



MÓDULO PROYECTO

CFGS Desarrollo de Aplicaciones
Multiplataforma
Informática y Comunicaciones

OFICINA VIRTUAL DE GESTIÓN ADMINISTRATIVA

Tutor individual: <el que corresponda>

Tutor colectivo: <el que corresponda>

Año: 2024

Fecha de presentación: 19/12/2024

Nombre y Apellidos: Manuel Lázaro Velasco
Email: manulazaro2000@gmail.com

**Foto actual
del alumno**

2 Organización de la memoria

1	2
2 Organización de la memoria	2
3 Descripción general del proyecto	3
3.1 Objetivos	3
3.2 Cuestiones metodológicas	3
3.3 Entorno de trabajo (tecnologías de desarrollo y herramientas)	5
4 Descripción general del producto	7
3.1 Visión general del sistema: límites del sistema, funcionalidades básicas, usuarios y/o otros sistemas con los que pueda interactuar.	7
3.3 Despliegue de la aplicación indicando plataforma tecnológica, instalación de la aplicación y puesta en marcha	9
5 Planificación y presupuesto	10
6 Documentación Técnica: análisis, diseño, implementación y pruebas.	12
6.1 Especificación de requisitos	12
6.2 Análisis del sistema	12
Gestión de Usuarios	12
Gestión de Clientes dentro de la interfaz de empleado	13
Gestión de Pagos	14
Gestión de Órdenes de Reparación	14
Visualización de Datos (Tablas y Gráficas)	15
6.3 Diseño del sistema:	16
6.3.1 Diseño de la Base de Datos	17
6.3.2 Diseño de la Interfaz de usuario.	18
6.4 Implementación:	19
6.4.1 Entorno de desarrollo.	19
6.4.2 Estructura del código.	19
6.4.3 Cuestiones de diseño e implementación reseñables.	21
Configuración de CORS	21
Programación Reactiva	22
6.5 Pruebas.Control de errores en el login:	23
7 Manuales de usuario	25
7.1 Manual de usuario	25
7.2 Manual de instalación	28
Requisitos básicos para la instalación	28
Proceso a seguir para instalar la aplicación	29

8 Conclusiones y posibles ampliaciones	29
Posibles ampliaciones	30
9 Bibliografía	31
Bibliografía	31
Angular y Desarrollo Frontend	31
JavaScript y Desarrollo Web	32
Spring y Desarrollo Backend	32
Fuentes y Estilización	32
Otros recursos útiles	32
10 Anexos	33

3 Descripción general del proyecto

3.1 Objetivos

El objetivo principal de este proyecto es diseñar y desarrollar un núcleo básico para una plataforma de gestión y administración destinada a cualquier centro deportivo. Esta plataforma busca ser fácilmente escalable, con una estructura limpia y comprensible, preparada para futuras ampliaciones y un desarrollo más avanzado.

Además, con este proyecto tenía la intención de ampliar y desarrollar el trabajo previamente realizado en la asignatura de *Empresa e iniciativa emprendedora* donde creaba una aplicación con las mismas bases que esta.

Como objetivos secundarios, me propuse aprender a utilizar Angular como nuevo framework y TypeScript como nuevo lenguaje de programación. También quise trabajar con un backend y un frontend separados de manera eficiente y organizada.

3.2 Cuestiones metodológicas

Para llevar a cabo este proyecto, he seguido una metodología de desarrollo basada en el modelo clásico de ciclo de vida en cascada.

El modelo en cascada es un enfoque que sigue pasos ordenados y secuenciales, como análisis, diseño, desarrollo y pruebas, completando cada fase antes de pasar a la siguiente.

Análisis:

En esta fase inicial, me enfoqué en repasar mi trabajo de *Empresa* y seleccionar qué partes serán más fáciles e interesantes de implementar. Analice todas las funcionalidades que podría añadir y priorice las más interesantes y visuales para empezar por ellas y asegurarme que me da tiempo a terminarlas, después pensé en otras muchas funcionalidades, herramientas o retoques pequeños para ir añadiendo a medida que viese el tiempo que me quede.

Diseño:

Tras escoger la vista de administrador como la más interesante para desarrollar me puse crear esquemas sobre la base de datos y la interfaz de la aplicación. Una vez que tenía una idea clara de cómo quería que fuese el proyecto dividi todo en dos partes:

- **FrontEnd:** Cree pequeños bocetos para definir cómo sería visualmente la aplicación y fui definiendo lo que haría cada módulo y como estaría dividido. En un principio creé 8 módulos y los dividí en 2 contenedores, el izquierdo con tablas de datos y el derecho con gráficas.
- **BackEnd:** Tras tener la base de datos bien creada me centre en desarrollar la estructura del backend, primero cree todas las entidades de mi proyecto, después cree todos los controladores con cada endpoint que podría usar en un futuro, luego cree el esquema que seguiría la lógica dentro de los servicios, delimitados con interfaces y por último cree varias Querys en mi repositorio para comprobar que todo funcionaba correctamente.

Implementación:

En esta etapa, me centré mucho más en el desarrollo de lógica y módulos funcionales:

- **frontend**, desarrolle componentes como tablas gráficas y el calendario rellenandolos con los datos pasados desde el backend.

Además fui haciendo pequeños cambios en el css hasta que obtuvo una imagen mucho mas seria y cuidada.

- **backend:** Configuré la seguridad básica, como la autenticación de usuarios y desarrolle la lógica para seleccionar la información idónea con la que poner rellenar las tablas y gráficas.

Pruebas:

En esta etapa, empecé a realizar pruebas, primero sobre el control de errores y avisos para el usuario, después me centré en comprobar la correcta funcionalidad de todo y por último la seguridad del login.

Aceptación:

El proceso de aceptación lo lleve a cabo mediante una serie de pruebas funcionales y de rendimiento, asegurándome de que la aplicación fuera fluida y sin fallos para el usuario final.

Accesibilidad:

Probe la aplicación en diferentes navegadores (Chrome, Firefox, Edge y Opera).

Control de errores: Verifique que el sistema manejara correctamente los errores y mostrará mensajes claros al usuario en caso de fallos.

Pruebas de rendimiento:

Realicen pruebas de carga y estrés (dentro de mis límites) para evaluar el rendimiento de las llamadas http.

Valide que las gráficas y tablas cargarán los datos necesarios en el menor tiempo posible.

3.3 Entorno de trabajo (tecnologías de desarrollo y herramientas)

Técnica de Desarrollo:

Para este proyecto, utilicé un modelo cliente-servidor, donde la aplicación se desarrolla con una arquitectura de aplicación web con un servidor remoto. En este enfoque, el frontend se encarga de interactuar con el usuario, mientras que el **backend** procesa las solicitudes HTTP y hace las operaciones CRUD necesarias. Esta separación permite una mayor flexibilidad, escalabilidad y mantenimiento de la aplicación.

Frontend: El cliente está desarrollado con **Angular**, un framework basado en TypeScript, que me permite crear aplicaciones web de una sola página (SPA) con interfaces dinámicas.

Backend: El servidor está desarrollado en **Spring Boot**, un framework para el desarrollo de aplicaciones Java, que proporciona una plataforma robusta y escalable para la creación de APIs RESTful.

Herramientas y tecnologías utilizadas:

Frontend:

- **Angular:** Utilicé Angular para desarrollar la parte frontend de la aplicación. Angular me permite construir aplicaciones web rápidas y dinámicas. Utilicé TypeScript como lenguaje principal. Y utilice la **Versión:** 18.0.0.
 - **HTML5 y CSS3:** Utilicé estos lenguajes para estructurar y dar estilo a la interfaz de usuario.
 - **Google Fonts:** Para la tipografía y los iconos, usé Google Fonts, que me permitió acceder a una amplia variedad de fuentes e iconos para mejorar fácilmente.

Backend :

- **Spring Boot:** Para el desarrollo del servidor, utilicé Spring Boot, que me permite crear aplicaciones Java de manera rápida y sencilla.
- **Hibernate:** Utilicé Hibernate para interactuar con la base de datos. Hibernate me permitió mapear las entidades a tablas de base de datos, facilitando el trabajo con PostgreSQL y evitando tener que escribir consultas SQL manualmente.

Java: El backend está desarrollado en Java 17, utilizando el JDK correspondiente para garantizar la compatibilidad y rendimiento.

Base de Datos:

PostgreSQL: La base de datos que utilicé para almacenar los datos de la aplicación es PostgreSQL.

Herramientas de desarrollo:

- **Visual Studio Code:** Utilicé Visual Studio Code para el desarrollo del frontend con Angular. Este editor es ligero, rápido y tiene una amplia gama de extensiones que facilitan el desarrollo, como la integración con Git y la depuración en tiempo real.
- **IntelliJ IDEA:** Para el desarrollo del backend en Spring Boot, usé IntelliJ IDEA, un potente IDE que me permitió escribir y gestionar código Java de manera eficiente, con soporte completo para Spring y Hibernate.

- **DBeaver:** Para gestionar la base de datos PostgreSQL, utilicé DBeaver ya que es muy fácil de usar y es la utilice durante mi periodo de prácticas.

GitHub: Usé **Git** como sistema de control de versiones para gestionar el código del proyecto. Git me permitió programar en diferentes equipos al mismo tiempo y tener todo controlado.

Postman: Utilicé Postman para probar los endpoints del backend, asegurándome de que todo funcionara correctamente y devolvieran los resultados esperados.

Chat GPT: Sobre todo utilice esta herramienta para conocer las anotaciones correctas en TypeScript y Angular ya que eran totalmente nuevos para mi y cometía muchos errores gramaticales. Además me ayudó a depurar grandes logs de errores imposibles de entender a simple vista.

4 Descripción general del producto

3.1 Visión general del sistema: límites del sistema, funcionalidades básicas, usuarios y/o otros sistemas con los que pueda interactuar.

El sistema es una aplicación web cliente-servidor que consta de un frontend desarrollado en Angular y un backend desarrollado con Spring Boot. Su propósito es gestionar información relacionada con un centro deportivo, facilitando a los empleados la administración de los datos de clientes, empleados y actividades del centro, a través de una interfaz intuitiva. La aplicación permite realizar diversas operaciones CRUD (crear, leer, actualizar y eliminar) sobre los usuarios, pagos y órdenes de reparación.

La aplicación es autónoma, ya que se ejecuta de manera independiente, sin necesidad de interactuar con nada externo, más allá de la base de datos local que gestiona el backend. El frontend y el backend se comunican de manera local. El sistema permite a los empleados realizar tareas administrativas relacionadas con la gestión de los usuarios y la actividad del centro deportivo, mientras que los clientes pueden acceder a su propia información y realizar pagos.

Como ya he dicho las principales funcionalidades de la aplicación están orientadas a la gestión y administración de datos relacionados con los empleados, clientes y pagos dentro de un centro deportivo:

- **Gestión de usuarios:** Los empleados gestionan las cuentas de los clientes, mientras que la creación de las cuentas de los empleados está limitada a un administrador absoluto.
- **Gestión de clientes:** Los empleados pueden visualizar, modificar y borrar información sobre los clientes del centro deportivo. Los clientes tienen la capacidad de actualizar su propia información o ver sus datos personales.
- **Gestión de pagos:** Los clientes pueden realizar pagos y visualizar su historial de pagos, mientras que los empleados pueden consultar la información de los pagos registrados en la aplicación.
- **Gestión de órdenes de reparación:** Los empleados pueden crear y gestionar órdenes de reparación para las instalaciones del centro deportivo.
- **Visualización de datos:** El sistema permite a los empleados visualizar información mediante **tablas y gráficas** que muestran el tránsito de clientes, pagos realizados, y diferente información interesante.

Existen dos tipos de usuarios en la aplicación:

- **Empleados:** Los empleados tienen la capacidad de gestionar los datos de los clientes, visualizar información sobre pagos y transacciones, y gestionar órdenes de reparación. Estos usuarios son creados por un administrador especial que tiene el control total sobre la creación de cuentas de empleados.
- **Clientes:** Los clientes pueden registrarse e iniciar sesión de manera autónoma. Una vez dentro, pueden ver y editar su propia información personal, realizar pagos y consultar su historial.

Sistemas con los que puede interactuar:

Dado que la aplicación es una aplicación web, puede ser ejecutada en cualquier sistema operativo con un navegador compatible, como Windows, macOS o Linux. La aplicación es multiplataforma, ya que está desarrollada para ser accesible a través de cualquier navegador, sin necesidad de instalación adicional más allá de tener acceso a una red local.

3.2 Descripción breve de métodos, técnicas o arquitecturas(m/t/a) utilizadas.

En cuanto a las técnicas y arquitecturas empleadas, he optado por un modelo cliente-servidor con un backend en Spring Boot y un frontend en Angular. Este enfoque me permite separar la lógica de presentación del procesamiento de datos y proporciona una estructura escalable. Para gestionar la persistencia de datos, se utiliza PostgreSQL.

3.3 Despliegue de la aplicación indicando plataforma tecnológica, instalación de la aplicación y puesta en marcha

La instalación de la aplicación se puede dividir en dos partes: la instalación del backend (Spring Boot) y del frontend (Angular).

Backend (Spring Boot):

Instalar Java 17: La aplicación backend está desarrollada en Java 17, por lo que es necesario tener instalada esta versión de Java. Puede descargarse desde el sitio web oficial de Oracle o usar OpenJDK.

Instalar PostgreSQL: La aplicación utiliza una base de datos PostgreSQL. La instalación de PostgreSQL se puede realizar desde el sitio web oficial.

Clonar el repositorio: Descargar el código fuente del backend desde el repositorio GitHub del proyecto usando el comando:

```
git clone
```

Configurar la base de datos: Crear una base de datos en PostgreSQL y configurar las credenciales en el archivo de configuración application.properties de Spring Boot:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/centro_deportivo
```

```
spring.datasource.username=usuario
```

```
spring.datasource.password=contraseña
```

```
spring.jpa.hibernate.ddl-auto=update
```

Frontend (Angular):

Instalar Node.js: Angular requiere Node.js para ejecutar el servidor de desarrollo. Se puede descargar desde el sitio web oficial o por línea de comandos.

Instalar dependencias: Clona el repositorio del frontend desde GitHub.

Instalar dependencias de Angular: npm install

Iniciar el servidor de desarrollo de Angular: ng serve

Esto arrancará el servidor de desarrollo en el puerto 4200 por defecto

5 Planificación y presupuesto

OCTUBRE

CRONOGRAMA DE UN PROYECTO

TÍTULO DEL PROYECTO

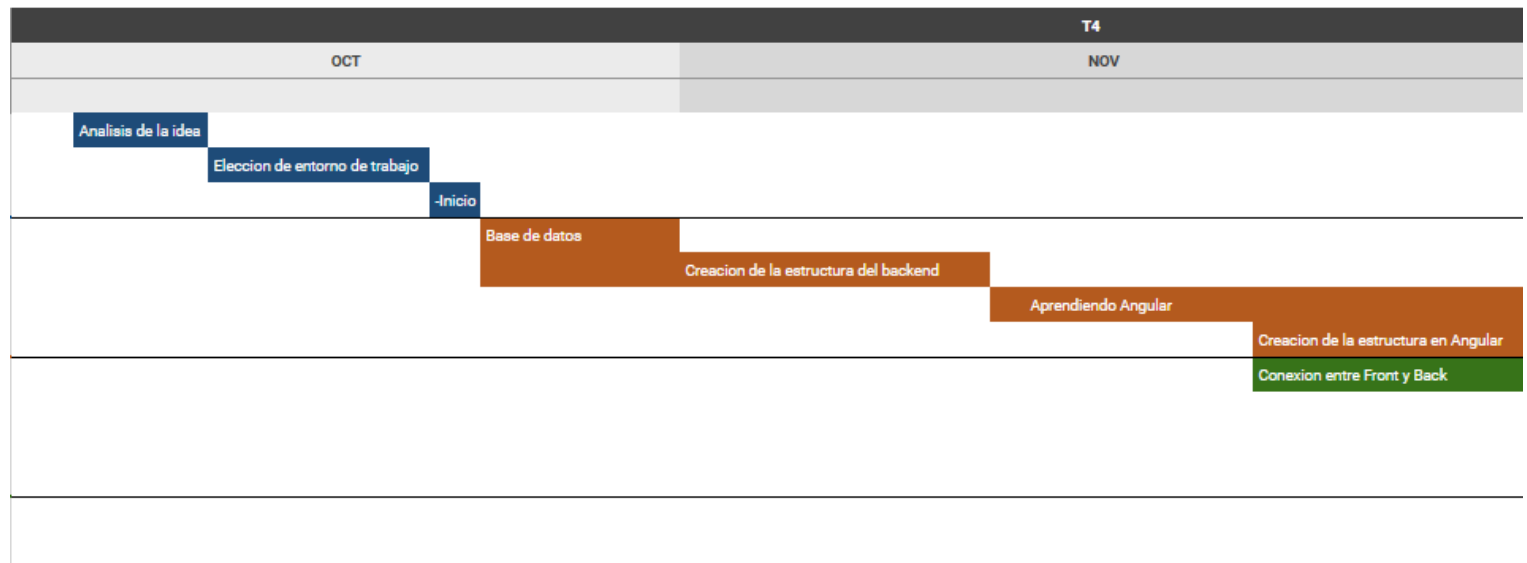
Oficina Virtual

RESPONSABLE DEL PROYECTO

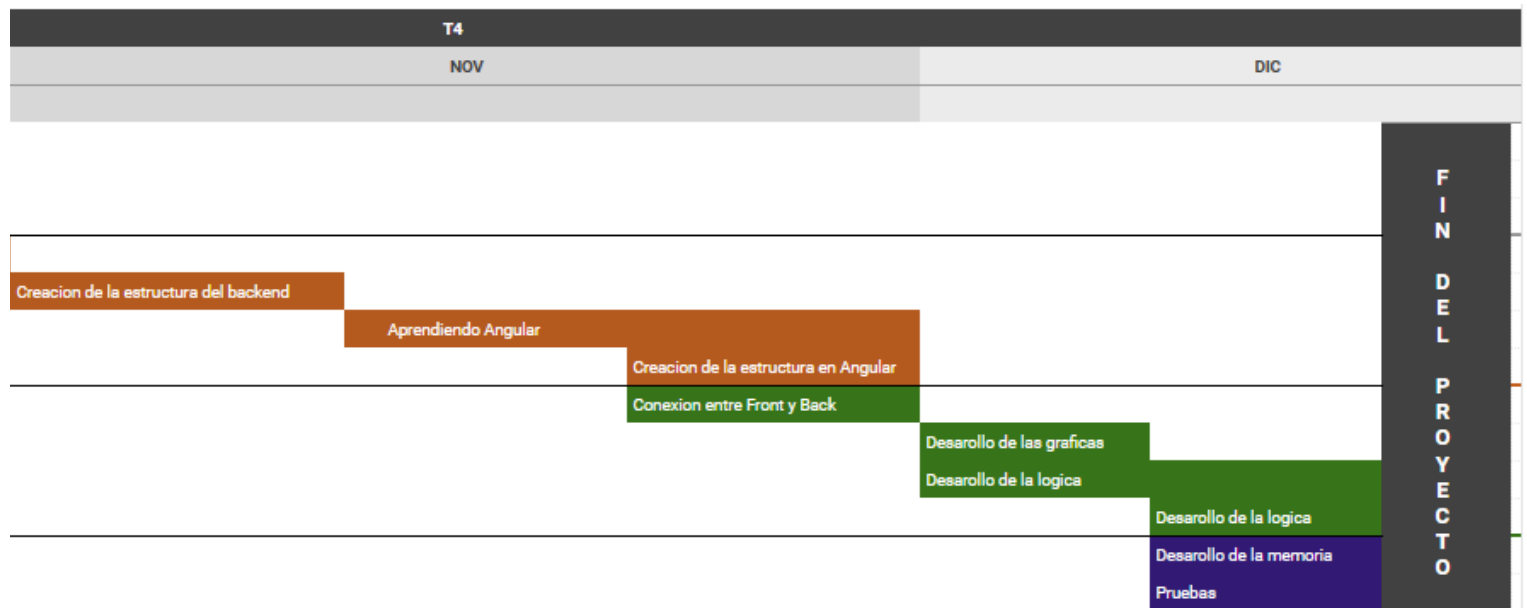
Manuel Lazaro Velasco

FASE	DETALLES	
		OCT
	SEMANA DEL PROYECTO: Introduce la fecha del primer lunes de cada mes -->	
1	Concepción e inicio del proyecto <ul style="list-style-type: none"> - Analisis de la idea - Eleccion de entorno de trabajo y estructura - Inicio 	<div>Analisis de la idea</div> <div>Eleccion de entorno de trabajo</div> <div>-Inicio</div>
3	Estructura del proyecto <ul style="list-style-type: none"> - Creacion de la base de datos - Creacion de la estructura del backend - Aprendiendo Angular - Creacion de la estructura en Angular 	<div>Base de datos</div>
4	Desarrollo <ul style="list-style-type: none"> - Conexion entre Front y Back - Desarrollo de graficas en Angular - Desarrollo de la logica - Control de errores 	
5	Cierre del proyecto <ul style="list-style-type: none"> - Desarrollo de la memoria - Pruebas 	

NOVIEMBRE



DICIEMBRE



Coste de desarrollo del código:

Según me he informado, el coste de la hora de trabajo de un programador junior en España durante 2024 está entre 10 € y 15 € por hora. He calculado las horas invertidas en función del tiempo realmente dedicado al proyecto. A continuación, detallo las horas invertidas en el desarrollo del código:

- Durante unos 5 días de puente que tuvimos, he dedicado unas **8 horas al día**, lo que suma un total de **40 horas**.

- Mientras realizaba las prácticas, intentaba dedicar un rato al llegar a casa, un par de veces a la semana, para no olvidar lo que estaba haciendo. En total, no suman más de **10 horas** en estos días.
- Durante los fines de semana, trabajaba alrededor de unas **10 horas en total** entre viernes, sábado y domingo. Algunos fines de semana me dediqué todo el día y otros solo un rato por la mañana. En total, no suman más de **40 horas** durante todos los fines de semana transcurridos.
- Finalmente, en los últimos días, tras haber terminado mis prácticas, seguí con mi rutina de trabajo dedicando las mañanas y las tardes a programar. En total, durante estos últimos 3 días, trabajé unas **8-10 horas diarias**, lo que suma **no más de 26 horas**.

En total, el número de horas invertidas en el desarrollo del código asciende a **116 horas**. Lo que serían **1160€**.

6 Documentación Técnica: análisis, diseño, implementación y pruebas.

6.1 Especificación de requisitos

6.2 Análisis del sistema

A continuación, especifico detalladamente las funcionalidades que la aplicación debe soportar, basándome en las funcionalidades básicas mencionadas anteriormente.

Gestión de Usuarios

La gestión de usuarios es una de las funcionalidades clave del sistema.

Altas de usuarios (empleados y clientes)

- **Campos obligatorios:**
 - Nombre completo
 - Correo electrónico
 - Contraseña
 - Tipo de suscripción
 - Fecha de cumpleaños
 - Teléfono
 - Método de Pago
- **Condiciones:**
 - El **correo electrónico** debe ser único en el sistema. No pueden existir dos usuarios con el mismo correo.

- Los **clientes** se registran directamente desde la interfaz de la aplicación, sin necesidad de intervención del administrador.

Bajas de clientes

- **Operación:** Se realizará seleccionando al usuario por su **ID** y pidiendo confirmación de la eliminación.
- **Condiciones de validación:**
 - El usuario a eliminar debe ser un **empleado** o **cliente** previamente registrado.
 - El sistema pedirá una confirmación para evitar eliminaciones accidentales.

Modificaciones de usuarios

- **Campos que se pueden modificar:**
 - Para **empleados**: nombre, correo electrónico (si es necesario), rol (solo el administrador puede cambiar el rol de un empleado).
 - Para **clientes**: nombre, correo electrónico, dirección y teléfono.
- **Condiciones de validación:**
 - El **correo electrónico** es modificable, pero debe seguir siendo único.
 - La **contraseña** se puede modificar solo si el usuario actual se autentica correctamente.

Gestión de Clientes dentro de la interfaz de empleado

La gestión de clientes se centra en permitir a los empleados interactuar con la información personal de los clientes.

Altas de clientes

- **Campos obligatorios:**
 - Nombre completo
 - Correo electrónico
 - Contraseña
 - Tipo de suscripción
 - Fecha de cumpleaños
 - Teléfono
 - Método de Pago

Bajas de clientes

- **Operación:** Seleccionando el cliente mediante su **ID** y pidiendo confirmación.

Modificaciones de clientes

- **Campos que se pueden modificar:**

- Nombre, correo electrónico, teléfono y dirección.
- El **correo electrónico** solo puede ser modificado si el nuevo correo no está registrado en la base de datos.

Gestión de Pagos

El sistema permite que los clientes realicen pagos, y los empleados gestionan los registros de estos pagos.

Realización de pagos

- **Campos obligatorios:**
 - Cliente
 - Monto a pagar.
 - Método de pago (por ejemplo, tarjeta, transferencia).
 - Fecha de pago.

Consultas de pagos

- **Operación:** Los empleados pueden consultar los pagos realizados.

Gestión de Órdenes de Reparación

Los empleados tienen la capacidad de crear y gestionar órdenes de reparación para el mantenimiento de las instalaciones del centro deportivo.

Altas de órdenes de reparación

- **Campos obligatorios:**
 - Descripción de la reparación.
 - Fecha de solicitud.
 - Responsable de la reparación.
 - Estado de la orden (pendiente, en proceso, completada).

Bajas de órdenes de reparación

- **Operación:** Al igual que con los usuarios, se selecciona la orden mediante su **ID** y se solicita confirmación para su eliminación.

Modificaciones de órdenes de reparación

- **Campos que se pueden modificar:**
 - Estado de la reparación.
 - Responsable de la reparación.
 - Descripción, si es necesario.

Visualización de Datos (Tablas y Gráficas)

La visualización de datos es muy importante para que los empleados del centro deportivo puedan tomar decisiones informadas.

Tablas de datos

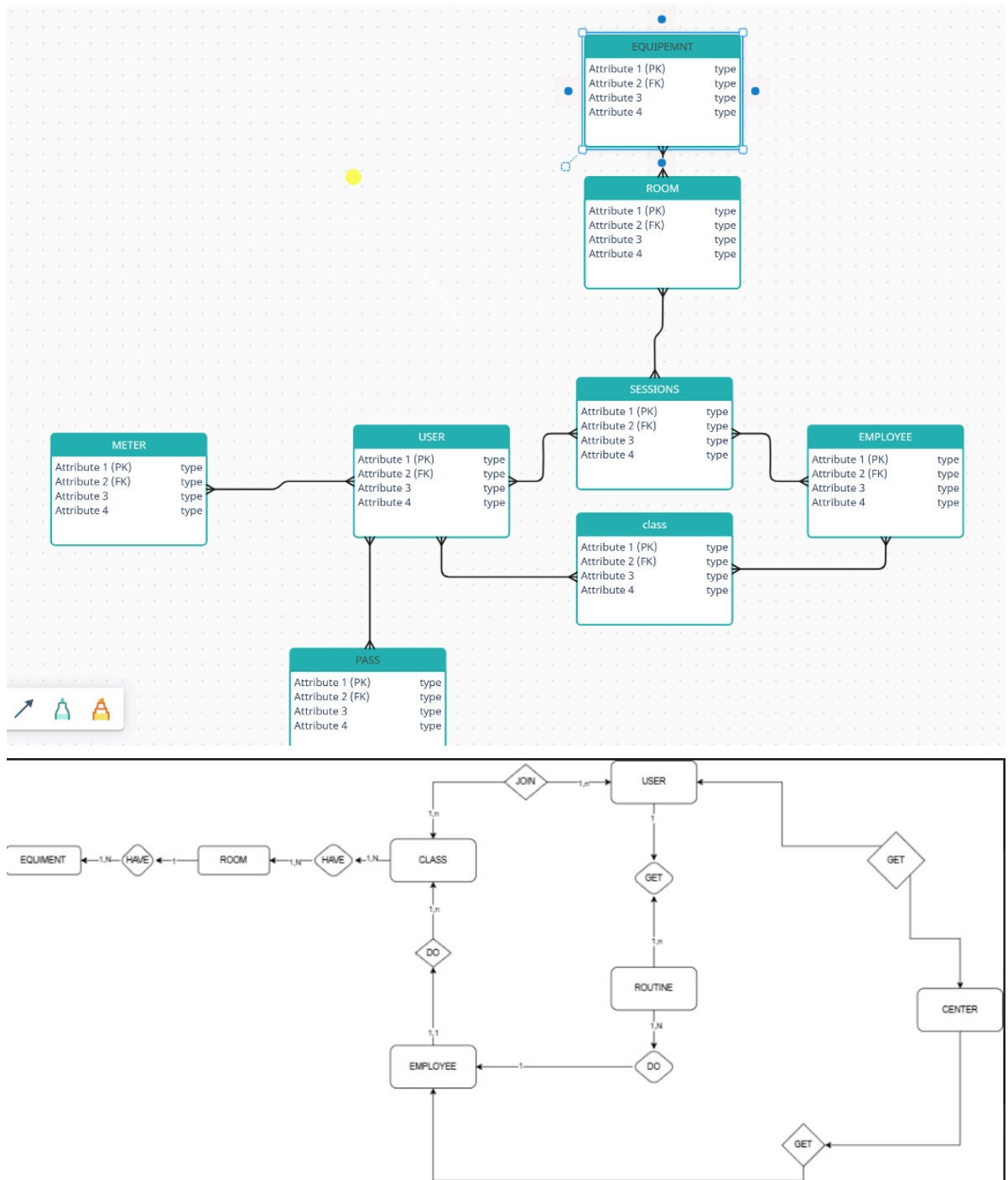
- **Información:** Los empleados pueden ver tablas con la información relacionada con los clientes, pagos y órdenes de reparación.

Gráficas de datos

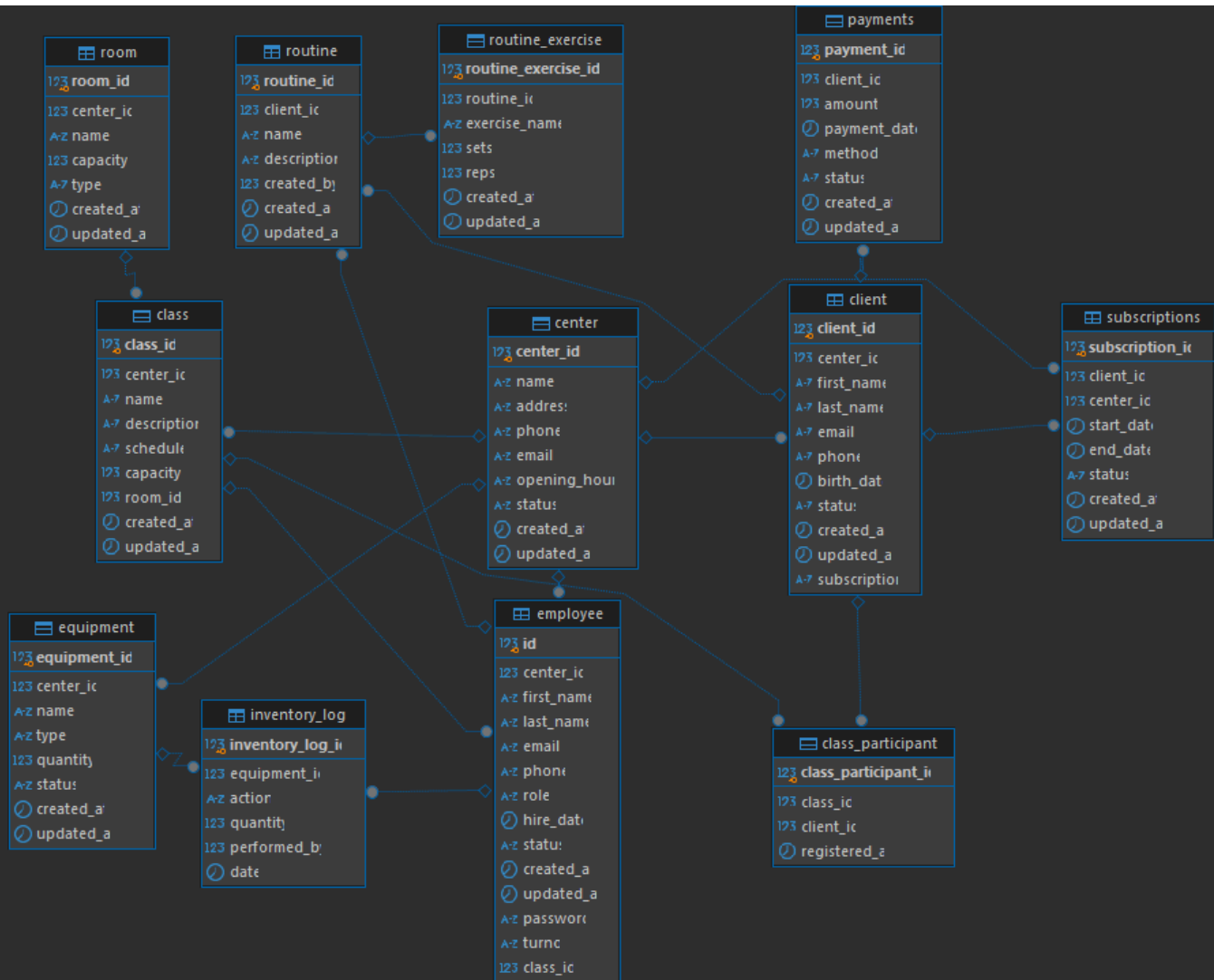
- **Información:** El sistema tiene gráficas dinámicas que muestren:
 - El **tránsito de clientes** a lo largo de los días y horas.
 - La **evolución de los pagos** a lo largo del tiempo.
 - Información de los clientes y empleados.

6.3 Diseño del sistema:

Estos fueron los primeros Diagramas E/R que realice para tener una idea en mi cabeza de cómo sería la base de datos:

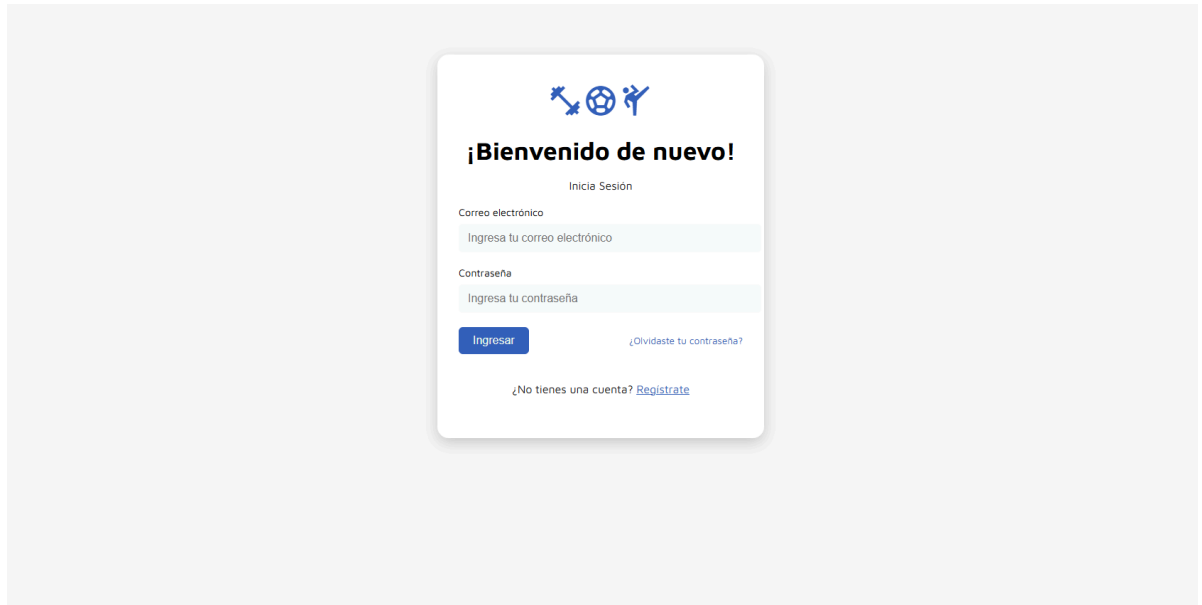


6.3.1 Diseño de la Base de Datos



6.3.2 Diseño de la Interfaz de usuario.

Pantalla de Login:



The login screen features a central white card on a light gray background. At the top of the card is a blue icon representing a key, a soccer ball, and a person. Below the icon is the heading "¡Bienvenido de nuevo!" followed by the subheading "Inicia Sesión". There are two input fields: "Correo electrónico" with the placeholder "Ingresa tu correo electrónico" and "Contraseña" with the placeholder "Ingresa tu contraseña". A blue "Ingresar" button is positioned below the password field. To the right of the button is a link "¿Olvidaste tu contraseña?". Below the button is another link "¿No tienes una cuenta? [Regístrate](#)".

Esta es la pantalla principal que tendrá mi interfaz de usuario:



6.4 Implementación:

6.4.1 Entorno de desarrollo.

Durante el desarrollo de este proyecto, he utilizado un entorno de trabajo compuesto por herramientas que me han permitido abordar tanto el backend como el frontend de forma eficiente. Para el backend, trabajé con **Spring Boot**, junto con **PostgreSQL** como base de datos. En el frontend, utilicé **Angular**.

Entre las herramientas clave que utilicé destacan:

- **IntelliJ IDEA**: Para el desarrollo del backend, ofreciendo integración completa con Spring y Maven.
- **Visual Studio Code**: Como IDE principal para el desarrollo del frontend en Angular.
- **DBeaver**: Herramienta para la gestión de la base de datos PostgreSQL.
- **Postman**: Para probar y depurar las APIs desarrolladas en el backend.
- **Angular CLI**: Para gestionar módulos, componentes y servicios del proyecto Angular.

6.4.2 Estructura del código.

Backend:

El backend sigue una arquitectura **MVC (Modelo-Vista-Controlador)** bien definida, estructurada en los siguientes niveles:

1. **Controladores**: Gestionan las solicitudes HTTP, dejando la lógica al nivel de servicio. Por ejemplo, en el controlador de clientes, implementé endpoints para operaciones CRUD como crear, editar y eliminar clientes.
2. **Servicios**: Encargados de la lógica de negocio, como validar datos, aplicar reglas específicas y coordinar las operaciones con los repositorios.
3. **Repositorios**: Implementados utilizando Spring Data JPA para interactuar directamente con la base de datos PostgreSQL.
4. **Modelos**: Cada modelo mapea las entidades de la base de datos, utilizando anotaciones como `@Entity` y `@Table`.

Frontend

El frontend Angular está organizado de manera modular, promoviendo la separación de preocupaciones y la reutilización de componentes:

Módulos: Cada funcionalidad principal (gestión de usuarios, pagos, clases, etc.) tiene su propio módulo.

Componentes: Representan las vistas y elementos visuales reutilizables, como tablas, formularios y gráficos.

Servicios: Encargados de conectar la API del backend mediante el uso de HttpClient de Angular. Por ejemplo:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ClientService {
  private apiUrl = 'http://localhost:8080/api/clients';

  constructor(private http: HttpClient) {}

  getClients(): Observable<Client[]> {
    return this.http.get<Client[]>(this.apiUrl);
  }

  createClient(client: Client): Observable<Client> {
    return this.http.post<Client>(this.apiUrl, client);
  }
}
```

Librerías utilizadas

Backend:

- Spring Boot starters para web, JPA y validación.
- jjwt para autenticación mediante JWT.
- log4j2 para el registro de logs.
- postgresql como driver de la base de datos.

Frontend:

- @angular/material para componentes visuales modernos.
- ng2-charts y chart.js para generar gráficos.
- rxjs para manejar eventos y programación reactiva.

6.4.3 Cuestiones de diseño e implementación reseñables.

Configuración de CORS

Para permitir la comunicación entre el frontend (Angular) y el backend (Spring Boot), configuré **CORS (Cross-Origin Resource Sharing)**. Ya que el frontend se ejecuta en un servidor local diferente al backend, y sin esta configuración los navegadores bloquearían las solicitudes por razones de seguridad.

En Spring Boot, configuré CORS globalmente mediante una clase de configuración:

```
1 package main.OV.config;
2
3 > import ...
4
5
6
7 new *
8 @Configuration
9 public class GlobalCorsConfig implements WebMvcConfigurer {
10
11     no usages new *
12     @Override
13     public void addCorsMappings(CorsRegistry registry) {
14         registry.addMapping(pathPattern: "/*")
15             .allowedOrigins("http://localhost:4200")
16             .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
17             .allowedHeaders("*")
18             .allowCredentials(true);
19     }
20 }
```

Con esta configuración, habilité el acceso controlado desde el frontend Angular hacia el backend, permitiendo métodos como GET, POST, PUT y DELETE. Además, utilicé

`allowCredentials(true)` para manejar cookies y encabezados de autenticación de manera segura.

Programación Reactiva

Utilizada en el frontend con RxJS, para manejar flujos de datos y eventos asíncronos.

La programación reactiva es básicamente un enfoque donde la aplicación "reacciona" automáticamente a los cambios en los datos o eventos. Lo usé mucho en el proyecto para manejar cosas como actualizaciones en tiempo real o cuando había que traer datos del backend al frontend y mostrar resultados

Por ejemplo, cuando necesitaba mostrar la lista de clientes, usé un "observable". Es como un canal que está escuchando constantemente al backend, y en cuanto llegan los datos, se actualiza la lista en la pantalla sin que yo tenga que hacer nada extra. Algo así:

```
this.clientService.getClients().subscribe((data) => {  
  
  this.clients = data;  
  
});
```

6.5 Pruebas. Control de errores en el login:



¡Bienvenido de nuevo!

Inicia Sesión

Correo electrónico

manulazaro2000@gmail.com

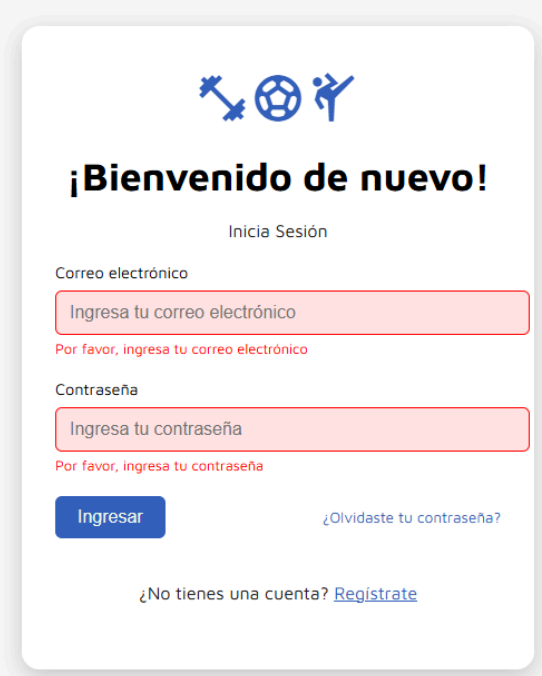
Contraseña

••

Ingresar [¿Olvidaste tu contraseña?](#)

Correo o contraseña incorrecta. Intentelo de nuevo

[¿No tienes una cuenta? Regístrate](#)



¡Bienvenido de nuevo!

Inicia Sesión

Correo electrónico

Ingresa tu correo electrónico

Por favor, ingresa tu correo electrónico

Contraseña

Ingresa tu contraseña

Por favor, ingresa tu contraseña

Ingresar [¿Olvidaste tu contraseña?](#)

[¿No tienes una cuenta? Regístrate](#)



¡Bienvenido de nuevo!

Inicia Sesión

Correo electrónico

manulazaro2000@gmail.com

Contraseña

••

Ingresar

[¿Olvidaste tu contraseña?](#)

Correo o contraseña incorrecta. Intentelo de nuevo

¿No tienes una cuenta? [Regístrate](#)

7 Manuales de usuario

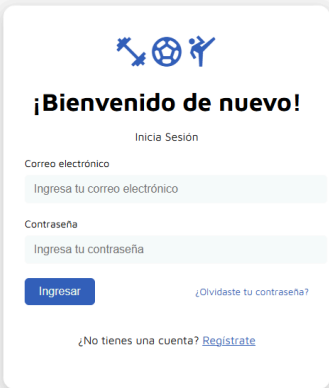
7.1 Manual de usuario

Pantalla de login:

- Las cuentas de empleados son limitas y solo las podría crear el administrador de aplicacion, es decir yo. he creado un default con email: empleado@gmail.com y contraseña: empleado

La base de datos esta en el repositorio.

- Aun asi las cuentas de clientes se podrán crear dando al botón de registrarse, pero la que nos interesa es la pantalla de empleado.



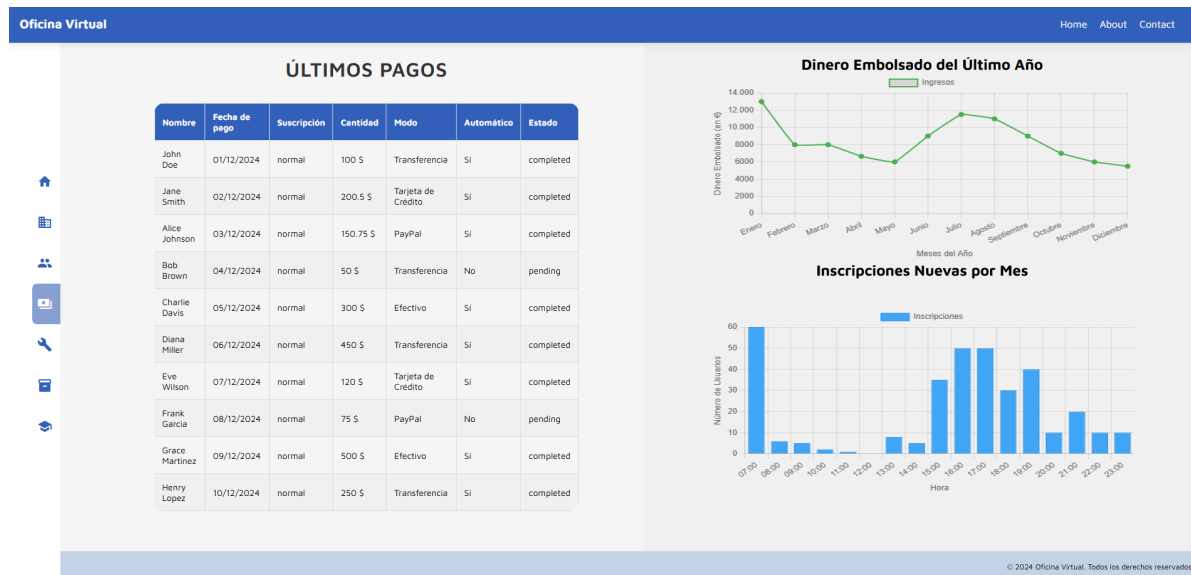
The image shows a login screen for an application. At the top, there is a logo consisting of three stylized figures in blue. Below the logo, the text "¡Bienvenido de nuevo!" is displayed in bold. Underneath, the text "Inicia Sesión" is shown. There are two input fields: "Correo electrónico" with the placeholder "Ingresa tu correo electrónico" and "Contraseña" with the placeholder "Ingresa tu contraseña". Below the password field is a blue button labeled "Ingresar". To the right of the button is a link that says "¿Olvidaste tu contraseña?". At the bottom, there is a link that says "¿No tienes una cuenta? [Regístrate](#)".

Pantalla inicial del empleado con datos de los monitores y de los clientes que se encuentran actualmente dentro del centro:
Además de una barra lateral que nos acompañara durante todos los modulos.



* Con el botón Home haríamos el logout y volveríamos al inicio.

Pantalla de pagos:



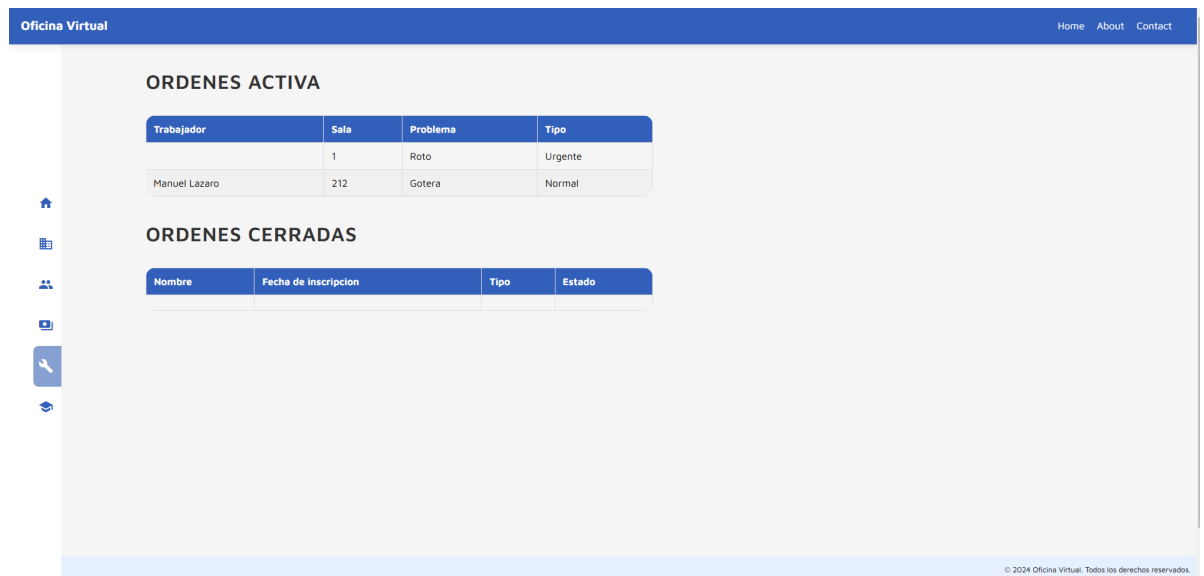
* Con el botón Home haríamos el logout y volveríamos al inicio.

Pantalla con información de los empleados:



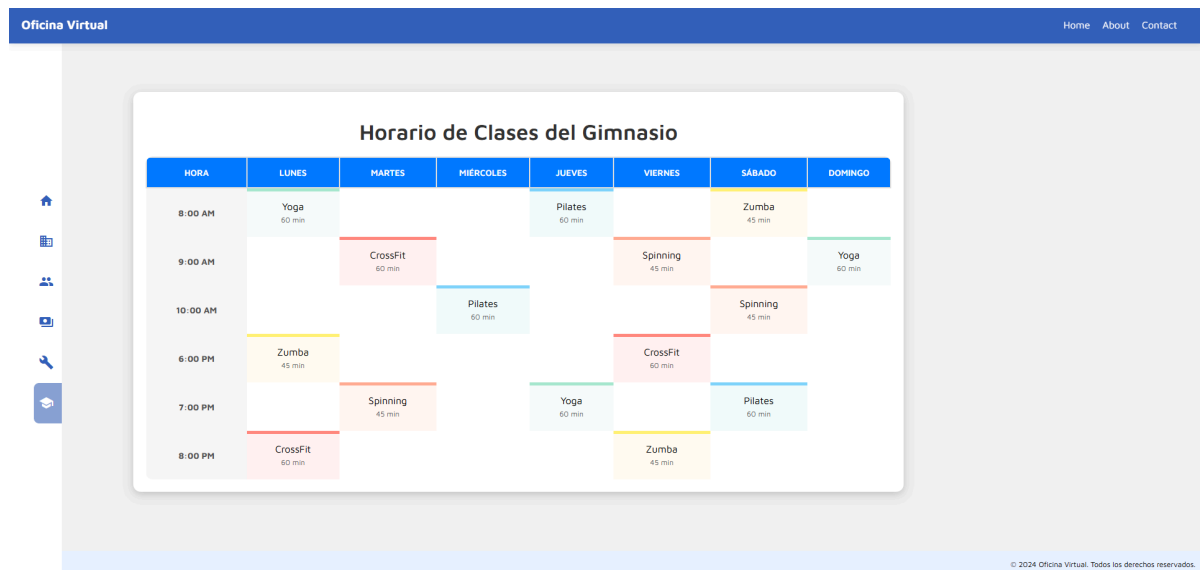
* Con el botón Home haríamos el logout y volveríamos al inicio.

Pantalla con las órdenes de trabajo



* Con el botón Home haríamos el logout y volveríamos al inicio.

Pantalla con el calendario:



* Con el botón Home haríamos el logout y volveríamos al inicio.

7.2 Manual de instalación

Requisitos básicos para la instalación

- **Java** (versión 11 o superior)
 - La aplicación usa Spring, que está basado en Java, por lo que necesitas tener Java instalado. Usar **java -version**
- **Node.js** (versión 14 o superior)
 - Angular necesita Node.js para funcionar. Usar **node -v**
- **Maven**
 - Maven es una herramienta que Spring usa para gestionar dependencias.
- **Git** (opcional pero recomendado)
 - Para clonar el repositorio directamente desde GitHub, necesitarás tener Git instalado. Usar **git clone**.

Proceso a seguir para instalar la aplicación

- Configurar Base de datos desde el fichero properties del Backend
- Levantar el backend
- Levantar el frontend usando **ng serve**
- Ir al puerto de tu navegador que hayas especificado, por defecto angular usa **http://localhost:4200/**

8 Conclusiones y posibles ampliaciones

A lo largo del desarrollo de este proyecto he podido sacar varias conclusiones importantes, especialmente después de enfrentarme al reto de trabajar tanto con un frontend como con un backend, usando lenguajes y frameworks completamente nuevos para mí, como Angular.

- **Análisis del proyecto:**
Una de las principales lecciones que he aprendido es lo fundamental que es sentar bien las bases y la estructura de un proyecto antes de comenzar a desarrollarlo. Este paso puede ahorrarte muchas horas de trabajo y evitar problemas innecesarios en el futuro. Si tuviera que empezar de nuevo, dedicaría más tiempo a estudiar a fondo el funcionamiento de Angular y las herramientas que ofrece, antes de escribir la primera línea de código. Tener una buena comprensión de los fundamentos te permite tomar decisiones más informadas a lo largo del desarrollo y evita que tengas que retroceder más tarde para corregir errores que podrías haber evitado.
- **Gestión del tiempo y errores:**
Mirando atrás, me doy cuenta del tiempo que he perdido trabajando en funcionalidades que finalmente no incluí en el proyecto. Esto ocurrió por diferentes razones, como que no me convencían del todo o porque complicaba el desarrollo innecesariamente. Aprendí que es crucial gestionar bien el tiempo y los recursos, dedicando esfuerzos solo a las funcionalidades que realmente van a aportar valor al proyecto. Hay que evitar invertir tiempo en características que no ofrecen mucho o que van a generar demasiados errores, lo cual puede ser frustrante y ralentiza el progreso.
- **Hibernate:**
Durante mis prácticas en la empresa, pude trabajar con Hibernate y poner en práctica lo que había aprendido sobre esta herramienta. Me di cuenta de que es una herramienta muy potente y que, con más experiencia y conocimientos, podría aprovecharla aún más. Con un dominio más profundo de Hibernate, sería capaz de construir un backend mucho más

robusto y eficiente, optimizando el rendimiento y la escalabilidad del sistema.

- **Arquitectura de la aplicación:**

Cada vez soy más consciente de la importancia de trabajar con una arquitectura bien organizada e independiente. A medida que el proyecto crece, el tener un código limpio y estructurado facilita enormemente el trabajo. El orden y la modularidad se agradecen mucho, ya que te permiten ahorrar tiempo, evitar duplicaciones y, lo más importante, mantener la flexibilidad para futuras modificaciones. Las limitaciones claras entre las diferentes funcionalidades también son clave para evitar problemas a largo plazo y mejorar la mantenibilidad del sistema.

- **Angular:**

Angular es una herramienta increíblemente poderosa y muy diferente a otros frameworks con los que había trabajado anteriormente. Su enfoque en componentes me costó entenderlo al principio, pero con el tiempo he llegado a apreciarlo. Angular tiene una gran cantidad de herramientas y funcionalidades que, aunque requieren una curva de aprendizaje, ofrecen un control mucho mayor sobre el desarrollo de la aplicación. Si bien al principio fue desafiante, el esfuerzo ha valido la pena, ya que ahora puedo gestionar la interfaz de usuario de una forma mucho más ordenada y eficiente.

- **Versiones:**

Al trabajar con diferentes versiones de herramientas y lenguajes, me encontré con los problemas típicos de las dependencias. Una de las cosas que me quedó clara es la diferencia abismal que existe entre versiones antiguas y nuevas, tanto de Angular como de Java. Por ejemplo, las diferencias entre AngularJS y Angular V18 son enormes, y lo mismo ocurre con Java 8 y Java 17. Es impresionante cómo los lenguajes y frameworks evolucionan a lo largo del tiempo, y cómo esas actualizaciones pueden afectar el desarrollo de la aplicación. Este desafío me ha permitido comprender mejor la importancia de mantenerse al día con las nuevas versiones y adaptarse rápidamente a los cambios.

Posibles ampliaciones

Las posibles ampliaciones de la aplicación son muchas, ya que de momento solo he creado una vista funcional, que es la vista de Empleado. Aún me queda por desarrollar la vista de **Administrador** y la de Usuario, pero me centraré en las posibles ampliaciones que puedo realizar en la vista de Empleado para seguir mejorando la funcionalidad.

- **Control de empleado:**

Una mejora importante podría ser la personalización del control de empleados según su tipo. Dependiendo de si el empleado es un monitor, jardinero, administrativo, entre otros, cada uno podría tener diferentes módulos o pantallas con funciones adaptadas a su rol. Además, se podrían agregar notificaciones para que cada tipo de empleado reciba alertas relacionadas con sus tareas o clases, como recordatorios de eventos, cambios de horario, o tareas pendientes.

- **Creación de nuevos módulos:**

Otro área de expansión podría ser la incorporación de nuevos módulos que mejoren la gestión diaria del empleado. Por ejemplo, podría añadirse un módulo de rutinas, donde cada empleado pueda interactuar con clientes y desarrollar rutinas de entrenamiento. También podría incluirse un calendario laboral para que los empleados puedan visualizar su agenda de manera más clara, o un módulo de gestión de clases, donde los empleados puedan asignar, editar y consultar las clases o tareas que tienen asignadas.

- Además, la aplicación podría integrar un sistema de evaluación del desempeño o un módulo de feedback donde los empleados reciban retroalimentación sobre su rendimiento, lo que sería útil para el crecimiento profesional y la mejora continua.
- Se podrían incluir opciones para realizar seguimientos más detallados de las horas trabajadas, crear informes personalizados, o incluso integrar una función de gestión de pagos o sueldos para los empleados, lo que podría convertir a la aplicación en una herramienta integral de gestión laboral.

En resumen, aunque el proyecto ha avanzado bastante, aún queda mucho por mejorar y añadir. Con el tiempo, se puede ir ampliando la aplicación para adaptarla a las necesidades cambiantes de los empleados o usuarios y hacerla aún más completa y útil.

9 Bibliografía

Bibliografía

Angular y Desarrollo Frontend

- [Angular Official Documentation](#)
Página oficial de Angular con guías completas, tutoriales y documentación detallada sobre el framework.

- [Angular Reactive Forms](#)
Documentación oficial sobre el uso de formularios reactivos en Angular.
- [Angular Material](#)
Conjunto de componentes de interfaz de usuario para Angular, diseñados con Material Design.
- [npm: Angular Calendar](#)
Biblioteca para implementar calendarios interactivos en proyectos Angular.
- [Angular Calendar - Demo y Documentación](#)
Sitio web con ejemplos prácticos y documentación sobre la biblioteca de calendarios para Angular.
- Chart.js for Angular
Biblioteca de gráficos basada en Chart.js, adaptada para proyectos Angular.
- [Highcharts for Angular](#)
Adaptación oficial de Highcharts para Angular, una potente herramienta para crear gráficos dinámicos.

JavaScript y Desarrollo Web

- [JavaScript.info](#)
Completa guía interactiva sobre JavaScript, desde conceptos básicos hasta avanzados.
- [npm](#)
Repositorio oficial para descargar bibliotecas y módulos de JavaScript.
- [Webpack Official Site](#)
Herramienta de empaquetado utilizada frecuentemente con proyectos Angular y otros frameworks.

Spring y Desarrollo Backend

- [Spring Official Documentation](#)
Página oficial de Spring Boot, con guías detalladas sobre configuración, desarrollo y despliegue.
- [Spring Data JPA Reference](#)
Documentación oficial sobre el manejo de bases de datos con Spring Data JPA.
- [Baeldung: Spring Tutorials](#)
Una excelente fuente de tutoriales y ejemplos prácticos sobre el ecosistema Spring.

Fuentes y Estilización

- [Google Fonts](#)
Biblioteca de fuentes gratuitas para usar en proyectos frontend.
- [Aula DIV - Recursos de diseño web](#)
Página de pruebas para html y css.

Otros recursos útiles

- [ChatGPT](#)
Herramienta basada en IA que puede servir para resolver dudas y aprender.

10 Anexos

<https://github.com/ManuLazaro/OficinaVirtual.git>