UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño y desarrollo de una aplicación web para unas oficinas de coworking, buscando agilizar y simplificar el sistema de reservas.

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Ruiz de la Torre Bertolín, Jorge

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Jorge Ruiz de la Torre Bertolín

Tutor: Pedro José Valderas Aranda

Curso Académico 2021-2022

Agradecimientos

En primer lugar, y lo más importante, agradecer a todas las personas de mi entorno, tanto con las que comparto sangre, como las que no, pero que poco a poco han ido formando parte de lo que considero mi familia. A mis abuelas, a mis padres y sus respectivas parejas, a mis tíos, a mis primos, a mis hermanos y a mis amigos más cercanos, tanto los que me han acompañado desde pequeño como los que he ido conociendo durante el camino.

A todos mis profesores por su paciencia y dedicación, desde maternidad hasta la universidad. Recalcando a mis profesores de matemáticas del instituto, quienes me motivaron a seguir con los estudios.

Gracias a todos vosotros.

Resumen

El objetivo del TFG consiste en desarrollar una aplicación web para unas oficinas de *coworking* que permita reservar espacios de *coworking* a través de internet, sin necesidad de ponerse en contacto con nadie.

La aplicación web se centrará en ofrecer servicios de reserva *online*, con pagos seguros, como también un sistema de suscripciones que nos permitirá acceder a mejores descuentos y a distintos eventos organizados por la empresa. Se hará una distinción en los tipos de reserva y en las salas, variando así el precio y la ubicación. El usuario podrá reservar en los horarios que más le convenga siempre y cuando no estén ocupados. Las reservas podrán realizarse por horas, y decidiendo si la reserva será para un solo día, durante varios días consecutivos o será una reserva recurrente semanalmente. Una vez creada la reserva, se podrá cancelar, una por una, reembolsando el 80% del montante de la reserva.

Asimismo, también enfocaremos nuestra página web en crear un vínculo social del usuario con la aplicación, teniendo así una cuenta, una cuenta de empresa y la información de éstas que nos quieran proporcionar.

Tanto la reserva como los pagos de ésta se podrán realizar de forma segura tanto *online* como presencial.

El administrador de la web tendrá un apartado de administrador, en el que podrá modificar a su gusto las salas, los usuarios y las reservas.

Palabras clave: Coworking, Desarrollo web, Aplicación web, Java Spring Framework, Angular, Front-end, Back-end, Software, Ingeniería del Software.

Abstract

The objective of our final degree project is to develop a web application for coworking offices. This application allows you to make reservations of our spaces through the internet, without the need to contact anyone.

This project will focus on offering online reservation services, with safe payments, as well as a subscription system that will give us some advantages, like access to better discounts and different events organized by the company. A distinction will be made in the types of reservation and in the rooms, thus varying the price and location. The user will be able to book at the times that suit him best if the room by that time is not occupied. Reservation can be made by hours, and the user will be able to decide if the reservation will be for one day, a recursive reservation or a day-by-day reservation. Once the reservation is created, it can be cancelled one by one, refunding 80% of the amount of that reservation.

We will also focus our website on creating a social link between the user and the application, thus having an account, a company account, and the information they want to provide to the web.

Both the reservation and the payments for it can be made safely online or in person. Otherwise, the subscription system will only be able online.

The web administrator will have an administrator section, in which he will have the power to modify, delete or create the rooms, users and reservations.

Keywords : Coworking, Web development, Web application, Java Spring Framework, Angular, Front end, Back end, Software, Software engineering.

Resum

L'objectiu del TFG és desenvolupar una aplicació web per a unes oficines de *coworking* que permeta reservar espais de *coworking* a través d'internet, sense necessitat de posar-se en contacte amb ningú.

L'aplicació web se centrarà en oferir serveis de reserva *online*, amb pagaments segurs, com també un sistema de subscripcions que ens permetrà accedir a millors descomptes i diferents esdeveniments organitzats per l'empresa. Es farà una distinció en els tipus de reserva i en les sales, variant així el preu i la ubicació. L'usuari podrà reservar en els horaris que més li convinguen sempre i quan no estiguen ocupats. Les reserves es podran fer per hores, i decidint si la reserva serà per a un sol dia, durant diversos dies consecutius o serà una reserva setmanal recurrent. Una vegada creada la reserva, es podrà cancel·lar, una per una, reembossant el 80% de l'import de la reserva.

Així mateix, també enfocarem la nostra pàgina web a crear un vincle social de l'usuari amb l'aplicació, tenint així un compte, un compte d'empresa i la informació que ens vulguen proporcionar.

Tant la reserva com els pagaments d'aquesta es podran realitzar de forma segura, tant *online* com de manera presencial.

L'administrador de la web tindrà un apartat d'administrador, on podrà modificar al seu gust les sales, els usuaris i les reserves.

Paraules clau: Coworking, Desenvolupament web, Aplicació web, Java Spring Framework, Angular, Front-end, Back-end, Software, Enginyeria del Software.

Tabla de contenidos

1. Introducción

En esta sección, vamos a introducir el proyecto, indicando la motivación que nos ha llevado a realizar el proyecto, los objetivos que queremos conseguir, el impacto esperado y vamos a presentar la estructura de la memoria.

1.1 Motivación

Tras una oportunidad de negocio que se le ofreció a un amigo, tras darle varias vueltas a cómo aprovechar un espacio, tuvo la idea de llevar a cabo unas oficinas de *coworking* en Valencia, ya que él había estado utilizándolas durante una etapa en Estados Unidos, y cree firmemente que, a raíz de la pandemia, las empresas van a dejar de tener sus propias oficinas y van a empezar a establecerse en espacios del estilo *coworking*, subcontratados para los trabajadores que necesiten un espacio para trabajar y no lo dispongan en sus viviendas.

Personalmente, viví una situación en la que me habría gustado tener la oportunidad de reservar un espacio acondicionado para el trabajo, ya que, con la implementación masiva del teletrabajo, durante un viaje que hice, debía realizar las horas estipuladas para las prácticas de empresa desde el ordenador portátil que me habían facilitado, solo que me encontraba en otro país y, por unas desafortunadas circunstancias que se dieron el día anterior y en un horario no laborable, no iba a disponer de los recursos ni el espacio para ponerme con ello. Después de varias búsquedas, conseguí un espacio a través de un contacto, pero pensándolo fríamente, creo que, si hubiese tecleado en Google algo relacionado con espacios de trabajo, me habría gustado poder reservar fácilmente uno de estos, y habría tenido en mente algunas oficinas de *coworking*.

Tras un estudio, nos dimos cuenta de que actualmente, las páginas web de *coworking* en Valencia no proporcionan un sistema automatizado de reservas y éstas se realizan a través del email o por teléfono.

Viendo que las oficinas de *coworking* es un mercado reciente y en auge, y está bastante relacionado con la tecnología, nos pareció absurdo que este campo no esté tan agilizado, teniendo que ponerte en contacto con alguna persona física y depender de la disponibilidad de esta. Pensamos que no se está adaptado al contexto tecnológico en el que vivimos y en el que se han creado.

1.2 Objetivos

El objetivo del TFG consiste en desarrollar una aplicación web para unas oficinas de *coworking* que permita reservar espacios a través de internet, sin necesidad de ponerse en contacto con nadie.

Nos gustaría poder cubrir la necesidad de las personas que, afectadas positiva o negativamente por esta nueva ola del teletrabajo, se vean obligadas a recurrir a un espacio acondicionado para el trabajo. Queremos ampliar la visión que se tiene hoy en día de un *coworking*, e ir un paso más allá. Crear un espacio social, en el que las personas, aunque trabajen en campos distintos, tengan la oportunidad de asistir presencialmente a unas oficinas y gozar de la vida social que se nos ofrecía antes de la pandemia.

También queremos abarcar el campo de afectados que se vean en la necesidad de encontrar un espacio de trabajo de manera urgente, en el que no necesiten ponerse en contacto con nadie ni depender de una persona, es decir, que de manera *online* puedan tener acceso a estos espacios. Así, abarcar tanto un mercado de gente más afianzada a este tipo de espacios de trabajo, como también a un público más casual, que soliciten estos servicios de manera puntual.

1.3 Impacto Esperado

Creemos firmemente que tarde o temprano, todas las webs de *coworking* van a acabar implementando estas funcionalidades, ya que no hemos identificado el motivo por el cual las empresas de *coworking* requieren de un contacto personal previo.

Esperamos ser los pioneros, y que a raíz de la exclusividad que ofrecemos con las reservas *online*, hacernos un hueco en un mercado creciente.

1.4 Estructura

Para llevar a cabo el trabajo, hemos realizado una memoria que consta de 9 puntos, con la intención de estructurar la explicación del proyecto.

En cada uno de estos puntos, vamos a estar haciendo un repaso de ciertos elementos que han ido apareciendo y que nos han sido de ayuda para el desarrollo.

El primer punto es la introducción a la memoria, en el cual nos situamos actualmente.

En el segundo punto, vamos a estar detallando el contexto tecnológico en el que nos encontramos actualmente, buscando dar una explicación de por qué hemos decidido realizar el proyecto con estas tecnologías.

Más adelante, vamos a estar concretando todos los preparativos para comenzar con los desarrollos, presentando así la metodología con la que hemos organizado la carga de trabajo, los requisitos acordados con el cliente, las tecnologías a emplear y todas las estructuras de datos que vamos a presentar. Esto corresponde a los puntos 3, 4, 5 y 6.

Para concluir, vamos a estar presentando los desarrollos. Primero, haremos un repaso a las tecnologías que hemos utilizado, explicando también su estructura, y algunos de los puntos clave que nos ayudarán a entender el producto desarrollado. Haremos una presentación de este, mostrando así las interfaces y cómo estas responden a los requisitos definidos. Esto corresponde a los puntos 7 y 8.

Finalmente, haremos una conclusión del proyecto con los resultados obtenidos, y un pequeño análisis de las mejoras que podrían llevarse a cabo. Añadiremos un apartado con todas las referencias y bibliografía que hemos utilizado durante el proyecto.

1.5 Colaboraciones

El trabajo de desarrollo ha sido realizado de forma completamente individual por mi parte.

El equipo del proyecto está compuesto por Alberto Monforte Gómez y Jorge Ruiz de la Torre Bertolín, siendo este segundo el alumno que presenta el TFG.

Las funciones del colaborador Alberto son las administrativas. Al ser el cliente del producto, sus labores consistirán principalmente en organizar y aconsejar cómo enseñar los datos por pantalla, además de decidir las tablas de precios para las reservas, las suscripciones, y definir el alcance del proyecto. Se le presentaron desde un comienzo los diagramas de clases y casos de uso, y estuvo involucrado en los desarrollos de forma externa aconsejando sobre cómo presentar los datos y organizar las tareas.

2. Contexto

En este apartado, vamos a hacer un estudio del escenario tecnológico actual en el que nos encontramos, y la repercusión que tendrá nuestro proyecto en este.

2.1 Internet y las páginas web

En este apartado hablaremos del contexto tecnológico.

Internet es una red de comunicación interconectada que se utiliza a nivel mundial, y, para gran parte de la población, su uso es cotidiano. Hoy en día, podemos acceder a esta red a través de ordenadores, dispositivos móviles, televisores, etc…

'Internet se ha convertido en una parte tan fundamental, tan intrínseca, tan arraigada a nuestras vidas que lo damos por sentado. Su presencia se ha hecho invisible porque está plenamente integrada en el engranaje del sistema, en su funcionamiento y en nuestras rutinas.' - Según [20]

Debido a la pandemia de CoVid-19 vivida mundialmente en el año 2020, su uso se ha visto más impulsado todavía. Los comercios que no se han adaptado a este cambio, están en una enorme desventaja frente a los que sí lo han hecho. La visibilidad es un pilar fundamental para estos, es por ello por lo que promocionarse en esta red y situarse en la cabeza de las búsquedas resulta cada vez más importante. Hay negocios que se dedican expresamente al posicionamiento en los motores de búsqueda que se utilizan en internet, y es por eso por lo que, desde un punto de vista empresarial, es fundamental la promoción que se pueda realizar de nuestro negocio en la plataforma de internet, apareciendo así en motores de búsqueda como en anuncios publicitarios.

Desde las bibliotecas hasta los restaurantes, se ha facilitado la programación de tu próximo movimiento a través de tu dispositivo móvil. Por ello, consideramos que dar con un sistema de reservas, en los que, en muchos casos, es necesario el pago *online*, sería fundamental para un proyecto de nuestras características. No hay que olvidar tampoco los métodos de pago tradicionales, dejar en el usuario la decisión de pagar de manera *online* o efectuar el pago efectivo en presencial. Independientemente de la decisión, nuestro objetivo es agilizar todos los procesos de pagos en la red.

2.2 Nacimiento y adaptación de las oficinas de coworking

"El concepto de coworking hace referencia a un espacio de trabajo compartido, donde varias empresas y profesionales llevan a cabo su actividad." – Según [1]

Fue en 1995 que se inauguró el C base en Berlín. A este se le conoce como uno de los primeros espacios de trabajo compartidos. Consiste en una asociación de personas con una mentalidad parecida y un interés común, que, en este caso, era un interés tecnológico, dónde se juntaban a compartir sus conocimientos bajo un mismo techo.

Aunque no fue hasta 1999 en Nueva York dónde Bernie DeKoven, un diseñador de videojuegos, acuñó el término de *coworking* (según [2]). Es una palabra de etimología inglesa, dónde el prefijo *"co"* hace referencia a la cooperación, sumado a la palabra *"working",* que se refiere a trabajo. Se podría definir como un lugar donde se va a trabajar juntos.

Los *coworkings* de hoy en día son un espacio enfocado principalmente hacia *freelancers* (personas que trabajan como autónomos) o *startups* (pequeños grupos de trabajo que buscan innovar y desarrollar una idea) que están empezando y se están dando a conocer. Rápidamente, se fue expandiendo por todo el mundo, principalmente en Europa y Estados Unidos, donde, a día de hoy, según [2] existen 5 800 espacios compartidos, de los cuales 2 500 están en Europa.

3. Metodología

Para la realización de un proyecto de *software*, se requiere una organización a medio - largo plazo. En este apartado, vamos a estar hablando de las metodologías que hemos aplicado durante el proyecto y de la planificación temporal que hemos planteado desde un comienzo, con todas las variaciones que esta conlleva.

3.1 Metodología AGILE

Para realizar una planificación temporal, al tener disponibilidad con nuestro cliente, hemos decidido aplicar una metodología AGILE Kanban.

Las metodologías AGILE están a la orden del día. Es muy difícil, en un mundo tan interconectado como el que vivimos, y más aún después de la pandemia, llevar a cabo un proyecto de *software* sin mantener ningún contacto con el exterior.

Esta metodología tiene un enfoque estructurado e interativo para los clientes. El cliente va a estar en contacto con el equipo de desarrollo, adaptando así las tareas a realizar a las necesidades del proyecto.

La metodología Kanban nos permite que el cliente participe en los desarrollos, estando presente en la definición de las tareas mediante un tablero, con breves descripciones de las tareas e indicando en el estado en el que se encuentran estas tareas. Consideramos que esta era la mejor alternativa para comenzar con el proyecto desde cero, ya que, después de asegurar las tareas iniciales, en cooperación con el cliente, definimos los límites y el alcance del proyecto. El desarrollador puede trabajar con cierta independencia, mientras mantiene contacto con el cliente. También nos permite limitar la carga de trabajo y concretar ciertos plazos temporales para las tareas.

Si está bien definido, en el momento en el que tengamos una primera versión y con cambios más visuales, podríamos realizar una transición a una metodología AGILE Scrum, dónde definiremos unos ‘*sprints’* que se traducirían en lapsos de tiempo de una duración fija, en el que vamos a realizar los desarrollos que nos presentaría nuestro cliente, y, al final de estos, le mostraríamos los progresos.

Vamos a hacer un repaso de las ventajas y desventajas que nos ofrecen las metodologías AGILE Kanban:

Ventajas:

- Los desarrollos se amoldan a las necesidades del cliente.
- Al haber tanto contacto entre cliente y desarrollador, no se suelen requerir grandes cambios ni grandes refactorizaciones del código, que podrían llegar a ser costosas.
- El desarrollador se involucra más en el proyecto y puede aportar ideas acordes con el desarrollo.
- El desarrollador está presente a la hora de definir el trabajo a realizar, por lo que va a estar al tanto de las tareas y el coste que estas van a llevar.

Desventajas:

- Al haber tanto contacto con el cliente, se podría perder tiempo de desarrollo.
- Es difícil gestionar grandes aumentos de la demanda.

3.2 Planificación tablero Kanban

Para el tablero Kanban con las tareas a realizar, hemos utilizado la nueva herramienta de tablero que ha implementado Github en una de sus últimas actualizaciones.

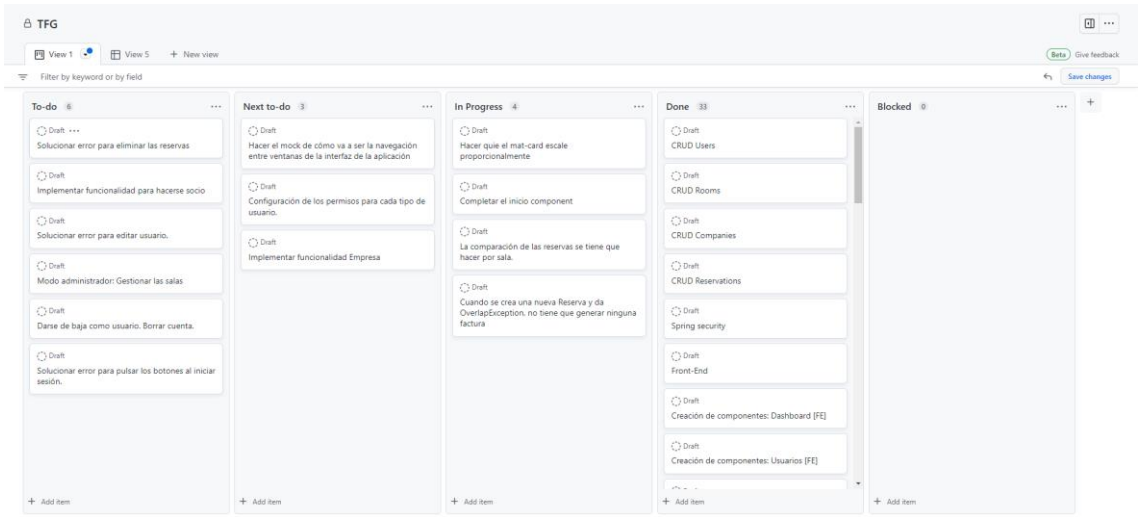


Imagen 1: Tablero Kanban con las tareas a realizar.

En esta imagen podemos observar el tablero Kanban que hemos utilizado. Como podemos ver, este tablero se divide en 5 columnas, y dentro de cada columna podemos ver las tareas descritas.

Vamos a explicar cada una de las columnas que se encuentran en el tablero:

To-Do

En esta columna, se almacenarán todas las tareas a realizar. La tarea más prioritaria se encontrará arriba, mientras, según bajemos, encontraremos el resto de tareas.

Next To-Do

Es el paso anterior para comenzar con los desarrollos. Dentro de todas las tareas futuras a realizar, estas son las más prioritarias.

In Progress

En esta columna irán las tareas que se estén desarrollando actualmente y que se encuentren en progreso.

Done

Las tareas que se encuentren en esta columna serán aquellas que hayan sido realizadas satisfactoriamente.

Blocked

La tarea está bloqueada. Esto puede darse bien por la necesidad de finalizar otra tarea antes de continuar con esta, o bien por falta de conocimientos del desarrollador, el cual tendrá que emplear tiempo en investigar la tarea.

4. Requisitos

Vamos a hacer un estudio de los requisitos funcionales y no funcionales del proyecto. Nos vamos a basar en las necesidades que ha planteado inicialmente el cliente y vamos a ir adaptándolas a nuestro proyecto.

4.1 Requisitos funcionales

Los requisitos funcionales se refieren a detalles técnicos y características que va a tener nuestro proyecto de *software*. En este caso, serían las necesidades de *software* que tendría el responsable del proyecto, junto con las necesidades de los usuarios al interactuar con la web. Estos requisitos describen cualquier actividad que el proyecto deba realizar.

Para poner en contexto la envergadura del proyecto, se presentan los requisitos iniciales y la problemática planteada por el cliente para el desarrollo de la página web:

1) Usabilidad de la página web:
   - El sistema debe presentar una web funcional con información sobre nuestras instalaciones y nuestros precios, en el que cualquiera, independientemente de si están registrados, tuviera acceso para consultarlo.
   - El sistema, más allá de ofrecer reservas para un espacio de trabajo, se debe presentar como un club social, en el que la gente se compromete y se siente identificado.
   - El usuario requerirá de una cuenta y estar registrado para realizar cualquier tipo de reserva sobre la página web.
   - El perfil de los usuarios constará de información básica, a rellenar por ellos, de cada uno. Además, tendremos información de si el usuario está abierto a escuchar ofertas por parte de otras empresas.
   - El usuario también indicará si está interesado en ser publicado en la página web, teniendo así un dato como publicable.
   - Siguiendo la premisa anterior, habrá un apartado donde cualquiera podrá consultar una lista de nuestros usuarios, con algo de información y facilitando su contacto, siempre y cuando estos hayan decidido ser publicables.
   - Lo que diferenciará nuestro sistema sobre nuestros competidores es la presencia en redes y las facilidades para reservar desde esta de forma inmediata.

2) Sistema de pagos:
   - Para las reservas, podremos elegir el método de pago, si pagar presencial u *online*. En el caso de las suscripciones, el pago sólo estará disponible *online*.
   - Las reservas podrán ser canceladas, reembolsando así el 80% del montante de la reserva. Haremos que se reembolsen una por una.

3) Sistema de asociados:
   - Existirá una función para hacerse socio, con una suscripción mensual establecida en 4,99€, donde los socios encontrarán mejores ofertas a la hora de realizar las reservas.
   - Los usuarios que sean socios tendrán la posibilidad de asociarse a una empresa que ya esté creada, o de crear una empresa. El creador de la empresa se establecerá como el administrador de esta.
   - Los usuarios que sean socios podrán tener un calendario personal donde pudiesen compartir sus disponibilidades a través de un enlace.
   - Los usuarios que estén dados de alta, también tendrán un chat en el que se podrán poner en contacto con el resto de los usuarios. Este chat será interno de la página web.
   - Por último, los usuarios que se hayan registrado como socios tendrán acceso a eventos que organicemos desde nuestro *coworking*.

4) Modo administrador:
   - 'El sistema tendrá un modo administrador, para facilitar a los secretarios presenciales y a los administradores la administración de este.'
   - 'El modo administrador podrá gestionar tanto las salas, como los usuarios, como las empresas, y como las propias reservas.'

4.2 Requisitos no funcionales

Los requisitos no funcionales se refieren a todos aquellos requisitos que describen las características de su funcionamiento.

Con lo cual, para describir los requisitos de calidad, vamos a ir recorriendo algunos atributos de calidad y las medidas que hemos tomado para asegurar que estos se cumplan.

El primer atributo de calidad que vamos a exponer es el de **disponibilidad**. Algo con lo que estamos comprometidos y por lo que queremos que se nos distinga es por la disponibilidad del proyecto. A diferencia de nuestros competidores, donde es necesario ponerte en contacto con alguna persona física, en nuestro caso habrá una disponibilidad 24/7 para realizar reservas y optar a ciertas ventajas (programa de socio). Para asegurar

esta disponibilidad, necesitaremos un servidor abierto. Esto lo lograremos arrancando un servidor web y que nuestro proyecto funcione sobre este.

El segundo atributo de calidad que vamos a exponer es el de **seguridad**. La seguridad informática es fundamental para el funcionamiento de cualquier proyecto web. Es por eso, que, para garantizar la seguridad de la información, hemos empleado las bibliotecas de *Spring Security*.

Lo primero que hacemos es encriptar las contraseñas, utilizando una librería que nos ofrece *Spring security*.

A la hora de crear un nuevo usuario, podemos ver cómo la contraseña introducida por el usuario se almacena de forma encriptada. Esto hará que las contraseñas, sin importar su valor, tengan un formato en la base de datos similar al de la imagen:

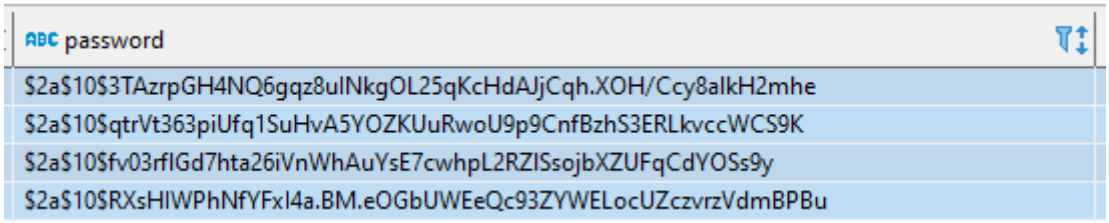


Imagen 2: Contraseñas que aparecen en la base de datos.

Como vemos en la imagen, las contraseñas van a estar encriptadas, y por mucho que intentemos iniciar sesión con este valor, el sistema no nos dejará.

Al tener la opción de pagos *online*, necesitamos fortalecer todavía más la seguridad de esta información. Stripe nos ofrece unas bibliotecas para nuestro código que van a ser las encargadas de efectuar correctamente los pagos. Gracias a estas, todos los intercambios de datos que realicemos relacionados con los pagos serán seguros. Los valores que pudiesen verse más afectados serán encriptados, como por ejemplo, la información de nuestra tarjeta de crédito. Lo que hará en su lugar es crear un identificador de esta tarjeta, y sólo podrá mostrarnos los últimos 4 dígitos de la tarjeta, el resto de información es completamente confidencial.

El último atributo de calidad que nos gustaría presentar es el de **privacidad**. Y es que, como hemos mencionado anteriormente, vamos a dar la opción al usuario de decidir si quiere que su cuenta aparezca pública en la web. Si el usuario decide que no la quiere pública, sólo los administradores del sistema podrán realizar llamadas para obtener los resultados de los usuarios. De cualquier otra manera, la llamada siempre filtrará sólo a los usuarios que hayan decidido que su cuenta sea pública.

5. Análisis

Para la realización de un proyecto de *software* en condiciones, es necesario realizar un análisis exhaustivo de este antes de comenzar con los desarrollos, puesto que un buen análisis nos permitirá saber cuáles van a ser los caminos que tomar y cuáles van a ser las tecnologías más eficientes que nos permitirán realizar el proyecto.

5.1 Casos de uso

Imagen 3: Diagrama de casos de uso de nuestro proyecto.

En este diagrama de casos de uso, podemos destacar las distintas opciones que van a tener tanto el cliente como el administrador.

Cliente:

C.U. 1. Consultar disponibilidad de las salas

Descripción	Desde la página web, el usuario tendrá la opción de consultar la disponibilidad de nuestras salas.
Actor	Usuario.
Precondición	Estar situado en la página que corresponde a la disponibilidad de las salas.
Postcondición	Se visualizan los horarios de disponibilidad de la sala.

C.U. 2. Servicio de contacto

Descripción	Tendremos una pestaña que nos facilitará el contacto con los administradores.
Actor	Usuario.
Precondición	Estar situado en la pestaña de contacto.
Postcondición	Se visualiza toda la información relativa de los administradores.

C.U. 3. Consultar precios

Descripción	Habrá una tabla con información sobre los precios.
Actor	Usuario.
Precondición	Estar situado en la pestaña de inicio.
Postcondición	Se visualizará una tabla con toda la información relativa a los precios de las reservas.

C.U. 4. Ver novedades, noticias

Descripción	Se mostrarán noticias relativas a nuestro *coworking*.
Actor	Usuario.
Precondición	Estar situado en la pestaña de inicio.
Postcondición	Se visualizarán las noticias.

C.U. 5. Crear cuenta

Descripción	Se permitirá al usuario crear una cuenta
Actor	Usuario
Precondición	Pulsar sobre el botón correspondiente a crear cuenta.
Postcondición	Se creará una cuenta para el usuario.

C.U. 6. Darse de baja

Descripción	Se permitirá al usuario dar de baja su cuenta.
Actor	Usuario autenticado.
Precondición	Pulsar sobre el botón de darse de baja en la pestaña de su perfil.
Postcondición	Se eliminará la cuenta del usuario.

C.U. 7. Editar información personal

Descripción	Se permitirá al usuario editar su información personal
Actor	Usuario autenticado.
Precondición	Pulsar sobre el botón de editar perfil, situado en la pestaña de su perfil.
Postcondición	Se editará la información del usuario.

C.U. 8. Solicitar visita a instalaciones

Descripción	Se permitirá al usuario solicitar una visita a las instalaciones.
Actor	Usuario autenticado.
Precondición	Pulsar sobre el botón de solicitar una visita presencial y seleccionar la fecha en la que se realizará la visita.
Postcondición	Se creará una nueva reserva para la fecha acordada.

C.U. 9. Consultar disponibilidad de las salas

Descripción	Desde la página web, el usuario tendrá la opción de consultar la disponibilidad de nuestras salas.
Actor	Usuario autenticado.
Precondición	Estar situado en la página que corresponde a la disponibilidad de las salas.
Postcondición	Se visualizan los horarios de disponibilidad de la sala.

C.U. 10. Solicitar reserva

Descripción	El usuario podrá solicitar una reserva de una sala en el horario que escoja.
Actor	Usuario autenticado.
Precondición	Estar situado en la pestaña de la sala correspondiente y haber seleccionado los horarios.
Postcondición	Se creará una o varias reservas tal y como haya seleccionado el usuario.

C.U. 11. Pagar en puerta

Descripción	El usuario podrá decidir si pagar en la puerta de nuestras oficinas
Actor	Usuario autenticado.
Precondición	Haber realizado una reserva y haber rellenado los campos correspondientes.
Postcondición	Se almacenará una factura y se cobrará al usuario cuando acuda presencialmente.

C.U. 12. Pago con tarjeta

Descripción	El usuario podrá decidir si pagar con tarjeta
Actor	Usuario autenticado.
Precondición	Haber realizado una reserva y haber rellenado los campos correspondientes con el pago *online*.
Postcondición	Se almacenará una factura y se cobrará al usuario.

C.U. 13. Hacerse socio

Descripción	El usuario podrá darse de alta de socio.
Actor	Usuario autenticado.
Precondición	Estar situado en la pestaña de inicio, y pulsar sobre el botón de hacerse socio. Rellenar todos los campos correspondientes para realizar el pago de la suscripción.
Postcondición	El usuario autenticado pasará a ser un usuario socio, contando así con nuevas funcionalidades y mejores descuentos.

C.U. 14. Darse de baja de socio

Descripción	El usuario podrá darse de baja de socio.
Actor	Usuario asociado.
Precondición	Estar situado en la pestaña de mi perfil y presionar el botón de darse de baja de socio.
Postcondición	El usuario dejará de contar con las ventajas de socio y se le devolverá el dinero correspondiente y proporcional del tiempo que le quede de su última suscripción.

C.U. 15. Acceso a mejores descuentos

Descripción	El usuario tendrá acceso a mejores descuentos.
Actor	Usuario asociado.
Precondición	Haber realizado una reserva y haber rellenado los campos correspondientes.
Postcondición	El usuario gozará de mejores descuentos a la hora de realizar los pagos.

C.U. 16. Crear cuenta de empresa

Descripción	El usuario podrá crear una cuenta de empresa.
Actor	Usuario asociado.
Precondición	Estar situado en la pestaña de mi perfil y presionar el botón para crear la cuenta de empresa. Rellenar los campos correspondientes.
Postcondición	El usuario pasará a ser el administrador de la empresa, y aparecerá relacionado con esta.

C.U. 17. Promocionarse en la web

Descripción	La empresa del usuario aparecerá promocionada en la web.
Actor	Usuario asociado.
Precondición	Tener una empresa asociada.
Postcondición	La empresa aparecerá en la pestaña de empresas colaboradoras.

Administrador:

C.U. 1. Modo administrador

Descripción	El administrador contará con un menú de administrador.
Actor	Administrador.
Precondición	Estar situado sobre la pestaña de modo administrador.
Postcondición	El administrador verá la interfaz preparada para los administradores.

C.U. 2. CRUD de reservas

Descripción	El administrador podrá crear, actualizar, visualizar y eliminar todas las reservas a su gusto.
Actor	Administrador.
Precondición	Estar situado sobre la pestaña de modo administrador y seleccionar la pestaña de reservas.
Postcondición	El administrador gestionará estas reservas.

C.U. 3. CRUD de usuarios

Descripción	El administrador podrá crear, actualizar, visualizar y eliminar todos los usuarios a su gusto.
Actor	Administrador.
Precondición	Estar situado sobre la pestaña de modo administrador y seleccionar la pestaña de usuarios.
Postcondición	El administrador gestionará los usuarios.

C.U. 4. CRUD de salas

Descripción	El administrador podrá crear, actualizar, visualizar y eliminar todas las salas a su gusto.
Actor	Administrador.
Precondición	Estar situado sobre la pestaña de modo administrador y seleccionar la pestaña de salas.
Postcondición	El administrador gestionará las salas.

5.2 Diagrama de clases

Un punto importante a tener en cuenta para el desarrollo del proyecto sería definir con precisión las entidades presentes en nuestro proyecto, lo que nos ayudará a formalizar una base de datos para nuestro sistema. Hemos realizado un diagrama UML que nos va a permitir hacernos una idea de cómo se almacenarán los datos:

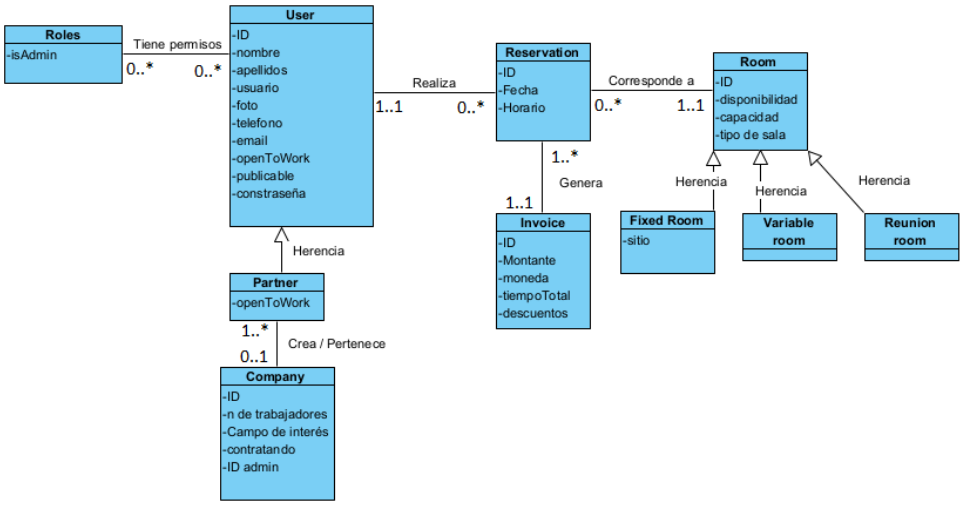


Imagen 4: Diagrama de clases UML de la aplicación.

Los números que hay al principio y al final de cada relación que hemos marcado en el diagrama representan la cardinalidad. Corresponden con la cantidad de objetos que este va a aceptar de la tabla con la que se relaciona.

Por ejemplo: Una factura va a ser la suma de una o más reservas, con lo cual, en la relación entre factura y reservas, la factura va a tener 1..*, que se traduce como una o más reservas, mientras que del lado de la reserva, cada una de estas va a corresponderse obligatoriamente con una sola factura, es decir, de este lado de la relación vamos a tener 1..1, que sería que obligatoriamente cada reserva tiene que estar correspondida con una factura.

Como podemos ver en el diagrama y basándonos en las necesidades del cliente, vamos a almacenar principalmente:

Usuarios (*User*)

Los usuarios estarán dotados de un ID, nombre y apellidos, un nombre de usuario y contraseña para la autentificación de este, y una foto para la cuenta. Un email y un teléfono de contacto, por si fuera necesario ponerse en contacto con dicho usuario, aunque estos últimos no son campos obligatorios. Finalmente, hemos añadido dos

variables: “openToWork”, para indicar que el usuario está buscando trabajo, y “publicable”, para indicar que el usuario no le importaría salir en el apartado donde aparecen nuestros *coworkers*.

De esta tabla de usuario, hereda la tabla de socio, que sería para indicar aquellos usuarios que se hayan asociado a nuestro sistema. Tendrán relaciones extra, como, por ejemplo: *Company* (Empresa), la cual dotaremos de un ID del administrador, además de registrar todos los usuarios que han decidido asociarse con dicha empresa.

Y finalmente, los socios podrán o bien crear o bien pertenecer a una empresa. Esta empresa tendrá, como cada una de las tablas, un ID, un número total de trabajadores, un campo de interés (ejemplo: Empresa de *software*, empresa farmacéutica…), el ID del administrador, y un indicador si la empresa está contratando gente, para que el resto de los usuarios puedan tenerlo en cuenta.

Reservas (*Reservation*)

Para las reservas, almacenaremos un ID, una fecha y un horario, no necesitaremos más. Ocasionalmente, podríamos indicar el sitio en el que nos sentaremos, pero esto estará presente en la sala.

Facturas (*Invoice*)

Estas facturas se crearán a partir de las reservas. Las facturas deberán contener entre una o más reservas. Las variables de la factura serán: un identificador ID, para reconocer la factura. Un montante total, para saber de cuanto es la factura, un descuento, para saber cuál es el descuento que se le aplica, una moneda, para saber la moneda en la que se va a facturar, y finalmente un tiempo total, para saber la cantidad de horas que se están facturando.

Salas (*Room*)

Las salas estarán dotadas de un identificador ID, una disponibilidad, para saber si la sala estará disponible, un horario en el que la sala estará operativa, y por último un aforo.

Hay unas relaciones de herencia en la sala, de las que distinguiremos entre los 3 tipos de sala principales: Espacio variable, dónde las plazas serán individuales y el sitio se asignará automáticamente en alguno que esté disponible. Espacio fijo, similar al espacio variable, solo que en este caso nosotros podremos elegir la plaza que queremos. Y por último las salas de reuniones, que serán salas privadas donde se podrán realizar reuniones.

Podremos, más adelante y según las necesidades de nuestro cliente, añadir distintas variables booleanas para saber si la sala tendrá internet, televisor, acondicionada para videollamadas, etc…

6. Diseño

Una vez realizados los análisis, vamos a comentar los diseños que hemos empleado para nuestro proyecto, tanto de la base de datos como de la interfaz de usuario que responde a los requisitos presentados inicialmente por el cliente.

6.1 Modelo de datos

En nuestro proyecto, la base de datos que utilizamos ha sido creada por *hibernate*.

Hibernate es una herramienta de *Java*. Su objetivo es facilitar el mapeo entre el modelo de objetos de nuestra aplicación y la base de datos relacional. En nuestro caso, lo que hicimos fue definir unas entidades (clases de *Java*) definiendo los valores que vamos a almacenar. *Hibernate* lo que hace es coger estas entidades y crear tablas similares al objeto.

Para nuestro proyecto, la base de datos va a tener una estructura similar a la de la imagen siguiente, correspondiendo cada una de las tablas con uno de nuestros objetos entidad:

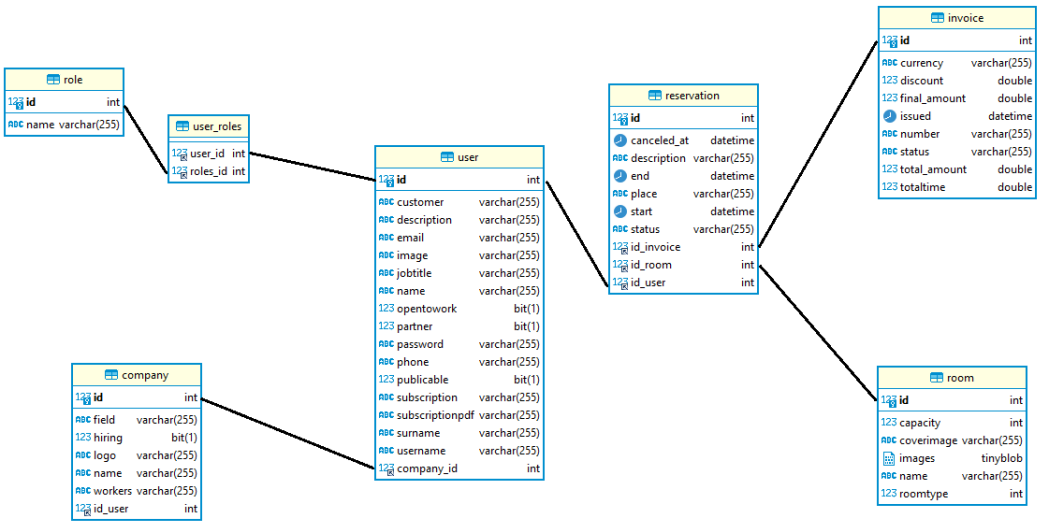


Imagen 5: Diagrama de la base de datos relacional del proyecto.

Como podemos ver, nuestra base de datos se compone de 7 tablas, y cada una de estas tablas contiene una serie de atributos. Por ejemplo, la tabla de Reservas (*reservation*), contiene 3 claves ajenas, es decir, va a contener 3 variables que van a depender del identificador de otra tabla. Y es que, para realizar una reserva y almacenarla, vamos a almacenar en esta tabla el id del usuario (tabla User) que realiza la reserva, el id de la

sala (tabla *Room*) y el id de la factura (tabla *Invoice*) a la que corresponde esta reserva. Estos casos los podemos seguir viendo a lo largo de todo el esquema.

6.2 Diseño de la interfaz

Para el diseño de la interfaz, vamos a mostrar algunas de las pantallas más representativas de nuestro proyecto y explicar por qué hemos decidido hacerlas así. Vamos a dividir nuestras interfaces en 3 partes, el inicio de sesión (*login*), el panel de control (*dashboard*) y el modo administrador.

Inicio de sesión

La interfaz de inicio de sesión, o *login*, es la pantalla que veremos cuando entremos en la página web de la aplicación.

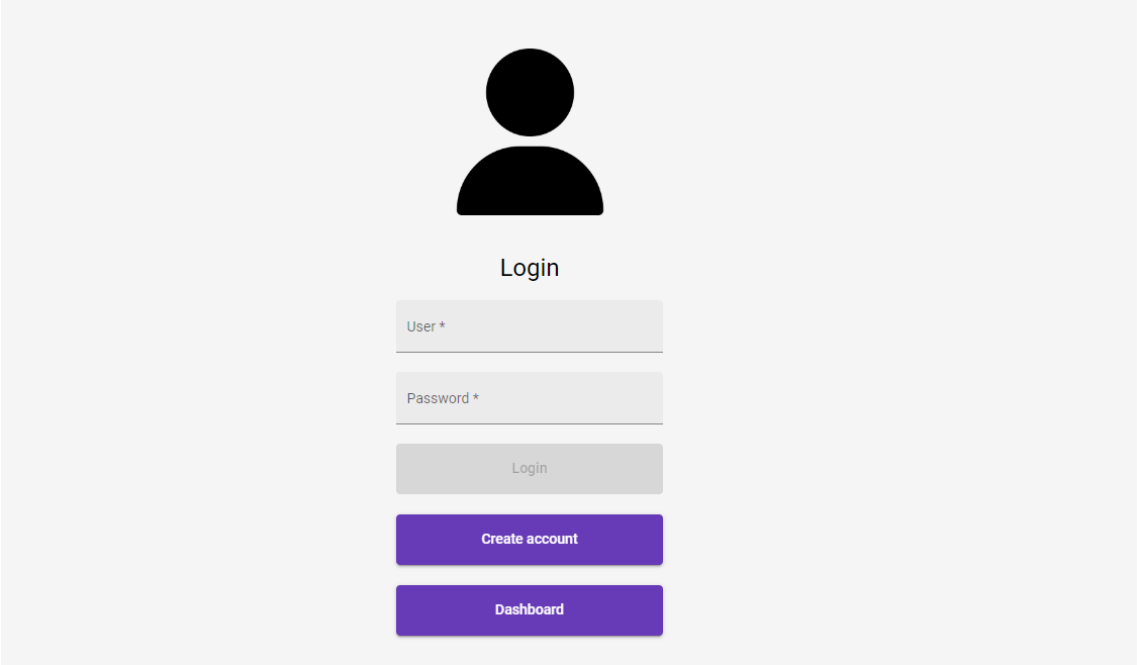


Imagen 6: Interfaz de usuario del login

Esta interfaz tiene un uso muy fácil e intuitivo. Hay un pequeño formulario con dos campos de texto, donde te pide el nombre de usuario y la contraseña. Una vez los rellenas, tendremos el botón de inicio de sesión habilitado. En caso de que las credenciales sean incorrectas, saltará un mensaje indicándonos el problema. En caso de que sean correctas, nos llevará al panel de control y en este podremos ver que estamos con la sesión iniciada.

Este formulario va acompañado de dos botones: Crear cuenta y acceder al panel de control (*dashboard*). Si decidimos crearnos una nueva cuenta, nos abrirá una ventana con un formulario, donde rellenaremos todos los campos necesarios para crear una

nueva cuenta. Si seleccionamos el panel de control, nos llevará directamente a este, sin haber iniciado sesión, siendo un usuario anónimo y teniendo funcionalidades limitadas.

Panel de control (*Dashboard*)

El panel de control va a ser donde vamos a encontrar toda la información de nuestro proyecto, y donde se va a acumular la mayoría de información. Vamos a mostrar la interfaz principal. Entraremos en detalle más tarde cuando analicemos todas las funcionalidades del sistema.

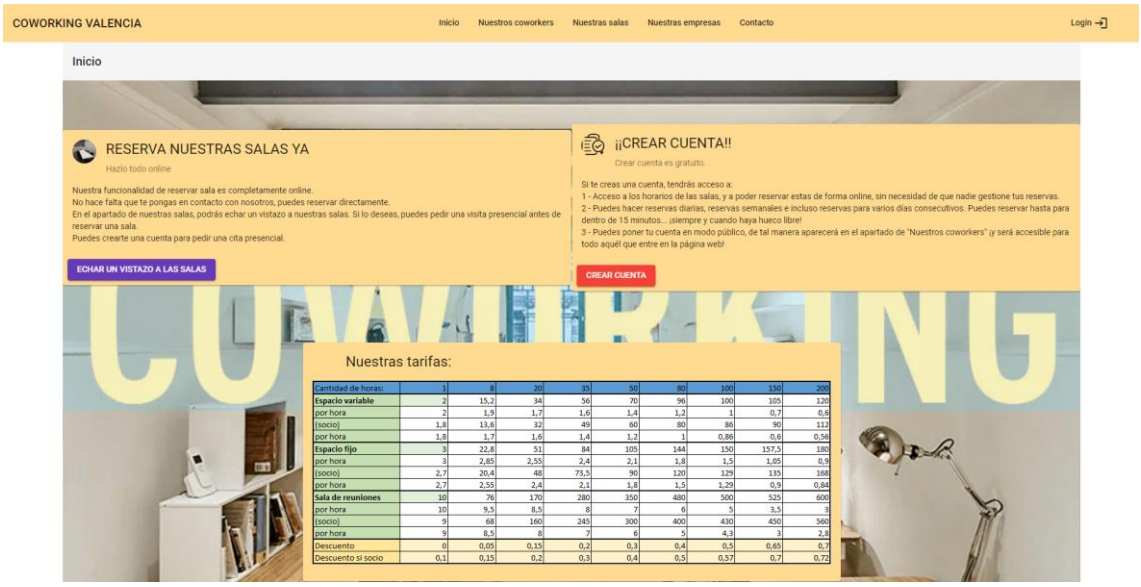


Imagen 7: Interfaz del panel de control.

En esta interfaz, podemos observar una barra horizontal en la parte superior donde podremos acceder a distintos menús, y en la parte superior derecha podremos acceder a la ventana de inicio de sesión o a la información relativa con nuestro perfil si tenemos la sesión iniciada.

Debajo de esta barra de navegación horizontal, vamos a encontrar un encabezado que nos indicará en qué pestaña estamos, y debajo, toda la información relativa con esta pestaña. Podremos navegar por los menús, y todos ellos tendrán la información organizada de la misma manera, acompañada del mismo fondo de pantalla. Es una interfaz intuitiva, donde seleccionamos qué tipo de información queremos ver en la parte superior y en el cuerpo de la web se nos mostrará dicha información.

Modo administrador

El modo administrador es una interfaz que sólo estará disponible para los usuarios que tengan permisos de administrador.

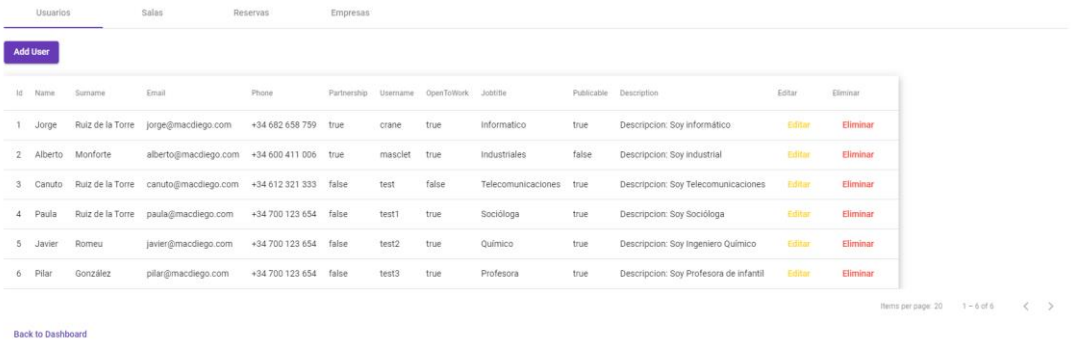


Imagen 8: Interfaz del modo administrador

Esta interfaz es más minimalista que las anteriores, aunque es más compleja de manejar. Esto se debe a que el usuario que controle esta pestaña ya tendrá conocimiento previo, con lo cual el aspecto visual no es determinante.

Se compone de una forma similar al panel de control: Una barra de navegación en la parte superior, y, dependiendo de la pestaña en la que nos encontremos, nos mostrará una tabla con toda la información que hay de nuestros usuarios, salas, reservas o empresas.

7. Desarrollo

Después de haber cerrado todos los análisis y diseños, podemos comenzar con los desarrollos. En este apartado, vamos a hacer una introducción al contexto tecnológico, presentando así todas las tecnologías que hemos utilizado en nuestro proyecto y, además, vamos a mostrar la estructura del proyecto y sus funcionalidades.

7.1 Contexto tecnológico

Vamos a comentar el contexto tecnológico y hacer una presentación de las tecnologías que vamos a utilizar.

7.1.1 Front end con Angular

El entorno de desarrollo (IDE) que hemos utilizado es el *Visual Studio Code*.

Introducción

Angular es un entorno de trabajo (*framework*) desarrollado por Google. Un *framework* es una plataforma que nos facilita mediante herramientas y bibliotecas la implementación de nuestro trabajo. Hablamos de bibliotecas informáticas, que se refiere a un conjunto de implementaciones ya desarrolladas y que nos ofrecen una interfaz definida para la funcionalidad que se invoca. En este caso, *Angular* nos ofrece una cantidad de opciones que nos va a facilitar organizar los datos y cuadrar así una interfaz de usuario intuitiva.

Para desarrollar en *Angular*, hemos utilizado 3 lenguajes distintos: *Typescript*, CSS y HTML.

TypeScript es un lenguaje de programación tipado basado en *Javascript*. Cuando se compila el código, se traduce automáticamente a *Javascript*.

HTML es un lenguaje de marcado, que se utiliza principalmente para páginas web.

CSS es un lenguaje de hojas de estilo, o lenguaje de diseño gráfico, se utiliza principalmente en conjunto con HTML. Es el encargado de organizar los datos en la interfaz, dando margen y formas a los elementos que este le proporciona.

Tanto *javascript*, como CSS como HTML son las tecnologías por excelencia para el desarrollo de las interfaces de usuario en las páginas web.

Historia

La fecha de lanzamiento de *Angular* fue el 15 de septiembre de 2015. Es un entorno de trabajo bastante reciente. Fue desarrollado y mantenido por Google.

Typescript se lanzó el 1 de octubre de 2012. Fue desarrollado y es mantenido por Microsoft.

CSS fue propuesto por primera vez el 10 de Octubre de 1994, y fue sólo dos años después cuando el consorcio internacional W3C lo recomendó. El consorcio W3C es el encargado de la estandarización, especificaciones y recomendaciones de la *World Wide Web*, que conforma una gran parte de lo que conocemos popularmente como internet.

Ante la necesidad de intercambiar y distribuir la información en la red, fue en 1993 cuando Tim Berners-Lee lanza HTML. Berners-Lee es a su vez el creador del consorcio W3C, con lo cual, HTML funciona como un estándar de datos.

Implementación en el proyecto

En el proyecto, hemos empleado *Angular* para el desarrollo de la parte de *front end*. Este se divide por componentes que explicaremos más adelante, pero cada uno de estos componentes contiene 3 ficheros relacionados entre sí: El fichero ".css", que se refiere al CSS, el ".html", que se refiere al HTML y el ".ts" que se refiere al TypeScript.

Para dar forma a nuestros objetos, hemos utilizado la biblioteca de *Angular Material*. Esta biblioteca, desarrollada por Google, nos ayuda con el diseño de la página, ya que nos proporciona los objetos con un diseño gráfico más atractivo.

7.1.2 Back end con Spring Framework

El entorno de desarrollo (IDE) que hemos utilizado es *IntellIJ IDEA*. Para la lectura de las bases de datos, hemos utilizado la herramienta *DBeaver*, que nos permite conectarnos a la base de datos SQL y ver los datos organizados en tablas.

Introducción

Spring framework es, como su nombre indica y al igual que *Angular*, un entorno de trabajo. Es un lenguaje de código abierto. El código abierto se refiere a todo programa cuyo código fuente es manipulable por el usuario para adaptarlo a sus necesidades. Lo que nos ofrece este entorno de trabajo es un soporte de infraestructura para *Java*.

Dentro de *Spring*, hay varias tecnologías que podemos utilizar, entre la cuales hemos empleado principalmente el *Spring Boot* y el *Spring Security*.

Historia

El lanzamiento inicial de *Spring* fue el 1 de octubre de 2002, pero no fue hasta junio de 2003 cuando se lanzó con la licencia de *Apache 2.0*, para, finalmente, sacar la versión 1.0 en marzo de 2004 que fue la que tuvo una gran repercusión.

Implementación en el proyecto

En el proyecto, hemos utilizado *Spring framework* con el lenguaje para el que está desarrollado, *Java*. El rol de *Spring* en nuestro proyecto va a ser el que gestione todos los eventos y la parte lógica del proyecto. En nuestro caso, el programa estará esperando peticiones que realizarán los usuarios (eventos) en el puerto 9090 dirigidas a los controladores, que serán los encargados en redirigirlas a la parte lógica del sistema hasta dar una respuesta, interactuando así con las bases de datos y con la interfaz para mostrar a los usuarios la información relacionada con sus peticiones.

7.1.3 Base de datos MySQL

Introducción

Una base de datos es un conjunto de datos estructurados que corresponden a un mismo contexto. Se utiliza para almacenar y proporcionar la información que requiera un servidor. - *Según [3]*

Existen varios tipos de bases de datos, que se diferencian por la manera en la que se comportan. En nuestro caso, vamos a utilizar el modelo de bases de datos relacional. Este modelo consiste en que los datos están almacenados en tablas, y estas tablas están relacionadas entre sí. Cada tabla contiene un registro identificador único, llamado ID o clave. Cada tabla corresponde con una entidad. Las columnas de estas tablas contienen los datos de dicha entidad, en el que cada registro contiene los datos correspondientes, distinguidos por el ID. El resto de las tablas se relacionan entre ellas mediante claves ajenas. Una clave ajena es un valor de un campo de la tabla que se corresponde con el id de otra tabla, y así es como se permite relacionarlas.

Historia

El término de “base de datos” fue mencionado por primera vez en 1963. No fue hasta la década de los 70 donde se definió la base de datos relacional. No fue hasta la década de los 90 dónde apareció el lenguaje SQL, que es un lenguaje programado para realizar consultas a la base de datos.

En España, las bases de datos se encuentran protegidas por la LOPD (Ley Orgánica de Protección de Datos de Carácter Personal).

Implementación en el proyecto

La implementación SQL en nuestro proyecto, es, mayoritariamente, nuestra base de datos. Para la creación y estructura de ésta, podemos revisar el punto **6.1 Modelo de datos**. La base de datos va a contener y a almacenar toda la información. Es donde van a persistir los datos de nuestro proyecto.

7.1.4 Stripe

Introducción

Stripe es una aplicación SaaS (*Software as a Service*) que nos ofrece un servicio de pagos *online*. Este programa se utiliza mediante llamadas API que van a crear nuestros pagos. Es ideal para las aplicaciones de venta online (*e-commerce*) y para aplicaciones móviles.

Implementación en el proyecto

En el proyecto, hemos utilizado Stripe para la realización de pagos *online* de las reservas, al igual que para el sistema de suscripciones.

Para la utilización de Stripe, hemos importado su librería a *Spring*, y, a partir de ahí, mediante llamadas a su API, hemos generado distintos tipos de facturas, suscripciones y reembolsos. Veremos varios ejemplos más adelante cuando expliquemos el desarrollo del proyecto.

7.2 Arquitectura

Para la realización del proyecto, hemos utilizado un estilo arquitectónico llamado Modelo Vista Controlador, ya que el entorno de trabajo *Spring* es ideal para aplicaciones web que siguen este modelo.

Este estilo se divide en tres partes:

El modelo, es dónde se contienen los datos que maneja el sistema, sus mecanismos de persistencia y toda la lógica del negocio.

La vista, sería la interfaz de usuario, contiene la información que va a recibir el usuario y va a responder a sus interacciones.

El controlador, es el que actúa de intermediario entre la vista y el modelo. Gestiona todo el flujo de información entre ellos.

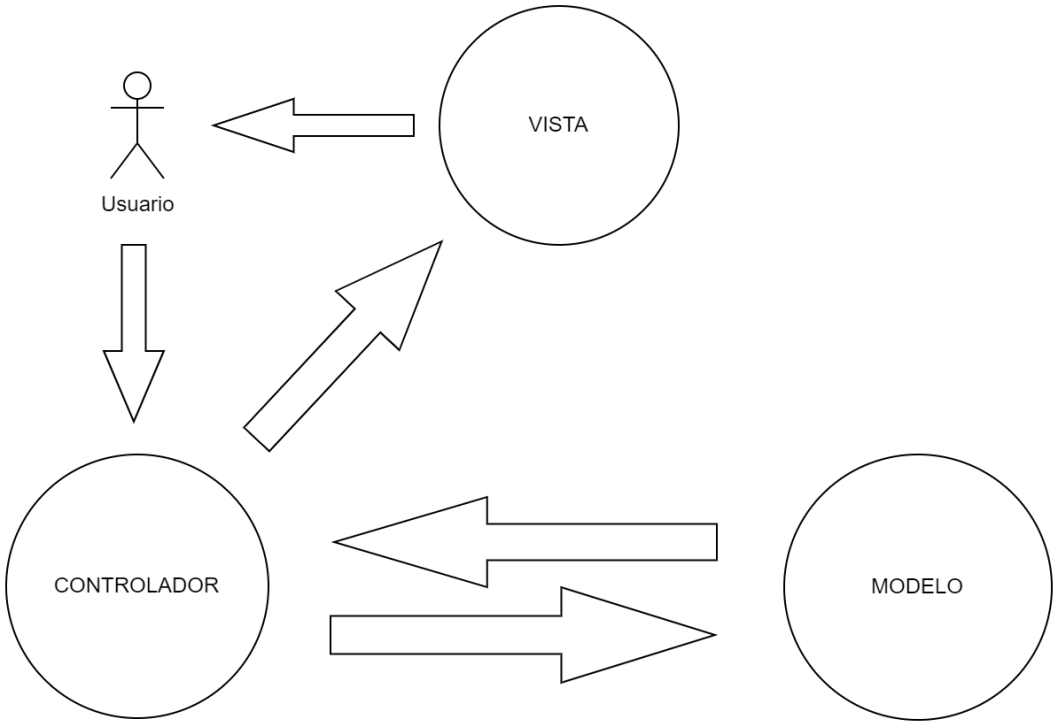


Imagen 10: Esquema de cómo funciona una arquitectura MVC (Modelo Vista Controlador)

Como vemos en el esquema, el usuario va a interactuar con la interfaz. En el momento en el que el usuario realiza alguna acción (pulsar un botón, rellenar un campo de texto…), va a provocar un evento.

El controlador será el encargado de capturar el evento y va a comenzar a ejecutar la lógica del programa. Frecuentemente, el intercambio de datos entre la interfaz de usuario y el controlador es distinta a los datos que se encuentran almacenados en el modelo o en la base de datos.

El controlador va a gestionar estos datos pasándoselos al modelo. Es posible que, durante esta interacción el controlador añada, modifique o elimine objetos del modelo, aunque también se pueden realizar operaciones de consulta. Una vez el controlador obtenga las respuestas del modelo, va a enviarle a la vista la información organizada, que es la que se va a encargar de mostrarla en la interfaz de usuario.

7.3 Implementación

En este apartado, vamos a explicar la estructura que sigue el código tanto en la parte de *Front End* como la de *Back end*. También vamos a hablar del porqué y cómo están organizadas las carpetas dentro de nuestro proyecto.

7.3.1 Front End

Vamos a comenzar hablando del explorador de archivos de *Visual Studio* y ver cómo están organizadas las carpetas.

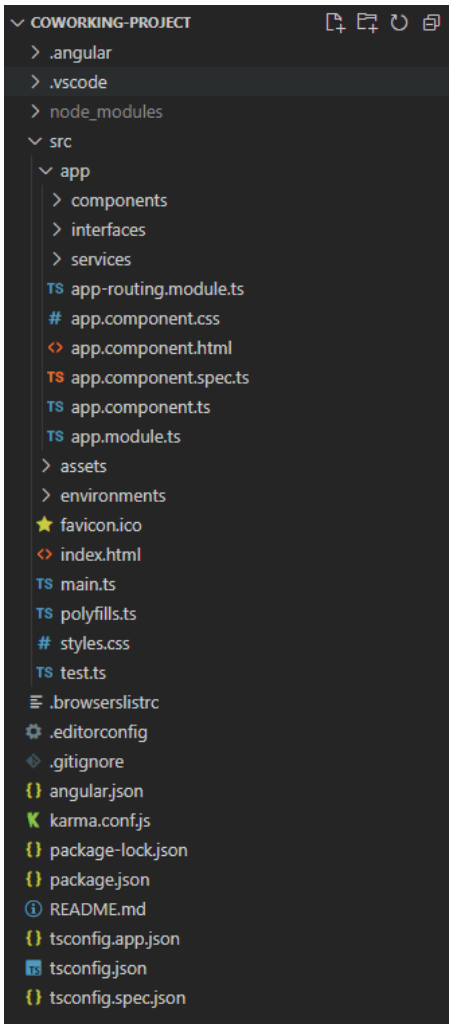


Imagen 11: Explorador de archivos de Visual Studio para nuestro proyecto Angular

Vamos a centrarnos en la carpeta de *src*, que es donde va a estar contenido todo nuestro proyecto, y donde vamos a generar todos los componentes y servicios que vamos a ir utilizando.

Antes de entrar en detalle, me gustaría resaltar el archivo "package.json", que es el archivo que se usa para administrar las dependencias, los *scripts* y las versiones de las distintas librerías que vamos a utilizar a lo largo del proyecto.

Carpeta *src*:

Dentro de esta, vamos a tener 3 carpetas más. Estas corresponden con *app*, *assets* y *environment*, de las cuales nos vamos a centrar principalmente en *app*.

Pero antes de ponernos con la carpeta *app*, vamos a dar una breve explicación del contenido de las otras dos carpetas. La carpeta de *assets* va a contener archivos arbitrarios y planos de texto, imágenes, fuentes y videos que se vayan a utilizar en la aplicación. En nuestro caso, guardaremos ciertas imágenes que vamos a utilizar y un archivo *json* plano para la barra de navegación, la cual entraremos en detalle más adelante. La carpeta *environment* va a contener las configuraciones del entorno, en nuestro caso, como solo funcionamos con uno, no se ha modificado esta carpeta durante los desarrollos.

Vamos ahora con la **carpeta *app*:**

En esta carpeta, vamos a tener primero los componentes y módulos que son comunes a todo el proyecto.

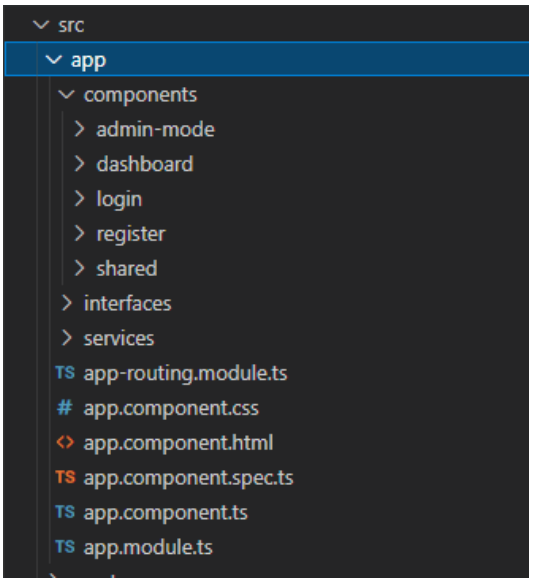


Imagen 12: Jerarquía de la carpeta app en el front end.

En el *front end*, un componente es el conjunto de elementos con una misma finalidad que se va a visualizar en la interfaz. Se compone de 4 archivos. El primero va a ser el archivo *typeScript*, que va a contener la script, es decir, la lógica que se va a aplicar en este componente. Vamos a tener también un archivo HTML, que contendrá la estructura basada en etiquetas de los elementos contenidos, y finalmente el archivo CSS, que será el encargado de dar diseño y hacer que la interfaz de estos sea visual e intuitiva para los usuarios. También contiene un archivo *spec.ts*, que es un archivo que se reserva para la generación de pruebas *(testing)* del componente, pero que no lo hemos utilizado durante el proyecto.

El módulo *app.module.ts* es un contenedor para un bloque de código, y va a ser el encargado de importar y exportar todos los componentes y proveedores de servicios a todo nuestro código.

Finalmente, vamos a tener el *app-routing.module.ts*, que es un módulo que nos va a crear unos enlaces, en los que dependiendo de la dirección *url* de la página web, nos va a mostrar el componente.

Imagen 13: app-routing.module.ts

Como podemos ver en la imagen anterior, el *app-routing.module.ts*, desde la línea 8 hasta la 14, tenemos indicados las direcciones a las que va a corresponder cada uno de los componentes. Es decir, en la directiva *'path'* vamos a indicar nombre que se va a añadir a la *url* raíz. En nuestro caso, vamos a estar trabajando por ahora en localhost, con lo que nuestra *url* raíz va a ser *'http://localhost:4200'*.

Con lo cual, si a esta dirección *url* le añadimos *'/login'* al final, nos cargará por pantalla el componente asignado a la directiva *'login'*. En caso de dejarla vacía, nos redirigirá al panel de control, como se indica en la línea 8, y en caso de que la *url* no sea reconocible, nos redirigirá igualmente al panel de control, como se indica en la línea 14.

Para las líneas 10 y 11, vemos que la directiva no nos indica el componente, y, en su lugar, tenemos un *loadChildren()*. Lo que hacemos con esta directiva es optimizar el rendimiento de la aplicación, ya que va a cargar el módulo de la ruta anidada y va a estar trabajando en ella con su módulo, en lugar de con el módulo raíz.

Vamos a pasar a revisar las subcarpetas que se encuentran en esta carpeta *app*. Como podemos ver en la imagen 12, tenemos una subcarpeta para cada tipo de archivos: Los servicios, los componentes y las interfaces. Vamos a explicar a qué corresponde cada uno de estos archivos.

Las interfaces

Las interfaces van a ser nuestros objetos planos que van a servir para los intercambios de datos con el servidor *back end*. Vamos a hacer un pequeño repaso de cómo va a ser el intercambio de datos en nuestra arquitectura de Modelo Vista Controlador.

El usuario va a realizar una acción (desde el front end), que va a provocar un evento en el controlador (servidor de *back end*) y este realizará la petición al modelo (base de datos), que, tras atravesar cierta lógica, se devolverá un objeto a la interfaz.

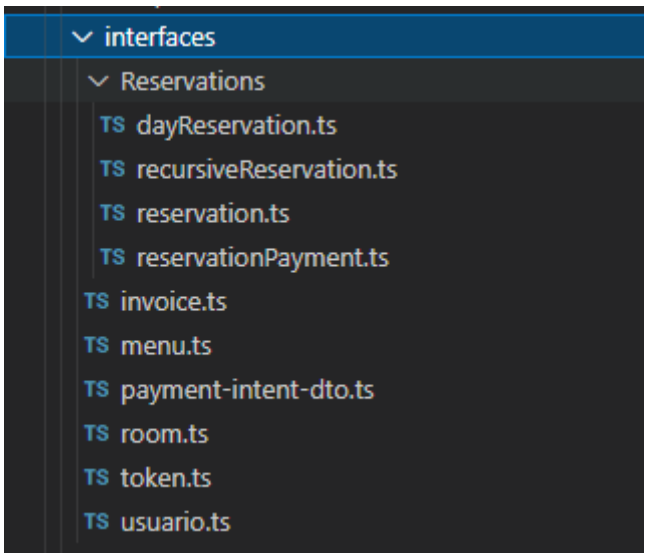


Imagen 14: Interfaces en el explorador de archivos de Visual Studio.

Estas van a ser todas las interfaces que vamos a utilizar a lo largo del proyecto.

Las interfaces son los objetos que se van a mostrar por pantalla, por lo que solo va a contener la información necesaria, no tienen por qué ser idénticos a los objetos en la base de datos.

Los servicios

Los servicios son los proveedores de datos. Se les llama desde los componentes, y son los responsables de acceder a la información de la lógica y realizar operaciones con esta. En nuestro caso, los servicios los estamos utilizando mayoritariamente para comunicarnos con el *back end*. Nuestros servicios se van a resumir en métodos que realizan llamadas para obtener o actualizar datos. Vamos a tener un servicio relacionado con cada funcionalidad de nuestro programa, es decir, uno para realizar todas las llamadas relacionadas con las empresas (*companies*), otro para realizar las llamadas referentes a las facturas (*invoice*), otro para los componentes que va a tener el menú, otro para los pagos (*payment*), para las reservas (*reservations*), para las salas (*room*) y finalmente para los usuarios (*users*). Existen 4 tipos de llamada que realizarán los servicios a la lógica de la aplicación:

La llamada GET, que sirve para que la llamada nos devuelva la información que queremos.

La llamada POST, que se utiliza para almacenar datos que vamos a enviar al *back end* desde el servicio.

La llamada DELETE, que la vamos a utilizar para eliminar datos de la base de datos.

La llamada PUT, que realizará modificaciones en el objeto que queramos.

Vamos a poner un ejemplo de alguno de nuestros servicios:

```
25
26  getPublicableUsers(page: number): Observable<Usuario> {
27    var params = new HttpParams().set("page", page);
28    return this.http.get<Usuario>("http://localhost:9090/api/user/publicableUsers", { params: params });
29  }
30
31  createUser(user : Usuario) {
32    return this.http.post("http://localhost:9090/api/user/add", user, { responseType: "text" });
33  }
34
35  deleteUser(id : number){
36    return this.http.delete("http://localhost:9090/api/user/delete/" + id);
37  }
38
```

Imagen 16: Extracto de código del users.service.ts

En esta imagen, podemos observar que hay 3 métodos, todos ellos están utilizando funciones de la clase *HttpClient* de *Angular*. Podemos ver, en la línea 28, que el primero de ellos es una llamada GET, en el que está buscando los usuarios publicados. Esta llamada va a retornar un objeto de la interfaz Usuario.

La segunda llamada es de tipo POST, en el que le enviamos un objeto que va a ser recogido por el *back end*, y va a añadirlo a la base de datos. Este objeto va a ser del tipo de la interfaz usuario.

La tercera llamada es de tipo DELETE, y le pasamos el id del usuario que queramos eliminar. Desde la lógica del *back end*, se eliminará el usuario con el identificador proporcionado.

Los componentes

Como hemos explicado anteriormente, un componente es el conjunto de elementos con una misma finalidad que se va a visualizar en la interfaz. Después de ver el direccionamiento de rutas, sabemos que, dependiendo de la ruta, vamos a cargar uno u otro. Pero esto no siempre es así. Dentro de un componente, podría haber otro embebido, según vayamos avanzando en la explicación del proyecto, lo podremos observar.

Vamos a hacer un repaso de cómo tenemos dividida la carpeta de los componentes:

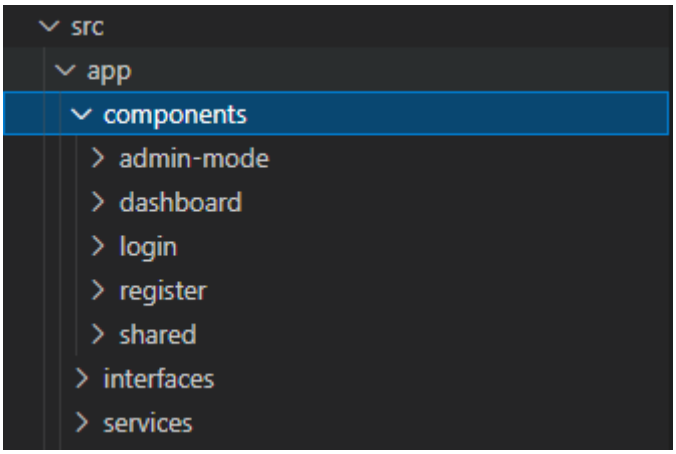


Imagen 17: Jerarquía de los componentes en el explorador.

En este caso, hemos organizado los componentes en subcarpetas. Cada carpeta va a tener los componentes relacionados con esta. Por ejemplo, en la carpeta de *admin-mode*, vamos a tener solo los componentes relacionados con el modo administrador. En la carpeta de *dashboard*, vamos a tener todos los componentes que aparecen en el panel de control. Más adelante, durante la presentación del proyecto final, veremos cómo están organizados los componentes.

7.3.2 Back End

Vamos a comenzar por echarle un vistazo a cómo están organizadas las carpetas del lado de *Back end*.

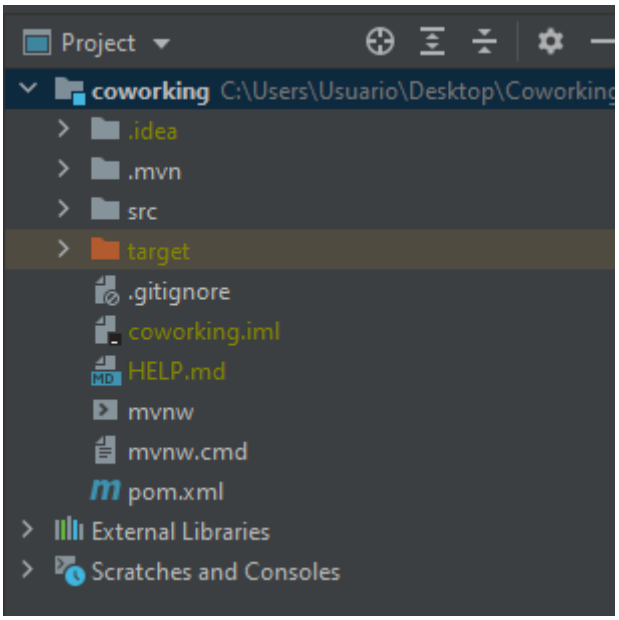


Imagen 18: Explorador de archivos de IntellIJ para el backend.

De todos estos archivos, los más relevantes y los que más vamos a utilizar a lo largo del proyecto son los siguientes:

El archivo *pom.xml* es el archivo que no puede faltar en ningún proyecto que utilice las dependencias de *Maven*. *Maven* es una herramienta potente para construir proyectos en *Java* y es la que vamos a estar utilizando. En este archivo *pom.xml*, es donde vamos a tener todas las dependencias y configuraciones que vamos a utilizar en el proyecto. Todas las librerías que se añadan de las dependencias se almacenarán en la carpeta de *External Libraries*.

La carpeta *target* es la carpeta donde se genera el código compilado y el que se va a ejecutar cuando generemos un ejecutable. Este *target* lo va a generar la primera vez que corramos el programa, y se puede eliminar fácilmente mediante el comando *mvn clean*, pudiendo crearlo de nuevo en la siguiente ejecución.

Y ahora, nos vamos a centrar en la carpeta que más nos interesa, la carpeta *src*, que, al igual que en el apartado de *front end*, es donde vamos a encontrar todo el código de nuestro proyecto.

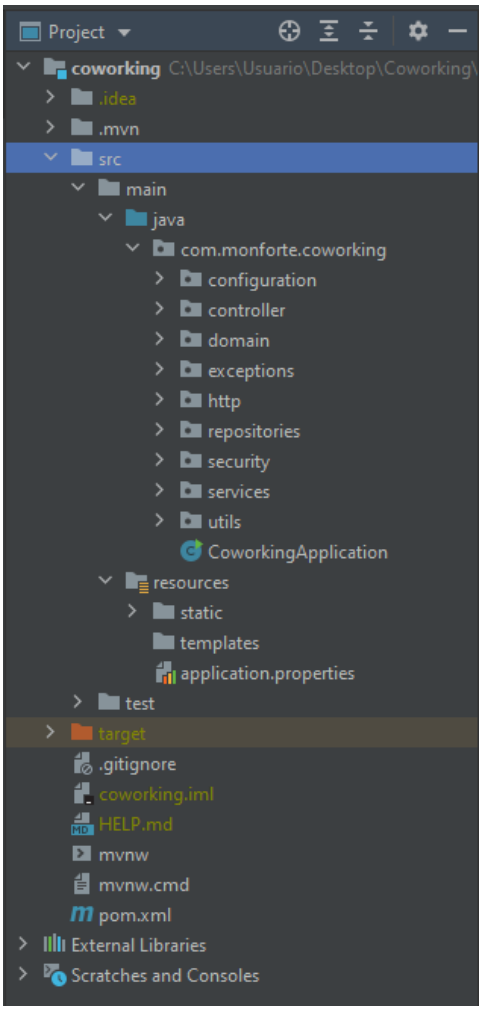


Imagen 19: Jerarquía del explorador de archivos en la carpeta src

Como podemos ver, hay dos carpetas dentro del *src*, que son la carpeta *main*, donde va a estar todo el código contenido, y la carpeta *test*, donde generaremos las pruebas de nuestro código. En nuestro caso, no hemos generado ningún test.

En la carpeta *main* se encuentran dos carpetas. La primera de ellas es donde va a estar todo el código: *com.monforte.coworking*, y la segunda de ellas, *resources*, es donde vamos a almacenar los recursos que podremos ir utilizando a lo largo del código. Dentro de la carpeta de recursos, cabría destacar el *aplication.properties*. Este es un documento de configuración del entorno. En este, vamos a indicar datos para conectarnos a la base de datos, en qué puerto va a arrancar el programa, así como todas las configuraciones que va a tener nuestro código en este entorno.

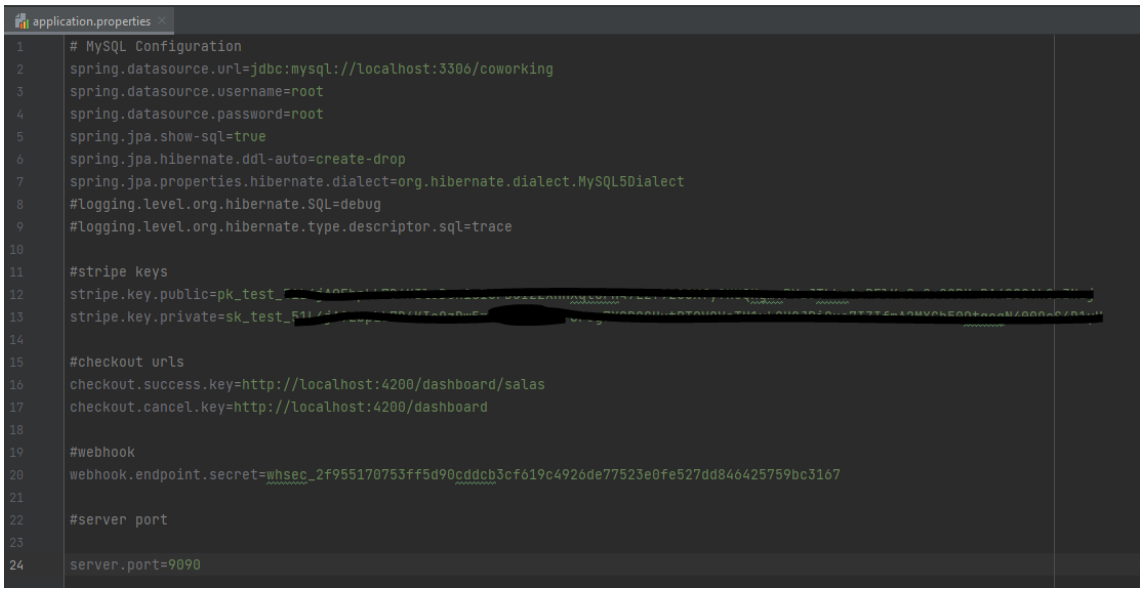


Imagen 20: Archivo application.properties

De la primera a la línea 7, estamos configurando la base de datos SQL: *url* y credenciales a las que se va a conectar, configuración de cómo se va a crear la base de datos. En las líneas 12 y 13 se encuentran las claves de Stripe, para que los pagos los lleguen a una misma cuenta. Las líneas 16 a 20, son configuraciones del entorno relacionadas con Stripe que veremos más adelante. Por último, en la línea 24, tenemos el puerto en el que se va a ejecutar nuestro servidor. Por defecto es el 8080, pero en nuestro caso lo hemos cambiado al 9090.

Volviendo a la jerarquía de carpetas de la imagen 17, vamos a dar una explicación de cómo hemos organizado cada carpeta y a que corresponden.

Configuración (configuration)

Esta carpeta tendrá contenido un solo archivo, con una configuración llamada CORS y es para que se puedan aceptar las llamadas que realiza nuestro *front end*.

Controladores (controller)

Esta carpeta va a contener los controladores de nuestro programa. Vamos a indicar a *Spring* con la etiqueta *@Controller* al comienzo de la clase y con esto ya sabrá que se trata de un controlador que está esperando a que le envíen eventos por el puerto indicado a la *url* indicada. Los controladores son los encargados de recoger y devolver las llamadas que se realizan desde el *front end*, es por eso por lo que cada operación de cada contenedor va a tener una *url* única, aunque se va a construir de manera parecida.

Vamos a ver ahora cómo está organizada esta carpeta y un ejemplo de un controlador:

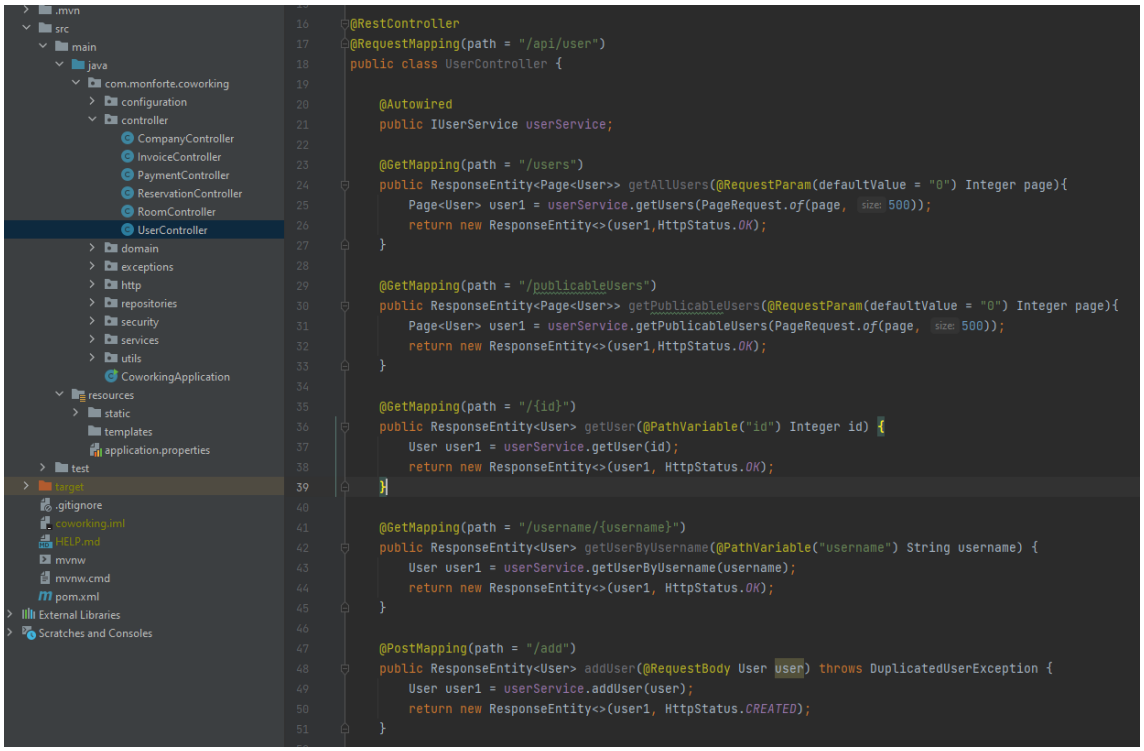


Imagen 21: Carpeta controladores y parte del código de la clase UserController

Dentro de cada uno de los controladores va a haber métodos, cada uno con su *url* única. La *url* por defecto se añade a *'http://localhost:9090'* y a esto le añadimos el valor que se indica en la variable *path* del *@RequestMapping* que encontramos en la línea 17, justo en la línea anterior de definir nuestra clase. Además de este, a cada método le añadimos el *path* que aparezca en su anotación. Por ejemplo: Si queremos hacer una llamada que llame a todos los usuarios publicables, sería el segundo método que aparece (línea 29), deberíamos hacer una llamada a 'http://localhost:9090/api/user/publicableUsers', que esta se hará desde los archivos de servicio del *front end*.

Existen dos anotaciones que vamos a utilizar a la hora de pasar parámetros a los métodos de los controladores.

La primera de ellas va a ser *@PathVariable*, la cual le vamos a pasar una variable como parámetro en la misma *url*. Esta variable suele ser útil para hacer búsquedas por algún tipo de parámetros. Se utiliza sobretodo para las llamadas GET de algún dato en

concreto y, para las llamadas de DELETE, le indicaremos el identificador del objeto que queramos eliminar.

La segunda de ellas va a ser el *@RequestBody*. La vamos a utilizar cuando, junto a la llamada http que se realiza, nos van a enviar un objeto. Este tipo de anotaciones es muy útil para añadir objetos a la base de datos, es decir, para las llamadas de tipo POST y PUT. Cuando leamos el objeto relacionado con esta anotación, este va a tener el formato JSON, y *Spring* lo va a traducir automáticamente en el objeto que espera. Esto pasa también con la respuesta que enviamos. El *ResponseEntity* que se encuentra junto al *return*, va a enviar una respuesta en formato JSON.

Entidades y DTOs (domain)

En esta carpeta vamos a encontrarnos con todas las entidades y DTOs que vamos a ir utilizando a lo largo del programa.

Las entidades son objetos de persistencia. Son objetos que se van a utilizar para mantener y recuperar la información de la base de datos. Cada tabla de la base de datos corresponde con una de estas entidades, y cada columna de la tabla corresponde con una variable. Con lo cual, cada fila de esta tabla corresponderá con una instancia de la entidad.

Un DTO (Data Transfer Object) es un patrón de diseño, que consiste en crear objetos que agregan y encapsulan información, y, posteriormente, se utiliza para intercambiar estos datos sobre la propia capa de lógica y el front end. Estos objetos DTO no tienen nada que ver con las tablas de nuestra base de datos.

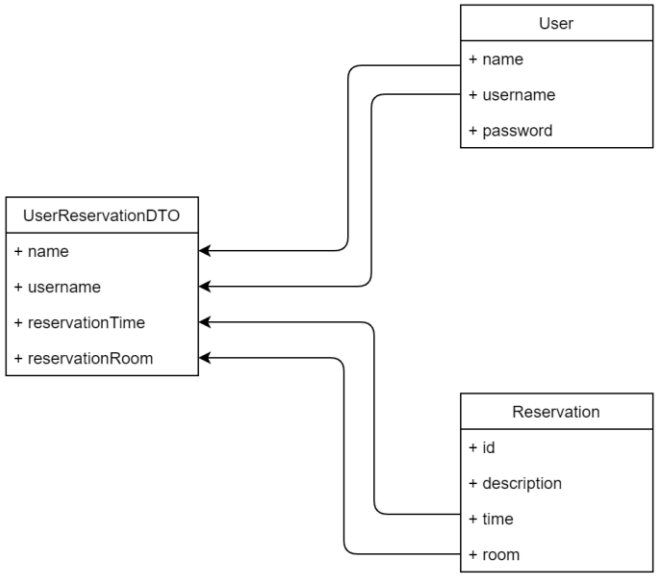


Imagen 22: Esquema del patrón de diseño DTO.

Con este esquema podemos ver el uso básico de cómo funciona este patrón de diseño, y como el objeto DTO va a estar construido a medida, según nuestras necesidades, a partir de otras entidades u otro tipo de cálculo.

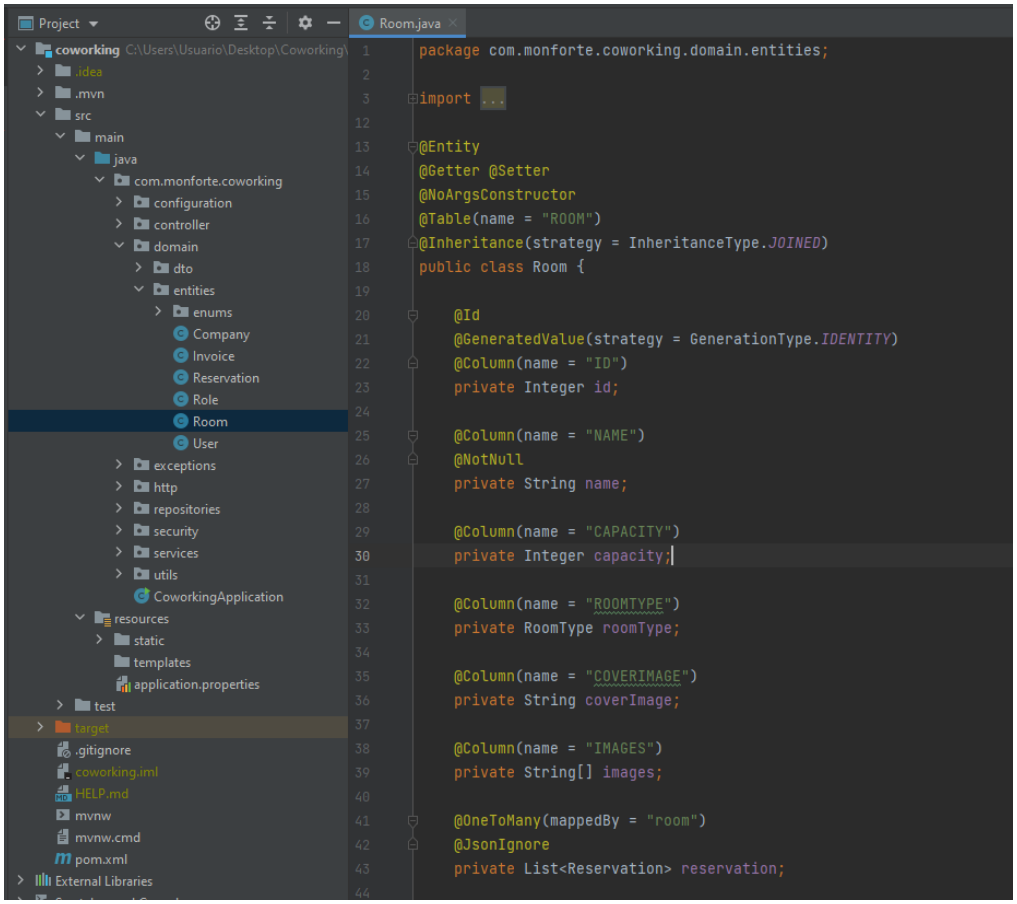


Imagen 23: Entidad Room y jerarquía de carpetas en el explorador.

Como podemos observar, cada una de las entidades que tenemos creadas en el explorador corresponde con cada una de las tablas que tenemos en la base de datos. Utilizamos la anotación *@Entity* para indicar a Spring que se va a tratar de una entidad.

Las anotaciones *@Getter*, *@Setter* y *@NoArgsContructor*, en las líneas 14 y 15, son anotaciones de la biblioteca de Lombok. La función de esta biblioteca es crear todos los *getters* y *setters* de todas las variables sin necesidad de escribir el código. Estas se utilizan para obtener o modificar los resultados de una instancia del objeto.

Al definir las variables, utilizamos varias anotaciones. Para el identificador ID, le indicamos a Spring que se trata del ID, y que lo genere como un valor de identidad. Hay que indicar el valor de la columna con la que se va a corresponder cada variable en la base de datos.

En las líneas 41 a 43, le indicamos a Spring que la variable *reservation* va a tener una relación de uno a muchos. Al ser la variable de tipo Lista de *Reservation*, que corresponde con otra entidad, Spring va a saber que se trata de una relación con esta. Y

es que, según la lógica del programa, queremos que cada sala contenga una lista de todas las reservas realizadas sobre esa habitación.

Excepciones (exceptions)

En esta carpeta, guardaremos las excepciones que serán propias de nuestro programa. Son clases que extienden de la clase de *Java Exception*. Las utilizaremos, por ejemplo, cuando dos reservas se colapsen, o cuando dado un ID de alguna entidad, no se encuentre en la base de datos, lo que provocará una excepción que informará a los usuarios del error.

Repositorios (repositories)

El repositorio es el encargado de realizar todas las operaciones de persistencia contra la base de datos. En nuestro caso, vamos a tener tantos repositorios como tablas en nuestra base de datos.

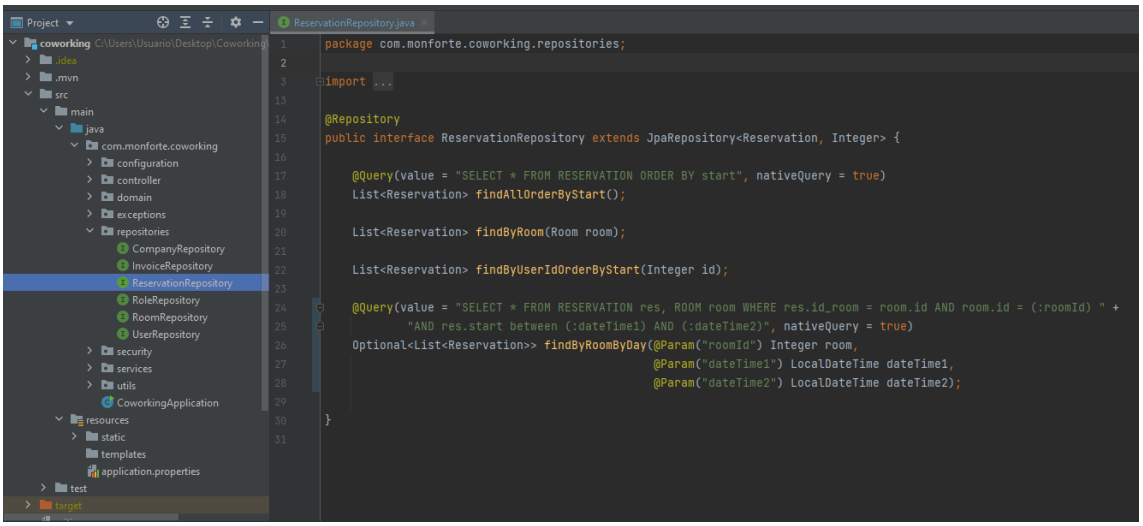


Imagen 24: Jerarquía de los repositorios y ReservationRepository

Le añadimos la anotación *@Repository* a la clase para indicarle a Spring que se trata de un repositorio. Este extiende a *JpaRepository*. A cada uno de ellos, le indicamos la entidad con la que va a corresponderse y el tipo de su identificador. Esto lo encontramos después del *'extends'* de la línea 15.

La implementación de este repositorio nos permite tener acceso a las llamadas a la base de datos *(queries)* predefinidas de *Jpa*, al igual que nos permite, mediante una nomenclatura propia, realizar nuestras llamadas. Por ejemplo, sin indicar el tipo de llamada que queremos realizar, en la línea 20, solo con el *findByRoom(Room room)* y pasandole un objeto de tipo *Room*, ya sabrá que tiene que buscar por el id del objeto que le proporcionemos, y nos devolverá una lista de reservas asociadas con esta sala.

La anotación *@Query* nos permite personalizar nuestras llamadas a la base de datos. Por defecto, esta directiva va a leerlo en un lenguaje de base de datos llamado HQL (*Hibernate Query language*) pero al añadirle la directiva *nativeQuery = true* al final de

esta anotación, le indicamos que la query que hemos escrito está en lenguaje SQL nativo. Esto es lo que ocurre en las líneas 24 y 25.

Seguridad (security)

En esta carpeta vamos a tener todo lo relacionado con la seguridad. Explicaremos con detalle cómo funciona la seguridad de nuestro proyecto más adelante.

Servicios (services)

En la carpeta de servicios es donde se va a ejecutar toda la lógica de nuestro proyecto.

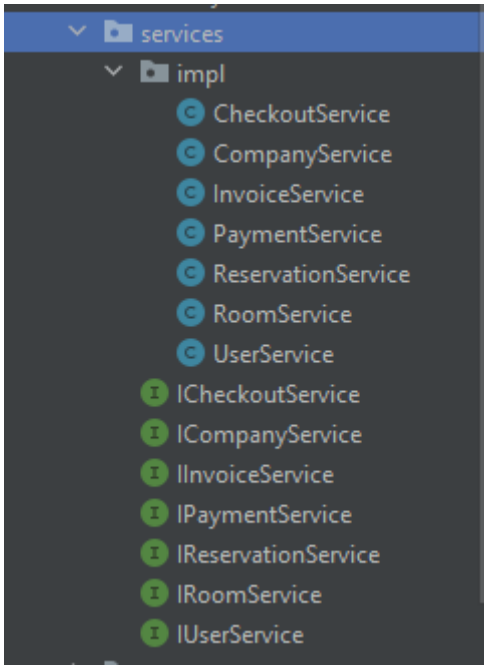


Imagen 25: Jerarquía de la carpeta de servicios

Podemos ver que todos nuestros servicios tienen una interfaz y una implementación. En este caso, estamos aplicando un patrón de diseño de inyección de dependencias.

El patrón de diseño de inyección de dependencias es una técnica que hace que la clase sea independiente de sus dependencias. Desacoplamos el objeto de su creación, por lo que nos permite crear instancias de este, aunque el objeto no tenga implementación. Este patrón de diseño nos puede ayudar para futuros cambios que pueda tener la aplicación, ya que no habrá que cambiar la clase, si no sus implementaciones, o incluso crear nuevas implementaciones. Es una mejora en la reusabilidad del código.

Estos servicios van a contener métodos, serán los responsables de aplicar la lógica del proyecto. Accederán al repositorio y harán los cambios oportunos a los objetos para responder con precisión la petición realizada. Serán los responsables de dar las instrucciones para que haya cambios en la base de datos, lo iremos viendo durante los siguientes apartados.

Utilidades (utils)

En esta carpeta vamos a tener documentos de utilidades que servirán a lo largo del proyecto y podrán ser accedidos por el resto de clases.

7.3.3 Autentificación con JWT

Una vez hemos explicado toda la estructura y varios de los conceptos de nuestro proyecto, en este apartado, vamos a ver cómo hemos realizado la parte del inicio de sesión y la seguridad del usuario.

Para la autentificación del usuario, hemos utilizado JWT (*JSON Web Token*). Es un estándar basado en JSON para la creación de un token de acceso. Cada vez que alguien inicie sesión, se generará un token que tendrá información relativa del usuario y sus roles.

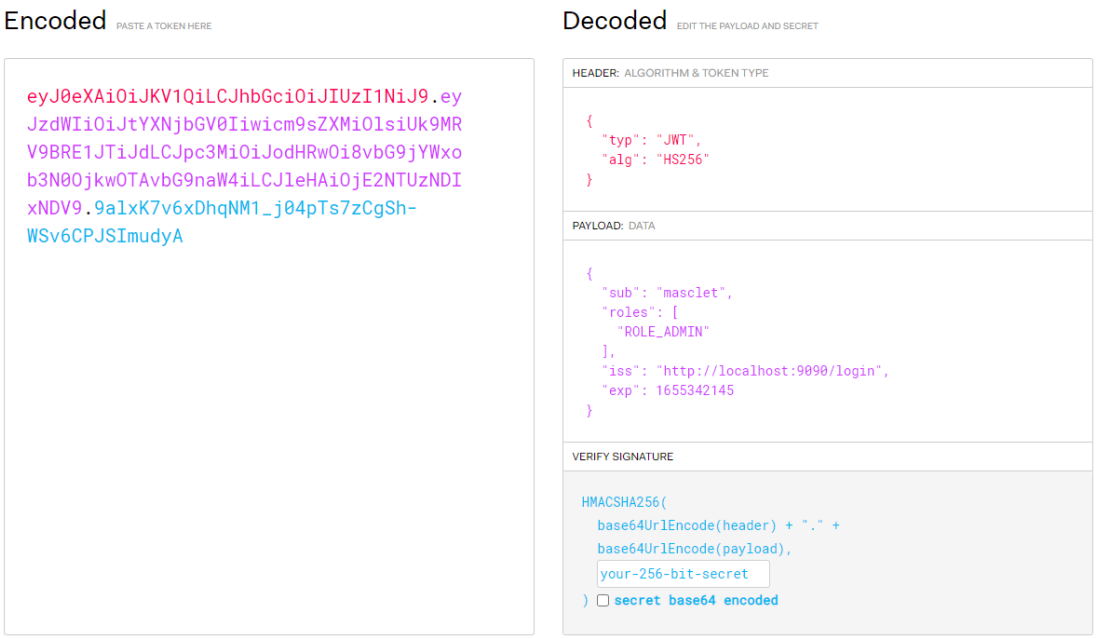


Imagen 26: Ejemplo de un JWT token.

Desde la página web oficial de JWT (Referencia [4]) podemos desencriptar los tokens para saber qué información contienen. Como podemos observar en la imagen anterior, el *token* se divide en 3 partes: la cabecera (*header*), el contenido (*data*) y la firma (*signature*). Estos *tokens* se utilizarán en las llamadas que sean necesarias para comprobar que efectivamente el usuario está iniciado sesión y si tiene o no los privilegios necesarios para dicha llamada. Es lo que nos permitirá distinguir si un usuario tiene los permisos para acceder o no al modo administrador.

Gracias a esto, cada vez que necesitamos información relativa con el usuario activo, y gracias a una librería de *jwt_decode*, hemos accedido al token, lo hemos decodificado y

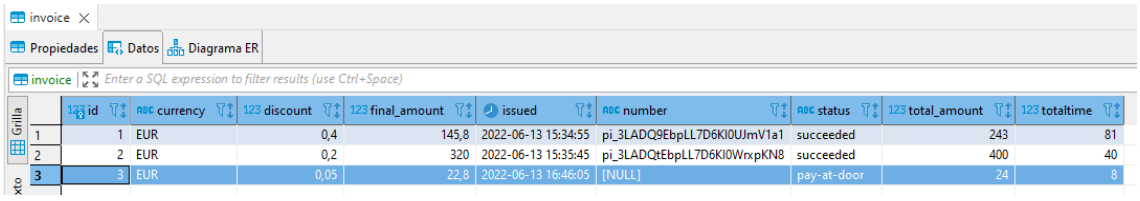
almacenado el nombre de usuario en una variable, del apartado de *front end*. Gracias a esta variable con el nombre de usuario, y, tras una consulta a la base de datos, nos ha permitido sacar toda la información del usuario. En caso de no haber token, esta variable se mantendría vacía.

7.3.4 Creación de pagos y facturas

Para crear los pagos *online* hemos utilizado ciertas funcionalidades de Stripe que nos ofrecen sus bibliotecas y que se conectan con nuestra cuenta mediante las claves que proporcionamos en el documento de *application.properties* en el *back end*.

Primero, generamos una intención de pago. Hacemos una llamada a una función de las bibliotecas de Stripe, lo que nos generará paralelamente este pago en su página web. Al mismo tiempo, nosotros almacenaremos la factura en nuestra base de datos. Después, tenemos que confirmar esta intención de pago, esto es lo que ocurrirá este ha sido efectuado con éxito. En caso de cancelar la reserva o de que no se haya podido efectuar, se cancelará. Tendremos un tiempo de respuesta hasta que se confirme, donde la pantalla quedará cargando, y nos saltará un letrero que nos dirá si se ha confirmado.

A la vez que generamos pagos con Stripe, también generaremos facturas en nuestra base de datos, recopilando toda la información necesaria. En la siguiente imagen, vamos a ver cómo se almacenan las facturas en nuestra base de datos, vamos a ver tres: dos de ellas ya se ha realizado el pago *online*, con lo que el estado está marcado como *'succeded'*, mientras que la última se ha indicado que se pagará en la puerta del local.


invoice

Propiedades | Datos | Diagrama ER

invoice — Enter a SQL expression to filter results (use Ctrl+Space)

Grilla

	id	currency	discount	final_amount	issued	number	status	total_amount	totaltime
1	1	EUR	0,4	145,8	2022-06-13 15:34:55	pi_3LADQ9EbpLL7D6KI0UJmV1a1	succeeded	243	81
2	2	EUR	0,2	320	2022-06-13 15:35:45	pi_3LADQtEbpLL7D6KI0WrxpKN8	succeeded	400	40
3	3	EUR	0,05	22,8	2022-06-13 16:46:05	[NULL]	pay-at-door	24	8

Imagen 28: Las facturas almacenadas en la base de datos.

Es por ello por lo que esta tercera factura no tiene ningún identificador de Stripe y el estado está marcado como *'pay-at-door'*, es decir, pagar en puerta. Se le cobrará al cliente una vez acuda presencialmente.

Para generar las facturas a partir de nuestras reservas, estamos agrupando todas las reservas en una lista, que luego se va a recorrer y va a ir sumando el montante total de la reserva. Dependiendo de la cantidad total de horas que contenga la lista, se aplicará un descuento u otro.

Para las reservas simples, para cada una se generará una factura, mientras que, para las que impliquen varias reservas, como, por ejemplo, las recursivas, seguiremos una

estructura que nos indique los días de la semana en los que vamos a querer que esta factura se repita.

Estos datos serán enviados desde las interfaces a los controladores en un DTO, y será desde el apartado lógico del *back end* donde se traducirá este objeto a su formato entidad, es decir a un objeto de formato *Reservation*.

Una vez obtengamos la reserva en su formato entidad, ya estará lista para ser almacenada en nuestra base de datos. Como, para estos tipos de reserva, tenemos que calcular el tiempo y el montante total sumado de cada una de las reservas, vamos a realizar varias comprobaciones. Primero, haremos un bucle, teniendo la fecha de inicio como anterior a la fecha de fin de la reserva, y lo iremos recorriendo. Conforme se vayan creando las reservas que correspondan con los días seleccionados, estas se añadirán a una lista. Al final del bucle, la lista de reservas sumará el tiempo total de las reservas, y mediante un cálculo, sacará el tiempo y el coste total de la factura, que la almacenará al mismo tiempo.

Imagen 30: Lista de precios como variables globales en el InvoiceService

Podemos ver cómo tenemos los precios base y los descuentos a aplicar en las variables globales de la clase. Los precios están indicados como números con decimales *double*, mientras que los descuentos se almacenarán en *arrays* de números decimales.

Después, para aplicar un descuento u otro, medimos la cantidad total de tiempo de todas las reservas, recorriendo así la lista mencionada anteriormente. Dependiendo de si el usuario está suscrito, aplicará un descuento distinto, seleccionando la posición del array correspondiente con su descuento.

7.3.5 Suscripciones de Stripe

El caso de las suscripciones funciona de manera distinta al de los pagos, ya que se tratará de objetos distintos que nos proporciona Stripe en sus bibliotecas. Para la realización de suscripciones, se nos abrirá una pestaña independiente de nuestro proyecto, donde rellenaremos toda la información relativa al pago *online*. Una vez rellenada toda la información, Stripe nos enviará respuestas que llamaremos *webhooks*, que sirve para poder recibir notificaciones directamente desde Stripe. De nuestro lado, tendremos un servidor funcionando que recibirá dichas peticiones, y que cada vez que reciba una, hará una llamada a nuestro controlador, quien gestionará estos eventos.

Dependiendo del tipo de evento que nos envíe Stripe, significará una cosa u otra, con lo cual, hemos implementado una lógica para escuchar solamente los eventos que nos interesen. En nuestro caso, como hemos comentado, a la hora de realizar una suscripción, nos redirige a una página web de Stripe que rellenaremos con nuestros datos. Cuando salgamos de ésta, nos enviará un evento de tipo: *checkout.session.completed*. Gestionaremos este evento para almacenar todos los datos que Stripe ha almacenado a partir de los que hemos rellenado en su web, como sería el número de *customer* (identificador del usuario para Stripe) o el número de la suscripción. Después, esperaremos a la llamada del evento *invoice.paid*, que indicará a nuestro sistema que el pago de la suscripción se ha realizado correctamente.

También vamos a gestionar eventos para cancelar suscripciones, o para indicar que el pago de la próxima suscripción es dentro de poco tiempo.

8. Producto desarrollado

Vamos a explicar el producto desarrollado y navegar por todas sus ventanas, respondiendo a la lista de requisitos funcionales que hemos presentado. Nos centraremos en explicar el uso de la aplicación de cara al usuario.

8.1 Crear cuenta e inicio de sesión

Lo primero que vamos a ver en nuestra página web final es el inicio de sesión que hemos presentado en apartados precedentes.

Como ya hemos visto, tendremos unos campos de texto para rellenar con nuestra cuenta, la opción de crear una cuenta, o la opción de acceder al panel de control directamente. Si rellenamos con nuestras credenciales, la aplicación va a comprobarlo con la base de datos, por lo que, si el usuario con el que intentamos acceder no está dado de alta, no nos dejará acceder a nuestra cuenta y nos aparecerá un aviso.

Si pulsamos la opción de crear una cuenta, se abrirá una ventana con un formulario que deberemos rellenar para crearla.

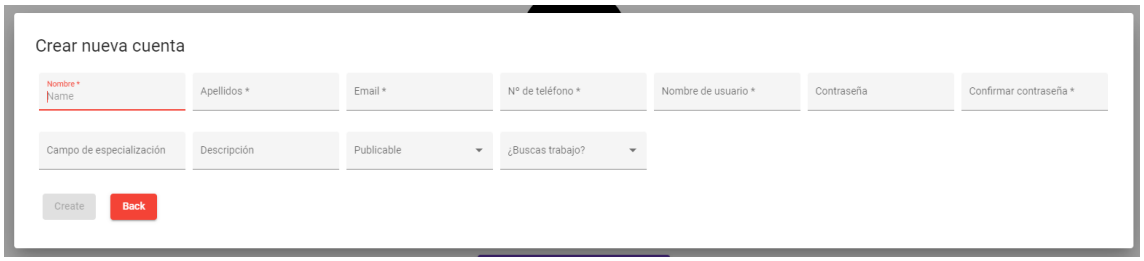


Imagen 32: Formulario para crear cuenta.

En este formulario, los campos dotados con un asterisco quieren decir que son obligatorios. Siguiendo los requisitos descritos anteriormente, hemos añadido dos más por si queremos que nuestra cuenta sea pública o privada, y si nos encontramos actualmente en una situación dónde buscamos trabajo.

8.2 Barra de Navegación

Vamos ahora a navegar por los menús de la aplicación y ver cómo hemos implementado todas sus funcionalidades.

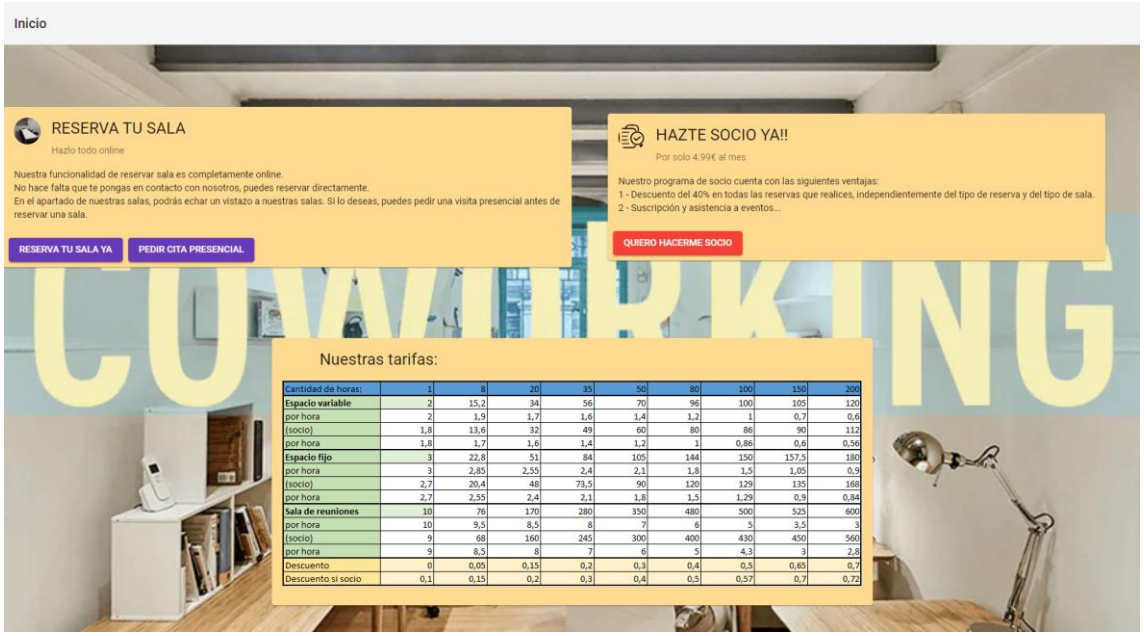
La barra de navegación se encuentra en el panel de control, y vamos a tener 2 versiones, dependiendo de si estamos con la sesión iniciada o no:

Imagen 25 y 26: Barra de navegación entre ventanas.

En la primera imagen, podemos ver que el usuario no ha iniciado sesión, con lo que, si pulsa en el botón de *login*, a la derecha de esta barra de navegación, aparecerá en la pestaña de inicio de sesión de la que hemos hablado en el punto anterior. En caso de tener la sesión iniciada, sí que saldrá su nombre, y un icono con el que podrá a acceder a cierto tipo de ajustes de su cuenta.

Vamos a hacer un repaso de izquierda a derecha de las opciones que tenemos en la barra de navegación. A la izquierda, se encuentra el nombre provisional de nuestra empresa, o podríamos poner un logotipo, algo que nos identifique de cara al usuario. Como todavía es provisional y no contamos con un logotipo oficial, he decidido dejar un texto plano que diga: "COWORKING VALENCIA".

La primera opción que vemos después del título es la opción de inicio. Ésta, nos llevará a la página de inicio donde tendremos información de nuestro *coworking*:



Cantidad de horas:	1	8	20	35	50	80	100	150	200
Espacio variable	2	15,2	34	56	70	96	100	105	120
por hora	2	1,9	1,7	1,6	1,4	1,2	1	0,7	0,6
(socio)	1,8	13,6	32	49	60	80	86	90	112
por hora	1,8	1,7	1,6	1,4	1,2	1	0,86	0,6	0,56
Espacio fijo	3	22,8	51	84	105	144	150	157,5	180
por hora	3	2,85	2,55	2,4	2,1	1,8	1,5	1,05	0,9
(socio)	2,7	20,4	48	73,5	90	120	129	135	168
por hora	2,7	2,55	2,4	2,1	1,8	1,5	1,29	0,9	0,84
Sala de reuniones	10	76	170	280	350	480	500	525	600
por hora	10	9,5	8,5	8	7	6	5	3,5	3
(socio)	9	68	160	245	300	400	430	450	560
por hora	9	8,5	8	7	6	5	4,3	3	2,8
Descuento	0	0,05	0,15	0,2	0,3	0,4	0,5	0,65	0,7
Descuento si socio	0,1	0,15	0,2	0,3	0,4	0,5	0,57	0,7	0,72

Imagen 33: Página de inicio de nuestro coworking.

Como podemos observar, tenemos en la imagen toda la información para hacerse socio, reservar una sala y una tabla con los precios y los descuentos que se nos va a ofrecer por horas. Esto respondería a la premisa de los requisitos funcionales, dónde se nos presenta toda la información acerca de nuestras salas y nuestros precios.

En el caso en el que no estemos con la sesión iniciada, en lugar del panel que nos sugiere hacernos socio, aparecerá uno para crearnos una cuenta. Hemos utilizado la

técnica que hemos mencionado en el apartado anterior, dónde a través del JWT token sacábamos el usuario activo con toda su información.

En esta ventana, tendremos la opción de pedir una cita presencial para hacer un tour por nuestras instalaciones, la cual nos abrirá una ventana que nos dará a elegir fecha en un calendario, y esta reserva se almacenará en la base de datos marcada como visita, con lo que no generará ninguna factura.

Después del inicio, vamos a encontrar la ventana de 'Nuestros coworkers'. Esta ventana nos mostrará una lista de todos los *coworkers* que hayan decidido que su cuenta sea pública.

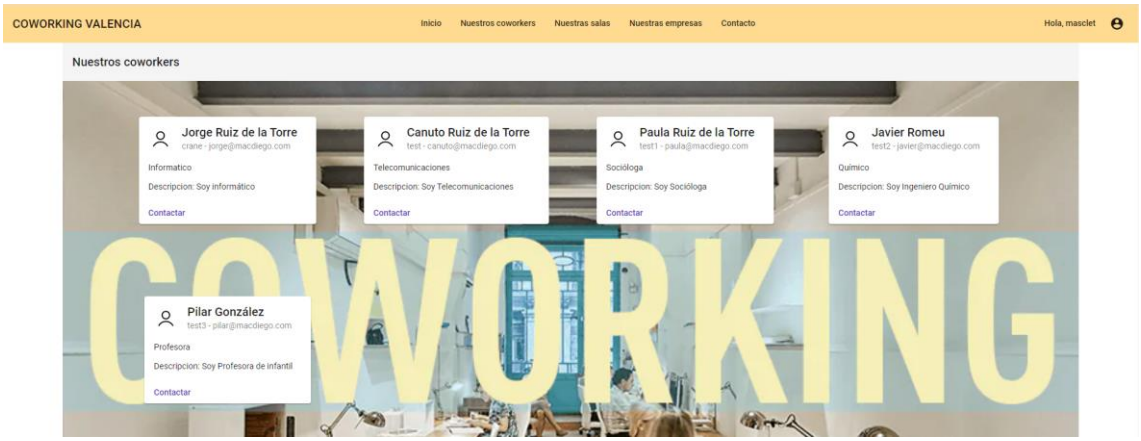


Imagen 35: Página de 'Nuestros coworkers'.

En este componente, hemos extraído del *Angular material* el componente de *grid*, es decir, nos genera una cuadrícula en la que vamos a ir organizando nuestra información. Hemos añadido tarjetas con los datos de nuestros usuarios, a la vez que un botón de contactar para que se pueda contactar con ellos.

Si nos fijamos, solo vemos 5 usuarios, aunque en nuestra base de datos hay 6.

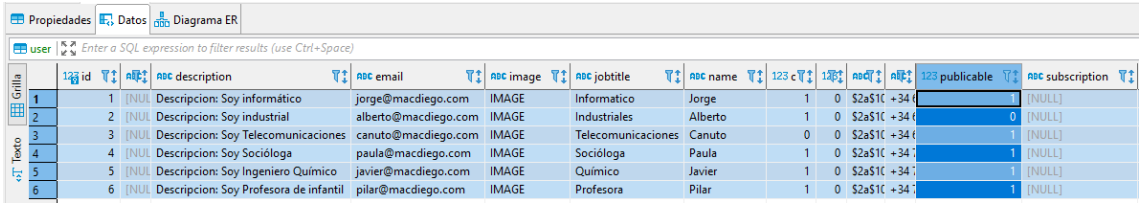


Imagen 36: Base de datos de los usuarios.

Esto se debe a que estamos haciendo una llamada a la base de datos con la premisa de que el atributo publicable sea cierto, con lo cual, sólo nos mostrará los usuarios cuya cuenta sea pública. En este caso, el segundo usuario ha decidido tener una cuenta privada, por lo que su tarjeta no aparecerá en la ventana de 'Nuestros coworkers'.

La siguiente opción que tenemos en nuestra barra de navegación es la de 'Nuestras salas'. Donde nos mostrará todas las salas que hay en nuestro *coworking*.

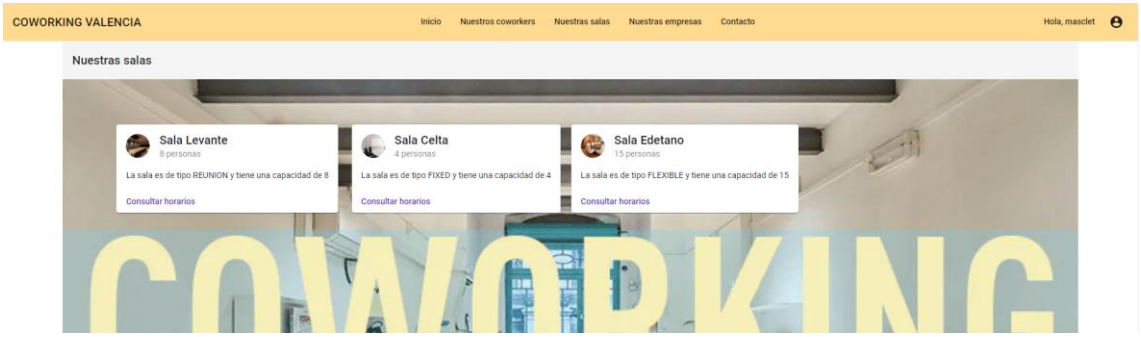


Imagen 37: Página de nuestras salas.

En esta imagen podemos ver que se ha utilizado una presentación similar a la página de 'Nuestros coworkers': Unas tarjetas puestas sobre una cuadrícula. En estas tarjetas, tenemos la información relevante respecto a la sala, donde podremos consultar los horarios y realizar las reservas. Lo veremos con más detalle en el próximo punto relativo a la reserva de salas.

Toda la información que aparece de la sala se recogerá directamente de la base de datos, mostrando así su nombre, tipo de sala, capacidad, etc…

La siguiente pestaña que tendremos, nos presentará la lista de las empresas que hayan sido creadas por los socios de nuestra aplicación.

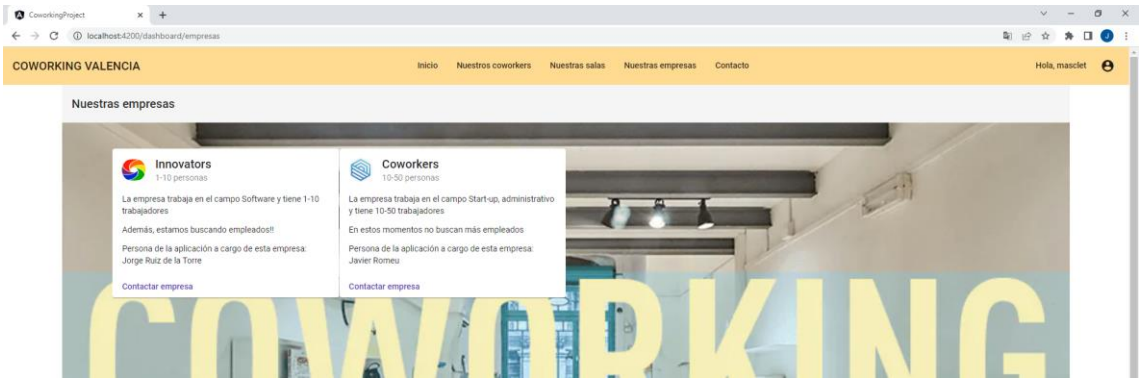


Imagen 38: Interfaz de presentación de las empresas

En este componente, se han utilizado recursos similares a las pestañas anteriores: cartas de presentación sobre cuadrícula, que contienen información relativa a lo que queremos presentar. Esto facilitará la lectura de la página, haciendo que las interfaces en las distintas páginas sean similares.

El usuario podrá tener una empresa asociada, estando la empresa anunciada en esta pestaña y, además, apareciendo algo de información del administrador de ésta, como el nombre y apellidos, para que sea más fácil de reconocer.

También hemos considerado que esta funcionalidad apoya a que se cree una comunidad entorno a nuestra página web de *coworking*, ya que vas a poder interactuar con ella y dar a conocer tu propia empresa.

La siguiente opción será la de contacto, y es que, si tenemos cualquier tipo de problema, nos gustaría ponernos en contacto con algún administrador.

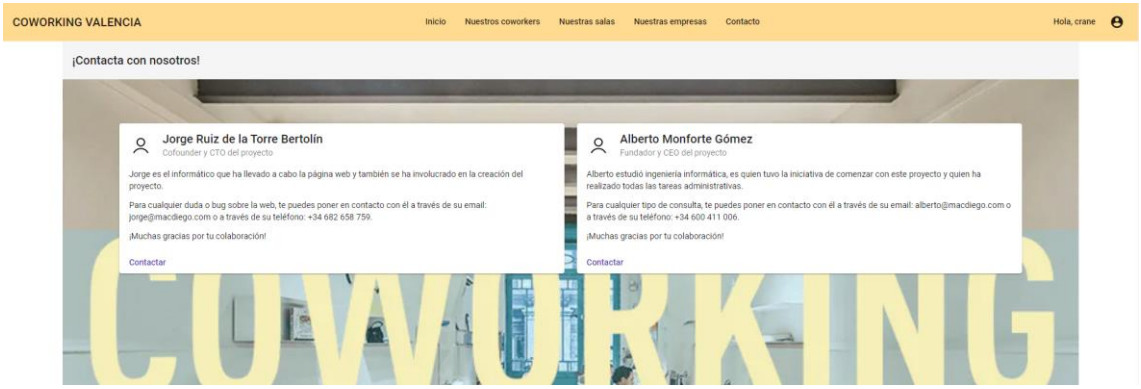


Imagen 39: Interfaz web de contacto

Siempre hay usuarios que prefieren hablar con una persona presencial para cualquier duda o incluso avisarnos de cualquier error *(bug)* que hayan podido sufrir. Es por eso por lo que, en la ventana de contacto, hemos creado dos tarjetas planas, es decir, la información no está extraída de la base de datos, sino que se trata de texto plano. En estas tarjetas, facilitamos nuestros números de teléfono y nuestros emails para los usuarios.

Finalmente, si no estamos con la sesión iniciada, veremos el botón de *login*, el cual nos redirigirá a la página de *login* que hemos visto anteriormente. En caso de haber iniciado sesión, tendremos en su lugar un desplegable con distintas opciones.

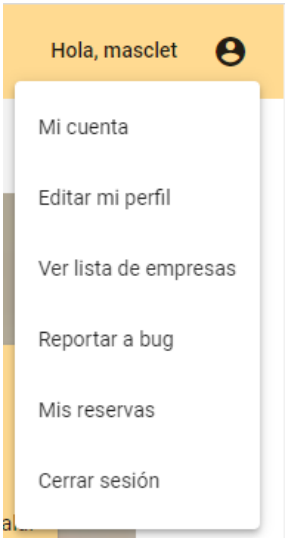


Imagen 40: Menú de ajustes para tu perfil.

En este menú, vamos a poder acceder a las distintas ventanas, entre las cuales vamos a destacar: “Mi cuenta”, donde llegaremos a la ventana que hemos enseñado en el apartado anterior, con toda la configuración de nuestra cuenta y la opción de editarla, y la ventana de “Mis reservas”, donde podremos ver una lista de todas las reservas.

Imagen 41: Pestaña de ‘Mi perfil’ cuando somos socios

Imagen 42: Pestaña de ‘Mi perfil’ sin ser socio

En esta ventana tendremos toda la información relativa con nuestro perfil. Podremos editarlo y darnos de baja. En caso de que estemos dentro del plan de socios, tendremos más funcionalidades, relativas con la creación y asociación de empresas, que, como vimos en los requisitos funcionales, estas opciones solo estarán disponibles para los socios. También tendremos un botón para gestionar nuestra reserva que veremos con detalle más adelante.

Vamos a ver la interfaz que se mostrará cuando pulsemos sobre ‘Mis reservas’:

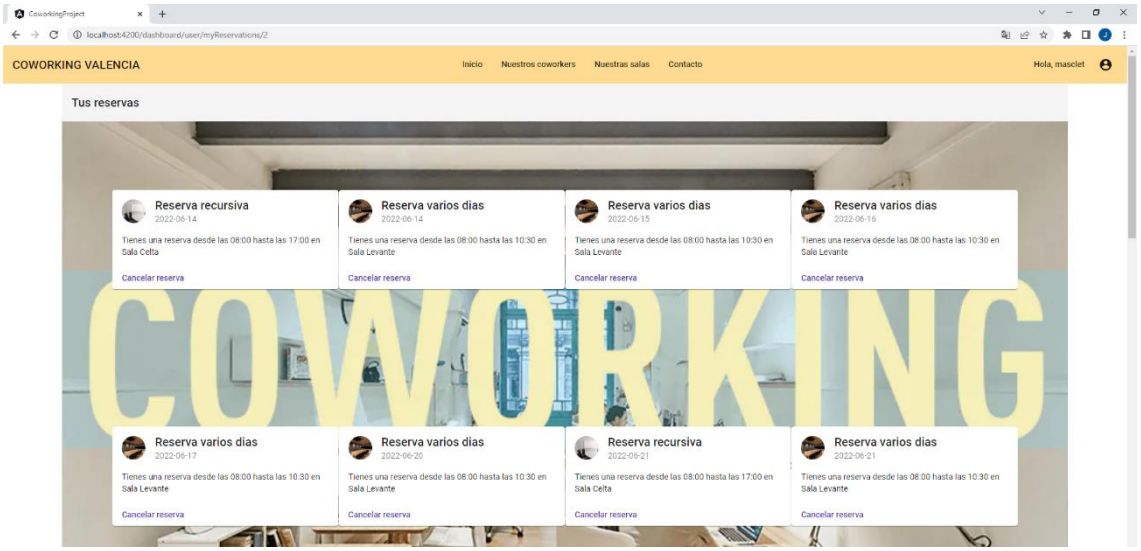


Imagen 43: Ventana de mis reservas

En esta ventana, vamos a tener todas las reservas que tenemos pendientes, utilizando el mismo modelo de tarjetas sobre una cuadrícula, y las podremos cancelar de una en una. Cuando las cancelemos, se nos devolverá el 80% de su coste. Lo explicaremos más adelante en el apartado de Reserva de salas.

Como hemos mencionado anteriormente, y como podemos ver en la base de datos, las reservas tienen asignado tanto el id del usuario que la realiza como el id de la sala que se está reservando. Gracias a esto, podemos ordenar las reservas para que a cada usuario le salgan las suyas sin que haya conflictos. En la misma tarjeta de la reserva, podemos ver que el nombre de la sala sobre la que hemos realizado la reserva está presente. En nuestra llamada a la base de datos para obtener la información relativa a nuestras reservas, hemos añadido la premisa de que estas se ordenen por fecha, con lo cual, las que aparezcan primero serán las más cercanas a la fecha actual.

8.3 Reserva de Salas

En este apartado, vamos a ver cómo se realizan las reservas y los pagos relacionados con estas. Cuando nos vamos al apartado de las salas, si ingresamos en el botón de consultar horarios, nos va a salir la ventana siguiente:

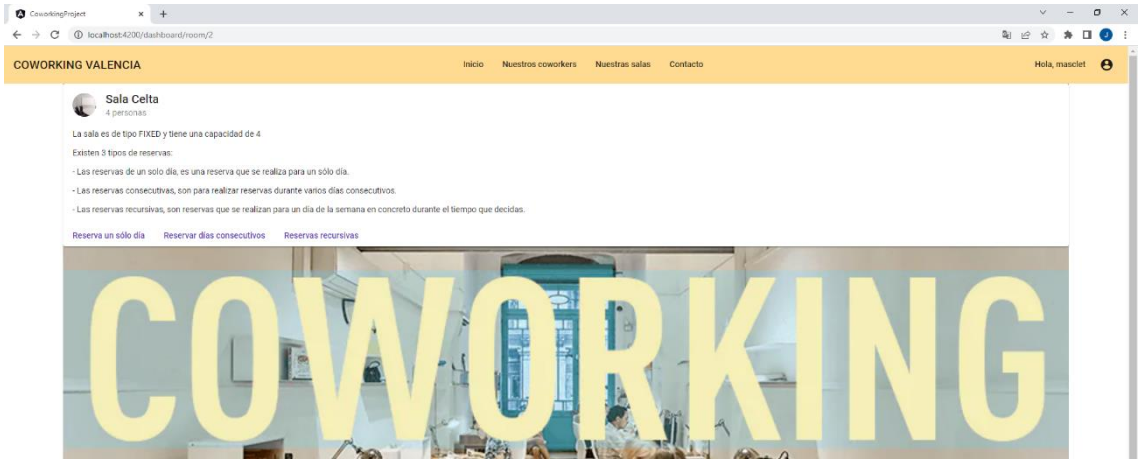


Imagen 44: Tarjeta de una sala.

Podemos ver que la *url* de la sala contiene su número identificador de la base de datos, y que toda la información relativa de la sala la saca de este mismo objeto. Existen 3 tipos de reserva, así que hemos escrito un pequeño texto para explicar a qué se refiere cada uno, y después, tenemos 3 botones correspondientes a cada uno de los tipos.

8.3.1 Reservar un solo día

La reserva de un solo día es simple: Reservar un solo día en un horario. Esto es lo que vemos cuando accedemos a las ventanas de realizar reservas un solo día:

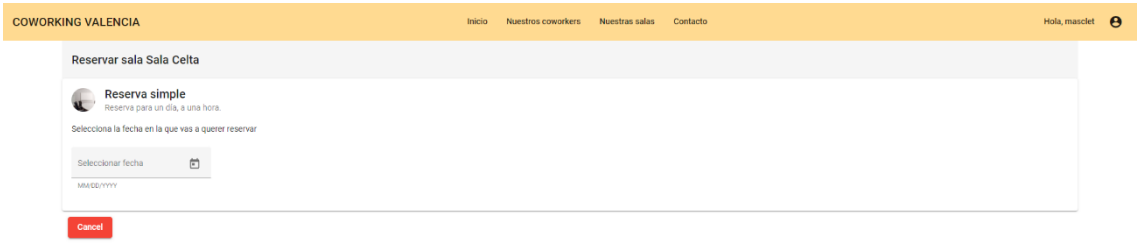


Imagen 45: Realizar reserva de un solo día

Primero, nos saldrá un formulario para indicar la fecha en la que nos interesaría realizar la reserva, una vez seleccionamos la fecha, lo que vemos es lo siguiente:

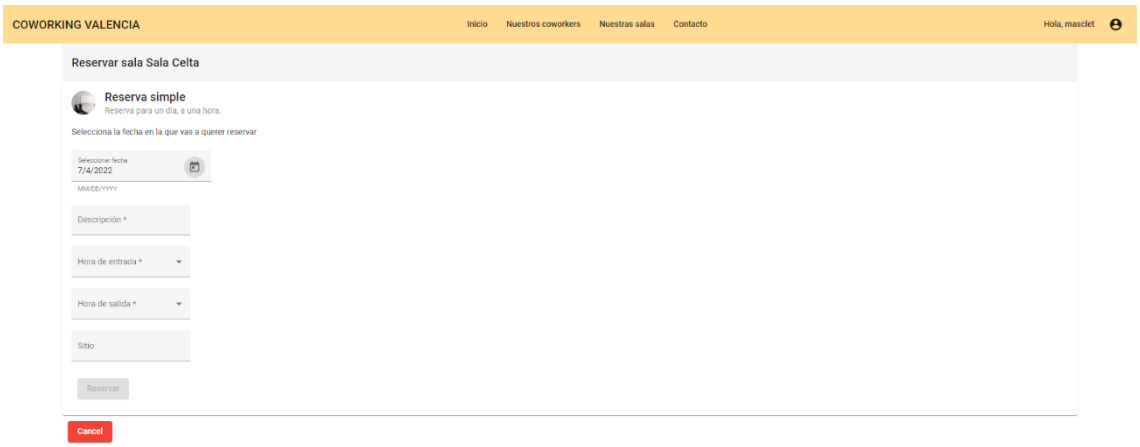


Imagen 46: Realizar reserva de un solo día, formulario completo.

Aparece todo el formulario para realizar la reserva. Esto se debe a que, para calcular las horas que están disponibles en dicha sala, tiene que saber qué día vamos a querer realizar la reserva, que producirá una serie de llamadas de nuestro *front end* sobre nuestro *back end*. Gracias a estas llamadas, solo nos aparecerán los horarios disponibles de la reserva ese mismo día. Vamos a acotar los horarios de 8:00h a 20:00h provisionalmente, ya que estos van a ser los horarios en los que nuestras oficinas de *coworking* van a estar disponibles y abiertas al público.

El funcionamiento para obtener las horas libres y ocupadas de cada sala para cada día consiste, en la parte lógica de nuestro proyecto, hemos creado un bucle que va avanzando cada 30 minutos, y va comprobando si esa hora ya está comprendida en alguna de las reservas existentes de ese día y en esa misma sala, en caso de no estarlo, lo añade a una lista de horarios, que será la lista que se presentará en esta casilla.

Cuando rellenemos el formulario de reserva de sala, y nos dispongamos a realizar la reserva, pulsaremos el botón de “Reservar”, lo que nos abrirá una ventana que nos preguntará cómo queremos realizar el pago:

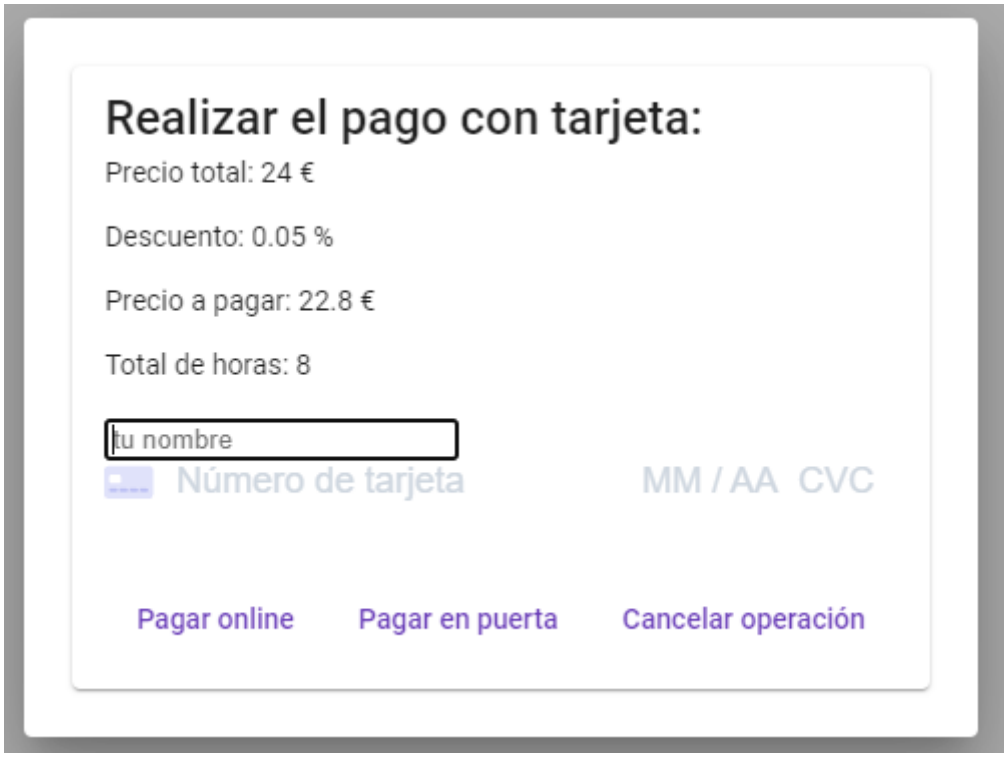


Imagen 47: Ventana para realizar los pagos de la reserva.

En esta ventana, a modo de tarjeta, podemos ver toda la información relevante de la factura. El precio total, el descuento que estamos aplicando, el precio total a pagar y las horas totales que estamos reservando. Tenemos varias opciones: Pagar *online*, pagar en puerta o cancelar la operación. Si cancelamos la operación, no ocurrirá nada, se cerrará la pestaña y podremos seguir navegando con total normalidad sobre la aplicación. En el caso de pagar en puerta, esto generará una factura en nuestra base de datos, indicando al secretario de la puerta que la factura se ha de pagar en el local. En el caso de pagar *online*, tendremos una breve pantalla de carga acompañada de un mensaje que nos confirmará el pago.

Vamos ahora a ver el comportamiento cuando seleccionamos la opción de pagar *online*.

Imagen 48: Tarjeta no válida cuando vamos a pagar.

La biblioteca de Stripe nos facilita un validador para la tarjeta a la hora de proceder al pago. Y es que, si detecta que el número de tarjeta no es válido, el número resaltará en rojo, y cuando intentemos darle a la opción de pagar *online*, nos saltará un mensaje indicándonos que este no es válido. Ocurrirá lo mismo si ponemos una fecha de caducidad de la tarjeta anterior a la fecha actual o un código CVC incorrecto.

Stripe, además, tiene opacidad para sacar el número de tarjeta, por lo que, aunque quisiéramos, sería imposible sacar tanto este como la información relativa si se rellena en los campos de Stripe, con lo cual no almacenaremos la tarjeta de nuestros usuarios en ningún punto de nuestra aplicación y los pagos se realizarán de forma segura.

Como ya hemos visto, en el archivo de application.properties de nuestro back end, hemos sincronizado nuestro proyecto con nuestra cuenta de Stripe, con lo cual, en cuanto realicemos un pago *online*, se sincronizará directamente esta.

Si entramos en la web de Stripe y entramos a nuestros pagos, podremos ver todos los pagos de reservas que se han realizado correctamente. En esta web solo aparecerán los pagos que se hayan realizado de forma *online*, es decir, aquellos que hayan decidido pagar en la puerta, no aparecerán en Stripe, pero sí en nuestra base de datos.

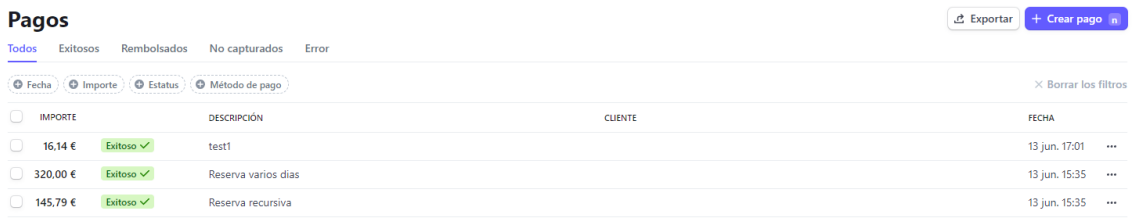


Imagen 49: Resumen de los pagos de Stripe desde la página web de Stripe.

Tendremos, dentro de la página web, un resumen de los pagos de Stripe en los que nos indicará si el pago se ha realizado, cancelado, e incluso reembolsado.

8.3.2 Reserva recursiva

Para las reservas recursivas, el formulario será similar, aunque necesitaremos algún dato extra.

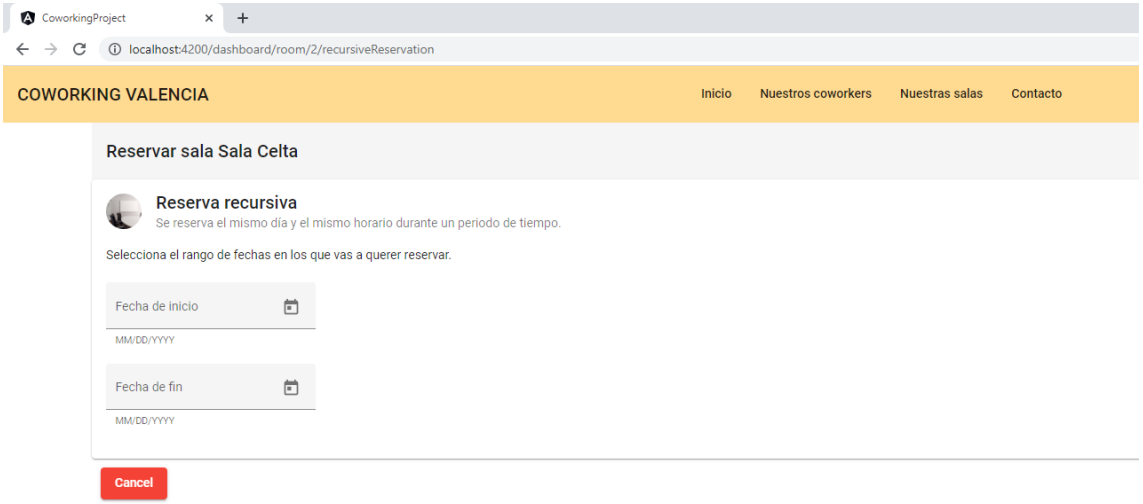


Imagen 50: Interfaz de las reservas recursivas.

Empezamos viendo solo dos campos del formulario, que nos piden la fecha de inicio y la fecha final en las que se va a comprender nuestra reserva recursiva. Una vez rellenados estos datos, al igual que con la reserva normal, nos aparecerán todos los campos del formulario.

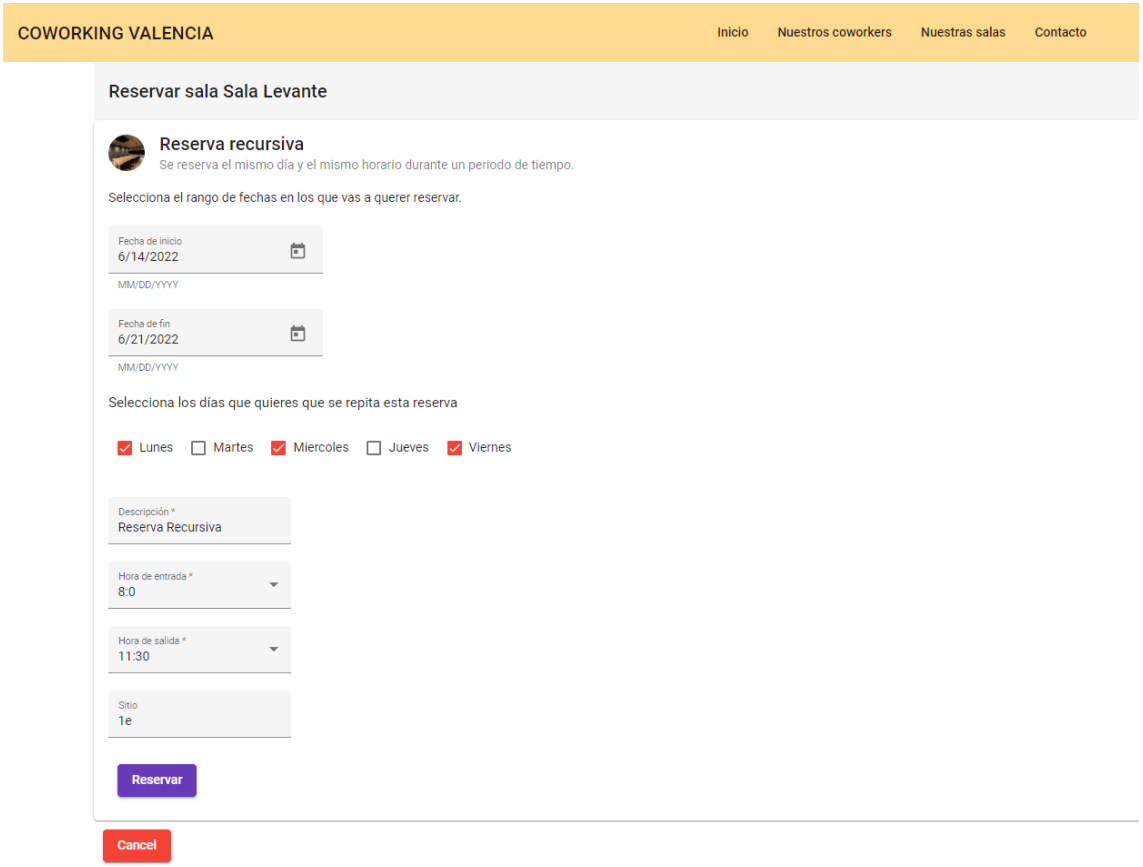


Imagen 51: Interfaz de la reserva recursiva con fecha seleccionada.

En este caso, podemos ver que el formulario es algo más completo, ya que, aparte de tener que seleccionar 2 fechas, tenemos que seleccionar los días de la semana en los que queremos que se repita esta reserva recursiva. Los desarrollos relativos a este tipo de reserva los hemos descrito en el apartado **7.3.4 Creación de los pagos y facturas**.

8.3.3 Devolución al cancelar reserva

Hemos aplicado un método de devolución sincronizado con Stripe, el cual devuelve el dinero en un plazo de 5 a 10 días, en caso de que, en una reserva recursiva, puedes eliminar las reservas a las que no vas a poder asistir, y este te devolverá el 80% del coste de esta.

Y es que, con una llamada a Stripe, le indicamos el montante a devolver. Esta cantidad la calcularemos a partir del precio total de la factura, lo dividimos entre el número total de reservas que tiene, y nos sacará el precio que ha costado esa reserva aislada, independientemente de las demás. Finalmente, multiplicaremos este precio por 0,8 lo que nos dará el importe total que se devolverá.

Imagen 52: Reembolso de un pago en Stripe.

En este caso, de una reserva recursiva con un importe total de 364€, hemos cancelado dos reservas, con lo que en Stripe se va a actualizar y nos va a realizar una devolución de 84,59€.

8.4 Suscripciones

Para las suscripciones, como hemos comentado antes, nos redirigirá a una ventana ajena a nuestro proyecto, lo que crea una sesión con la que nos vamos a comunicar a partir de los eventos del servidor de Stripe. Y es que, Stripe está preparado para este tipo de aplicaciones, con lo cual, ofrece por defecto algunas interfaces de usuario con las que puedes interactuar, sin necesidad de programarlas por código.

En la página de inicio, una vez pulsemos en el botón para suscribirnos, nos aparecerá esta otra pantalla:

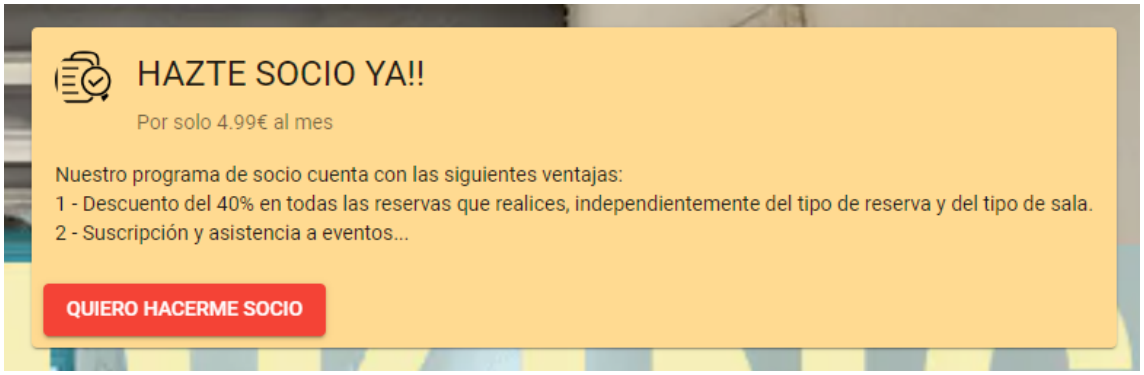


Imagen 53: Tarjeta para hacernos socios en el panel de control

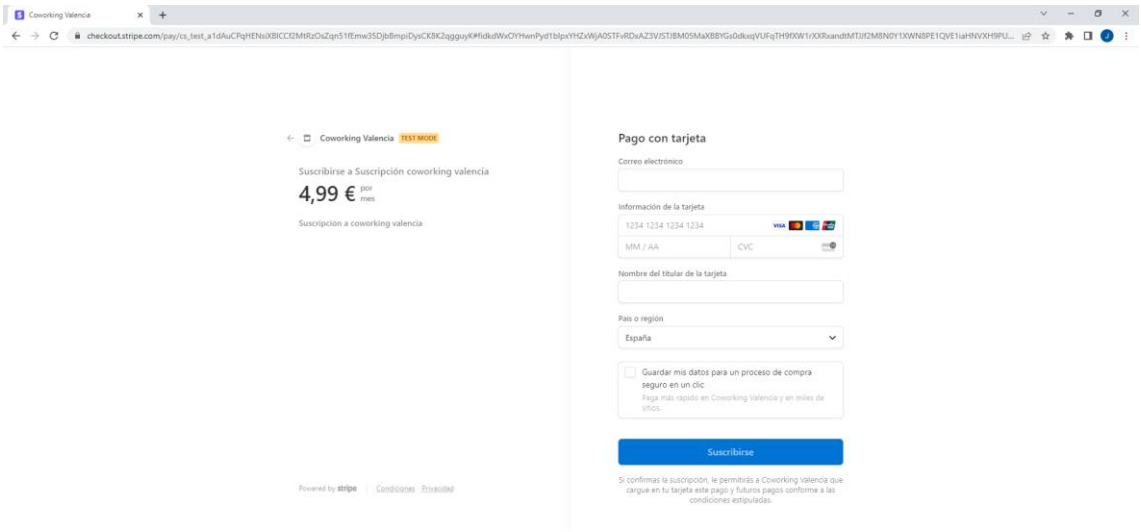


Imagen 54: Interfaz para realizar las suscripciones.

Esta ventana, que como hemos comentado, nos la ha ofrecido Stripe, tiene verificaciones para la tarjeta, es decir que no podremos poner una tarjeta falsa. En caso de que haya algún problema con el montante del pago, nos lo notificará en el sistema con un evento y tendremos la capacidad de reaccionar al respecto.

Una vez rellenados los datos, cuando le demos al botón de suscribir, nos devolverá a nuestra página web, y una vez procesado el pago, se almacenará en la base de datos el usuario con todos los atributos relativos a la suscripción actualizados.

Una vez seamos socios y tengamos todas las ventajas que se nos ofrece, en el apartado de nuestro perfil, aparecerá un botón para gestionar las suscripciones.

Imagen 55: Botón de gestionar suscripción en nuestro perfil.

Pulsando este botón, nos llevará, al igual que antes, a un portal donde podremos ver nuestra suscripción, suscripciones anteriores, y, en caso de desearlo, podremos cancelarla, y se nos devolverá el porcentaje del importe correspondiente al tiempo que ha pasado mientras hemos estado suscritos. Este cálculo lo realiza Stripe automáticamente.

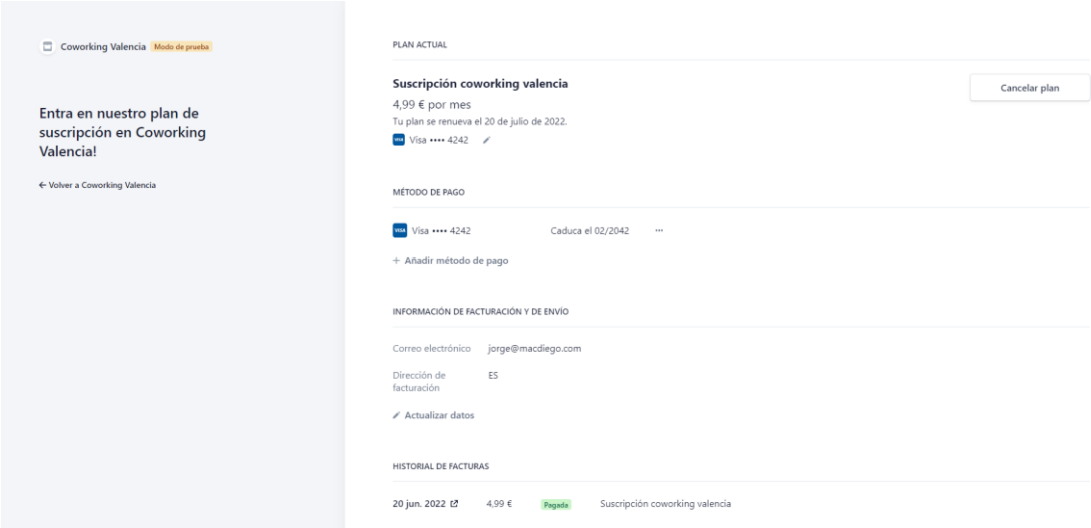


Imagen 56: Interfaz de gestión de nuestras suscripciones.

Como podemos ver en esta interfaz, tendremos la opción de cancelar el plan arriba a la derecha.

8.5 Modo administrador

El modo administrador es una página extra en la que solo los que tengan el rol de administrador podrán acceder. El objetivo de este modo es que esté disponible para los administradores y la persona encargada que se encuentre de manera presencial en las oficinas de *coworking*.

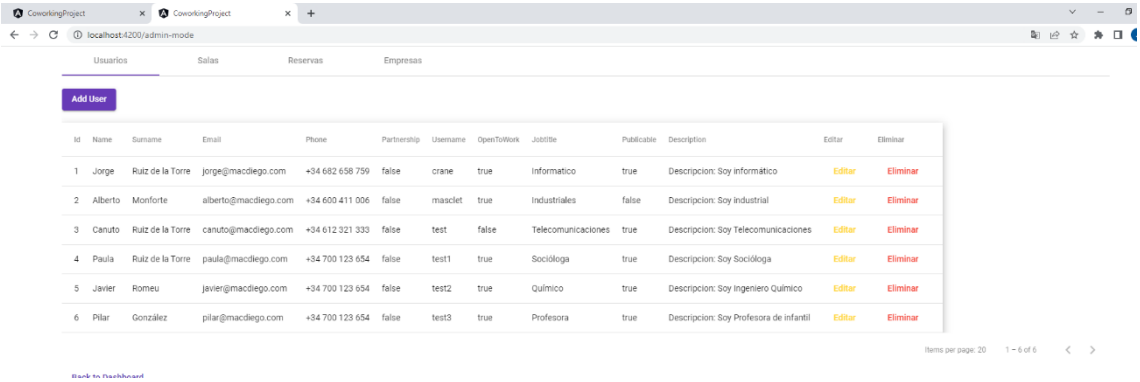


Imagen 57: Interfaz de usuario del modo administrador.

Como vemos, la interfaz es minimalista. Vemos una lista con todos los usuarios, otra con todas las salas, las reservas y una última para todas las empresas. Dentro de cada una, podremos o bien editar la información de cada una de las filas, o bien eliminarlo.

9. Conclusiones y trabajos futuros

9.1 Trabajos futuros

Creemos que todavía quedan ciertos trabajos futuros y mejoras sobre nuestro programa que no hemos podido realizar por falta de conocimientos y/o tiempo. Deberíamos dedicar algo de trabajo en solucionar ciertos errores que han ido apareciendo con los desarrollos. Que suelen ser menores, pero podrían causar ciertos conflictos.

También sería ideal realizar los *tests* de *JUnit* pertinentes a nuestros métodos de nuestro código *Java*, para, en las futuras ampliaciones de nuestro codigo, asegurarnos que los cambios no alterasen los resultados, y para reducir la deuda técnica.

De los requisitos iniciales que se presentan por parte del cliente, podríamos trabajar en aplicar un chat dentro de nuestra web entre los usuarios que sean socios, y además, poder contactar tanto con los que lo permitan como con las empresas directamente. Al igual que la propuesta del calendario donde los usuarios puedan compartir un enlace con la disponibilidad que tienen esa semana, que sería una aplicación extra de nuestro proyecto.

Otra mejora sería, a la hora de registrar imágenes, que estas se registrasen en *base64*, y se tradujesen a un servidor de almacenamiento, que nos provisione de enlaces *url* para cada una. Un ejemplo de un servidor que podría ayudarnos con estas funciones es el Fuga Cloud.

Otra mejora para los usuarios sería fraccionar los pagos, de alguna forma, generar varias facturas o dividir la factura.

Por último, crear algún buscador dentro de nuestra página de modo administrador, para que sea más fácil dar con el valor deseado.

9.2 Conclusión

El proyecto actual se ha realizado con el propósito de facilitar a los usuarios de *coworking* que reserven sus espacios sin necesidad de ponerse en contacto con personas físicas. Además de esto, permite tener una página web personalizable e innovadora con funcionalidades que hacen resaltar nuestra página web frente a las de nuestros competidores.

La facilidad para obtener el servicio que se ofrece en nuestro proyecto frente a la de nuestros competidores nos podría situar en la cabeza del mercado. Hemos creado un principio de una red social con la que la gente va a poder involucrarse.

Hemos adaptado nuestro sistema para que abarque distintas necesidades, ya sean las personas que necesitan de un espacio de trabajo a diario, o para las personas que van a necesitar de un espacio de trabajo de forma urgente, para aquellas pequeñas *start ups* o autónomos que requieran de una sala de reuniones para reunirse con sus clientes, y que todos estos puedan hacerlo de forma online y segura.

El desarrollo del proyecto ha sido un éxito, ya que hemos conseguido añadir todas las funcionalidades básicas que nos pidió el cliente, y tenemos claro cómo realizar los trabajos futuros, además que está acondicionado para la realización de estos. La metodología AGILE que hemos utilizado ha sido un factor clave para la realización del proyecto, ya que ha permitido que, gracias al contacto con el cliente, hemos podido realizarlo a su medida.

Creo que gracias a nuestra aplicación y las facilidades que nos ofrece, podremos ampliar el mercado, centrándonos en una clientela *online* y en una clientela que podría ser más pasajera, o que incluso, después de probarlo, puedan darse cuenta de las ventajas que te pueden ofrecer los espacios de *coworking*.

Del lado de la empresa, esto nos permitirá generar ganancias, no solo con el sistema de reservas, que sería la fuente tradicional de ingresos de cualquier local de *coworking*, sino que además con un sistema de suscripciones.

A parte, otra de nuestras intenciones era dar con un público más casual que el que suelen tener este tipo de instalaciones. Mediante la agilización del sistema de reservas, creemos que tenemos la oportunidad de llegar a este público más cómodamente.

Referencias

1. Economipedia. Coworking. https://economipedia.com/definiciones/coworking.html – Accedido por última vez el 14 de junio de 2022
2. Como nació el coworking. Oficina24. https://www.oficina24.es/noticias/como-nacio-el-coworking/#:~:text=La%20historia%20de%20los%20espacios,juntos%2C%20pero%20de%20forma%20independiente – Accedido por última vez el 14 de junio de 2022
3. Orígenes y antecedentes de las bases de datos. Data Prix. https://www.dataprix.com/es/mineria-datos-aplicada-encuesta-permanente-hogares/24-origenes-y-antecedentes-bases-datos#:~:text=El%20t%C3%A9rmino%20Base%20de%20Datos,un%20simposio%20celebrado%20en%20California.&text=Edgar%20Frank%20Codd%20defini%C3%B3%20el,las%20bases%20de%20datos%20relacionales. – Accedido por última vez el 14 de junio de 2022
4. Página web oficial del JWT token. JSON Web Tokens. https://jwt.io/ - Accedido por última vez el 20 de junio de 2022
5. Securizar un API REST utilizando JSON Web Tokens. https://www.adictosaltrabajo.com/2017/09/25/securizar-un-api-rest-utilizando-json-web-tokens/ - Accedido por última vez el 28 de marzo de 2022
6. Spring Boot Full Stack with Angular. Canal de youtube de Amigoscode. https://www.youtube.com/watch?v=Gx4iBLKLVHk&ab_channel=Amigoscode – Accedido por última vez el 28 de marzo de 2022
7. Manual para comandos MySQL. MySQL Commands. http://g2pc1.bu.edu/~qzpeng/manual/MySQL%20Commands.htm – Accedido por última vez el 29 de marzo de 2022
8. Iconos de las fuentes de Google. Icons Google fonts. https://fonts.google.com/icons?selected=Material+Icons – Accedido por última vez el 12 de Abril
9. Angular Material Tutorial. Canal de youtube de Tomas Ruiz Diaz. https://www.youtube.com/watch?v=rWOwTVSMfPw&ab_channel=TomasRuizDiaz – Accedido por última vez el 26 de mayo de 2022
10. Documentación de Angular. Angular guide. https://angular.io/guide - Accedido por última vez el 1 de junio de 2022
11. JSON Web Token with Spring security and Angular. Canal de youtube de Get Arrays. https://www.youtube.com/watch?v=FMGQsW_B9Rs&t=242s&ab_channel=GetArrays – Accedido por última vez el 4 de abril de 2022

12. Documentación de Spring. Spring guides. https://spring.io/guides - Accedido por última vez el 30 de mayo de 2022
13. Implementación de pagos con Stripe, Spring-Boot y Angular 8. Canal de youtube de LuigiCode. https://www.youtube.com/watch?v=3_saElBbmbg&ab_channel=LuigiCode – Accedido por última vez el 31 de mayo de 2022
14. Documentación de Stripe. Stripe Docs. https://stripe.com/docs - Accedido por última vez el 20 de junio de 2022
15. Baeldung. Guías de Spring. https://www.baeldung.com/ - Accedido por última vez el 15 de junio de 2022
16. Stackoverflow. https://stackoverflow.com/ - Accedido por última vez el 17 de junio de 2022.
17. Head First Design Patterns – Eric Freeman & Elisabeth Freeman
18. Kanban: una breve introducción. https://www.atlassian.com/es/agile/kanban - Accedido por última vez el 10 de junio de 2022
19. Github. https://github.com/ - Accedido por última vez el 20 de junio de 2022
20. Error 404: ¿Preparados para un mundo sin Internet? – Esther Paniagua

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**			X	
ODS 4. **Educación de calidad.**			X	
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**	X			

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Hemos relacionado nuestro TFG con algunos de los ODS presentados. Dentro de todos los objetivos propuestos, creemos que nuestro proyecto va a estar relacionado con 4 de ellos. Vamos a repasar todos los objetivos relacionados con nuestro proyecto y dar una explicación del porqué del grado de relación con estos.

El primer ODS en el que puede estar relacionado nuestro TFG es el ODS 3. Salud y bienestar. Nuestro trabajo consiste en proveer unas instalaciones para trabajo colaborativo y montar un grupo social, pues pensamos que, dentro de la salud y bienestar, y sobre todo después de lo vivido con la pandemia mundial del Covid-19, es muy importante la salud psicológica. Creemos que el teletrabajo es un gran avance para el que decida hacerlo, y es cómodo y eficaz, pero puede desgastar psicológicamente que tu puesto de trabajo se encuentre en tu hogar, y en caso de tener jornadas de trabajo exhaustivas, no poder salir de su hogar es, para mucha gente, algo agotador a nivel mental. Proponiendo otro sistema de trabajo, que puede ser independiente de tu empresa y que invita a mantener relaciones sociales, esperamos poder ayudar a nuestros usuarios a mantener su salud en términos sociales. Es por esto por lo que hemos concluido con que este ODS está relacionado con nuestro proyecto, marcándolo como nivel Medio.

El segundo ODS que hemos relacionado con un grado bajo a nuestro sistema es el ODS 4 Educación de calidad. Esto se debe a que, al tener la oportunidad de reservar salas de forma online, podría utilizarse para distintos tipos de formaciones que se pudieran realizar, ya sea por parte de empresas o entidades públicas. Al facilitar los espacios al público y fomentar que se utilicen estos espacios de forma grupal, también fomentamos que la gente se forme en los campos que desee. Teniendo, además, la oportunidad de trabajar junto con distintas empresas de distintos ámbitos.

El ODS 13 Acción por el clima, esta también relacionado en bajo nivel, ya que, al ofrecer instalaciones compartidas, los gatos de electricidad, luz, etcétera serían compartidos. Buscaremos siempre que nuestras instalaciones sean lo más respetuosas con el medioambiente posibles. Creemos que unas instalaciones de espacio compartido donde se comparten a su vez los gastos domésticos puede favorecer tanto al usuario, por los ahorros de coste de sus hogares, como al coste total energético.

Por último, y el que consideramos más importante de nuestro proyecto, hemos indicado un grado de relación alto para el ODS 17 Alianzas para lograr objetivos. Es uno de los principales objetivos que tenemos con el proyecto de coworking: Que se forme una comunidad y que los usuarios puedan colaborar y progresar entre ellos. Como hemos comentado, más allá de un espacio de trabajo, queremos posicionarnos como un club o lugar social. Los espacios de coworking suelen estar acondicionados para acoger a

pequeñas empresas tecnológicas que están comenzando y, a falta de un local, prefieren subcontratar espacios donde poder reunirse. También se utiliza para aquellas personas que son autónomas y que prefieren tener un espacio de trabajo. Es por esto por lo que pensamos que nuestro proyecto concuerda perfectamente con este objetivo.

ETS Enginyeria Informàtica
Camí de Vera, s/n. 46022. València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es