaje de éxito.
Flujo alternativo	4a. El usuario no acepta los términos y condiciones de uso. 5a. El usuario pulsa el botón de “Create Account”. 6a. El sistema muestra un mensaje de advertencia indicando que es necesario aceptar las condiciones. 6b. El sistema determina que hay algún error en los datos introducidos y no crea el nuevo usuario. 7b. El sistema muestra un mensaje de advertencia.
Post-condición	Se registra un nuevo usuario en el sistema. Se vuelve a la pantalla “Home”.

UC02	Login de usuario
Descripción	Un usuario registrado podrá identificarse en el portal introduciendo su email y contraseña.
Actores	Administrador, anfitrión, cliente.
Pre-condición	El usuario debe existir en el sistema.
Flujo principal	1. El usuario pulsa el botón de “Login”. 2. El sistema solicita email y contraseña. 3. El usuario introduce sus credenciales. 4. El sistema valida las credenciales.

	5. El sistema muestra un mensaje de éxito y el email del usuario en la barra superior.
Flujo alternativo	4a. El sistema determina que la combinación de email de usuario y contraseña es incorrecta. 5a. El sistema muestra un mensaje de advertencia.
Post-condición	El usuario queda identificado en el portal, adquiriendo los permisos correspondientes al rol que desempeñe.

UC03	**Logout**
Descripción	Un usuario podrá cerrar sesión en el portal.
Actores	Administrador, anfitrión, cliente.
Pre-condición	El usuario debe existir en el sistema y estar identificado en el portal.
Flujo principal	1. El usuario pulsa el botón de “Logout”. 2. El sistema cierra la sesión del usuario. 3. El sistema muestra la pantalla “Home” y los botones de “Login” y “Sign-up”.
Flujo alternativo	No aplica.
Post-condición	El usuario queda sin identificar en el sistema.

UC04	**Visualizar el perfil de usuario**
Descripción	Un usuario registrado podrá consultar los datos de su perfil.
Actores	Anfitrión, cliente.
Pre-condición	El usuario debe existir en el sistema y haberse identificado en el portal.
Flujo principal	1. El usuario pulsa el icono de “Profile” en la barra de navegación superior. 2. El sistema muestra la pantalla del perfil con los datos personales del usuario.
Flujo alternativo	No aplica.
Post-condición	Se muestran por pantalla los datos de perfil de usuario.

UC05	**Modificar los datos personales**
Descripción	Un usuario registrado podrá modificar sus datos personales.
Actores	Anfitrión, cliente.
Pre-condición	El usuario debe existir en el sistema y haberse identificado en el portal.

Flujo principal	1. El usuario pulsa el icono de “Profile” en la barra de navegación superior. 2. El sistema muestra la pantalla de perfil con los datos personales. 3. El usuario pulsa el botón de “Update”. 4. El sistema muestra los campos del formulario como editables. 5. El usuario modifica los datos. 6. El usuario pulsa el botón “Save”. 7. El sistema valida los datos y actualiza los registros correspondientes. 8. El sistema muestra un mensaje de éxito.
Flujo alternativo	7a. El sistema determina que el formato de los datos introducidos no es correcto y no actualiza los registros. 8a. El sistema muestra un mensaje de advertencia para indicar al usuario que debe corregir los datos de algún campo del formulario.
Post-condición	Los datos de usuario quedan modificados en el sistema.

2.3.2.2. Casos de uso relacionados con la gestión de espacios

UC06	**Publicar un espacio de trabajo**
Descripción	Un anfitrión podrá dar de alta un nuevo espacio de trabajo en el sistema.
Actores	Anfitrión.
Pre-condición	El anfitrión debe existir en el sistema y haberse identificado en el portal.
Flujo principal	1. El anfitrión pulsa el botón “My Spaces” en la barra de navegación superior. 2. El sistema muestra la pantalla con el listado de espacios del anfitrión. 3. El usuario pulsa el botón de “Create”. 4. El sistema muestra el formulario para crear un nuevo espacio de trabajo. 5. El anfitrión rellena los campos del formulario. 6. El anfitrión pulsa el botón de “Publish”. 7. El sistema valida los datos introducidos y crea el nuevo espacio. 8. El sistema muestra un mensaje de éxito.
Flujo alternativo	7a. El sistema determina que el formato de los datos introducidos no es correcto y no crea el espacio.

	8a. El sistema muestra un mensaje de advertencia para indicar al usuario que debe corregir los datos de algún campo del formulario.
Post-condición	Se crea un nuevo espacio en el sistema.

UC07	**Modificar los datos de un espacio**
Descripción	Un anfitrión podrá actualizar los datos de un espacio que haya registrado con anterioridad.
Actores	Anfitrión.
Pre-condición	El espacio debe existir en el sistema. El anfitrión debe estar identificado en el sistema.
Flujo principal	1. El anfitrión pulsa el botón de “My spaces”. 2. El sistema muestra la pantalla con el listado de espacios dados de alta por el anfitrión. 3. El anfitrión selecciona el botón “Ver detalles” de un espacio. 4. El sistema muestra la pantalla con los detalles de un espacio. 5. El anfitrión pulsa el botón de “Update”. 6. El sistema configura los campos del formulario como editables. 7. El anfitrión modifica los campos y pulsa el botón de “Save”. 8. El sistema valida los datos introducidos. 9. El sistema muestra un mensaje de éxito.
Flujo alternativo	8a. El sistema determina que el formato de los datos introducidos no es correcto. 9a. El sistema muestra un mensaje de advertencia para indicar al usuario que debe corregir los datos de algún campo del formulario.
Post-condición	Los datos del espacio quedan actualizados en el sistema.

UC08	**Consultar los detalles de un espacio**
Descripción	Los usuarios registrados podrán consultar los detalles de un espacio de trabajo.
Actores	Administrador, Anfitrión, cliente.
Pre-condición	El espacio debe existir en el sistema. El usuario debe estar identificado en el sistema. El sistema debe encontrarse mostrando un listado de espacios.
Flujo principal	1. El usuario pulsa el botón “Mostrar detalles” de un espacio del listado.

	2. El sistema muestra la pantalla del detalle de un espacio.
Flujo alternativo	No aplica
Post-condición	Se muestran por pantalla los datos de un espacio.

UC09	**Listar los espacios de un anfitrión**
Descripción	Un anfitrión podrá consultar el listado de espacios que ha dado de alta en el portal.
Actores	Anfitrión.
Pre-condición	El anfitrión de existir en el sistema y haberse identificado en el portal.
Flujo principal	1. El anfitrión pulsa el icono de “My spaces” en la barra de navegación superior. 2. El sistema consulta los espacios de trabajo asociados y obtiene datos. 3. El sistema muestra al usuario una pantalla con un listado de espacios de trabajo.
Flujo alternativo	2a. El sistema consulta los espacios de trabajo asociados al anfitrión y no obtiene datos. 3a. El sistema muestra un mensaje informando de que no existe ningún espacio en el sistema que cumpla los requisitos de búsqueda.
Post-condición	Se muestra el listado de espacios de un anfitrión.

UC10	**Eliminar un espacio**
Descripción	Un anfitrión podrá eliminar un espacio propio del sistema.
Actores	Anfitrión.
Pre-condición	El espacio debe existir en el sistema. El anfitrión debe estar identificado en el portal.
Flujo principal	1. El anfitrión pulsa el icono de “My spaces” en la barra de navegación superior. 2. El sistema muestra al usuario una pantalla con un listado de espacios de trabajo. 3. El anfitrión pulsa el botón “Delete” del espacio. 4. El sistema muestra un mensaje solicitando confirmación. 5. El usuario confirma la acción. 6. El sistema elimina el espacio de trabajo del sistema. 7. El sistema muestra un mensaje de éxito.
Flujo alternativo	4a. El anfitrión cancela la eliminación del espacio. 5a. El sistema no realiza la acción de borrado.
Post-condición	El espacio queda eliminado del sistema.

2.3.2.3. Casos de uso relacionados con la búsqueda y alquiler de espacios

UC11	Realizar una búsqueda
Descripción	Un usuario podrá realizar la búsqueda de los espacios disponibles entre dos fechas y en una ciudad determinada.
Actores	Cliente, usuario no registrado.
Pre-condición	Deben existir espacios dados de alta en el sistema.
Flujo principal	1. El usuario selecciona la ciudad y las fechas en las que desea realizar la reserva y pulsa el botón “Search”. 2. El sistema consulta los espacios disponibles según los criterios introducidos y encuentra resultados. 3. El sistema muestra un listado de espacios de trabajo.
Flujo alternativo	2a. El sistema no encuentra resultados según los datos introducidos. 3a. El sistema muestra un mensaje informando que no se han encontrado espacios según los criterios proporcionados.
Post-condición	El sistema muestra un listado de espacios disponibles.

UC12	Aplicar filtros a la búsqueda
Descripción	El usuario podrá aplicar filtros a los resultados de una búsqueda.
Actores	Cliente, usuario no registrado.
Pre-condición	Deben existir espacios datos de alta en el sistema. El sistema debe haber encontrado resultados para los criterios de búsqueda.
Flujo principal	1. El usuario selecciona uno o varios filtros y pulsa el botón “Search”. 2. El sistema consulta los espacios disponibles según los filtros aplicados y encuentra resultados. 3. El sistema muestra un nuevo listado de espacios de trabajo.
Flujo alternativo	2a. El sistema no encuentra resultados según los datos introducidos. 3a. El sistema muestra un mensaje informando que no se han encontrado espacios según los criterios proporcionados.
Post-condición	El sistema muestra un listado de espacios disponibles según los filtros aplicados.

UC13	Realizar la reserva de un espacio
Descripción	Un cliente podrá alquilar un puesto en un espacio de trabajo durante un periodo determinado.
Actores	Cliente.
Pre-condición	El espacio debe existir en el sistema. El espacio debe tener plazas disponibles para el periodo solicitado. El sistema debe encontrarse mostrando la pantalla con el listado de espacios disponibles.
Flujo principal	1. El usuario pulsa la opción “View” del espacio que desee alquilar. 2. El sistema muestra la pantalla con los detalles del espacio y el resumen de la reserva. 3. El usuario pulsa el botón reservar. 4. El sistema solicita confirmación. 5. El usuario confirma la reserva. 6. El sistema le redirige hacia la plataforma de pagos externa. 7. La plataforma de pagos confirma que el pago se ha realizado correctamente. 8. El sistema crea una nueva reserva y muestra un mensaje de éxito.
Flujo alternativo	5a. El usuario cancela la solicitud de reserva. 6a. No se crea ninguna reserva en el sistema. 7b. La plataforma de pagos notifica que ha habido un error durante el proceso de pago. 8b. No se crea ninguna reserva en el sistema. 9b. El sistema muestra un mensaje de error.
Post-condición	Se crea una reserva nueva en el sistema.

UC14	Visualizar el historial de reservas
Descripción	Un usuario registrado podrá visualizar el historial de reservas realizadas. En el caso de un anfitrión, se mostrarán las reservas que hayan tenido lugar en sus espacios de trabajo publicados. Si se trata de un usuario, se mostrarán las reservas que el usuario haya realizado en los diferentes espacios.
Actores	Anfitrión, cliente.
Pre-condición	El usuario debe existir en el sistema y haberse identificado en el portal.
Flujo principal	1. El usuario pulsa el icono de “My Reservations” en la barra de navegación superior.

	2. El sistema consulta las reservas asociadas al usuario y obtiene datos. 3. El sistema muestra al usuario una pantalla con un listado de reservas.
Flujo alternativo	2a. El sistema consulta las reservas asociadas al usuario y no obtiene datos. 3a. El sistema muestra un mensaje informando de que no existe ninguna reserva en el sistema que cumpla los requisitos de búsqueda.
Post-condición	Se muestra un listado de reservas según el rol del usuario que realiza la consulta.

UC15	**Cancelar una reserva**
Descripción	El usuario podrá cancelar una reserva pendiente.
Actores	Cliente.
Pre-condición	El usuario debe estar registrado. El usuario debe haber realizado una reserva en el espacio con anterioridad. La reserva debe encontrarse activa.
Flujo principal	1. El usuario pulsa el icono de “My Reservations” en la barra de navegación superior. 2. El sistema consulta las reservas asociadas al usuario y obtiene datos. 3. El sistema muestra al usuario una pantalla con un listado de reservas. 4. El usuario selecciona la opción “Cancel” de la reserva activa. 5. El sistema muestra un mensaje solicitando confirmación. 6. El usuario confirma la cancelación de la reserva. 7. El sistema valida las condiciones para eliminar la reserva y procede con la operación. 8. El sistema muestra un mensaje de éxito.
Flujo alternativo	6a. El usuario cancela la solicitud de borrado. 7a. El sistema no elimina la reserva del sistema. 7b. El sistema determina que no se cumplen los requisitos para la cancelación de la reserva y no procede con la operación. 8b. EL sistema muestra un mensaje de error.
Post-condición	La reserva queda eliminada del sistema.

2.3.2.4. Casos de uso relacionados con la administración del portal

UC16	Listar espacios de trabajo
Descripción	Un administrador podrá acceder al listado completo de espacios de trabajo registrados en el portal a través de un menú de administración.
Actores	Administrador.
Pre-condición	El administrador debe estar identificado en el sistema.
Flujo principal	1. El Administrador pulsa el enlace de “Administration”. 2. El sistema muestra el menú de administración con la lista de entidades del portal. 3. El administrador selecciona la opción de “Spaces”. 4. El sistema muestra la pantalla con el listado de espacios.
Flujo alternativo	No aplica
Post-condición	Se muestran por pantalla el listado completo de espacios de trabajo.

UC17	Listar reservas
Descripción	Un administrador podrá acceder al listado completo de reservas realizadas en el portal a través de un menú de administración.
Actores	Administrador.
Pre-condición	El administrador debe estar identificado en el sistema.
Flujo principal	1. El Administrador pulsa el enlace de “Administration”. 2. El sistema muestra el menú de administración con la lista de entidades del portal. 3. El administrador selecciona la opción de “Reservations”. 4. El sistema muestra la pantalla con el listado de reservas.
Flujo alternativo	No aplica
Post-condición	Se muestran por pantalla el listado completo de reservas.

UC18	Gestionar usuarios
Descripción	Un administrador podrá realizar operaciones CRUD sobre los usuarios del sistema. Esto es la creación, consulta, actualización y borrado de usuarios.
Actores	Administrador.
Pre-condición	El administrador deber haberse identificado en el portal.
Flujo principal	1. El Administrador pulsa el enlace de “Administration”. 2. El sistema muestra el menú de administración con la lista de entidades del portal.

	3. El administrador selecciona la opción “Users”. 4. El sistema muestra la pantalla con el listado de usuarios existentes en el sistema. 5. El administrador pulsa el botón de “Create”, “Update”, “View” o “Delete” sobre un usuario. 6. El sistema realiza la operación conveniente.
Flujo alternativo	No aplica
Post-condición	Los datos del usuario quedan modificados en el sistema.

UC19	**Gestionar servicios**
Descripción	Un administrador podrá realizar operaciones CRUD sobre los servicios del sistema. Esto es la creación, consulta, actualización y borrado de servicios.
Actores	Administrador.
Pre-condición	El administrador debe estar identificado en el sistema.
Flujo principal	1. El Administrador pulsa el enlace de “Administration”. 2. El sistema muestra el menú de administración con la lista de entidades del portal. 3. El administrador selecciona la opción “Services”. 4. El sistema muestra la pantalla con el listado de servicios existentes en el sistema. 5. El administrador pulsa el botón de “Create”, “Update”, “View” o “Delete” sobre un servicio. 6. El sistema realiza la operación conveniente.
Flujo alternativo	No aplica
Post-condición	Los datos del servicio quedan modificados en el sistema.

2.3.3. Diagramas de casos de uso

A continuación, se muestran los diagramas de casos de uso agrupados en los bloques definidos en el apartado anterior.

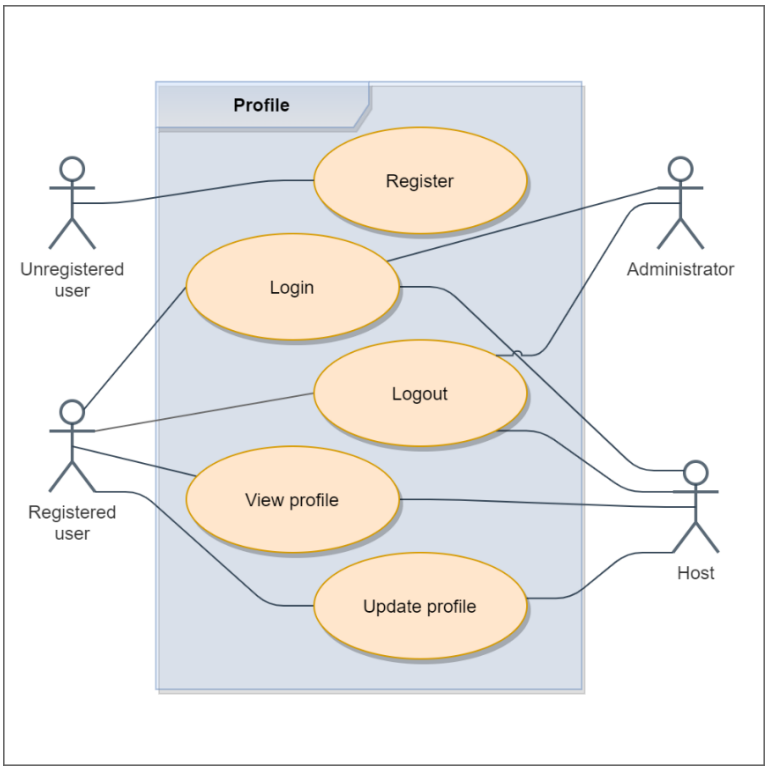


Ilustración 2. Casos de uso sobre el perfil de usuario

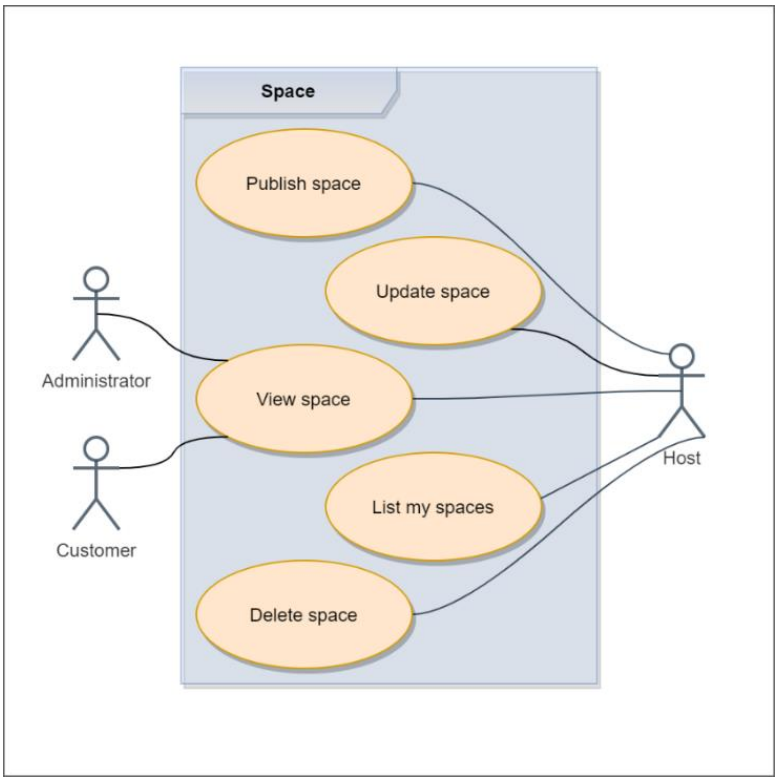


Ilustración 3. Casos de uso sobre los espacios

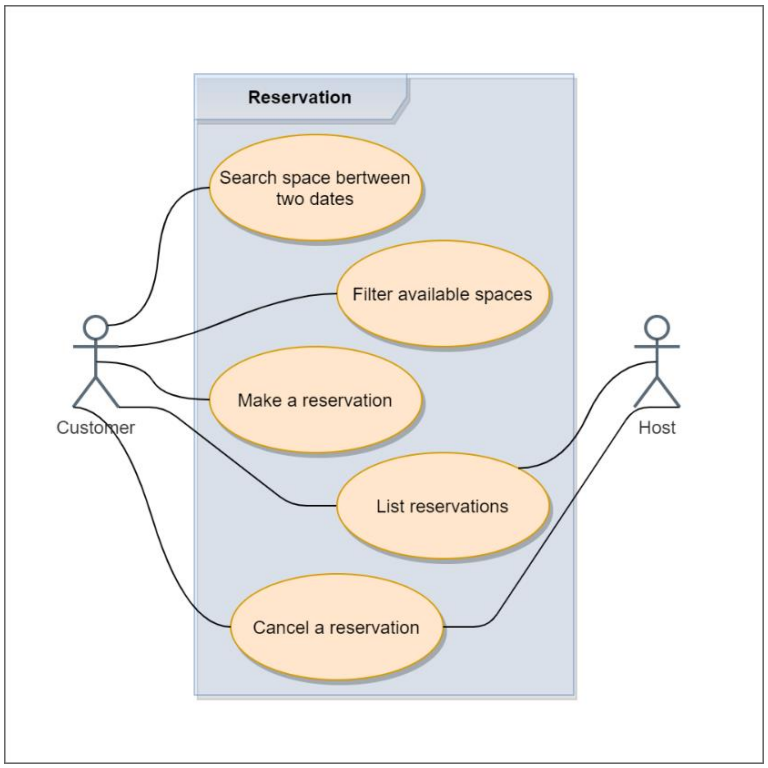


Ilustración 4. Casos de uso sobre las reservas

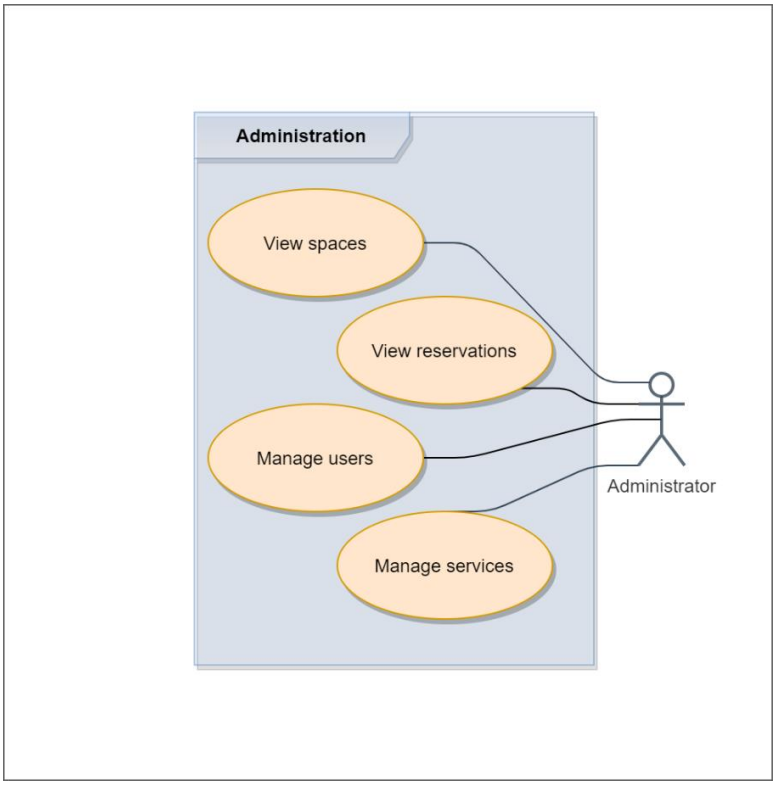


Ilustración 5. Casos de uso sobre la gestión del portal

3. Diseño

3.1. Arquitectura del sistema

En este trabajo hemos realizado una propuesta de arquitectura limpia, apoyada en patrones de diseño, que nos permitirá organizar nuestra aplicación en capas siguiendo principalmente una regla: las capas superiores deben depender únicamente de las inferiores.

Esta regla está basada en el principio de "Inversión de dependencia" de los denominados principios SOLID de diseño de software. El objetivo de estos principios es la creación de estructuras de software que permitan la tolerancia a cambios, la fácil comprensión del código y la reutilización de los componentes[5].

Volviendo a nuestra arquitectura, dispondríamos en el núcleo de la aplicación las entidades de dominio, rodeadas por los casos de uso. Estos casos de uso no son más que servicios que cumplen con una función concreta y tienen una única responsabilidad, siguiendo, esta vez, el principio de responsabilidad única de SOLID.

Alrededor de este núcleo se encuentra el resto de los componentes que apoyan las reglas de negocio y que proporcionarán la persistencia, la interfaz de usuario o los adaptadores para la comunicación con componentes externos. Como hemos comentado anteriormente, la capa de dominio no debe saber nada sobre los detalles de esta, por lo que podrá centrarse en las reglas de negocio[6].

En nuestra aplicación la capa exterior estará compuesta, principalmente, por los adaptadores de persistencia a una base de datos MySQL y los controladores para exponer los *endpoints* de una API. Estos *endpoints* serán el punto de comunicación entre los componentes del lado del usuario y los del lado del servidor.

Como ejemplo de arquitectura limpia podríamos mencionar la arquitectura hexagonal, en la que encontramos el dominio de la aplicación en el centro de un hexágono. Fuera estarían los adaptadores que interaccionan con la aplicación,

[5] Martin, R. C., Grenning, J., & Brown, S. (2018). *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall. (cap. III. Design principles, p. 352)

[6] Hombergs, Tom. (2019). *Get Your Hands Dirty on Clean Architecture: A hands-on guide to creating clean web applications with code examples in Java*. Packt Publishing Ltd. (p. 15-16)

como adaptadores web o de base de datos, comunicándose con esta mediante puertos[7].

Aprovecharemos esta última definición para mostrar el siguiente gráfico, en el que se representa la arquitectura limpia mediante un hexágono, dividiendo las capas por colores y representando la regla de dependencia mediante flechas que apuntan hacia el núcleo.

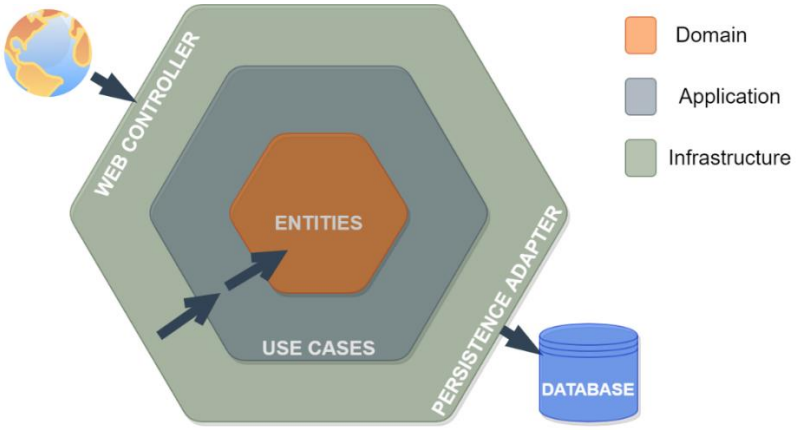


Ilustración 6. Representación de la arquitectura del sistema

3.1.1. Base de datos

Para la persistencia del sistema se ha seleccionado MySQL como Sistema de gestión de Base de Datos Relacional, por ser una de las bases de datos *Open Source* más populares del mundo[8]. Además, como se describe en su propia web, el servidor de bases de datos MySQl es rápido, fiable, escalable y fácil de usar.

3.1.2. API REST

Para la comunicación entre la parte *backend* y *frontend* de nuestro sistema se ha optado por el desarrollo de una API REST (*Representational State Transfer*), por ser la elección más empleada en los últimos tiempos. Este paradigma expone los datos como recursos y utiliza métodos estándar HTTP para representar las

[7] Alistair, Cockburn. (2005). *Hexagonal Architecture*. Consultado desde https://alistair.cockburn.us

[8] MySQL Documentation Team. *MySql 8.0 Reference Manual: What is MySQL?* Consultado desde https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

transacciones de creación, lectura, actualización y eliminación (CRUD) de estos recursos[9].

A continuación, hemos elaborado una tabla en la que se define el tipo de operaciones, el verbo HTTP que las lleva a cabo, una *url* de ejemplo y la acción que se realizará en cada caso. La entidad que se ha tomado como referencia es la de *Space*, que representa un espacio de trabajo.

Operación	Verbo HTTP	URL de ejemplo	Acción
Create	POST	/spaces	Creación de un espacio
Read	GET	/spaces /spaces/1	Lista todos los espacios Obtiene el espacio con id 1
Update	PUT	/spaces/1	Actualiza los datos del espacio con id 1
Delete	DELETE	/spaces/1	Elimina el espacio con id 1

Siguiendo las buenas prácticas en el diseño de las APIs, se ha decidido que las respuestas tendrán una misma estructura para todas las peticiones y que incluirán el código HTTP, un código de estado que se pueda manejar internamente, un mensaje legible para el usuario y el cuerpo de la respuesta[10].

En la siguiente tabla se definen los tipos de respuestas esperados en la aplicación.

Estado	Código HTTP	Mensaje	Descripción
Consulta correcta	200	OK	
Recurso creado	201	created	“A new {resource} was created successfully.”
Error del sistema	500	internal_server_error	“An error occurred in the server side. Cause: {exception message}”

[9] Jin, B., Sahni, S., & Shevat, A. (2018). *Designing Web APIs: Building APIs That Developers Love*. O'Reilly Media, Inc. (p. 10)

[10] Ídem. (p. 52-54, 71-72)

Error de formato en la petición	400	missing_required_parameters	“The request is missing a {parameter} parameter”
Recurso no encontrado	404	not_found	“The requested {resource} was not found.”
Conflicto en la petición	409	request_conflict	“There was a conflict in your requested data. Cause: {exception message}”.

3.1.3. Lenguaje y tecnologías utilizados

A continuación, se presenta el listado de lenguajes y tecnologías que se esperan utilizar.

- JavaEE como plataforma de programación y Spring Boot como *framework* principal para el desarrollo de la parte *backend* de la aplicación.
- Intellij como IDE.
- Maven como herramienta organización y gestión de proyectos.
- JUnit5 y Mockito para test unitarios.
- MySQL como sistema de gestión de base de datos.
- React y Bootstrap para la implementación de la parte *frontend*.
- Postman y DBeaver para pruebas en la API y la base de datos.

3.2. Diagrama de clases

En el siguiente diagrama se han modelado las clases que compondrán nuestro sistema. Se ha optado por hacer uso de una relación de herencia entre la clase User y las clases Host, Customer y Administrator, ya que los tres usuarios compartirán múltiples campos.

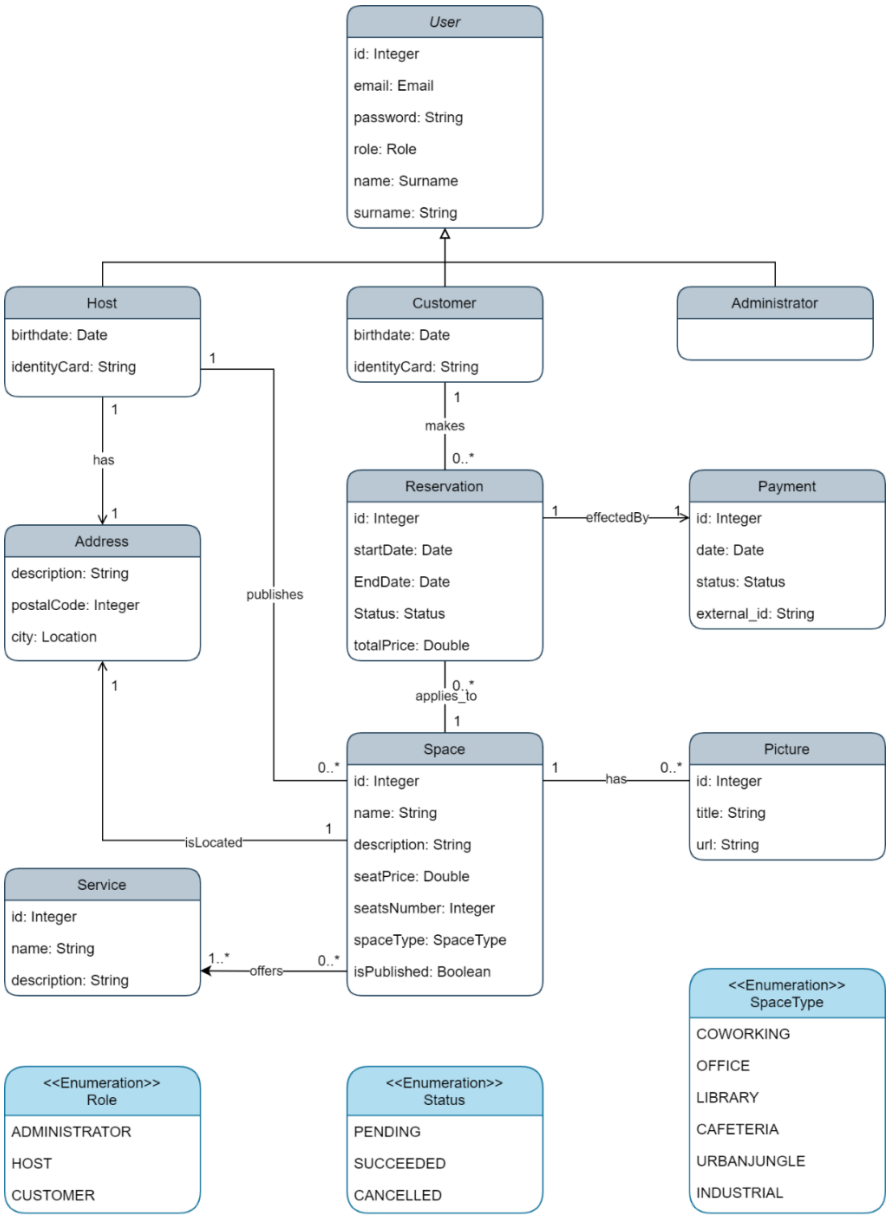


Ilustración 7. Diagrama de clases

Por otro lado, se ha diseñado una clase Address, que será reutilizada por las clases Host y Space, encapsulando así el conjunto de campos que componen la dirección. La decisión de añadir un campo de dirección al anfitrión es que resulta imprescindible disponer del domicilio fiscal de la persona para proceder con los cobros derivados de las reservas.

En cuanto al tipo de espacio, ha sido modelado como un tipo enumerado, ya que será uno de los campos clave para el filtrado de resultados y se deseaba tener acotadas las posibilidades.

Por último, se ha decidido simplificar la entidad Picture, incluyendo únicamente los campos de título y la URL, ya que supondremos que de la carga de imágenes

de encargará un sistema externo. Se manejarán, por lo tanto, las URLs donde se encuentran alojadas estas imágenes.

3.3. Diagrama de Entidad-Relación

En el diseño de la base de datos relacional se ha intentado simplificar el modelo lo máximo posible. Las entidades Space, Payment, Picture, Service y Reservation mantienen una estructura similar a la definida en el diagrama de clases. Sin embargo, se han aplicado algunas adaptaciones al resto.

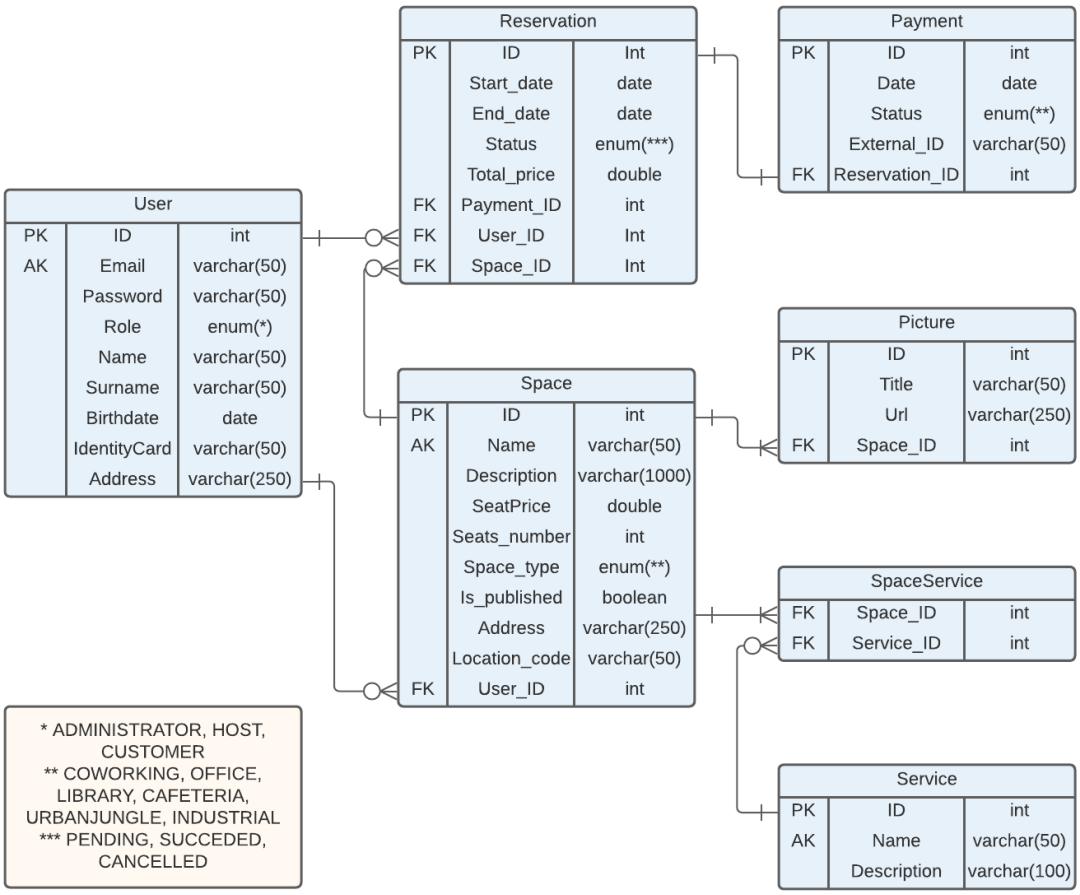


Ilustración 8. Diagrama de Entidad-Relación

En el caso de los usuarios advertimos que los campos que diferenciaban los distintos roles eran escasos, por lo que se decidió simplificar la herencia y diseñar una única tabla de usuario con los campos de todos los roles.

Como desventaja nos encontramos con que no es posible establecer algunas restricciones sobre campos no nulos, como es el caso del número de identidad, que solo es obligatorio para los anfitriones y los clientes, o la dirección postal, que solo debe incluirla el anfitrión. Sin embargo, se consiguen obtener los datos

de todos los usuarios en una sola consulta a la base de datos, lo que agilizará el proceso de identificación en el portal y visualización del perfil.

En el caso de la dirección se han tomado diferentes decisiones de diseño. Esta entidad estaría relacionada, por un lado, con el usuario y, por otro lado, con el espacio de trabajo. En la entidad User sería apropiado utilizar un tipo de datos JSON, ya que no es necesario procesar campos concretos de la dirección.

Sin embargo, en el caso de la dirección de un espacio el tratamiento de los datos será distinto, ya que no solo vamos a necesitar la cadena de caracteres que la componen, sino que utilizaremos el campo *City* para el filtrado de resultados en la pantalla con el listado de espacios. Como solución, se ha decidido generar un código de localización que se almacenará en la propia tabla Space y que nos servirá para aplicar el filtrado sin tener que acceder a los campos de dirección.

Por último, entre las entidades de Space y Service nos encontramos con una relación Many-to-many, por lo que ha sido conveniente diseñar una tabla intermedia SpaceService.

3.4. Diagramas de arquitectura

En este apartado se presentarán los diagramas desde el punto de vista de la computación. Para una mejor legibilidad, en el primer diagrama se muestran los componentes más relevantes de nuestro sistema y las interfaces colapsadas. En los siguientes se ofrecerá un mayor refinamiento de los elementos y la paquetería concreta de cada componente.

Como podemos observar, se han definido tres capas siguiendo nuestra arquitectura limpia, en la que la regla de dependencia ocurre desde la capa superior a la inferior. Si nos fijamos en las flechas de punta negra, comprobamos que los componentes de las capas superiores solo utilizan las interfaces de las capas inferiores, siguiendo esta norma.

Las interfaces de dominio son implementadas en la capa de infraestructura permitiendo, de esta manera, que los componentes interiores hagan uso de los exteriores sin vulnerar nuestro principio.

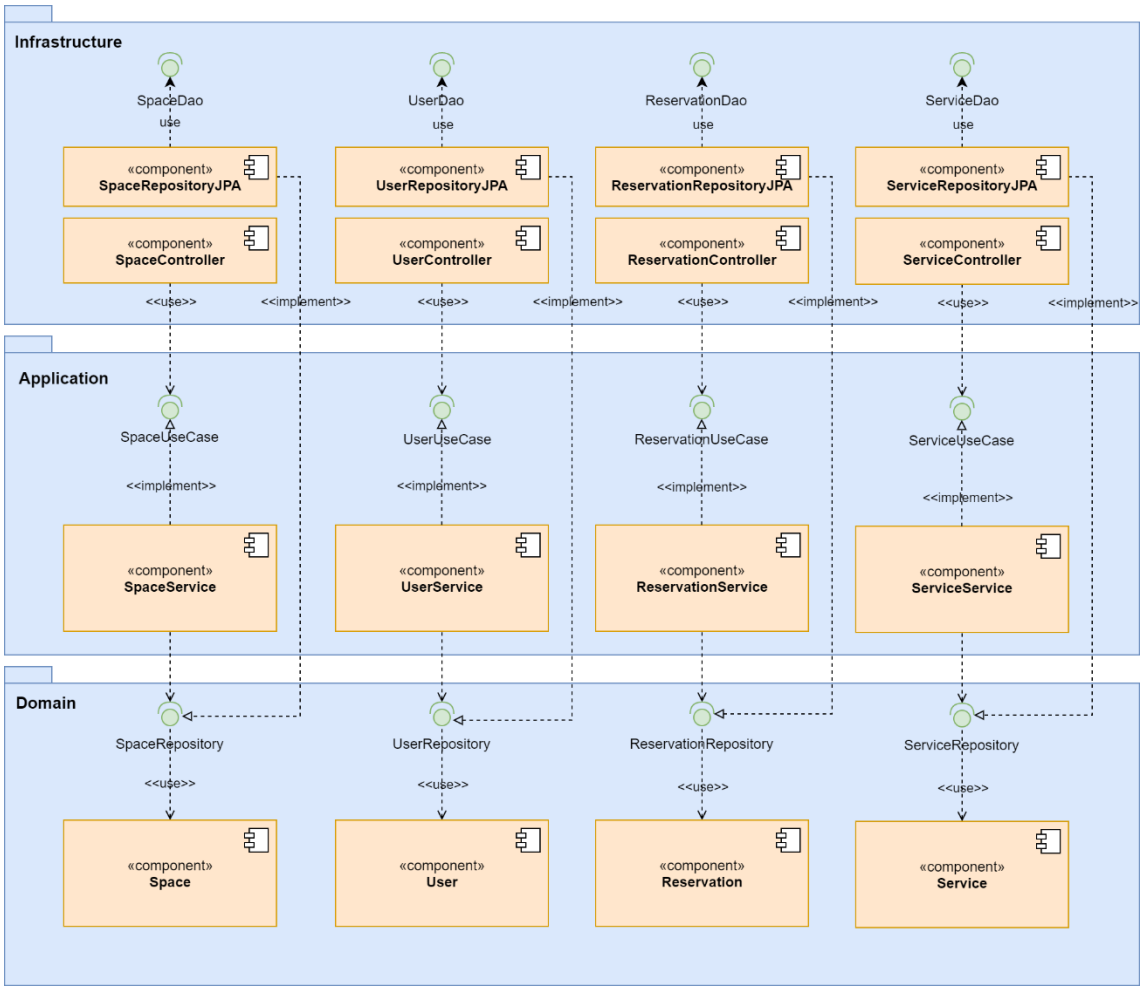


Ilustración 9. Diagrama de arquitectura

3.4.1. Detalle de interfaces y componentes

En la presente sección se describirán las interfaces y los componentes del sistema, organizados según las capas descritas.

3.4.1.1. Capa de infraestructura

Como ya se ha mencionado, en esta capa se implementarán los controladores y los componentes encargados de la persistencia.

3.4.1.1.1. Persistencia

La arquitectura del módulo de persistencia está basada en el patrón DAO, que se traduce en una interfaz encargada de hacer las consultas a la base de datos y de manejar entidades JPA. Esta interfaz será llamada desde los servicios de la

capa de aplicación, por lo que la persistencia quedará desacoplada de la lógica de negocio[11].

JPA hace referencia a Java Persistence API y fue creada para proporcionar un mecanismo de mapeo de los objetos de bases de datos relacionales, es decir, una entidad con la que se puedan realizar operaciones CRUD en la base de datos[12].

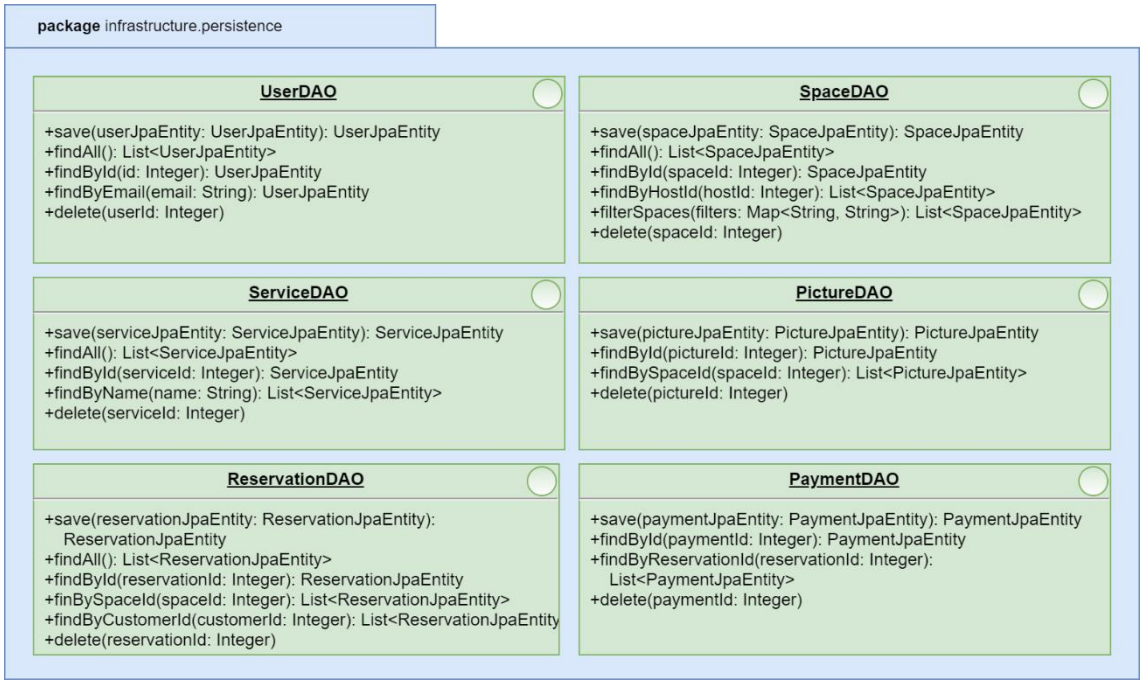


Ilustración 10. Interfaces de la capa de infraestructura-persistencia

En el diagrama que mostramos a continuación se observan los detalles de los componentes de esta capa y su relación con las interfaces que se acaban de presentar. Además de las clases que se han mencionado previamente encontramos *mappers,* que transformarán el objeto de dominio en uno de tipo JPAEntity.

[11] Garrido Tejero, A. (2016). *Implementación del patrón DAO (Data Access Objecto)*. Universidad Politécnica de Valencia. Consultado desde https://polimedia.upv.es/visor/?id=d55b1e80-c327-f14f-93f6-e55ee68525fa

[12] Pech-May, F., Gomez-Rodriguez, M. A., Luis, A., & Lara-Jeronimo, S. U. (2012). *Desarrollo de Aplicaciones web con JPA, EJB, JSF y PrimeFaces*. Instituto Tecnológico Superior de los Ríos, Tabasco, México.

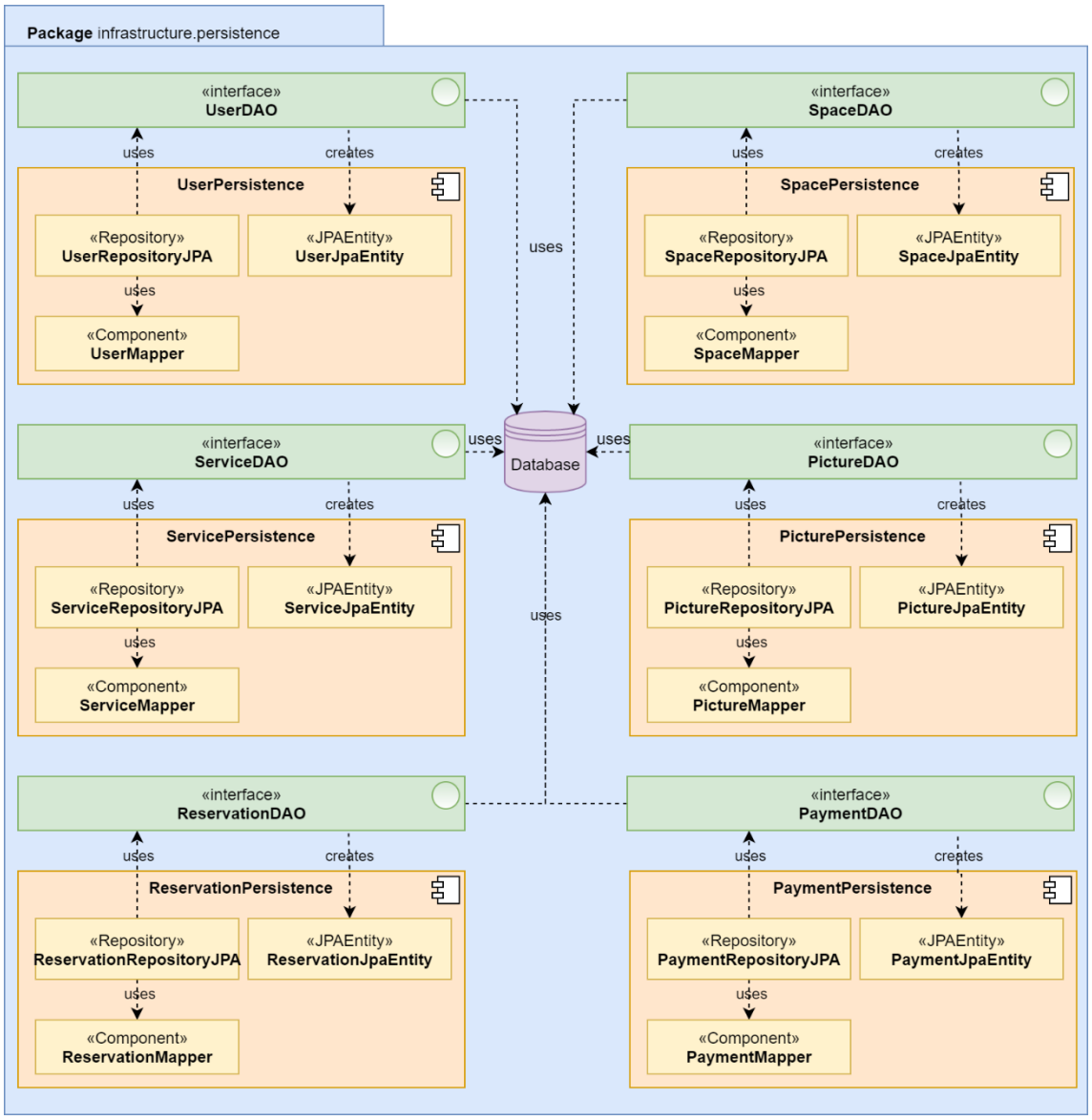


Ilustración 11. Componentes de la capa de infraestructura-persistencia

Seguidamente, se presentará el refinamiento de los componentes de la capa de persistencia. Dado que en nuestra arquitectura se hace uso de componentes e interfaces entre capas, se ha decidido presentar los elementos más relevantes de cada componente, aunque formen parte de una capa diferente.

Precisamente, en la capa de persistencia, los componentes de tipo RepositoryJPA implementarán la interfaz de repositorio de la capa de dominio. Para mostrar unos diagramas más compactos, se ha optado por ilustrar el paquete de dominio dentro del de persistencia, pero hay que recordar que estos dos paquetes se encontrarían al mismo nivel.

Por último, hay que añadir que en cada diagrama de refinamiento se han especificado únicamente los detalles de los componentes de ese paquete

concreto. El resto de los componentes se presenta para comprender la relación que existe entre ellos.

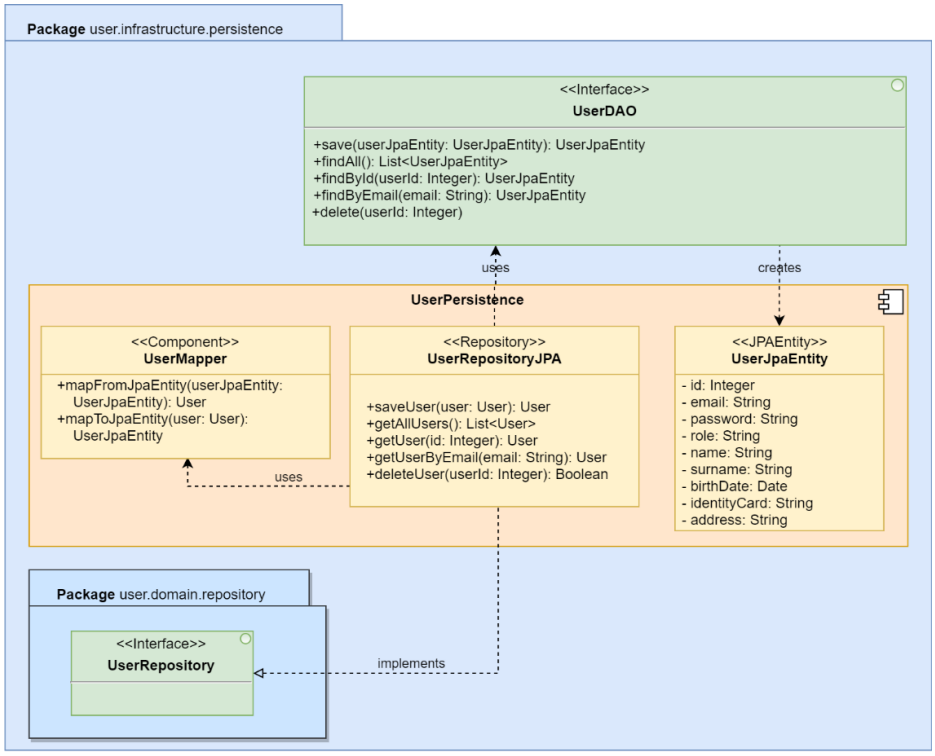


Ilustración 12. Refinamiento del paquete de persistencia de usuario

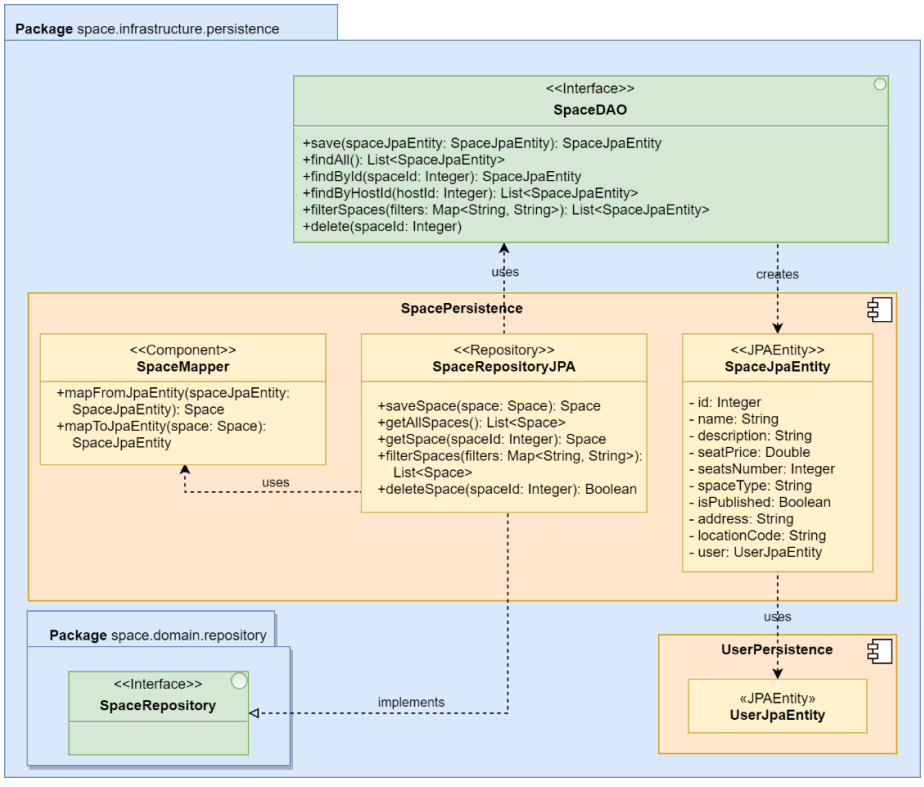


Ilustración 13. Refinamiento del paquete de persistencia de espacio

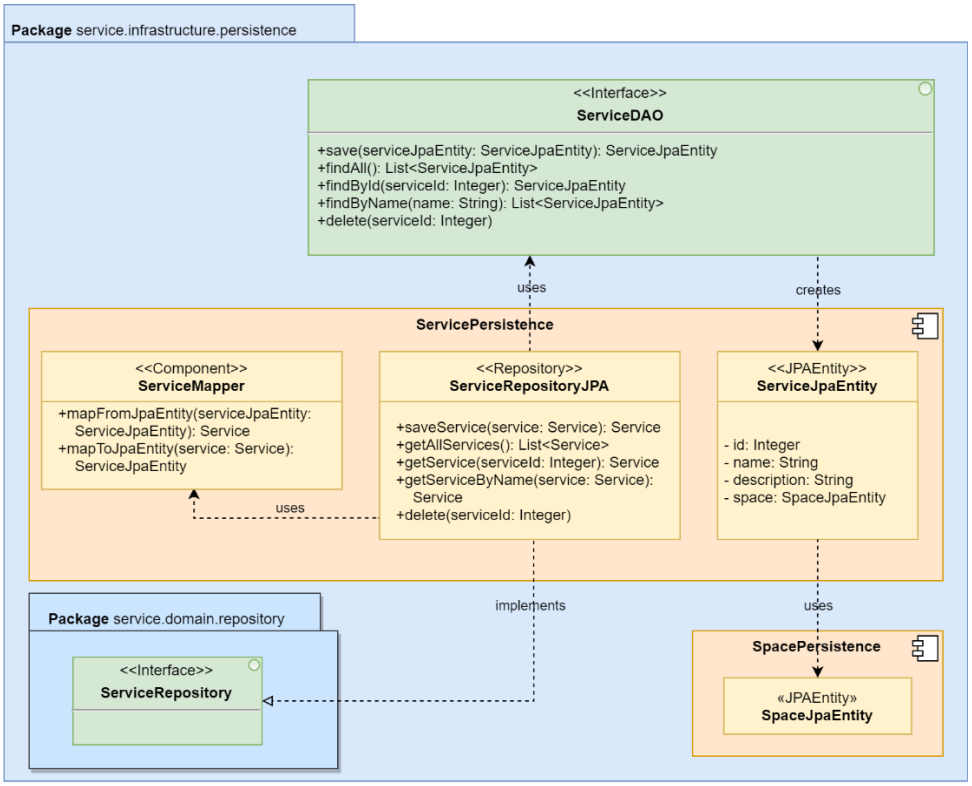


Ilustración 14. Refinamiento del paquete de persistencia del servicio

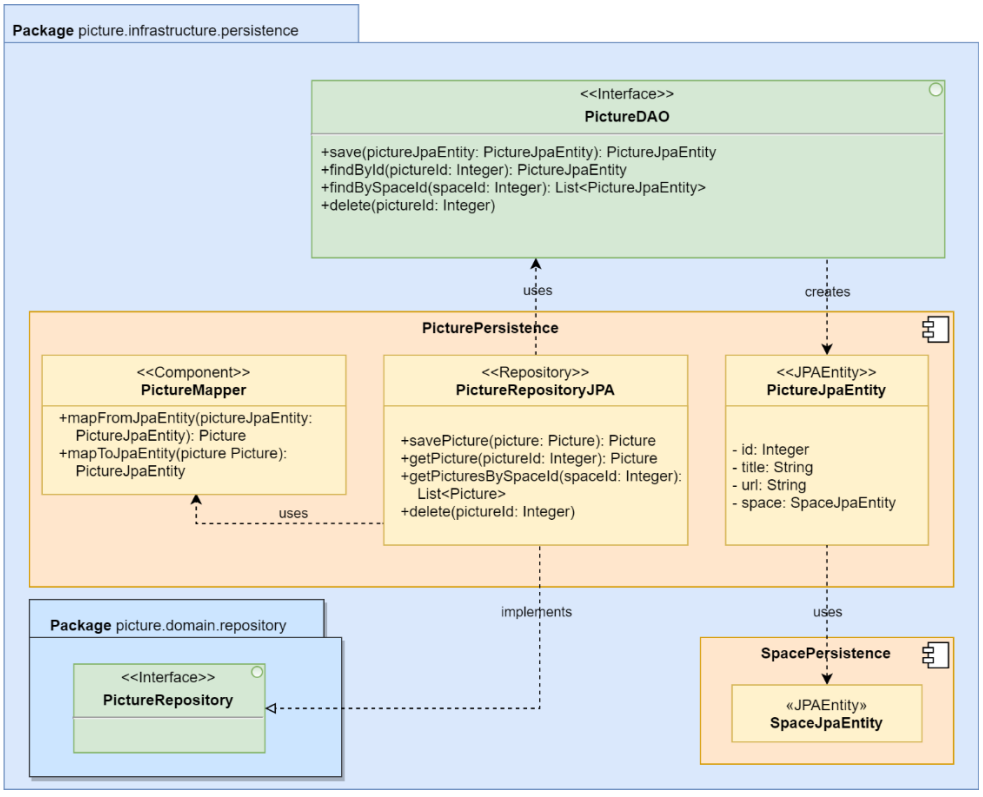


Ilustración 15. Refinamiento del paquete de persistencia de la imagen

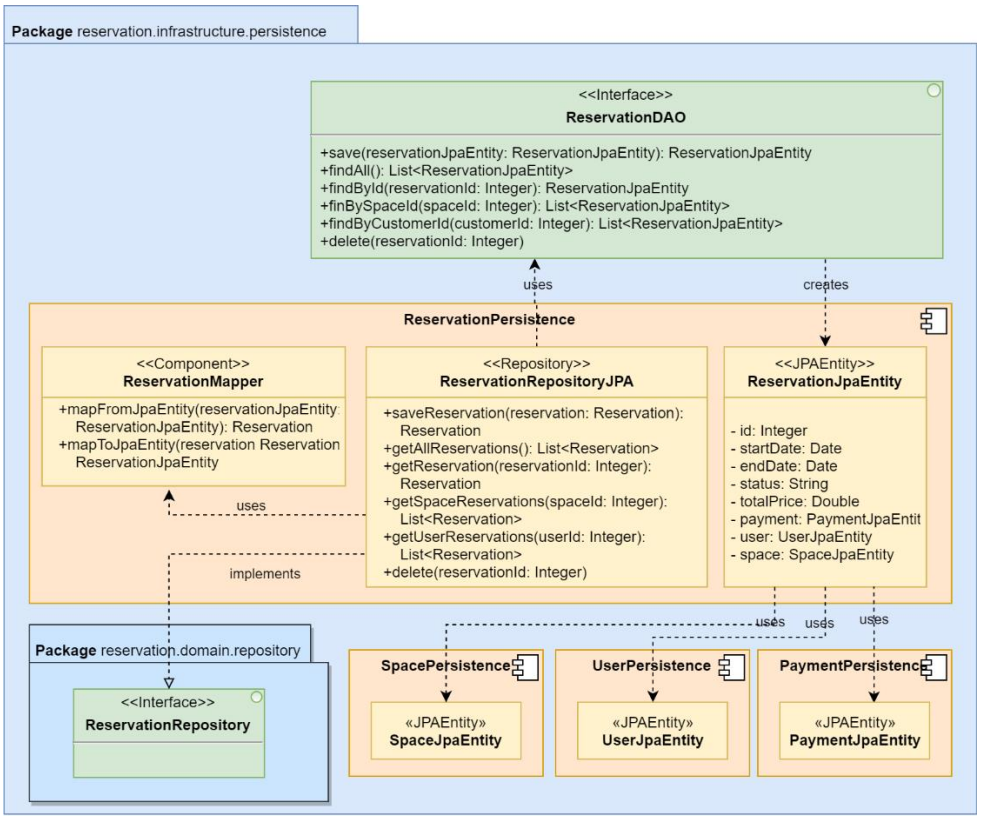


Ilustración 16. Refinamiento del paquete de persistencia de la reserva

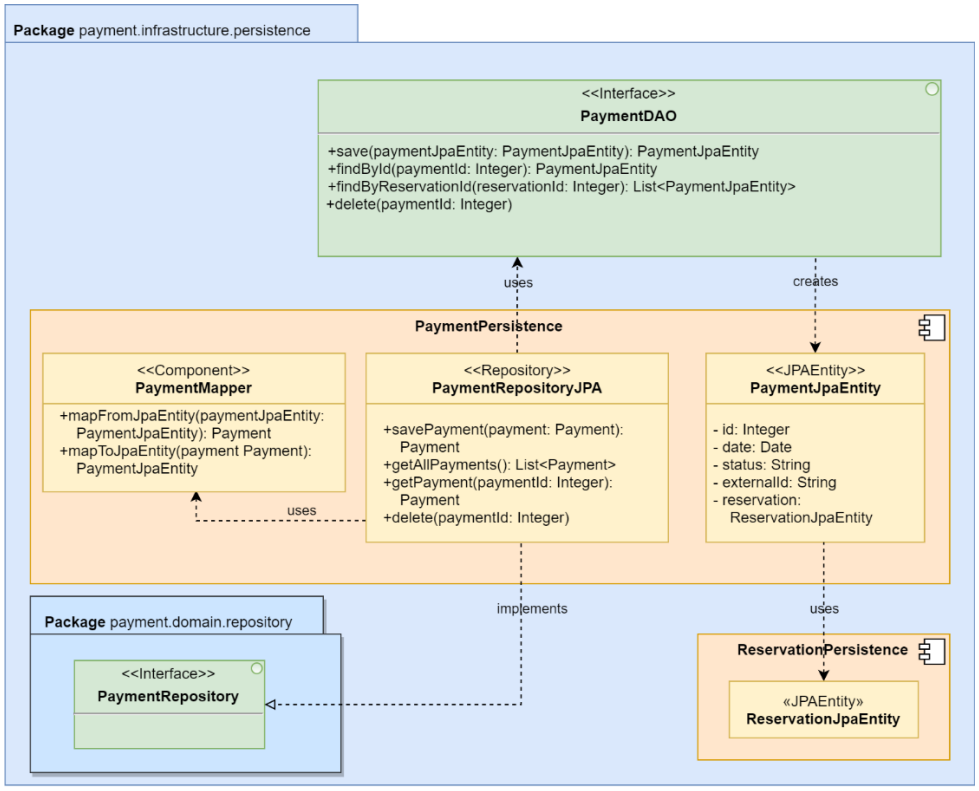


Ilustración 17. Refinamiento del paquete de persistencia de pago

3.4.1.1.2. Web

Dentro de la capa de infraestructura se encuentra el módulo web, que será el encargado de implementar los controladores de la API. Estos controladores recibirán un objeto tipo Request con el esquema de datos de entrada y devolverán un objeto ResponseEntity del *framework* Spring, que incluirá algunas particularidades de las respuestas HTTP.

Al igual que en el componente anterior, encontramos servicios de mapeo entre las entidades de dominio y las propias del componente. Estos objetos de consulta y respuesta se denominan DTO (Data Transfer Object) y se utilizan para encapsular y transportar datos entre los distintos componentes[13].

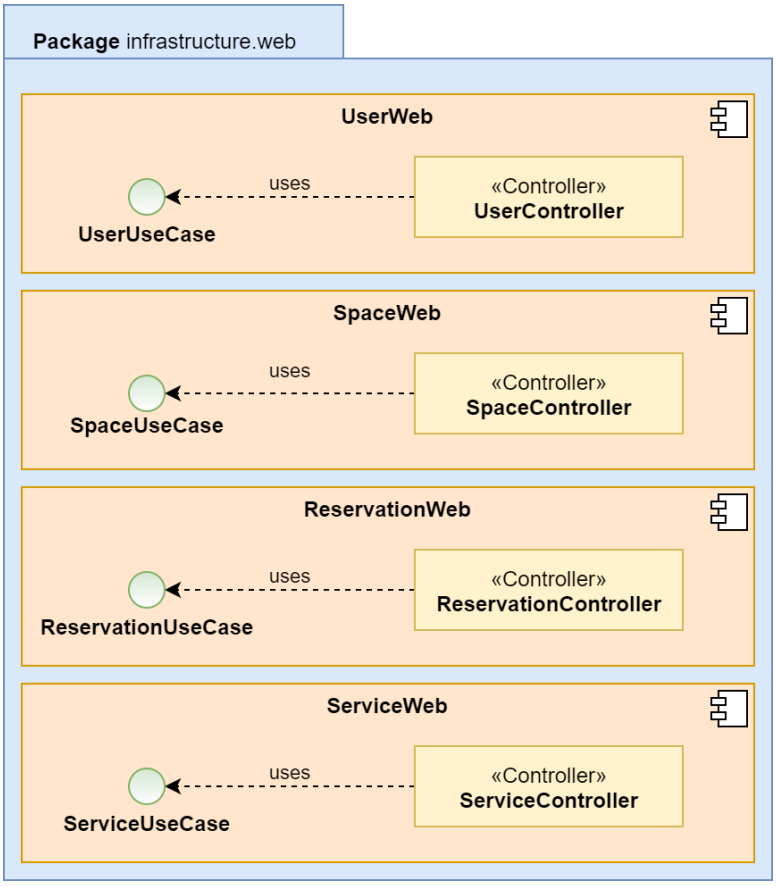


Ilustración 18. Componentes de la capa de infraestructura-web

[13] MSDN (2010). *Data Transfer Object. Microsoft MSDN Library*. Consultado desde http://msdn.microsoft.com/en-us/library/ms978717.aspx

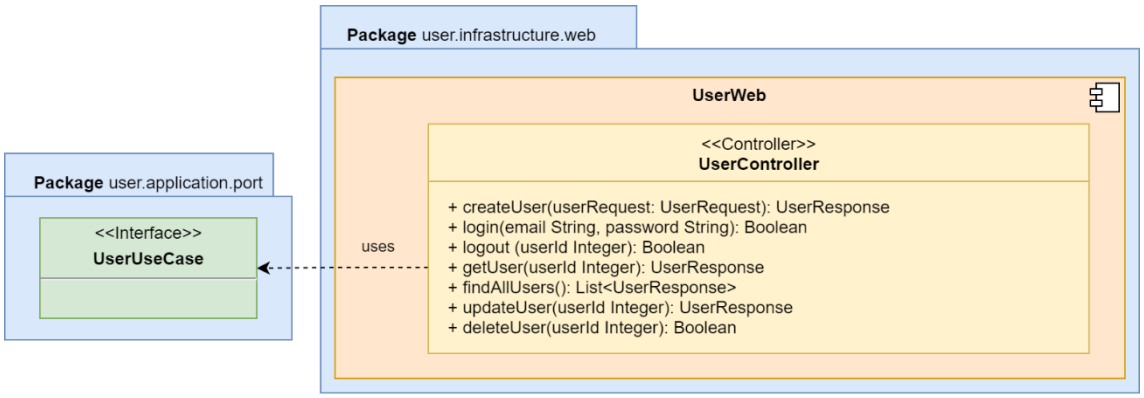


Ilustración 19. Refinamiento del paquete web de usuario

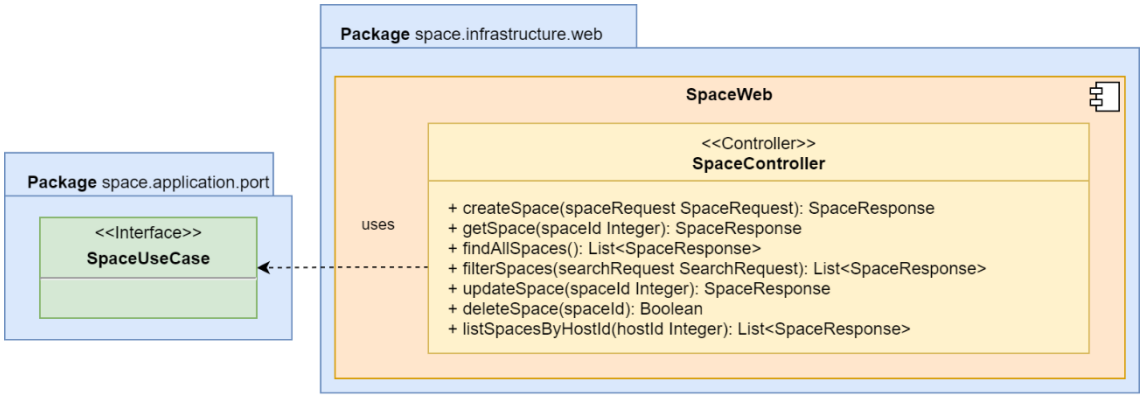


Ilustración 20. Refinamiento del paquete web de espacio de trabajo

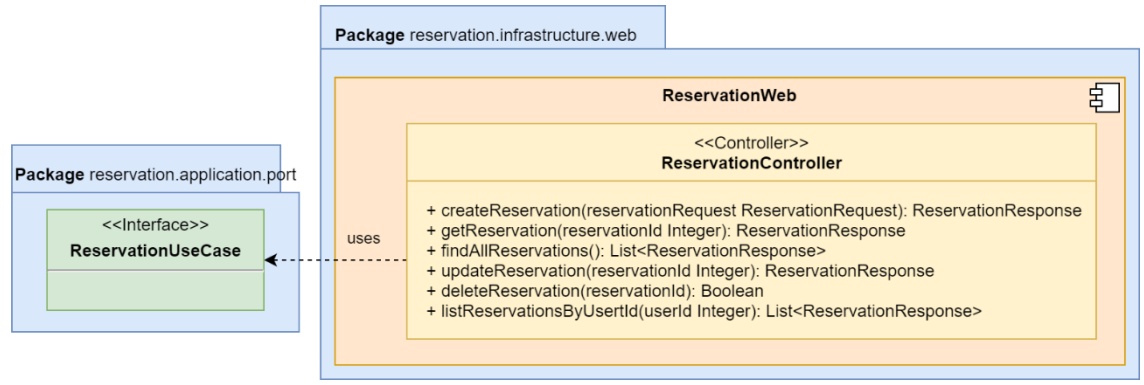


Ilustración 21. Refinamiento del paquete web de reserva

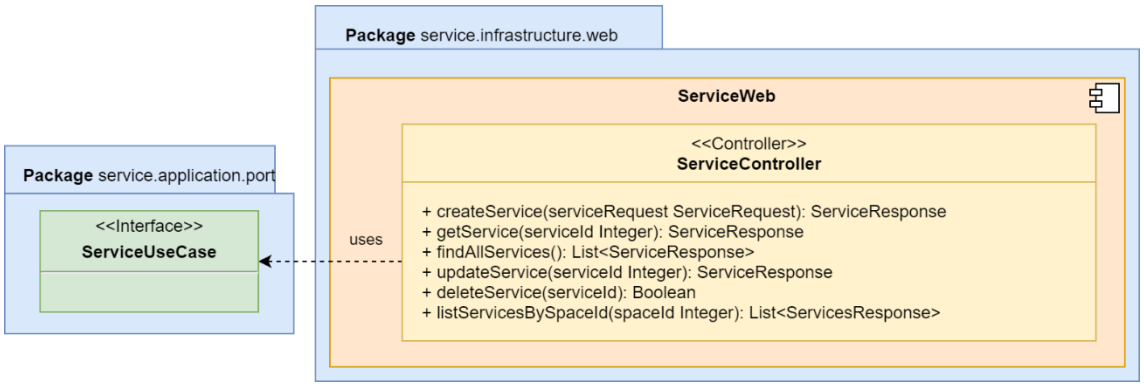


Ilustración 22. Refinamiento del paquete web de servicio

3.4.1.2. Capa de aplicación

En esta capa se definen las interfaces de casos de uso y los servicios que las implementan. Estos servicios reciben los objetos Request desde el controlador, los mapean a objetos de dominio y aplican sobre ellos la lógica oportuna. Una vez se hayan procesado se transformarán en objetos tipo Response y se devolverán al controlador.

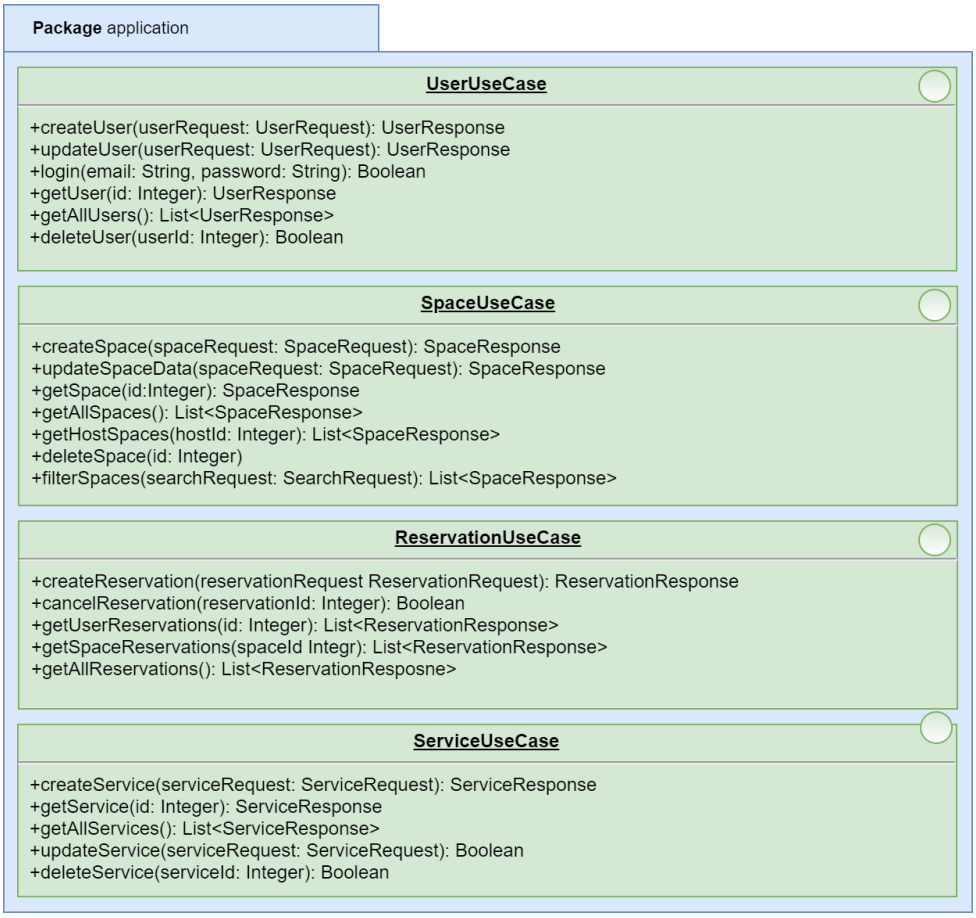


Ilustración 23. Interfaces de la capa de aplicación

Como podemos observar en el diagrama siguiente, los servicios de la capa de aplicación utilizan la interfaz de repositorio de la capa de dominio para interactuar con la base de datos.

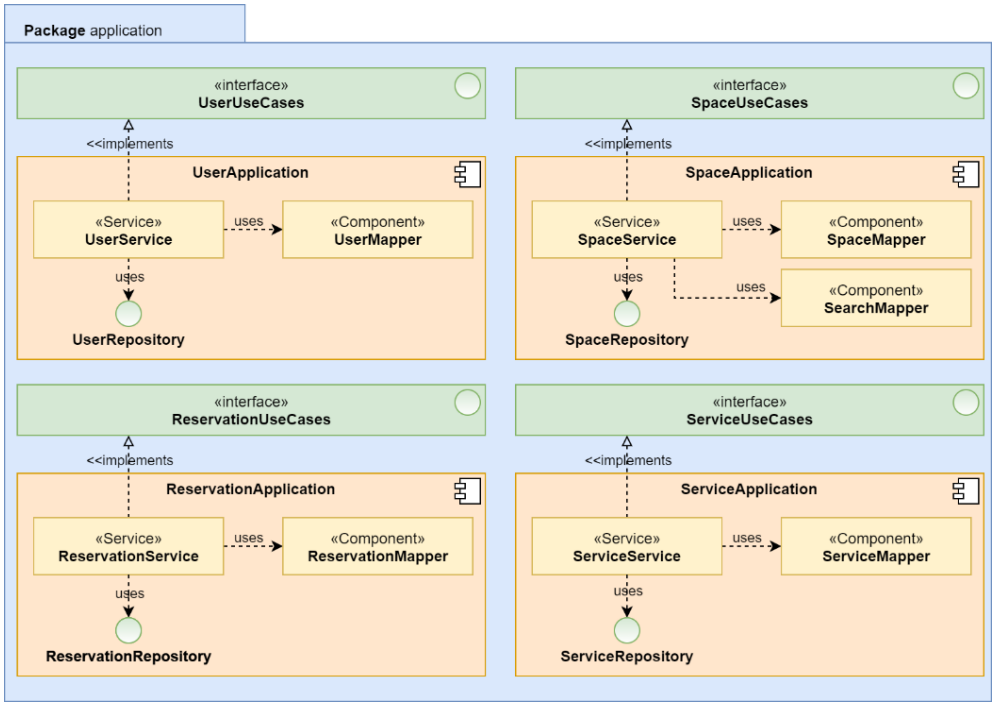


Ilustración 24. Componentes de la capa de aplicación

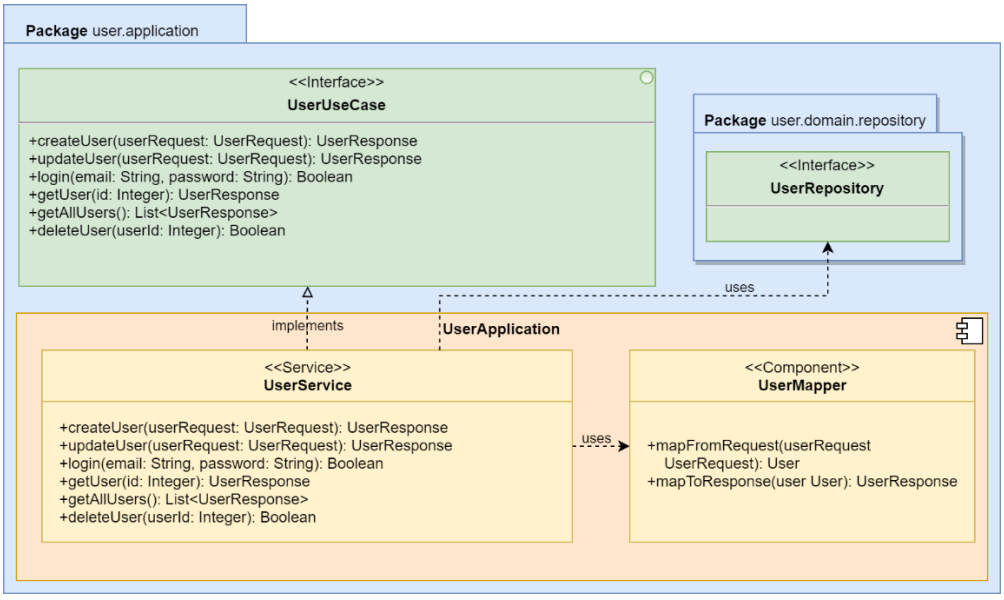


Ilustración 25. Refinamiento del paquete aplicación de usuario

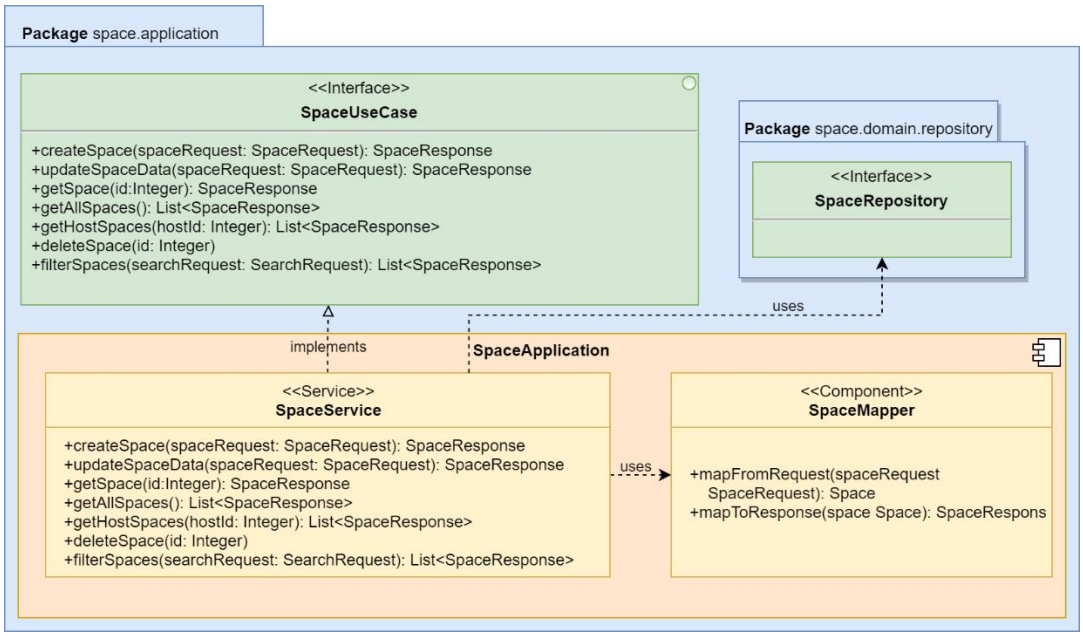


Ilustración 26. Refinamiento del paquete aplicación de espacio

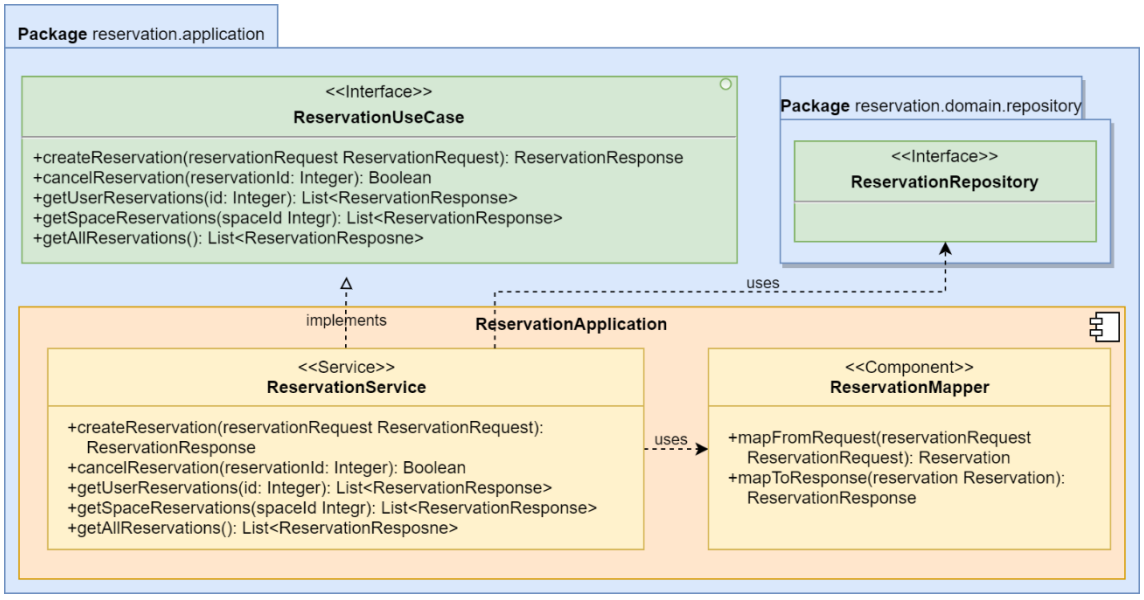


Ilustración 27. Refinamiento del paquete aplicación de reserva

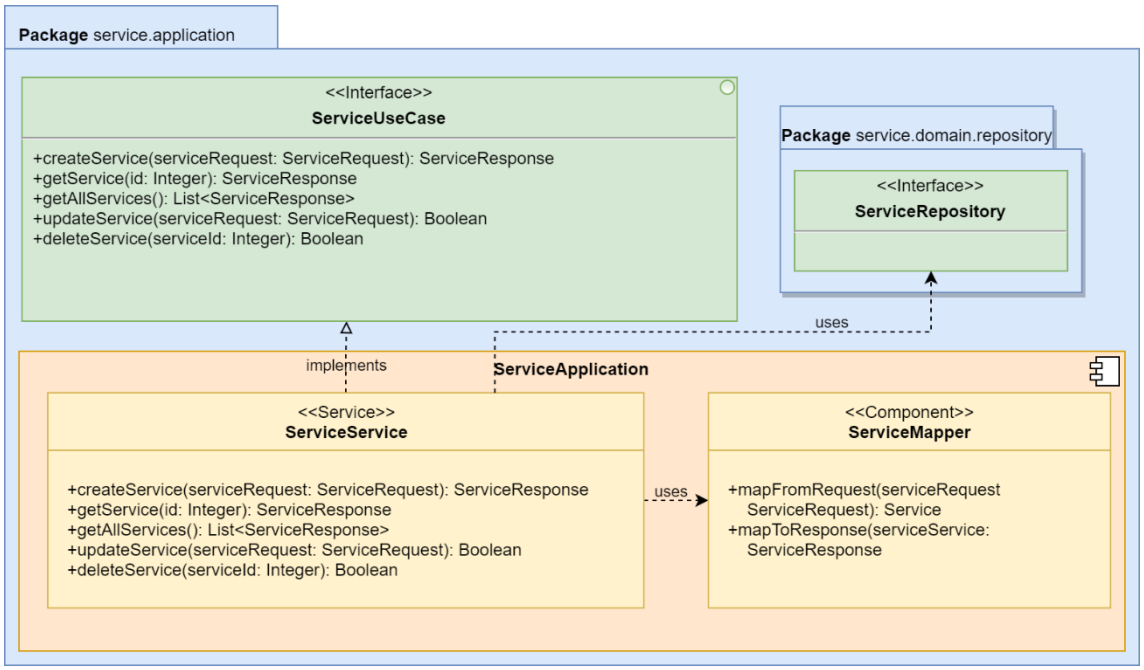


Ilustración 28. Refinamiento del paquete aplicación de servicio

3.4.1.3. Capa de dominio

En la capa de dominio están las interfaces de los repositorios y las entidades de dominio, cuyo detalle ya quedaría reflejado en el diagrama de clases del apartado 3.2, por lo que no necesitamos aplicar un mayor refinamiento.

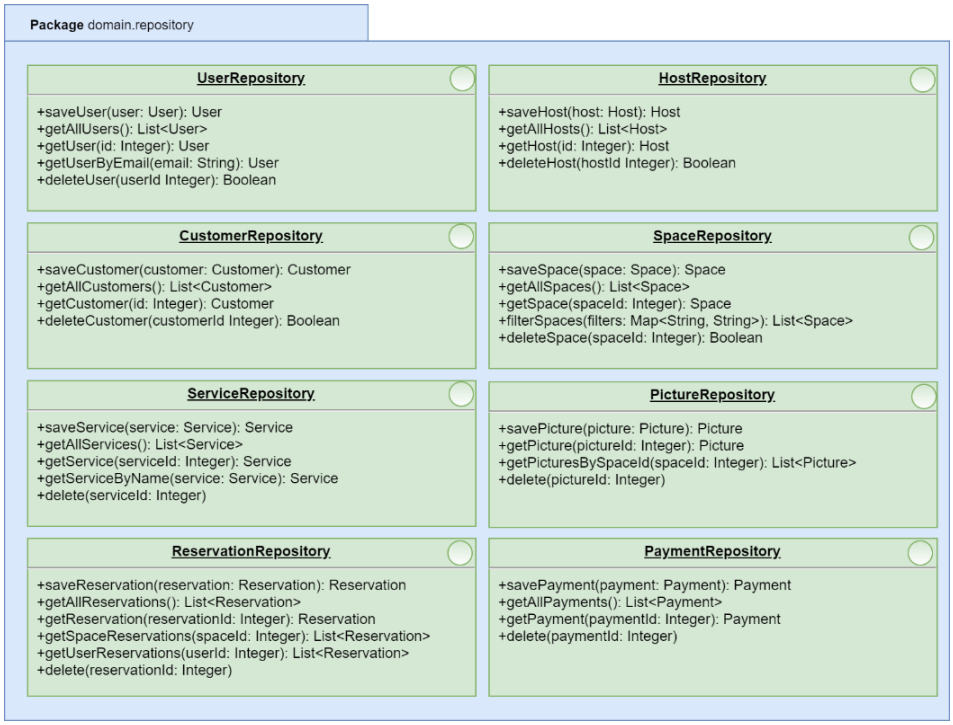


Ilustración 29. Interfaces de la capa de dominio

3.5. Diseño de pantallas

En este apartado se definirán las pantallas principales de la aplicación. No se incluyen algunos formularios, como el de dar de alta al usuario o el registrar un nuevo espacio en el sistema.

La pantalla inicial o “Home” permite introducir los datos de búsqueda de un espacio, darse de alta como usuario o identificarse.

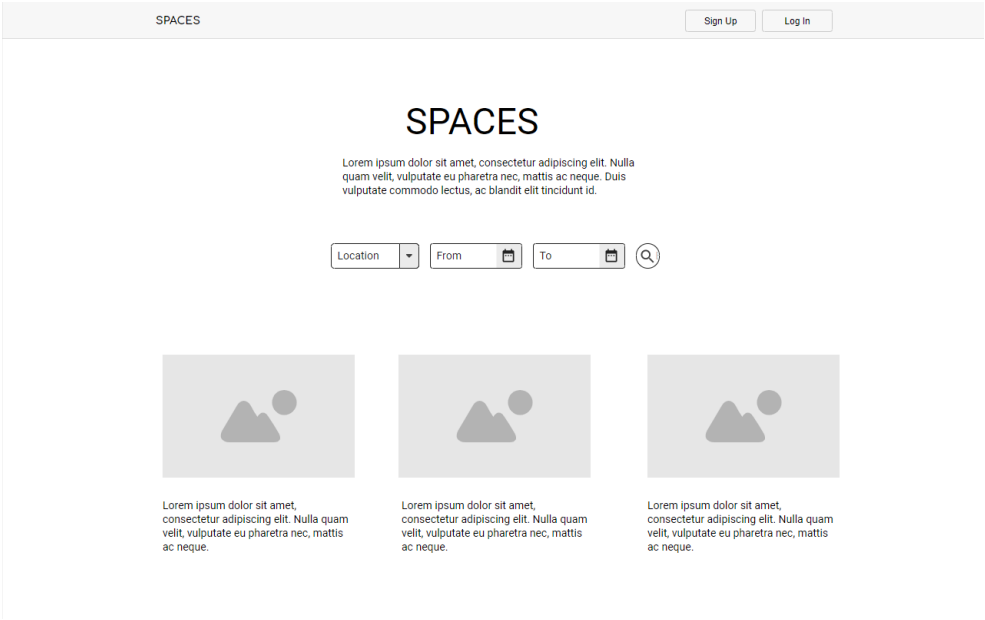


Ilustración 30. Pantalla "Home"

La pantalla de “Listado de espacios de trabajo” ofrece los espacios disponibles dentro de la ubicación y fechas seleccionados. Permite, además, visualizar las características principales de los espacios y aplicar filtros a los resultados.

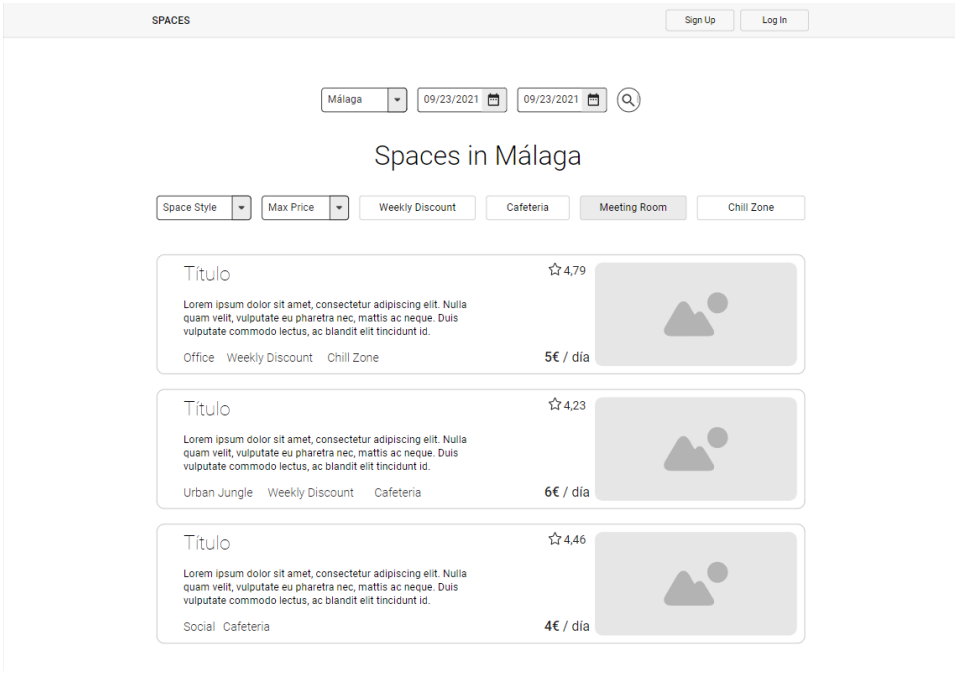


Ilustración 31. Pantalla "Listado de espacios de trabajo"

La pantalla de “Detalle de un espacio” muestra todas las propiedades del espacio seleccionado y permite visualizar el resto de las imágenes que haya disponibles. En esta se muestra también un cuadro resumen de la reserva y se permite efectuar esta acción.

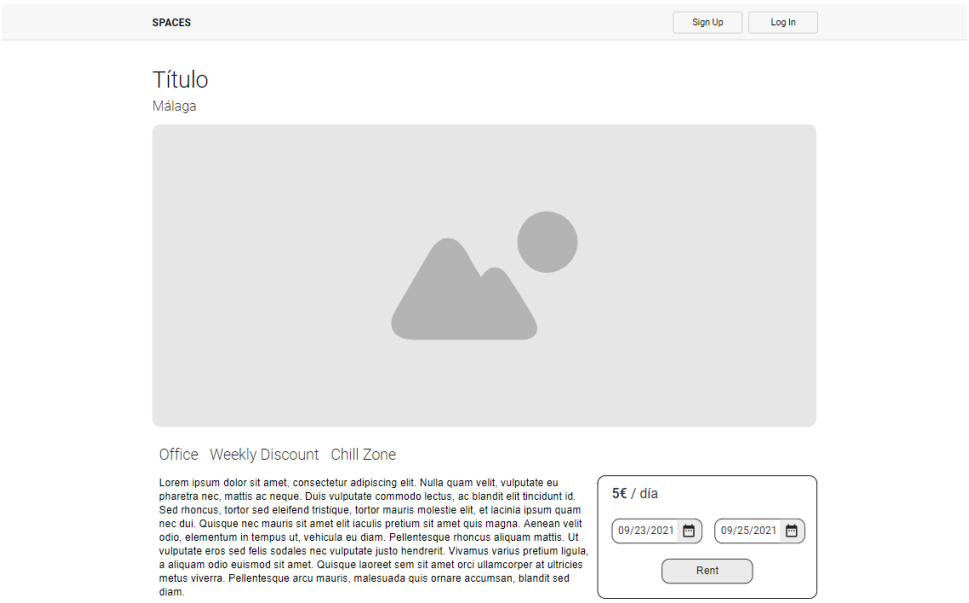


Ilustración 32. Pantalla "Detalle del espacio"

La pantalla de “Perfil de usuario” muestra los datos personales de este y permite modificarlos.

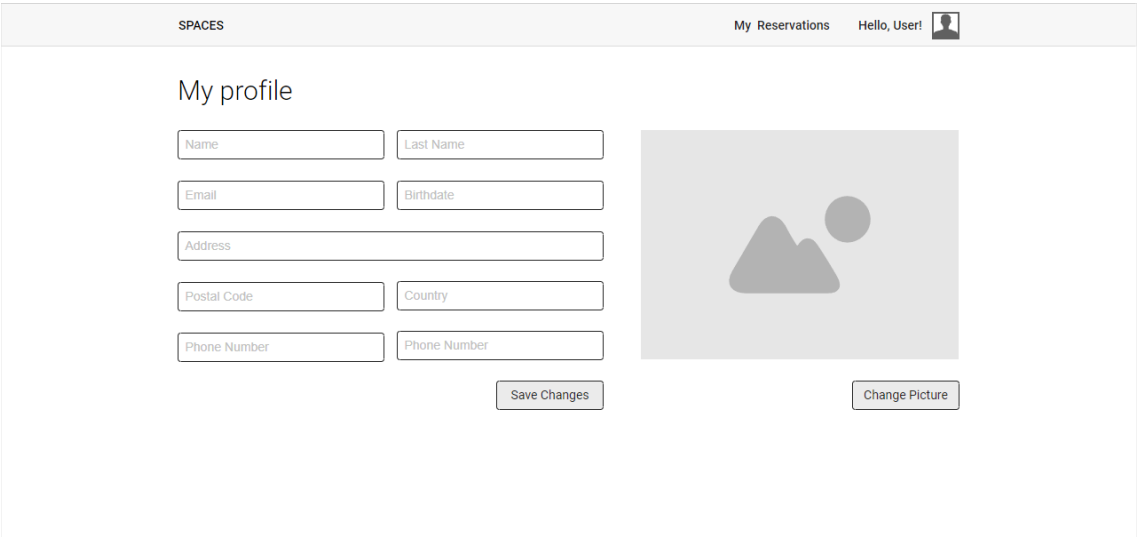


Ilustración 33. Pantalla "Perfil de usuario"

La pantalla “Mis reservas” muestra el historial de reservas del usuario, incluyendo detalles como el espacio, las fechas o el estado de la reserva.

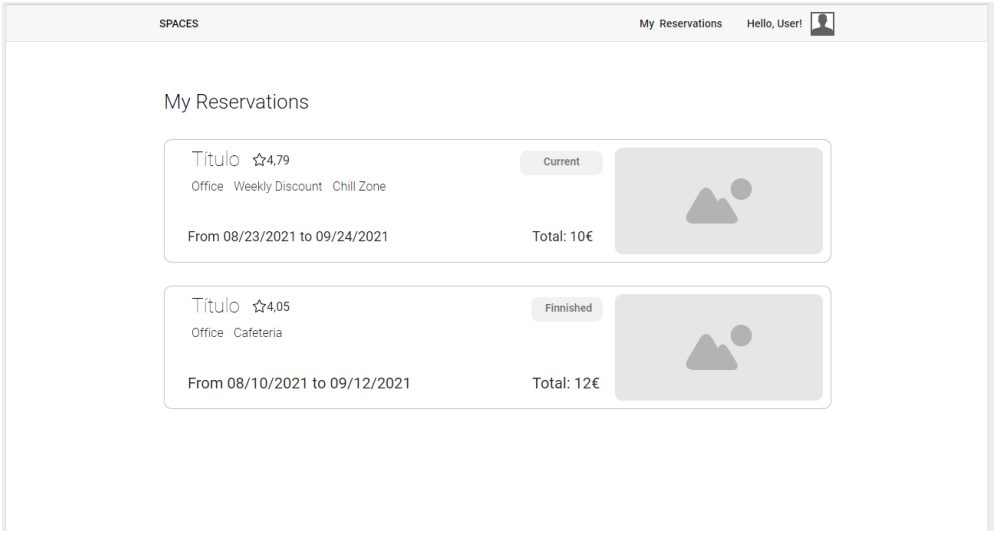


Ilustración 34. Pantalla "Mis reservas"

Los anfitriones tendrán la opción de visualizar los espacios de trabajo que están registrados a su nombre accediendo a la pantalla “Mis espacios”.

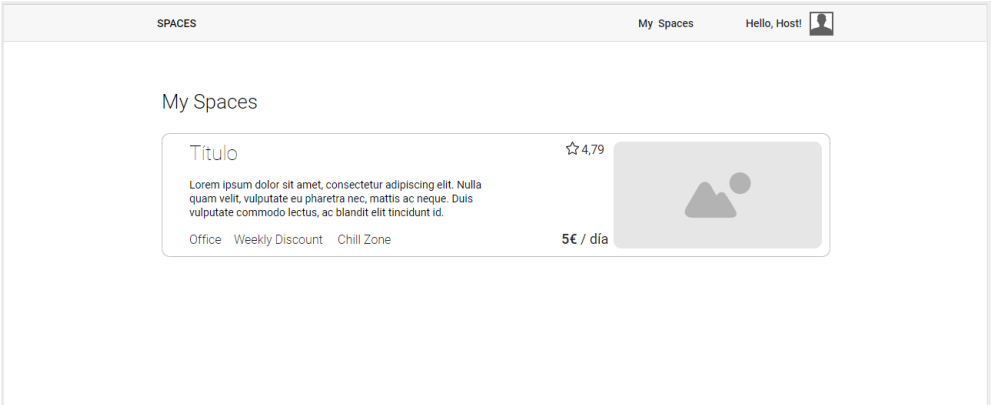


Ilustración 35. Pantalla "Mis espacios"

La pantalla “Panel de administración” muestra el menú de listados al que puede acceder el administrador.

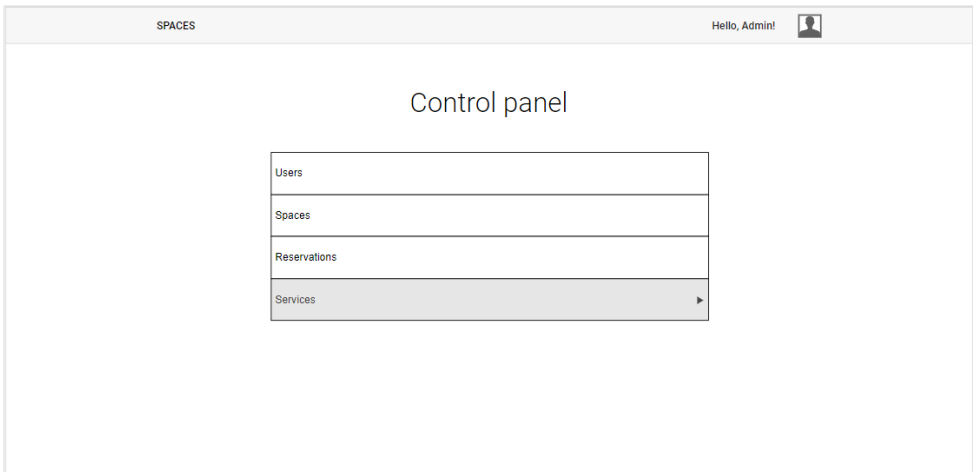


Ilustración 36. Pantalla "Panel de administración"

Como gran parte del contenido del portal lo aportan los usuarios, el administrador podrá gestionar únicamente las entidades de usuarios y servicios. Tanto los espacios de trabajo como las reservas serán elementos con permisos de lectura.

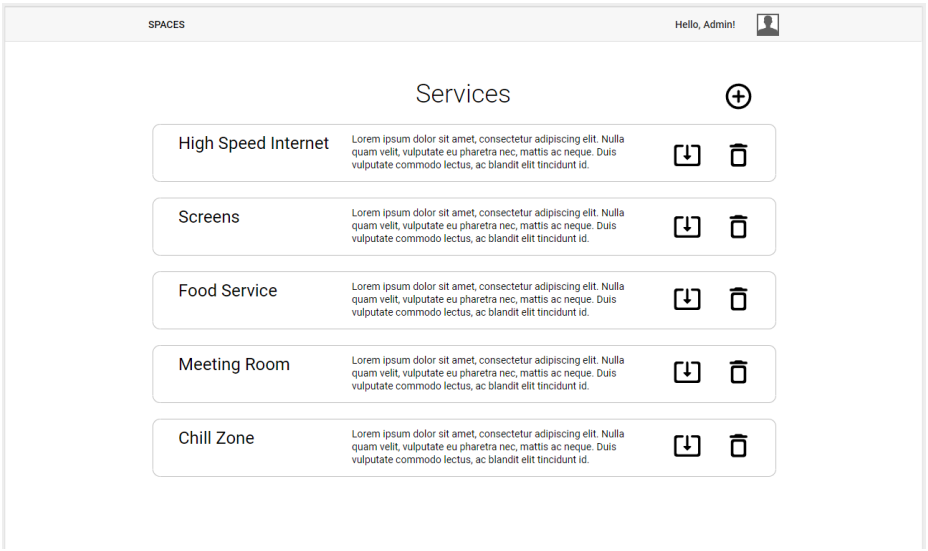


Ilustración 37. Pantalla "Administración de servicios"

4. Implementación

En este apartado se describirán algunas de las características y decisiones más relevantes durante el proceso de implementación de la aplicación.

4.1. Aplicación Java

La aplicación Java contendrá el código encargado de la parte *backend* del portal, incluyendo los controladores, los servicios y el acceso a datos.

4.1.1. Controladores

El principal producto obtenido tras el desarrollo de nuestro programa en Java es una API REST que permitirá la comunicación con la Web gracias a los controladores que se han definido. Estos se encuentran en la capa de infraestructura y se encargan de recibir datos en formato JSON y enviarlos a la capa de aplicación, que ejecutará la lógica de negocio sobre estos.

Para cada una de las entidades principales del sistema se ha diseñado una clase con la anotación @RestController de Spring y, dentro de esta, se han creado los *endpoints* para realizar operaciones CRUD y otras acciones necesarias para satisfacer los requisitos propuestos.

```
@PostMapping(path = "/services")
ResponseEntity<Object> createService(@RequestBody ServiceRequest request) {
    try {
        ServiceResponse service = serviceUseCase.createService(request);
        return ResponseHandler.createResponse(service, HttpStatus.CREATED, ResponseMessage.created,
                description: "A new service was created successfully!");

    }catch (DataIntegrityViolationException de){
        logger.error(String.valueOf(de.getRootCause()));
        return ResponseHandler.createResponse( responseObj: null, HttpStatus.CONFLICT, ResponseMessage.request_conflict,
                description: "There is a conflict with the data you requested. Cause: " + de.getRootCause());

    } catch (Exception e){
        return ResponseHandler.createResponse( responseObj: null, HttpStatus.INTERNAL_SERVER_ERROR,
                ResponseMessage.internal_server_error, description: "An error occurred in the server side. Cause: " +
                        e.getLocalizedMessage());
    }
}
```

Ilustración 38. Endpoint para crear un servicio

Una de las decisiones de implementación más relevantes en este punto fue la definición de la estructura de datos que se iban a manejar para las peticiones y las respuestas. Para los datos de entrada se han creado clases de tipo Request, que se han configurado con la anotación @Value de Spring para que contengan únicamente métodos *getter*.

Para precisar qué campos son obligatorios se ha utilizado la anotación @NotNull, además de un método llamado *validateSelf()*, que se llamará en el constructor de la clase y comprobará si se cumple la condición. Si alguno de los datos introducidos no es del tipo correcto o si es obligatorio y no está presente en la petición HTTP se devolverá un error de tipo Bad Request.

Por otro lado, en el objeto de respuesta, se ha recurrido a la clase ResponseEntity que ofrece Spring, añadiendo algunos campos personalizados. Utilizando un manejador de respuestas (ResponseHandler), se construye un objeto que conserva la misma estructura, independientemente de si se devuelven resultados correctos o un error. Los campos que se han incluido son:

- Data: datos de respuesta.
- Status: código de estado HTTP.
- Message: mensaje identificativo.
- Description: descripción del mensaje.

Se han empleado bloques try-catch para la clasificación de errores y la correcta determinación del estado HTTP que se debe devolver. Asimismo, nos hemos apoyado en la clase HttpStatus de Spring para la gestión de estos códigos de estado.

4.1.2. Servicios

En la capa de aplicación se encuentran los servicios encargados de recibir los objetos tipo Request enviados desde el controlador, mapearlos a entidades de dominio, aplicar la lógica de negocio y devolver un objeto tipo respuesta.

```
@Override
public ReservationResponse createReservation(ReservationRequest reservationRequest) throws ReservationNotValidException {
    if (reservationRequest == null) return new ReservationResponse();

    if (!validate(reservationRequest)) {
        return new ReservationResponse();
    }

    Reservation reservation = reservationRepository.saveReservation(reservationRequest.getStartDate(),
            reservationRequest.getEndDate(), Status.PENDING.toString(), reservationRequest.getSpaceId(),
            reservationRequest.getUserId());

    return reservationMapper.mapToResponse(reservation);
}
```

Ilustración 39. Servicio de creación de reserva

Si nos fijamos en el servicio de crear una reserva, podemos observar el flujo que se sigue. Como parámetro de entrada se recibe un objeto ReservationRequest que llega desde el controlador. En este caso se llama al método *validate()*, que

realiza una serie de comprobaciones según las fechas solicitadas y la disponibilidad del espacio. Una vez validada la petición se solicita al repositorio de reservas que realice el guardado de esta en base de datos.

```
public boolean validate(ReservationRequest reservationRequest) throws ReservationNotValidException {
    if (!datesAreValid(reservationRequest.getStartDate(), reservationRequest.getEndDate())) {
        throw new ReservationNotValidException("Start date must be before today, end date must be after start date " +
                "and the maximum time period to rent is 32 days");
    }

    if (userAlreadyBooked(reservationRequest)) {
        throw new ReservationNotValidException("User already has an active reservation between these dates.");
    }

    if (!hasAvailableSeats(reservationRequest.getSpaceId(), reservationRequest.getStartDate(), reservationRequest.getEndDate(),
            reservationId: null)) {
        throw new ReservationNotValidException("There are no available seats for these dates.");
    }

    return true;
}
```

Ilustración 40. Método para la validación de la reserva

Se ha decidido definir la lógica de validación en esta capa ya que se considera que forma parte de la lógica de negocio. Se comprobará no sólo que las fechas sean coherentes, sino también, el periodo máximo de reserva que ha sido establecido en los requisitos y la disponibilidad de asientos en el espacio solicitado.

La interfaz de repositorio devolverá un objeto Reservation de dominio, que tendrá que ser mapeado a uno de tipo ReservationResponse y devolverse al controlador. Este mapeo entre objetos se realiza a través de componentes llamados *mappers,* que, como ya hemos mencionado en el apartado de arquitectura, se encargan de transformar un objeto en otro para poder utilizarlo en diferentes capas.

4.1.3. Persistencia

En la capa de infraestructura también se hallan las clases dedicadas a la persistencia de datos. Si retomamos el flujo del apartado anterior, cuando el servicio llama a la interfaz de repositorio, esta es implementada por los adaptadores JPA. Estas clases utilizarán una interfaz DAO, que se encargará de realizar los accesos a la base de datos y devolver los resultados.

Gracias a que la interfaz DAO extiende de la clase JpaRepository de Spring, es posible prescindir de la definición de consultas básicas CRUD en SQL, debido a que ya están definidas por esta clase. Solo en algunas consultas de mayor complejidad ha sido necesario añadir una anotación con la consulta SQL.

Una de las decisiones de implementación tomadas en esta capa ha sido la de incluir los métodos de filtrado es espacios. Aunque estos contienen cierta lógica que podría considerarse de negocio, se ha decidido incluirlos en la clase SpaceJpaRepository. El motivo es que se considera que este filtrado debe ser responsabilidad de las clases que acceden a los datos y que estos métodos podrían incluso no ser requeridos si, en un futuro, se opta por otro *framework* o tipo de base de datos.

En el módulo de persistencia también se encuentran los *mappers* destinados a traducir entre objetos de dominio y de persistencia. Tienen, por lo tanto, una función análoga a la de los *mappers* de la capa de servicio.

Para la representación de tablas se han utilizado entidades JPA. JPA hace referencia a la API de persistencia desarrollada para Java EE[14] y permite la definición de tablas, columnas y restricciones propias de una base de datos relacional. En nuestro caso se han definido las claves primarias, claves foráneas y restricciones de unicidad y nulos para que la base de datos pueda ser generada automáticamente al iniciar la aplicación.

4.1.4. Dominio

En la capa de dominio se han situado las entidades, los tipos enumerados y las excepciones. No entraremos en mayor detalle sobre esta capa, ya que solo contiene objetos y métodos básicos de Java.

4.2. Aplicación React

La aplicación React supone la interfaz de usuario que conectará con la API. Hemos establecido cuatro flujos según el rol de usuario, incluyendo el acceso restringido a ciertas secciones y diferentes funcionalidades disponibles. En este apartado se describirán algunas de las funcionalidades más interesantes, aunque no las nombraremos todas.

La primera vez que accedemos a la aplicación nos encontramos con la pantalla principal y nuestro usuario tendrá el rol de usuario no registrado. Los principales elementos que se podrán observar son:

- Barra de navegación superior: da acceso a los menús principales de cada usuario y contiene los enlaces para volver a la pantalla “Home”, registrarse, loguearse, consultar los datos de usuario o salir del sistema.

[14] *Spring Data JPA*. Spring. Consultado desde https://spring.io/projects/spring-data-jpa

- Componente central: ofrece una acción diferente según el rol del usuario que esté identificado en el portal.
- Tarjetas de acceso a los menús de usuario: encontramos los mismos enlaces que en la barra superior, pero incluyendo una imagen y un breve texto explicativo.
- Footer: sección en la parte inferior con algunos enlaces de interés.

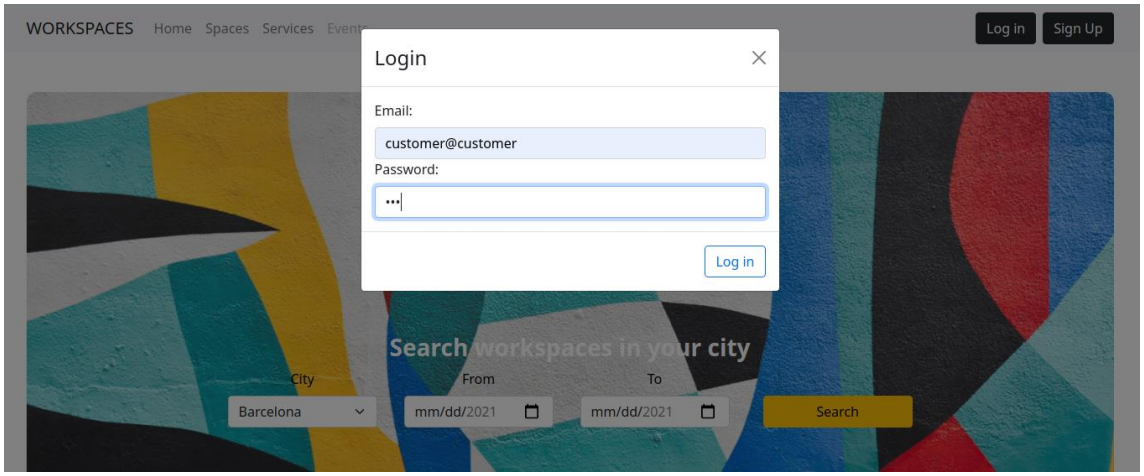


Ilustración 41. Pantalla de Login

Desde la pantalla “Home” del usuario no registrado se puede acceder al formulario de registro o de Login y se pueden realizar búsquedas de espacios por ciudad entre dos fechas. También se podrá visualizar el listado de todos los espacios y de los servicios que se ofrecen.

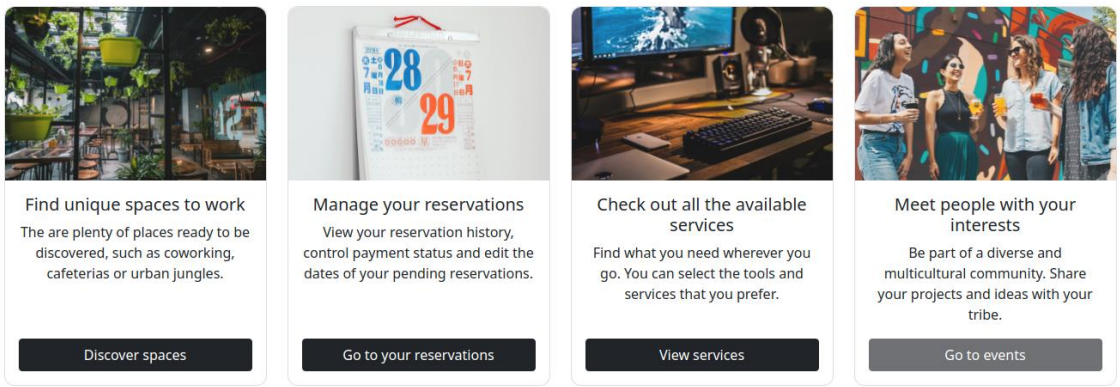


Ilustración 42. Sección de tarjetas de usuario registrado

Tanto los usuarios no registrados como los registrados tendrán una sección con tarjetas que darán acceso a las páginas que tienen disponibles según su rol.

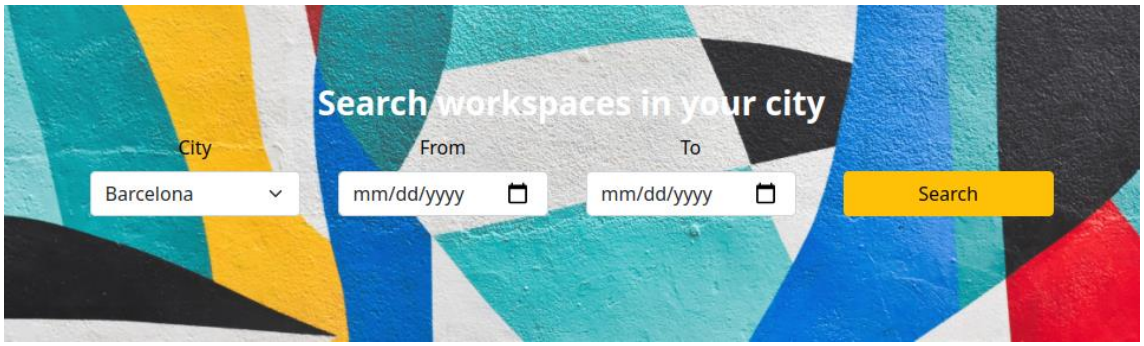


Ilustración 43. Cuadro de búsqueda de espacios entre dos fechas

Cuando se realiza una búsqueda, la interfaz redirigirá al listado de espacios disponibles, donde se podrán aplicar filtros para establecer el tipo de espacio, el precio máximo por día y la lista de servicios que deben ofrecer.

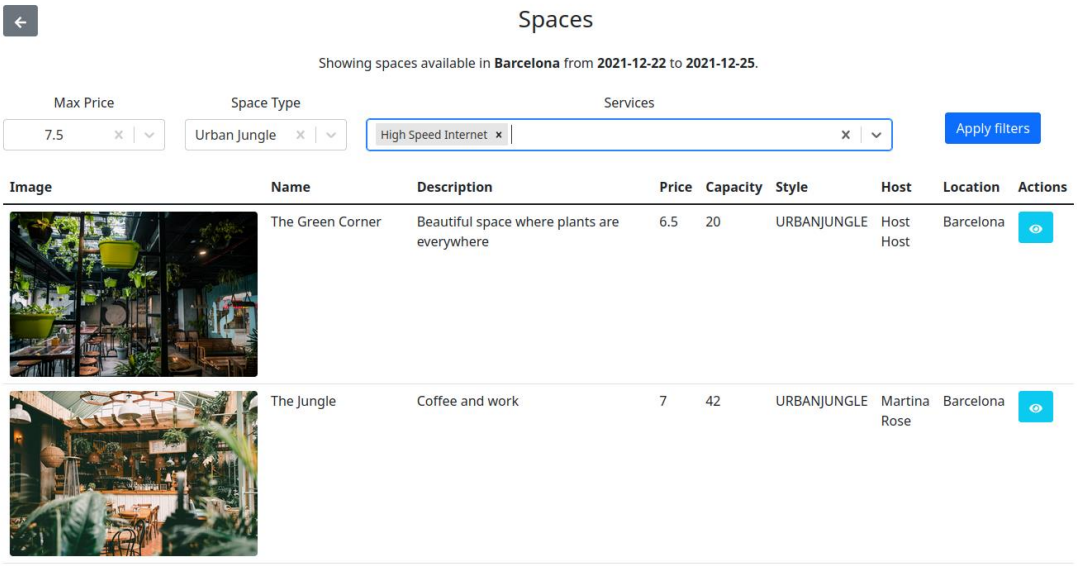


Ilustración 44. Pantalla de espacios filtrados

Seleccionando el botón con el icono del ojo azul se accede a la pantalla del detalle del espacio y, si el cliente está identificado en el sistema, podrá realizar la reserva presionando el botón de “Make Reservation”.

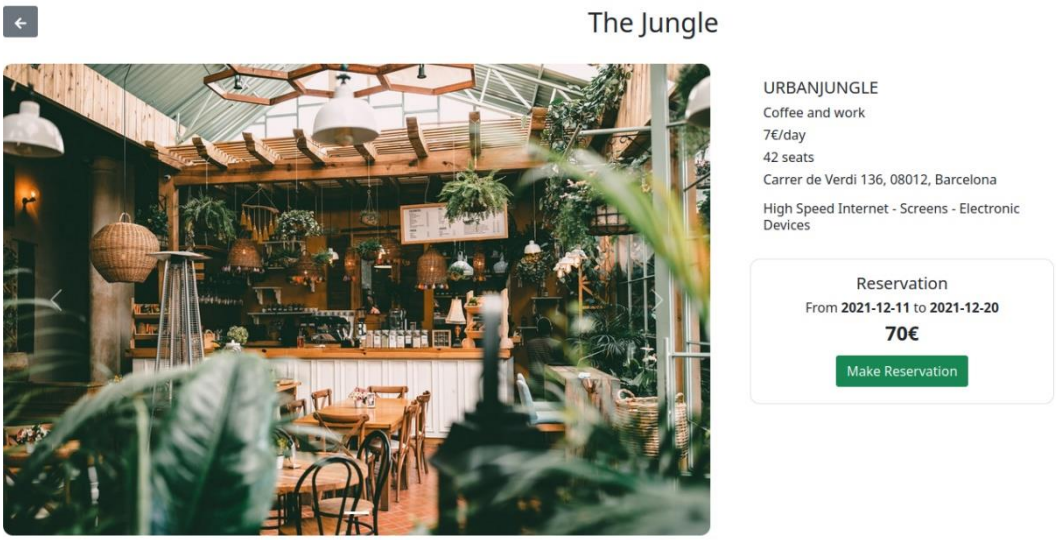


Ilustración 45. Pantalla de detalle del espacio

Una vez realizada la reserva, aparecerá en el listado de reservas del usuario, que hemos clasificado en dos grupos para una mejor legibilidad: reservas pendientes e histórico. El histórico contendrá las reservas completadas y canceladas.

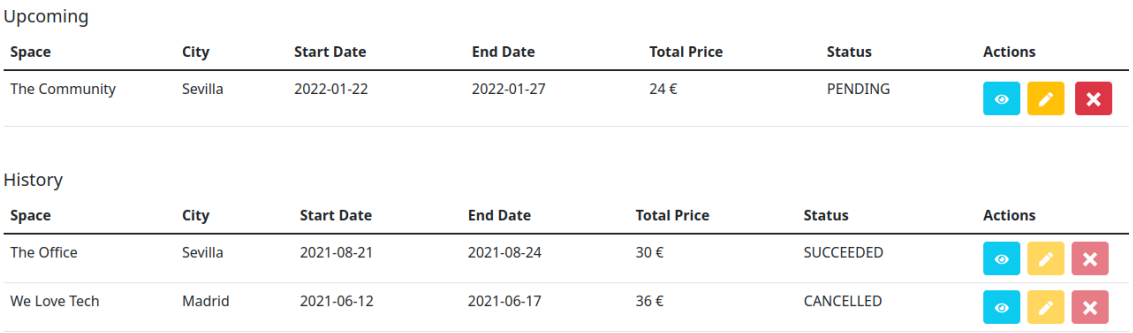


Ilustración 46. Pantalla de listado de reservas del usuario

Desde el listado de reservas de usuario se podrá acceder a los detalles de esta, donde se mostrarán las características del espacio, las fechas, el precio total y el estado de la reserva. También se podrán modificar las fechas o cancelar la reserva en cuestión.

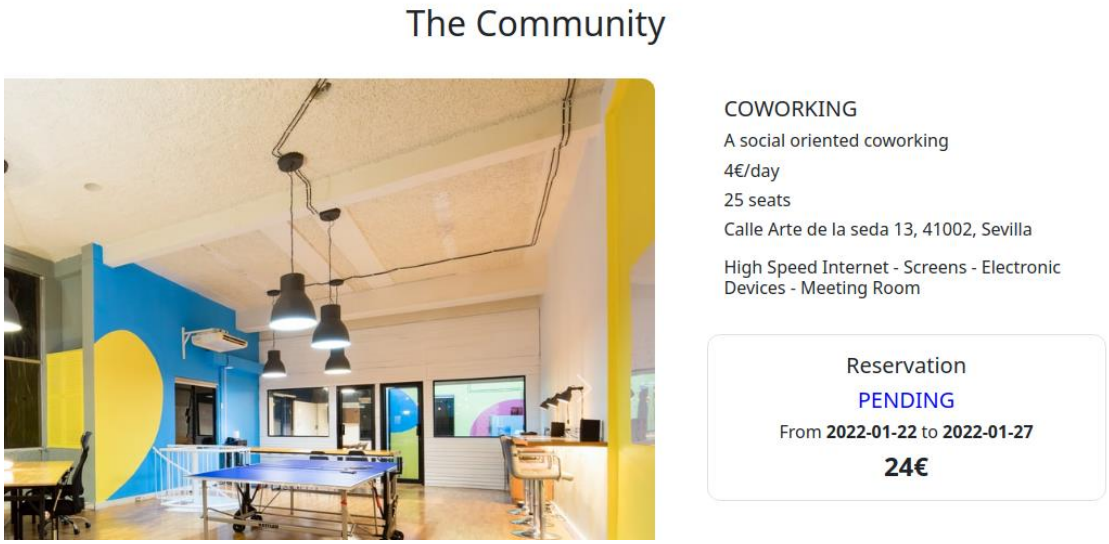


Ilustración 47. Pantalla con el detalle de la reserva

Si nos identificamos en el sistema como anfitrión comprobaremos que cambian algunos detalles de la “Home”, además de los menús con las secciones a las que tenemos acceso, tanto en la barra de navegación superior como en las tarjetas de la parte inferior.

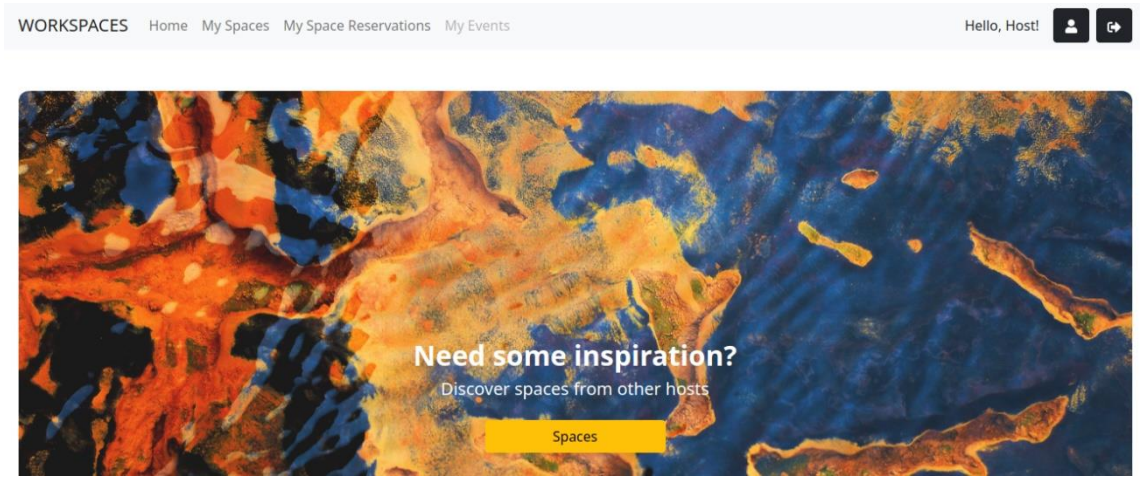
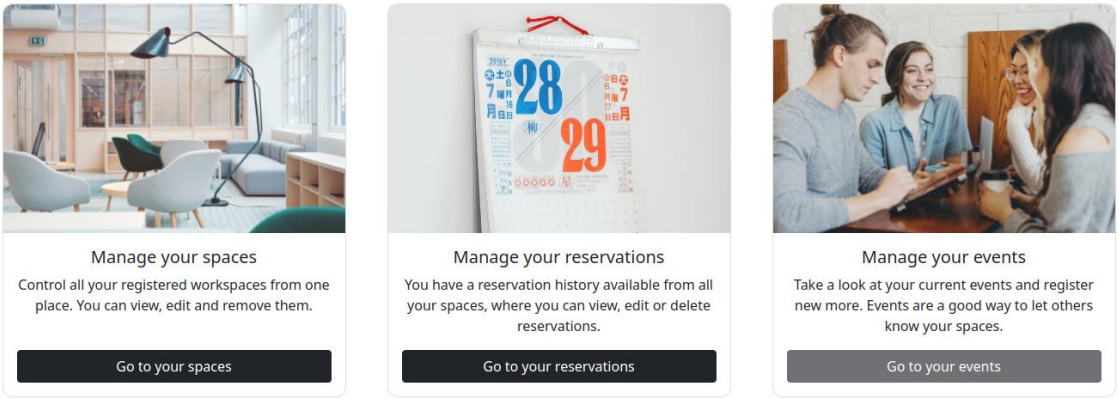


Ilustración 48. Pantalla Home del anfitrión

El anfitrión podrá acceder al listado de espacios que tenga registrados en el portal y al listado de reservas que los usuarios hayan realizado en sus espacios. Se ha incluido la sección “Eventos” sin habilitar para comentar esta funcionalidad en el apartado de Trabajo Futuro.



En el caso del administrador también dispondrá de su propia página principal y tendrá permisos para acceder a la gestión de las entidades principales del sistema a través de listados similares a los que han sido presentados para otros roles.

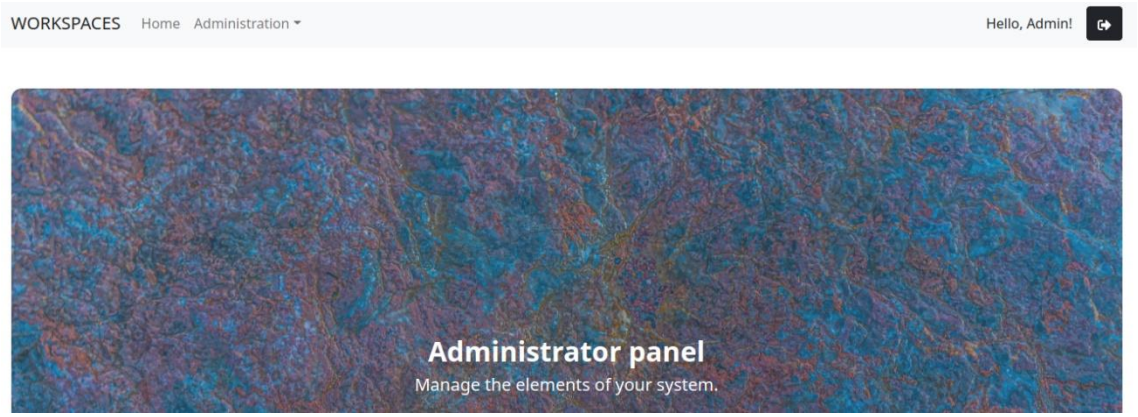


Ilustración 49. Pantalla Home del administrador

Por otro, nos gustaría comentar que se han añadido validadores en todos los formularios, que verificarán que se han introducido todos los datos obligatorios y con el tipo correcto antes de enviar la petición al *backend*.

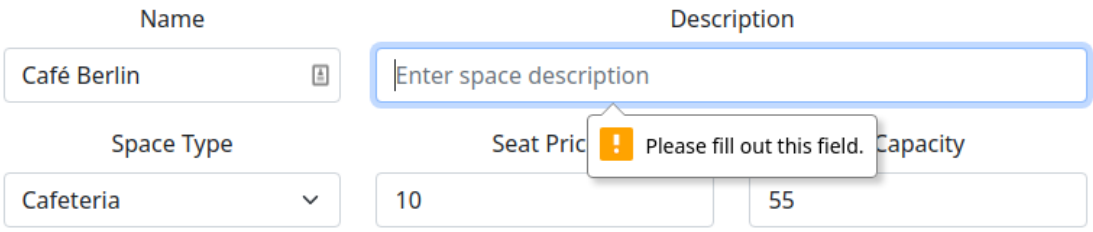


Ilustración 50. Ejemplo de validación de un formulario

Además, contamos con un sistema de alertas que indican al usuario si la acción se ha podido llevar a cabo o si ha ocurrido algún error. También se han incluido cuadros de diálogo para confirmar acciones más delicadas, como el borrado de los usuarios o espacios.

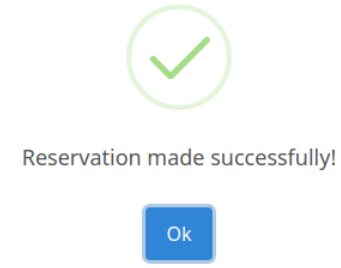


Ilustración 51. Mensaje de confirmación de creación de reserva

Por último, nos gustaría comentar que se ha incorporado un sistema de manejo de errores en cada llamada a la API, que permite al usuario entender qué ha ocurrido y cómo puede solucionarlo en el caso de que sea posible. Además, se ha restringido el acceso a páginas según el rol de usuario, en el caso de que este intente introducir manualmente una URL para la que no tiene acceso.

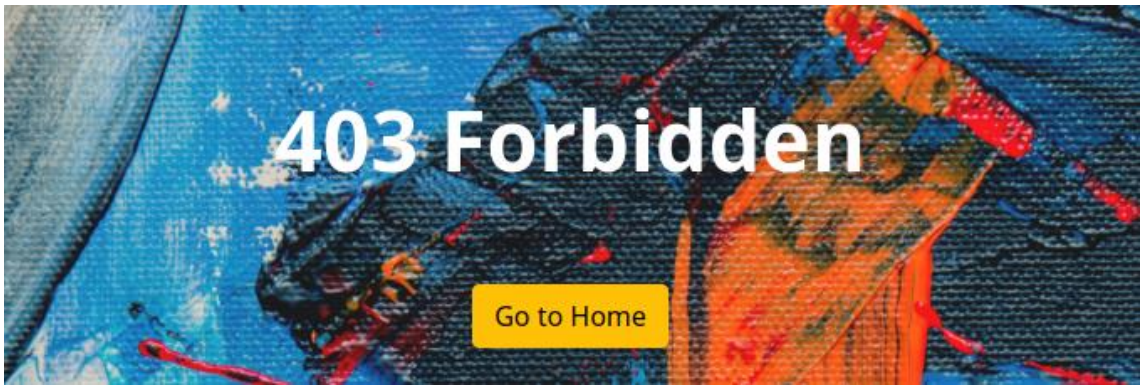


Ilustración 52. Pantalla de acceso restringido

4.3. Seguridad

Para añadir una capa de seguridad a la aplicación se ha utilizado el marco Spring Security, que proporciona opciones de autenticación y autorización, además de otras características relacionadas con la seguridad[15].

En primer lugar, se ha empleado la interfaz PasswordEncoder para encriptar la contraseña del usuario antes de guardarla. Cuando un usuario se registra, su contraseña se envía a la aplicación *backend*, que se encarga de encriptarla y almacenarla en la base de datos.

Cuando el usuario realiza la identificación en el sistema, se comprueba si la contraseña enviada obtiene *match* con la contraseña encriptada. De esta forma se logran dos ventajas:

1. Si ocurre un acceso no autorizado a la base de datos, el intruso no podrá revelar las contraseñas.
2. La API comprueba que las contraseñas coinciden, pero nunca las desencripta ni las devuelve como respuesta al *frontend*.

En segundo lugar, se ha añadido un token a los *endpoints* restringidos de la API utilizando el estándar JWT (JSON Web Token) y, nuevamente, el módulo de seguridad de Spring. El procedimiento es el siguiente:

1. El usuario se identifica en el sistema con sus credenciales.
2. El sistema valida las credenciales y devuelve un token.
3. El usuario puede realizar peticiones a la API utilizando ese token por un periodo de tiempo limitado.

Para facilitar las tareas de testeo de la aplicación, los tokens que se generan tienen una fecha de caducidad de un año, pero esta y otras configuraciones se podrían modificar con la implementación actual.

[15] *Spring Security*. Spring. Consultado desde https://spring.io/projects/spring-security

5. Evaluación y pruebas

En esta sección se llevará a cabo un repaso de los métodos utilizados para testear la aplicación y se comentarán los resultados obtenidos.

5.1. Pruebas funcionales

Para probar los *endpoints* de la API se ha optado por crear una colección en el programa Postman y definir una petición HTTP por cada uno de los casos de uso que se deben verificar. En líneas generales, se ha puesto a prueba el proceso completo de las acciones CRUD, desde que se envía la petición a la API hasta la persistencia en la base de datos.

Además, no solo se han demostrado los casos de éxito, sino que también se han evaluado las diferentes respuestas de error que, según nuestro diseño, deberían esperarse. Por ejemplo, se ha probado enviar tipos de datos no permitidos o nulos en campos que son obligatorios.

Una muestra de las diversas pruebas funcionales ejecutadas la podemos encontrar en el archivo “Pruebas” adjunto, donde se han descrito algunas de las más interesantes. No obstante, si se desean realizar comprobaciones adicionales, también se ha incluido una colección de peticiones HTTP ya definidas que podrá ser importada en el programa Postman.

5.2. Test unitarios

Los test unitarios han sido utilizados, principalmente, para probar la capa de aplicación y la lógica de negocio, incluyendo las clases de servicios y los *mappers*. También se han empleado para evaluar ciertas lógicas de filtrado de los módulos de persistencia. Contamos con un total de 82 test unitarios agrupados en las siguientes clases:

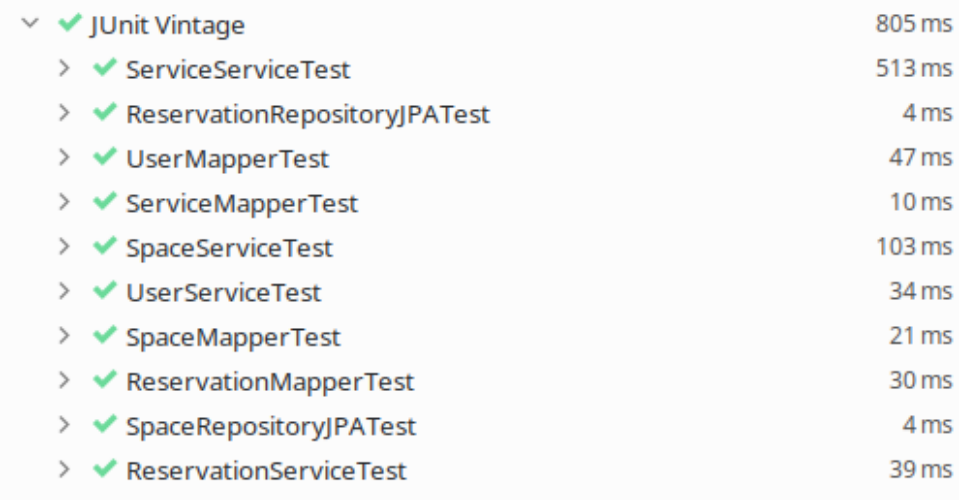


Ilustración 53. Clases que contienen las pruebas unitarias

La plataforma que se ha seleccionado para implementar los test en Java ha sido JUnit5, complementada con el *framework* Mockito. JUnit5 permite definir las pruebas utilizando la anotación @Test y pone a nuestra disposición una gran variedad de utilidades, mientras que Mockito está especialmente orientado a la creación de objetos de prueba.

En cada clase de servicio hemos probado todos los métodos principales devuelven el resultado esperado y que se ha ejecutado la lógica de negocio correspondiente, haciendo uso de respuestas *mock* en la llamada a servicios de otras capas, como es el caso de las interfaces de repositorio o de los *mappers*. De esta manera, se prueba únicamente el código concreto de cada clase.

En la clase SpaceRepositoryJpa se han testeado los métodos de filtrado, incluyendo la cobertura total de estos y los casos de tipos nulos. Entre los métodos se incluyen el de los espacios disponibles entre dos fechas o si un espacio dispone de todos los servicios que se han solicitado.

En la clase SpaceRepositoryJpa se ha evaluado el método *calculateTotalPrice()* que se utiliza para calcular el precio total de una reserva. En este método interviene el cálculo de días a pagar entre dos fechas determinadas y el precio por asiento del espacio en cuestión.

Nuevamente, en el documento “Pruebas”, podremos encontrar el listado completo de test unitarios que se han desarrollado. Gracias al uso de nombres descriptivos en la definición de cada método podemos deducir su finalidad, los datos proporcionados y el resultado esperado.

5.3. Otras pruebas

Además de las pruebas ejecutadas con Postman, hemos hecho uso del programa DBeaver para evaluar las reglas de la base de datos, ejecutando consultas SQL directamente desde su editor. Este programa nos ha permitido, asimismo, verificar que la definición de las tablas era correcta o los datos insertados en la base de datos mediante su interfaz.

5.4. Evaluación

Tras la realización de estas pruebas y el uso intensivo de la aplicación completa podemos concluir que se satisfacen los requisitos propuestos en este trabajo y que se pueden llevar a cabo todos los casos de uso descritos.

Como dificultades encontradas, durante la etapa de implementación se ha tenido que desarrollar algún caso de uso nuevo para conseguir que el comportamiento fuera el adecuado. También se han puesto en evidencia la falta de definición de cierta lógica de negocio y algunas limitaciones propias de la parte *frontend*, que no se había diseñado tan exhaustivamente.

Por otro lado, se han tenido que realizar cambios en los permisos que habían sido establecidos para cada tipo de usuario. Por ejemplo, en la etapa de diseño se determinó que el administrador podría tener todos los permisos CRUD sobre entidades creadas por él, pero solo de lectura en las entidades dadas de alta por los usuarios, como los espacios y las reservas.

Este comportamiento no permitía al administrador eliminar algunos usuarios del sistema, ya que en las reglas de negocio se definió que un usuario solo podría ser borrado si no contaba con espacios registrados a su nombre. Algo similar ocurrió posteriormente cuando se otorgaron permisos de borrado de espacios al administrador, pero no los de eliminación de reservas: los espacios que tenían reservas activas no podían ser eliminados.

6. Conclusiones

La realización del trabajo Final de Grado me ha dado la posibilidad de desarrollar un proyecto de software desde el principio y de participar en todas las etapas de su ciclo de desarrollo.

Desde el comienzo, he tenido que idear una nueva aplicación que pudiera tener algún impacto en nuestro contexto actual, he planificado las etapas de trabajo y los requisitos en la fase de análisis. Se ha decidido y diseñado la arquitectura del sistema, sus entidades y la comunicación entre sus componentes. Por último, he puesto en marcha el desarrollo de esta idea, implementándola y evaluando los resultados.

Escogí el área de Java EE debido a mi interés en las tecnologías de desarrollo *backend* y en la programación orientada a objetos. Una vez estuvieron cubiertos los objetivos establecidos en la planificación decidí dedicar tiempo adicional a la documentación y desarrollo de la interfaz de usuario en React, con el objetivo de aprovechar este trabajo para la experimentación con tecnologías que desconocía y de conseguir un producto más completo.

Por otro lado, implementar la aplicación utilizando una arquitectura limpia ha supuesto un reto añadido, que se aceptó con la intención de construir un software tolerante a cambios y con bajo acoplamiento. Aunque, en un principio, ha sido costosa de implementar, ha facilitado las modificaciones posteriores.

Con respecto a la metodología seguida, me ha parecido la adecuada para las características del proyecto, ya que ha permitido la realización secuencial de unas tareas que estaban bien definidas. Sin embargo, no la he podido cumplir de forma estricta, ya que no ha sido posible prever algunas necesidades que se han presentado ya en la etapa de implementación del código.

En conclusión, se han logrado cumplir los objetivos propuestos a pesar de las dificultades encontradas. Esto ha sido posible, gracias a la aplicación de los conocimientos académicos que se han adquirido en las distintas áreas de la Ingeniería informática a lo largo de estos años.

6.1. Trabajo futuro

Uno de los mayores retos abordados en la realización de este proyecto ha sido la planificación del trabajo para adaptarnos a las fechas de entrega. El tiempo disponible para cada etapa ha sido limitado, por lo que se han tenido que limitar el alcance y los objetivos.

En este apartado vamos a repasar algunas de las funcionalidades que nos gustaría haber podido implementar o que supondrían una mejora en futuras versiones:

- Interfaz de usuario mejorada: queda pendiente el refinamiento de estilos en las diferentes pantallas, la sustitución de las tablas por elementos más estéticos y la aplicación de un comportamiento responsive para añadir compatibilidad a un mayor número de dispositivos.
- Subida de imágenes: aunque no era objetivo de esta entrega gestionar la subida de imágenes a una plataforma externa, sería un requisito fundamental si la aplicación quisiera lanzarse en entornos de producción.
- Plataforma de pagos: actualmente existe una interfaz en la aplicación que devuelve un estado del pago fijo y un identificador aleatorio. Quedaría pendiente configurar una plataforma de pagos real y conectarla al sistema.
- Paginación: aunque no ha resultado necesaria para el prototipo de aplicación que se ha realizado, sería conveniente contar con la paginación de resultados cuando el sistema contara con una mayor cantidad de datos.
- Evaluación de espacios: con esta nueva funcionalidad los usuarios podrían evaluar un espacio que hayan reservado, incluyendo puntuación y un mensaje para compartir su experiencia con otros usuarios.
- Publicación de eventos: esta aplicación ha sido ideada para ofrecer un espacio y unos servicios adecuados para el trabajo, pero también para poner en contacto a personas que trabajan en remoto y que deseen compartir su jornada con otras personas o participar en eventos colectivos.

7. Glosario

- API REST: Interfaz de Programación de Aplicaciones web que utiliza HTTP para realizar operaciones sobre datos.
- Backend: Parte del desarrollo web del lado del servidor que se encarga de la lógica para que esta funcione.
- Base de datos relacional: Conjunto de datos que se almacenan relacionados entre sí mediante la representación por tablas.
- CRUD: Acrónimo de “Create, Read, Update y Delete” que utilizamos para referirnos a las operaciones básicas que se pueden realizar sobre los datos.
- Endpoint: Punto de intercambio de datos que posee una API para comunicarse con el exterior.
- Framework: Conjunto de librerías y herramientas de un lenguaje concreto y que sirven de base para el desarrollo de software.
- Frontend: Parte del desarrollo web del lado del cliente que se encarga de los elementos con los que interactuará el usuario.
- IDE: Acrónimo de “Entorno de Desarrollo Integrado”. Programa empleado para desarrollar software.
- JSON: Formato de texto sencillo para el intercambio de datos utilizado comúnmente en los servicios REST.
- Mapper: En nuestra aplicación nos referimos a un servicio que recibe un tipo de objeto y lo transforma en otro con el objetivo de permitir operar con objetos entre distintas capas.
- Open source: Referencia al software de código abierto, es decir, cuyo código es accesible al público.

8. Bibliografía

1. Observatorio Nacional de Tecnología y Sociedad. (septiembre de 2021). *Flash datos de teletrabajo en el segundo en el segundo trimestre de 2021*.
2. González González, F., & Calero Castañeda, S. L. (2019*). Comparación de las metodologías cascada y ágil para el aumento de la productividad en el desarrollo de software* (Doctoral dissertation, Universidad Santiago de Cali).
3. Molina Hernández, Y., Granda Dihigo, A., & Velázquez Cintra, A. (2019). *Los requisitos no funcionales de software. Una estrategia para su desarrollo en el Centro de Informática Médica*. Revista Cubana de Ciencias Informáticas, 13(2), 77-90.
4. Hombergs, Tom. (2019*). Get Your Hands Dirty on Clean Architecture: A hands-on guide to creating clean web applications with code examples in Java*. Packt Publishing Ltd.
5. MySQL Documentation Team. *MySql 8.0 Reference Manual: What is MySQL?.* Consultado desde https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html
6. Jin, B., Sahni, S., & Shevat, A. (2018). *Designing Web APIs: Building APIs That Developers Love*. O'Reilly Media, Inc.
7. Garrido Tejero, A. (2016). *Implementación del patrón DAO (Data Access Objecto).* Universidad Politécnica de Valencia. Consultado desde https://polimedia.upv.es/visor/?id=d55b1e80-c327-f14f-93f6-e55ee68525fa
8. Pech-May, F., Gomez-Rodriguez, M. A., Luis, A., & Lara-Jeronimo, S. U. (2012*). Desarrollo de Aplicaciones web con JPA, EJB, JSF y PrimeFaces*. Instituto Tecnológico Superior de los Ríos, Tabasco, México.
9. MSDN (2010). *Data Transfer Object. Microsoft MSDN Library*. Consultado desde http://msdn.microsoft.com/en-us/library/ms978717.aspx
10. *Spring Data JPA*. Spring. Consultado desde https://spring.io/projects/spring-data-jpa
11. *Spring Security*. Spring. Consultado desde https://spring.io/projects/spring-security