



Facultade de Informática

UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO  
GRAO EN ENXEÑARÍA INFORMÁTICA  
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

# Aplicación Web para la gestión de una cadena de gimnasios

**Estudiante:** Juan Jeremy Ortiz Ortega

**Dirección:** Carlos Vázquez Regueiro

A Coruña, febreiro de 2020.



*A mi madre y a mi abuelo*



### **Agradecimientos**

En primer lugar, agradecer a Carlos Vázquez Regueiro su infinita paciencia, su confianza y todo el apoyo y ayuda que me ha proporcionado para poder realizar este TFG. Agradecer también a mi madre su sacrificio, gracias a ella ha sido posible estudiar lo que quería y gracias a su apoyo y confianza he logrado llegar a este punto. A mis compañeros de facultad a lo largo de estos años, por todas las veces que pensamos que no llegaríamos al final pero cogíamos fuerzas para seguir adelante.

Muchas gracias a todos.



## **Resumen**

En la actualidad, toda empresa que pretenda ser competitiva en el mercado debe necesariamente adoptar tecnologías que le ayuden a una organización mejor, puesto que el gasto en tiempo y dinero que supone la gestión manual de los citados procesos provocaría una clara desventaja frente a sus competidores.

La mayoría de software comerciales de las actividades de un gimnasio se centran en entrenamientos personales. Este proyecto se presenta como un programa de organización para un gimnasio facilitando una herramienta para gestionar sus empleados, máquinas, clientes, cursos, clases.

Este proyecto se enfoca en la organización de sus actividades, una cadena de gimnasios, posiblemente situados en diferentes localidades, que cuentan con una serie de actividades organizadas en cursos e impartidas en clases, que a su vez cuenta con empleados que imparten dichos cursos mediante clases. También se contempla la gestión de clientes que se inscriben a las clases obteniendo un horario.

El proyecto proporciona las herramientas necesarias para gestionar al personal del gimnasio, desde un responsable de un determinado gimnasio en una localidad, pasando por monitores que imparten los cursos y los posibles cambios que surgan en cada curso y gimnasio. A su vez también permite gestionar las aulas asignadas a cada curso, así como las máquinas y los recursos (p.e. aulas) necesarios para la realización de cada actividad en cada clase.

Es importante destacar que este proyecto se lleva a cabo en forma de aplicación web, lo que permite el acceso a los usuarios desde cualquier dispositivo que tenga un navegador de internet.

## **Abstract**

Nowadays, every company that pretends to be competitive in the market must necessarily adopt technologies that help it to be a better organization, since the expense in time and money that the manual management of the aforementioned processes entails would cause a clear disadvantage compared to its competitors.

This project is presented as an organization program for a gym providing a tool to manage its employees, machines, customers, courses, classes. Most commercial software for gym activities focuses on personal training. This project focuses on the organization in its activities: a chain of gyms located in different locations that have a series of activities organized in courses and taught in classes. At the same time, it has employees who teach these courses

---

through classes, also adding the management of clients that enroll in the classes obtaining a schedule.

The project provides the necessary tools to manage the gym staff, from a person in charge of a certain gym in a locality, through monitors that teach the courses and the possible changes that arise in each course and gym.

It also allow to manage the possible classrooms assigned to each course and the machines and their types for the realization of the activities as well as each realization of the classes.

It is important to note that this project is carried out in the form of a web application, which will allow access to users from any device that has an internet browser.

**Palabras clave:**

- Gestión cursos
- Gestión empleados
- Aplicación web
- Java
- Spring
- Bootstrap

**Keywords:**

- Course management
- Employee Management
- Web application
- Java
- Spring
- Bootstrap



# Índice general

---

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Motivación . . . . .	1
1.2	Objetivos . . . . .	2
1.3	Organización de la memoria . . . . .	2
<b>2</b>	<b>Estado del arte</b>	<b>5</b>
2.1	Alternativas . . . . .	5
2.1.1	Gym community app . . . . .	5
2.1.2	IsMyGym . . . . .	6
2.1.3	Viday . . . . .	6
2.2	Conclusiones . . . . .	7
<b>3</b>	<b>Metodología</b>	<b>9</b>
3.1	Proceso Unificado de Desarrollo de Software . . . . .	9
<b>4</b>	<b>Fundamentos tecnológicos</b>	<b>11</b>
4.1	Tecnologías y herramientas de la parte servidor . . . . .	11
4.2	Tecnologías y herramientas de la parte web . . . . .	12
4.3	Tecnologías y herramientas de la parte web . . . . .	12
<b>5</b>	<b>Análisis de requisitos</b>	<b>13</b>
5.1	Actores . . . . .	13
5.2	Casos de Uso . . . . .	13
5.2.1	Gestión de empleados . . . . .	14
5.2.2	Gestión de gimnasio . . . . .	14
5.2.3	Gestión de Cursos . . . . .	16
5.2.4	Gestión de Clientes . . . . .	17
5.2.5	Gestión de Máquinas . . . . .	17

---

5.3	Casos de uso de cada actor . . . . .	17
5.3.1	Casos de uso del rol Administrador . . . . .	17
5.3.2	Casos de uso del rol Encargado . . . . .	17
5.3.3	Casos de uso del rol Monitor . . . . .	19
5.3.4	Casos de uso del rol Cliente . . . . .	19
<b>6</b>	<b>Planificación y costes</b>	<b>21</b>
6.1	Planificación . . . . .	21
6.1.1	Planificación de las iteraciones . . . . .	21
6.1.2	Seguimiento . . . . .	24
6.2	Coste . . . . .	24
6.2.1	Coste del material . . . . .	25
6.2.2	Coste de los recursos humanos . . . . .	25
6.2.3	Coste final del proyecto . . . . .	25
<b>7</b>	<b>Diseño y Arquitectura del Sistema</b>	<b>27</b>
7.1	Modelo de datos . . . . .	27
7.2	Diseño de la arquitectura del sistema . . . . .	29
<b>8</b>	<b>Implementación</b>	<b>33</b>
8.1	Implementación del servidor . . . . .	33
8.1.1	El patrón DAO . . . . .	36
8.1.2	El patrón DTO . . . . .	37
8.1.3	El patrón <i>Facade</i> . . . . .	37
8.1.4	Controladores del servidor . . . . .	40
8.2	Implementación del cliente . . . . .	40
8.2.1	Elementos y conceptos de AngularJS . . . . .	42
8.2.2	División de AngularJS en dos áreas . . . . .	42
8.2.3	Aplicación de página única (SPA) . . . . .	42
8.2.4	Servicios en AngularJS . . . . .	43
8.2.5	Componentes de la aplicación en AngularJS . . . . .	45
8.3	Interfaz Gráfica para el usuario . . . . .	47
8.3.1	Gestión Empleados . . . . .	47
8.3.2	Gestión Cursos . . . . .	49
8.3.3	Gestión Clientes . . . . .	50
8.3.4	Gestión Máquina . . . . .	54
8.3.5	Gestión Gimnasio . . . . .	54

<b>9 Pruebas</b>	<b>59</b>
9.1 Pruebas unitarias . . . . .	59
9.2 Pruebas de aceptación . . . . .	59
9.3 Pruebas de integración . . . . .	61
9.4 Otras pruebas . . . . .	61
<b>10 Conclusiones y trabajo futuro</b>	<b>63</b>
10.1 Conclusiones . . . . .	63
10.2 Trabajo futuro . . . . .	64
<b>A Material adicional</b>	<b>65</b>
A.1 Instalación y ejecución de la aplicación . . . . .	65
A.1.1 Despliegue del servidor Web . . . . .	65
A.1.2 Instalación Node Package Manager (NPM) . . . . .	65
<b>B Servicios</b>	<b>69</b>
<b>Bibliografía</b>	<b>75</b>



# Índice de figuras

---

2.1	Captura de la página principal de Gym Community . . . . .	5
2.2	Captura de la pagina principal de IsMyGym . . . . .	6
3.1	Proceso Unificado de Desarrollo de Software . . . . .	10
5.1	Las dos jerarquías de actores de nuestra aplicación. . . . .	14
5.2	Casos de uso pertenecientes a los bloques Gestión de Empleados y Gestión de Gimnasios. . . . .	15
5.3	Casos de uso pertenecientes a los bloques Gestión de Cursos, Gestión de Clientes y Gestión de Máquinas. . . . .	16
5.4	Casos de uso pertenecientes al rol Administrador. . . . .	18
5.5	Casos de uso pertenecientes al rol Encargado. . . . .	19
5.6	Casos de uso pertenecientes al rol Monitor. . . . .	20
5.7	Casos de uso pertenecientes al rol Cliente. . . . .	20
6.1	Diagrama de Gantt del proyecto. . . . .	22
6.2	Diagrama de Gantt correspondiente a la iteración 1. . . . .	23
6.3	Diagrama de Gantt correspondiente a la iteración 2. . . . .	23
6.4	Diagrama de Gantt correspondiente a la iteración 3. . . . .	23
6.5	Diagrama de Gantt correspondiente a la iteración 4. . . . .	24
6.6	Diagrama de Gantt correspondiente a la iteración 5. . . . .	24
7.1	Diagrama Entidad-Relación . . . . .	28
7.2	Funcionamiento del patrón MVC. . . . .	29
7.3	Arquitectura general de nuestro sistema. . . . .	31
8.1	Mapeado de las clases que representan las distintas entidades de la aplicación	34
8.2	Diagrama de paquetes de la parte "Entities". . . . .	35
8.3	Diagrama de paquetes de la parte servidor. . . . .	36

8.4	Ejemplo de la interfaz <i>EmpleadoDAO</i> con dos de las búsquedas implementadas.	36
8.5	Ejemplo del patrón DTO en la clase <i>EmpleadoDto</i> .	37
8.6	Diagrama de clases del paquete "dto".	38
8.7	Ejemplo del patrón FACADE del servicio <i>EmpleadoServiceImpl</i> .	39
8.8	Extracto del código controlador <i>UserController</i> .	40
8.9	Extracto del código del controlador <i>HomeController</i> .	41
8.10	Ejemplo de implementación del servicio <i>routing</i> en nuestra aplicación.	43
8.11	Ejemplo de aplicación del servicio <i>http</i> en nuestra aplicación para crear un empleado.	44
8.12	Captura conforme se muestra que el administrador puede crear empleados y visualizar todos los empleados en la tabla.	48
8.13	Captura conforme se muestra que el encargado no puede crear empleados y solo puede visualizar sus datos como empleado.	48
8.14	Captura conforme se muestra que el administrador puede modificar los datos rol y jefe de cada empleado.	49
8.15	Captura conforme se muestra que el administrador tiene acceso a las secciones sobre la gestión Cursos, además visualiza todos los cursos disponibles.	50
8.16	Captura conforme se muestra que el encargado tiene acceso a las secciones sobre la gestión Cursos, pero solo visualiza los cursos disponibles en el gimnasio del que es encargado.	51
8.17	Captura conforme se muestra que el Monitor tiene acceso a las secciones sobre la gestión Cursos pero solo visualiza los cursos que impartirá.	51
8.18	Captura conforme se muestra que el Monitor <b>lorena@app.com</b> tiene acceso a las secciones sobre la gestión clases, pero solo visualiza clases que impartirá.	52
8.19	Captura conforme se muestra que el Admin puede dar de alta a nuevos clientes y visualizar a todos los clientes.	52
8.20	Captura conforme se muestra que el cliente " <b>adriana@app.com</b> " puede visualizar solo sus datos y además los botones para, poder modificar sus datos, poder inscribirse y mostrar los cursos del que está inscrito.	53
8.21	Captura conforme se muestra como el usuario " <b>adriana@app.com</b> " se inscribe a un curso.	53
8.22	Lista de los cursos al que está inscrito el cliente " <b>adriana@app.com</b> ".	54
8.23	Lista de las máquinas para el encargado " <b>jenny@app.com</b> ".	55
8.24	Lista de las máquinas para el encargado " <b>cesar@app.com</b> ".	55
8.25	Captura donde el usuario " <b>jeremy@app.com</b> " da de alta a una localidad.	56
8.26	Captura donde el usuario " <b>jeremy@app.com</b> " da de alta a un responsable del Gimnasio.	57

8.27	Captura donde el usuario "jenny@app.com" muestra las aulas del Gimnasio del que es encargada. . . . .	57
8.28	Captura donde el usuario "cesar@app.com" muestra las aulas del Gimnasio del que es encargado. . . . .	58
9.1	Ejemplo de la interfaz <i>GymDAO</i> con dos de las búsquedas implementadas. . .	60
9.2	Captura de pantalla de una petición GET a la url /gym. . . . .	60
A.1	Ejemplo de como se despliega la aplicación servidor con el comando Maven. .	66
A.2	Ejemplo que explica el <i>package.js</i> de esta aplicación . . . . .	67
A.3	Ejemplo del archivo declarado en la seccion <b>main en el archivo package.js</b> . .	67
A.4	Ejemplo del comando <b>npm start</b> para desplegar la aplicación cliente. . . . .	68
B.1	Diagrama de clases de <i>ActividadService</i> . . . . .	69
B.2	Diagrama de clases de <i>EmpleadoService</i> . . . . .	70
B.3	Diagrama de clases de <i>ClaseService</i> . . . . .	71
B.4	Diagrama de clases de <i>ClienteService</i> . . . . .	71
B.5	Diagrama de clases de <i>CursoService</i> . . . . .	72
B.6	Diagrama de clases de <i>GimnasioService</i> . . . . .	73
B.7	Diagrama de clases de <i>MaquinaService</i> . . . . .	74





# Índice de tablas

---

5.1	Descripción de los casos de uso de Gestión de Empleados. . . . .	14
5.2	Descripción de los casos de uso de Gestión de Gimnasios. . . . .	15
5.3	Descripción de los casos de uso de Gestión de Cursos. . . . .	16
5.4	Descripción de los casos de uso de Gestión de Clases. . . . .	17
5.5	Descripción de los casos de uso de Gestión de Máquina. . . . .	17
6.1	Coste del Material empleado en el proyecto. . . . .	25
6.2	Detalle del coste por tareas del analista-programador. . . . .	25
6.3	Detalle del coste por tareas del jefe de proyecto. . . . .	26
6.4	Detalle del coste del personal humano. . . . .	26



# Introducción

---

**E**N el primer capítulo de la memoria se explicará de forma detallada la motivación de este proyecto y los objetivos a alcanzar, así como la organización de la mismas

## 1.1 Motivación

La informatización de los procesos internos de la gestión de un gimnasio es un factor determinante en el éxito de cualquier empresa. No debería concebirse que a día de hoy existan grandes cadenas de gimnasios que no utilicen sofisticados programas de gestión para controlar sus recursos, pero no ocurre lo mismo con las empresas más pequeñas, debido a que no pueden asumir el coste que les supondría pagar las licencias de dichos programas. Es por esto que deben recurrir a aplicaciones más sencillas que no siempre cubren la totalidad de funcionalidades necesarias.

Este proyecto nace con el objetivo de cubrir esa demanda: pequeñas y medianas cadenas de gimnasios que aspiran a expandirse y además tener una organización detallada de los cursos que pueden impartir y tener un control de ello mediante clases y convertirse en una organización modélica que satisfaga las necesidades tanto de sus empleados y sobre todo para sus clientes.

Un punto importante de este proyecto es que se trata de una aplicación web, con un servidor centralizado donde se almacena toda la información a la que acceden los distintos usuarios, por lo que sería posible la gestión de múltiples sucursales en distintas localizaciones dado que todos estarían accediendo a los mismo datos de una forma organizada, cosa que evita los errores y dificultades técnicas de tener los datos de cada gimnasio en una sucursal físicamente en la misma.

En definitiva, el proyecto se presenta como una alternativa viable a pequeñas y medianas empresas que les permita obtener como resultado final un horario donde poder consultar sus distintas actividades en diferentes sucursales de su gimnasio y poder llevar un mayor control

de ello.

## 1.2 Objetivos

El objetivo principal de este proyecto es el desarrollo de una aplicación web que permita la gestión de los procesos internos de una cadena de gimnasios, de manera sencilla e intuitiva, relativos a:

- **Gestión de Empleados:** este módulo proporcionará las funcionalidades para la gestión de los usuarios, los responsables de cada gimnasio y los responsables de cada actividad tendrán en su mano las herramientas necesarias para gestionar sus actividades y cursos impartidos mediante clases.
- **Gestión de Cursos:** este módulo permitirá a los empleados tener controlado y organizado sus actividades con una fecha inicial y una fecha final para cada curso.
- **Gestión de Gimnasio:** en este módulo se gestionará físicamente un gimnasio en una sucursal, además se controlarán las aulas asignadas a cada curso.
- **Gestión de Clases:** en este módulo se gestionará la impartición de los cursos con una fecha establecida, monitor y asistentes.
- **Gestión de Máquinas:** en este módulo se gestionará los aparatos para realizar la distintas actividades asignadas a aulas correspondientes.

## 1.3 Organización de la memoria

El presente documento se ha dividido en los siguientes capítulos:

- **Capítulo 1: Introducción**  
Se introduce la motivación del proyecto y los objetivos a cumplir al finalizar su desarrollo.
- **Capítulo 2: Estados del arte**  
Se valoran las distintas alternativas existentes en el mercado a este proyecto.
- **Capítulo 3: Metodología**  
Se explica la metodología seguida para llevar a cabo este proyecto.
- **Capítulo 4: Fundamentos tecnológicos**  
Se indican las tecnologías que se han utilizado a lo largo del desarrollo del proyecto.

- **Capítulo 5: Análisis de requisitos**  
Se describen las diferentes funcionalidades que permite la aplicación y se detallan los actores y sus casos de uso.
- **Capítulo 6: Planificación y costes**  
Se indica la planificación del proyecto y el coste del mismo.
- **Capítulo 7: Diseño y Arquitectura del Sistema**  
Se describe el diseño de los datos y de la arquitectura del sistema.
- **Capítulo 8: Implementación**  
Se explica como se desarrollo el proyecto.
- **Capítulo 9: Pruebas**  
Se indican las distintas pruebas realizadas.
- **Capítulo 10: Conclusiones y trabajo futuro**  
Se explican las conclusiones obtenidas al finalizar el proyecto y las líneas futuras de desarrollo.



## Estado del arte

---

EN este capítulo se destacarán algunas de las aplicaciones existentes hoy en día en el mercado que guarden relación con el proyecto llevado a cabo.

### 2.1 Alternativas

#### 2.1.1 Gym community app

Es una aplicación enfocada para la captación de clientes en gimnasios utilizando estrategias de retención para evitar que los clientes de un gimnasio se marchen invirtiendo en GRC (Gestión de Relaciones con Clientes).

Entre sus funcionalidades que incluyen destacan: Fidelización, Captación, Actividades, horarios, Reservas, mensajes, eventos, ofertas, entrenamientos y dietas.

En la figura 2.1 podemos ver la pantalla principal de Gym community app



Figura 2.1: Captura de la página principal de Gym Community

### 2.1.2 IsMyGym

Es una aplicación enfocada para la generación automática de una web para un gimnasio con posibilidad de tener un dominio personalizado.

Entre sus funcionalidades destacan: el personal podrá acceder como administrador y modificar horarios, actividades, añadir noticias que serán notificadas automáticamente a todos los clientes en la App del móvil y/o por email.

Entre sus funcionalidades que incluyen destacan: Fidelización, Captación, Actividades, horarios, Reservas, mensajes, eventos, ofertas, entrenamientos y dietas

En la figura 2.2 podemos ver la pantalla principal de Is My Gym



Figura 2.2: Captura de la pagina principal de IsMyGym

### 2.1.3 Viday

Viday crea una propia aplicación móvil para un negocio, con la imagen y marca que uno escoga. A través de ella, los clientes podrán reservar una cita en cualquier momento sincronizando automáticamente con una agenda accesible desde cualquier dispositivo. Dicho esto, será un sistema de reservas online que permitirá optimizar un negocio. En otras palabras es una App para cualquier negocio que permite tener una agenda organizada con recordatorios y notificaciones, gestionar reservas en función de la disponibilidad.



## 2.2 Conclusiones

Tras analizar las principales alternativas existentes en el mercado a este proyecto, se llega a las siguientes conclusiones:

- Las dos primeras herramientas **GymCommunity e IsMyGym** son aplicaciones de pago con un coste elevado alrededor de 30€ al mes por usuario. Aunque cabe destacar que gymCommunity es una herramienta mas completa que IsMyGym.
- La herramienta **Viday** que aunque puede servir perfectamente para éste proyecto está orientado para cualquier tipo de negocio, por lo que seguramente tenga algunas funcionalidades que no nos harían falta y sería un despilfarro de gasto .

Una vez comprobado que no existen alternativas viables en el mercado, se decide la realización de este proyecto para conseguir:

- Una interfaz simple e intuitiva con la que los usuarios puedan familiarizarse desde el primer minuto.
- Permitirá la gestión de empleados, actividades, cursos, clases, aparatos en una aplicación sin necesidad de instalación de módulos de pago.
- Que la aplicación sea web, para poder acceder a ella desde cualquier lugar y equipo.



# Metodología

---

**P**ara la realización de este proyecto se ha utilizado el Proceso Unificado de Desarrollo de Software (PUDS). En este capítulo se explicará brevemente en qué consiste, cuáles son sus principales características, por qué motivos se ha seleccionado y, por último, cómo se ha aplicado al desarrollo de este proyecto.

### 3.1 Proceso Unificado de Desarrollo de Software

El desarrollo de una aplicación informática es un desafío complejo que conlleva grandes riesgos, es por ello que debe de realizarse siguiendo una metodología que minimice la posibilidad de cometer errores.

El Proceso Unificado de Desarrollo de Software (PUDS) [1] es una de las metodologías más utilizadas en desarrollo de software, se caracteriza por estar dirigido por los casos de uso, estar centrado en la arquitectura y ser iterativo e incremental:

- **Dirigido por casos de uso:** Los casos de uso son las distintas acciones que los usuarios pueden llevar a cabo en el sistema. El hecho de dirigir el desarrollo de la aplicación por esas acciones implica que los requisitos funcionales y las iteraciones estén basados en dichos casos de uso.
- **Centrado en la arquitectura:** La arquitectura del sistema es una parte crítica del mismo que puede acarrear unos costes enormes si hay errores en su concepción. Es por ello que debe de establecerse de manera que se minimicen los riesgos.
- **Iterativo e incremental:** El PUDS consta de cuatro fases **inicio, elaboración, construcción y transición**. Cada una de ellas dividida en una serie de interacciones compuestas de las subfases: **análisis de requisitos, diseño, implementación y pruebas**.

El resultado de cada iteración es un incremento de las funcionalidades del sistema.

## El CV del proceso unificado

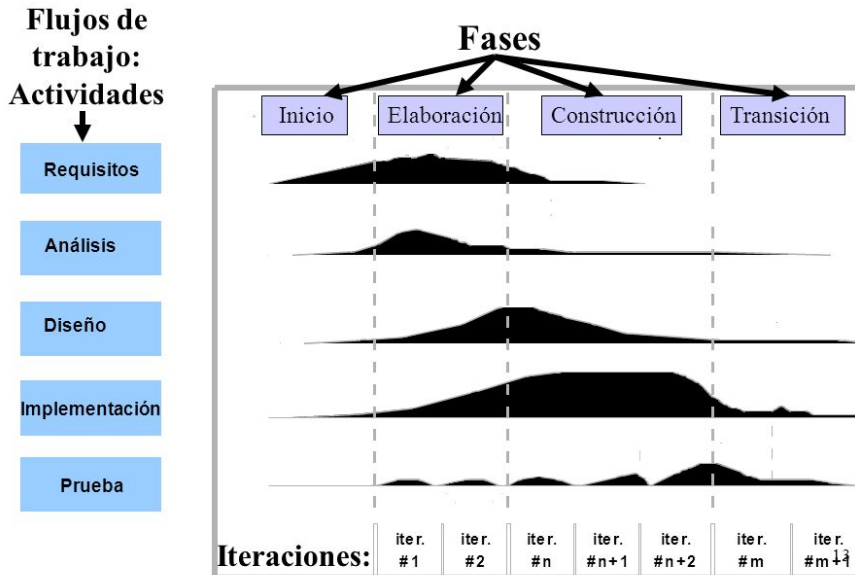


Figura 3.1: Proceso Unificado de Desarrollo de Software

EL motivo por el que se eligió el Proceso Unificado de Desarrollo es porque se acopla perfectamente a las características y necesidades de esta aplicación, como usar los diagramas UML que ayudarán en el momento de escribir el código, así como la elaboración de la aplicación de forma incremental que ayudará a ir añadiendo mejoras en el desarrollo a la vez que existirán unos determinados usuarios con unos roles específicos que nos determinará cuales son los mejores escenarios para cada usuario mediante los casos de uso.

En la figura 3.1 podemos apreciar las distintas fases y sus subfases con la carga de trabajo de cada una.

# Fundamentos tecnológicos

---

EN este capítulo se indican las distintas tecnologías y herramientas que se han utilizado para desarrollar este proyecto. La principal razón de la elección de estas tecnologías es que son de código abierto y gratuitas, pero aportan prestaciones destacadas.

## 4.1 Tecnologías y herramientas de la parte servidor

- **PostgreSQL**: sistema de Gestión de Datos Relacionales orientado a objetos (SGBDR), de las más populares del mundo utilizado por aplicaciones web muy importantes como Skype y Sony online. Entre sus principales características de PostgreSQL destacamos su alto grado de concurrencia y amplia variedad de tipos nativos. [2]
- **Java**: es el lenguaje de programación más utilizado en el mundo, esto provoca que haya una enorme cantidad de librerías y frameworks disponibles que nos facilitan el desarrollo. Podemos citar que grandes aplicaciones como Twitter y Netflix utilizan el lenguaje JAVA en su parte servidor.[3]
- **Spring**: Framework diseñado para facilitar el desarrollo de aplicaciones Java.[4]
- **Spring Boot**: herramienta de Spring que simplifica las tareas de configuración de aplicaciones web.[5]
- **Spring Data**: herramienta de Spring que simplifica las tareas de acceso a datos.[6]
- **JPA**: *framework* que nos permite establecer una correlación entre una base de datos relacional (MySQL) y un sistema orientado a objetos (Java). [6]
- **Maven**: herramienta para la gestión de dependencias de un proyecto Java.
- **Spring Security**: framework de seguridad que gestiona la autenticación y la autorización. [6]

## 4.2 Tecnologías y herramientas de la parte web

- **JavaScript:** lenguaje de programación utilizado en clientes web. [7]
- **HTML5:** lenguaje de marcado utilizado para el desarrollo de páginas web. [8]
- **CSS:** lenguaje de diseño gráfico que se utiliza para dar estilo a una página web. [8]
- **jQuery:** librería de JavaScript que permite simplificar la manera de interactuar con los documentos HTML. [8]
- **AngularJS:** AngularJS es un framework estructural para aplicaciones web dinámicas.[9]
- **Bootstrap:** librería de JavaScript que facilita la maquetación y diseño de sitios web. [10]

## 4.3 Tecnologías y herramientas de la parte web

- **Eclipse:** IDE de código abierto y multiplataforma para desarrollar aplicaciones.
- **Postman:** herramienta que permite la creación de peticiones REST a APIs.
- **JUnit:** framework utilizado para testear proyectos Java.[11]
- **LaTeX:** sistema de composición de textos orientado a la creación de documentos escritos.
- **Dia:** herramienta que permite crear diagramas
- **WhiteStar UML:** herramienta que permite crear diagramas. [12]
- **SmartSheet:** herramienta utilizada para crear los diagramas de Gantt.

# Análisis de requisitos

---

EN este capítulo se detallarán los requisitos que debe satisfacer la aplicación. Para ello se realizará un estudio de las funcionalidades necesarias y se identificarán los casos de uso que podrán llevar a cabo los distintos actores.

## 5.1 Actores

Los actores representan los distintos roles que pueden adoptar los usuarios al interactuar con el sistema, A continuación se describen los distintos tipos de actores:

- **Administrador:** Se trata del rol con mayor número de funcionalidades en el sistema, ya que tiene control sobre toda la aplicación, principalmente en la gestión de cursos y todos los empleados asignados a las actividades, cursos y clases.
- **Encargado de Gimnasio:** Representa el rol de responsable de un centro, sus funcionalidades están restringidas a su centro asignado, además las máquinas y sus tipos así como el aula donde se impartirá el curso son responsabilidad del encargado.
- **Monitor o empleado:** Representa el rol de impartir un curso mediante clases.
- **Cliente:** Representa el rol de inscribirse a un curso y ver sus horarios asignados.

En la figura 5.1 se muestra la jerarquía existente entre los distintos actores.

## 5.2 Casos de Uso

Los casos de uso definen las distintas acciones que podrán llevar a cabo los usuarios, en este proyecto se han agrupado en cinco bloques que se detallan a continuación:

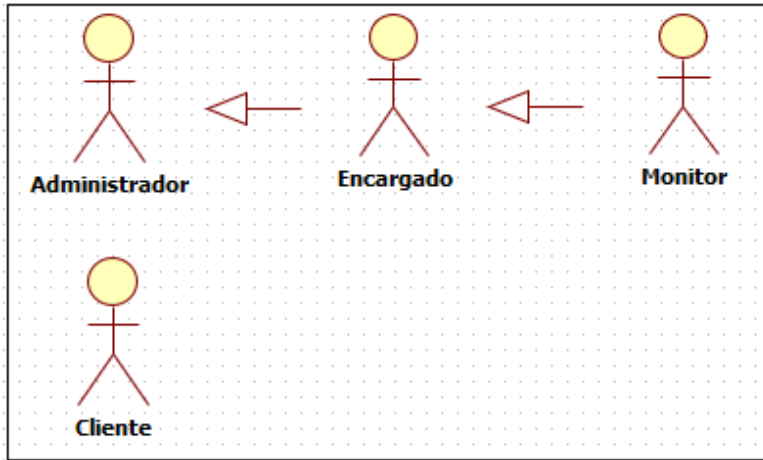


Figura 5.1: Las dos jerarquías de actores de nuestra aplicación.

### 5.2.1 Gestión de empleados

En este apartado se agrupan las acciones referentes a la gestión de empleados. Las funcionalidades que tendrá disponible un usuario estarán siempre limitadas al rol que se le asigne.

En la tabla 5.1 se muestra una descripción resumida de cada caso de uso, mientras que en la figura 5.2a podemos ver el diagrama que refleja los citados casos.

### 5.2.2 Gestión de gimnasio

En este bloque se recogen las funcionalidades relativas a la gestión de un gimnasio por parte de los empleados. Además se recoge las funcionalidades sobre la localidad y las aulas dentro de un gimnasio. Del mismo modo que en la gestión de empleados, un usuario solo podrá ejecutar acciones si tiene el rol asignado para poder hacerlo, y sobre un gimnasio que se encuentra a su cargo.

En la tabla 5.2 se muestra una descripción resumida de cada caso de uso, mientras que en la figura 5.2b podemos ver el diagrama que se refleja los citados casos.

Tabla 5.1: Descripción de los casos de uso de Gestión de Empleados.

Caso de Uso	Descripción
Alta Nuevo Empleado	Se realiza el alta de un nuevo empleado asignándole un jefe si procede
Ver Subordinados	El usuario registrado obtiene la lista de empleados que tiene a su mando
Baja Empleado	Se da de baja un empleado
Asignar un Jefe	Se cambia o se le asigna un jefe al empleado



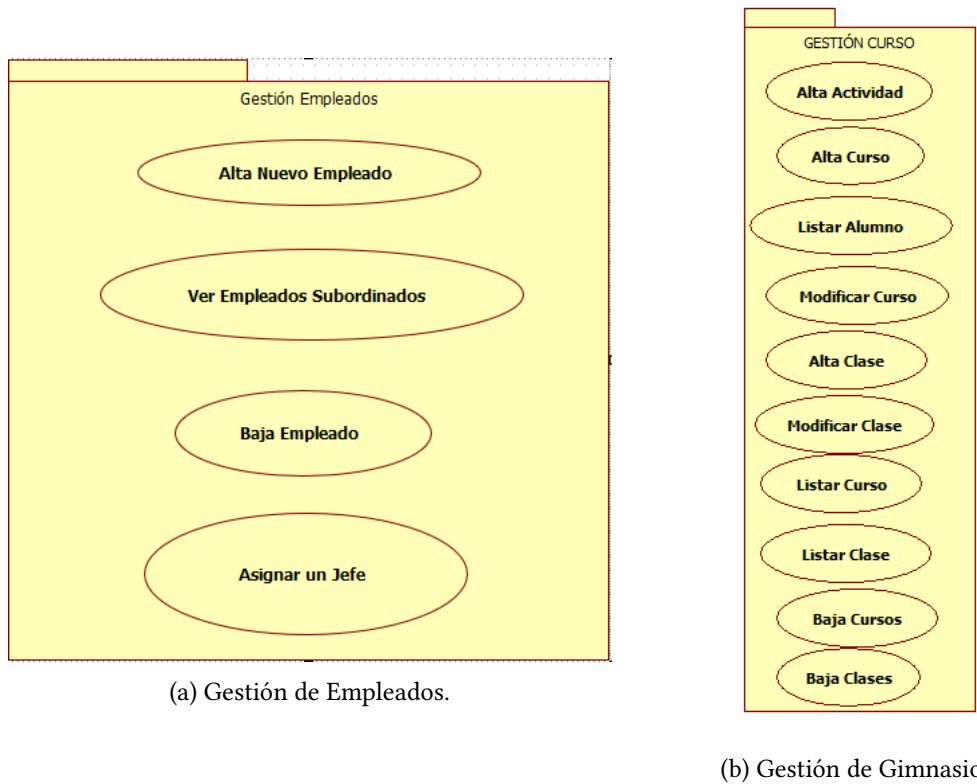


Figura 5.2: Casos de uso pertenecientes a los bloques Gestión de Empleados y Gestión de Gimnasios.

Tabla 5.2: Descripción de los casos de uso de Gestión de Gimnasios.

Caso de Uso	Descripción
Alta Localidad	El administrador realiza el alta de una localidad
Alta Gimnasio	El administrador realiza el alta de un gimnasio asociado a una localidad y encargado
Crear Aula	El encargado crea un aula asociado a un gimnasio
Baja Gimnasio	El administrador realiza la baja de un gimnasio
Baja Aula	El encargado realiza una baja asociado a un gimnasio
Modificar Gimnasio	El Administrador modifica datos de un gimnasio como añadir o cambiar de encargado

Tabla 5.3: Descripción de los casos de uso de Gestión de Cursos.

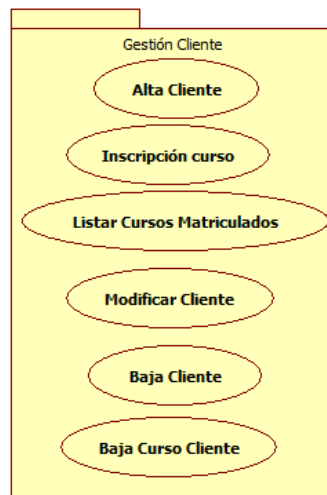
Caso de Uso	Descripción
Alta Actividad	El usuario realiza el alta de una actividad
Alta Curso	El usuario realiza el alta de un curso asociado a un aula, un monitor y una actividad
Listar alumnos	El usuario lista todos los alumnos inscritos a un curso y generar un documento PDF
Modificar Curso	El usuario modifica datos del curso
Alta Clase	El usuario da de alta a una clase
Modificar Clase	El usuario modifica datos de una clase
Baja Curso	El usuario da de baja a un curso
Baja Clase	El usuario da de baja a una clase

### 5.2.3 Gestión de Cursos

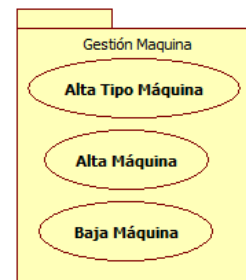
Este apartado reúne la funcionalidades asociadas a la gestión de cursos y relacionadas al tipo de rol que el usuario tenga. En la tabla 5.3 se describe brevemente cada caso de uso y en la figura 5.3a se muestra el diagrama correspondiente a los citados casos.



(a) Gestión de Cursos.



(b) Gestión de Clientes.



(c) Gestión de Máquinas.

Figura 5.3: Casos de uso pertenecientes a los bloques Gestión de Cursos, Gestión de Clientes y Gestión de Máquinas.

Tabla 5.4: Descripción de los casos de uso de Gestión de Clases.

Caso de Uso	Descripción
Alta Cliente	El usuario da de alta a un cliente
Inscripción Curso	El usuario se inscribe a un curso
Listar cursos matriculados	El usuario lista los cursos a los que esta matriculado
Modificar Cliente	El usuario modifica datos de un cliente
baja cliente	El usuario da de baja a un cliente
Baja Curso	El usuario da de baja a un curso a un cliente

Tabla 5.5: Descripción de los casos de uso de Gestión de Máquina.

Caso de Uso	Descripción
Alta Tipo Máquina	El usuario da de alta a un tipo de máquina
Alta Máquina	El usuario da de alta a una máquina asociado a un tipo de máquina y un aula
Baja Máquina	El usuario da de baja a una máquina

#### 5.2.4 Gestión de Clientes

Este apartado reúne las funcionalidades asociadas a la gestión de clientes. En la tabla 5.4 se describe brevemente cada caso de uso y en la figura 5.3b correspondiente a los citados casos.

#### 5.2.5 Gestión de Máquinas

Este apartado reúne las funcionalidades asociadas a la gestión de Máquinas. En la tabla 5.5 se describe brevemente cada caso de uso y en la figura 5.3c correspondiente a los citados casos.

### 5.3 Casos de uso de cada actor

En este apartado indicaremos los distintos roles que adoptan los usuarios en esta aplicación

#### 5.3.1 Casos de uso del rol Administrador

El rol "Administrador" tiene disponibles todos las funcionalidades disponibles, gestión empleados, gestión cursos, gestión clientes, gestión máquinas y gestión Gimnasios. En la figura 5.4 podemos apreciar las relaciones entre este rol y los casos de uso que tiene a su disposición.

#### 5.3.2 Casos de uso del rol Encargado

El encargado tiene disponibles las funcionalidades sobre las gestión de Máquinas, cursos impartidos en el gimnasio del que es responsable y las aulas donde se van a realizar los cursos

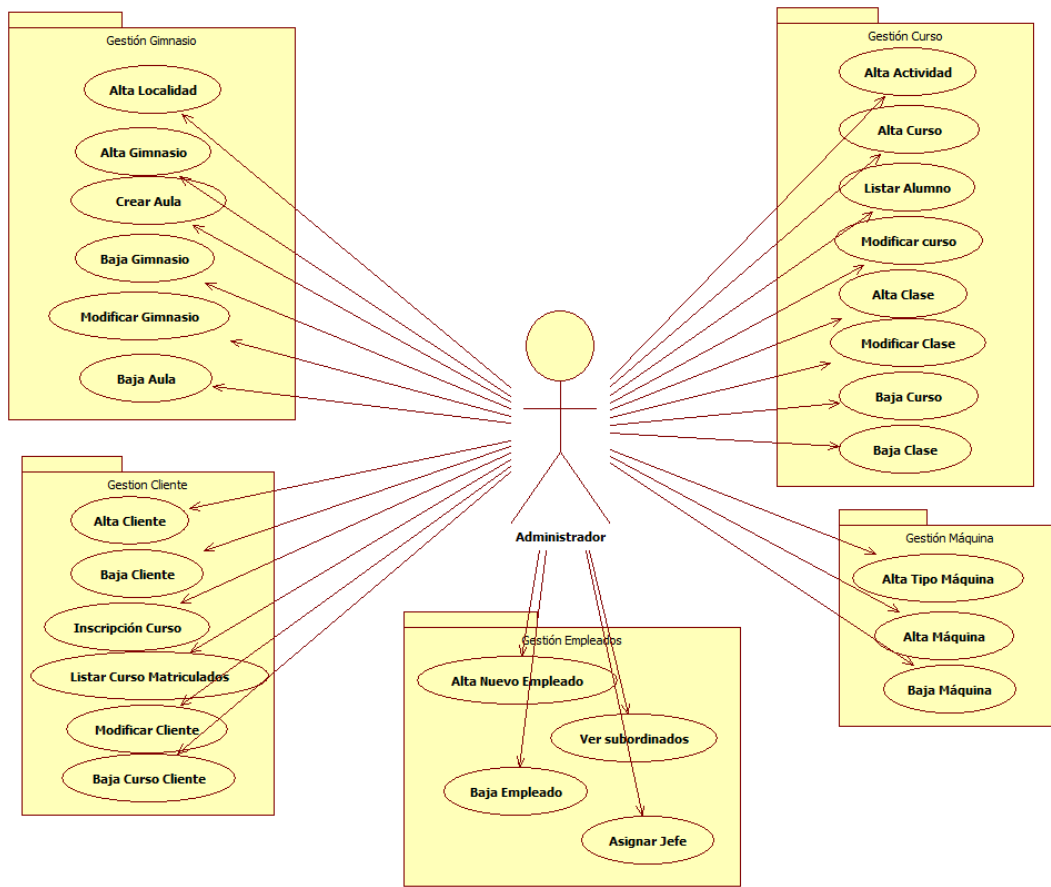


Figura 5.4: Casos de uso pertenecientes al rol Administrador.

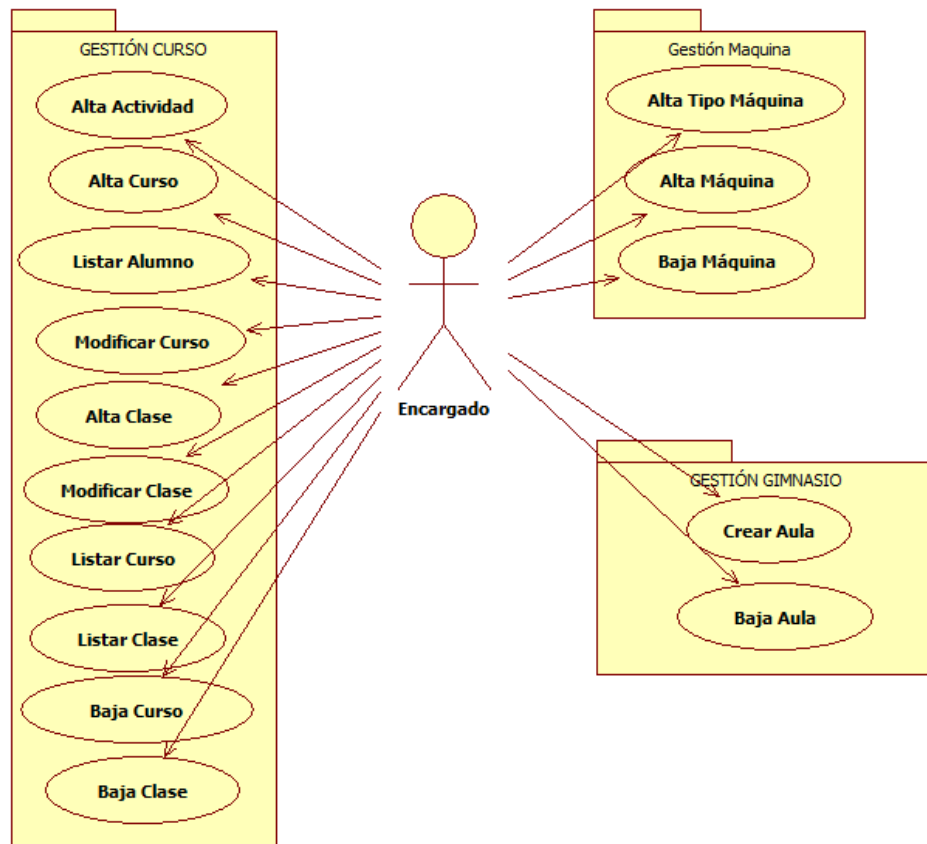


Figura 5.5: Casos de uso pertenecientes al rol Encargado.

. En la figura 5.5 podemos apreciar las relaciones entre este rol y los casos de uso que tiene a su disposición.

### 5.3.3 Casos de uso del rol Monitor

El monitor tiene disponible la funcionalidad de gestión de Monitores donde solo podrá visualizar datos concernientes a si mismo. En la figura 5.6 podemos apreciar las relaciones entre este rol y los casos de uso que tiene a su disposición.

### 5.3.4 Casos de uso del rol Cliente

El cliente tiene disponible la funcionalidad de gestión de Clientes. Del mismo modo que los monitores, los clientes solo podrán tomar acciones concernientes a si mismo.

En la figura 5.7 podemos apreciar las relaciones entre este rol y los casos de uso que tiene a su disposición.

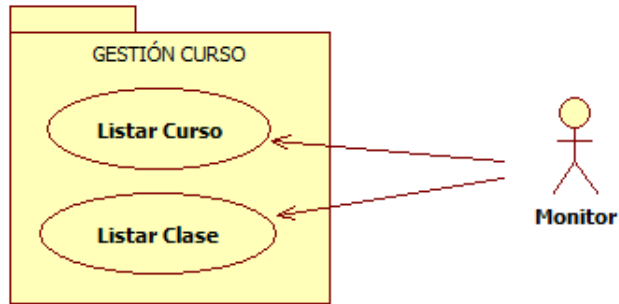


Figura 5.6: Casos de uso pertenecientes al rol Monitor.

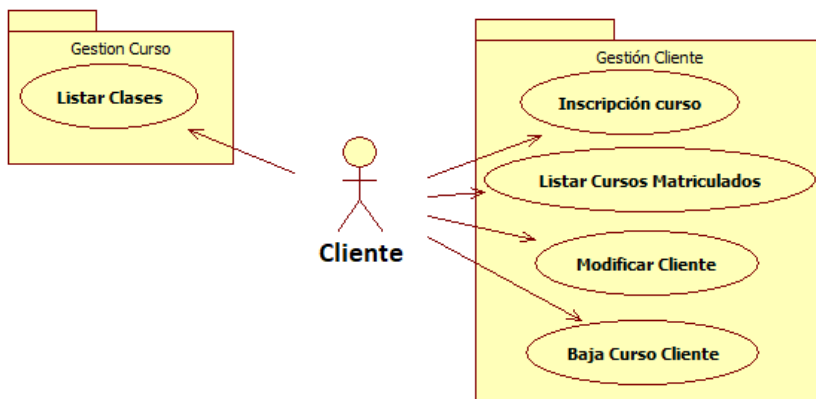


Figura 5.7: Casos de uso pertenecientes al rol Cliente.

# Planificación y costes

---

**E**N este capítulo de la memoria se detalla el proceso seguido para la realización de este proyecto y el coste del mismo.

## 6.1 Planificación

Según nuestra aplicación del proceso unificado de desarrollo, la primera fase del proyecto consistió en un análisis previo de requisitos en la que se definieron los distintos módulos o apartados de los que consta el proyecto. Tal y como se ha descrito en el capítulo correspondiente, se han previsto cinco módulos: uno que engloba las funcionalidades relativas a la **gestión de empleados**, otro para las funcionalidades relativas a la gestión de un **gestión de gimnasio**, otro apartado para la **gestión de máquinas**, un cuarto para la **gestión de clientes**, y un último que recoge las funcionalidades de **gestión de cursos**.

El siguiente paso, ha consistido en seleccionar las herramientas y tecnologías a utilizar para el desarrollo de la aplicación. Una vez elegidas se procedió a realizar un estudio más detallado y una fase de aprendizaje de aquellas herramientas que eran desconocidas para el desarrollador.

Finalmente, se procedió a la fase de pruebas y a la finalización de la memoria, puesto que la documentación del proyecto se realizó al mismo tiempo que se desarrollaban las distintas iteraciones.

En la figura 6.1 puede verse el diagrama de Gantt general del proyecto. Para su confección se ha estimado una media de 8 horas diarias durante todo el ciclo de desarrollo del proyecto.

### 6.1.1 Planificación de las iteraciones

Las iteraciones para el desarrollo del software coinciden con los módulos descritos en el apartado anterior.

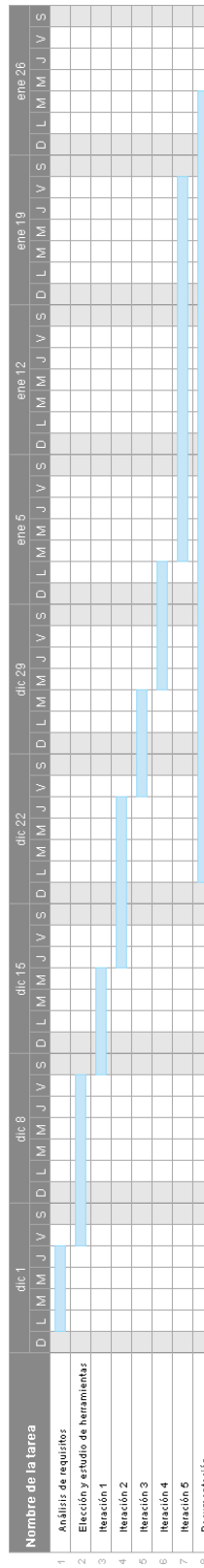


Figura 6.1: Diagrama de Gantt del proyecto.





Figura 6.2: Diagrama de Gantt correspondiente a la iteración 1.

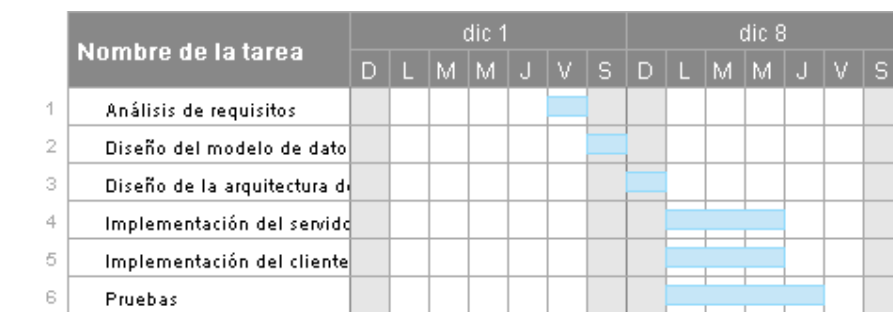


Figura 6.3: Diagrama de Gantt correspondiente a la iteración 2.

En la primera iteración (figura 6.2 se implementaron las funcionalidades recogidas en los apartados de gestión de empleados, gestión de localidades y gestión de gimnasio. Se tomó la decisión de realizarlas en la misma iteración debido a la fuerte dependencia existente entre ellas. En esta primera iteración también se llevó a cabo el diseño de la arquitectura del sistema.

A continuación, en la segunda iteración (figura 6.3) se desarrollaron las funcionalidades relativas al módulo de gestión de máquinas.

En la tercera iteración (figura 6.4) se desarrolló la gestión de los clientes.

En la cuarta iteración (figura 6.5) se desarrollaron las funcionalidades referentes a los cursos.

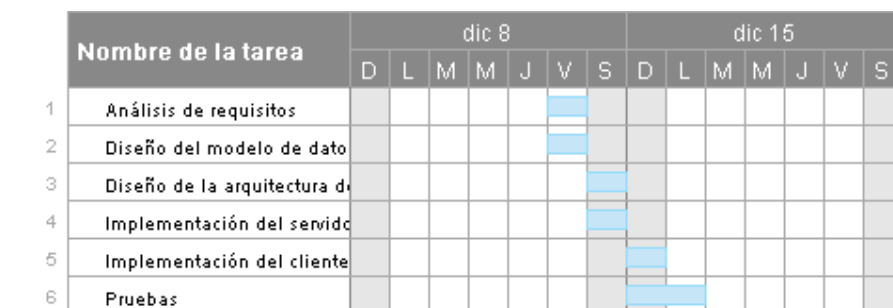


Figura 6.4: Diagrama de Gantt correspondiente a la iteración 3.

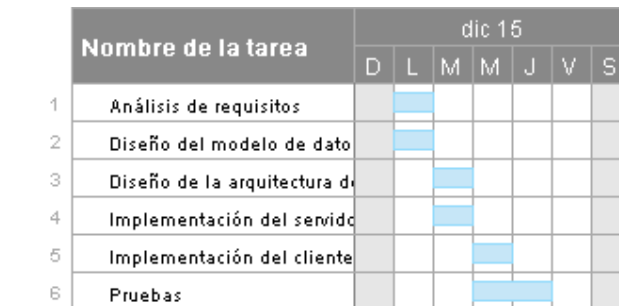


Figura 6.5: Diagrama de Gantt correspondiente a la iteración 4.

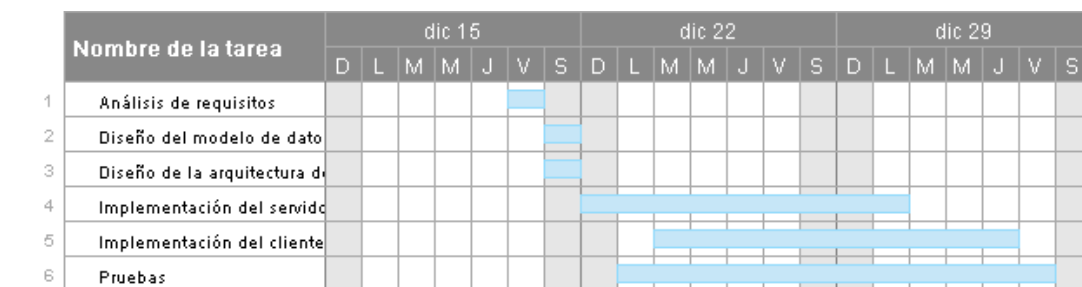


Figura 6.6: Diagrama de Gantt correspondiente a la iteración 5.

En la quinta y última iteración (figura 6.6) se desarrollaron las funcionalidades referentes a las clases. Esta iteración se desarrolló en esta última fase ya que es la más dependiente de todas y se esperó a que las otras iteraciones acabaran de forma correcta.

A pesar de que la realización de este proyecto se haya dividido en 5 iteraciones, los capítulos de esta memoria "Análisis de requisitos", "Diseño", "Desarrollo" y "Pruebas" agrupan las funcionalidades de las 5 iteraciones para una mayor claridad a la hora de leerla.

### 6.1.2 Seguimiento

Se llevaron a cabo las acciones para la comprobación de la correcta ejecución de las actividades del proyecto establecidas en la planificación del mismo y todo ha salido según lo previsto y sin incidencias relevantes.

## 6.2 Coste

Para calcular el coste de este proyecto se tienen en cuenta dos aspectos, el coste del material hardware y software empleado y el coste de los recursos humanos.

Tabla 6.1: Coste del Material empleado en el proyecto.

Material	Descripción	Coste imputable
SmartSheet	Herramienta para la elaboración de los diagramas de Gantt	0 €.
Ordenador Portátil	Portátil ASUS F550C, procesador Intel Core i5 y 4 GB de memoria RAM.	0 €

Tabla 6.2: Detalle del coste por tareas del analista-programador.

Tarea	Horas	Coste
Análisis previo de requisitos	24	720€
Elección y estudio de herramientas	36	1080€
Iteración 1	24	720€
Iteración 2	36	1.080€
Iteración 3	18	540€
Iteración 4	24	720€
Iteración 5	84	2.520€
Documentación	108	3.240€
Total		10.620€

### 6.2.1 Coste del material

El coste del material empleado en el proyecto se reduce al valor del equipo informático en el que se ha desarrollado la aplicación, dado que se ha utilizado software libre para la realización del mismo. En la tabla 6.1 se describe el coste del material de pago.

### 6.2.2 Coste de los recursos humanos

En la realización del proyecto han intervenido dos recursos humanos: el alumno, ejerciendo el rol de analista-programador y el tutor, con un rol de jefe de proyecto. Para elaborar la estimación de coste del proyecto se asume que el salario de un jefe de proyecto es de 50€/hora y el de analista-programador es de 30€/hora. Como ya se indicó en el apartado anterior, la jornada laboral es de 8 horas, en los días que coinciden el desarrollo de las iteraciones con la elaboración de la documentación, se ha destinado 3h a la documentación y 5 al desarrollo. En la tabla 6.2 se indica el desglose del coste de las tareas realizadas por el analista mientras que en la tabla 6.3 se muestra el coste de las tareas realizadas por el jefe de proyecto.

### 6.2.3 Coste final del proyecto

El coste total del proyecto, sumando el coste del material y el de los recursos humanos se indica en la tabla 6.4. No se incluye el coste del software porque se ha seleccionado *open-source*. Excepto en el caso de *SmartSheet* que se usó una licencia de prueba.

Tabla 6.3: Detalle del coste por tareas del jefe de proyecto.

<b>Tarea</b>	<b>Horas</b>	<b>Coste</b>
Análisis previo de requisitos	7	350€
Elección y estudio de herramientas	2	100€
Iteración 1	2	100€
Iteración 2	2	100€
Iteración 3	2	100€
Iteración 4	2	100€
Iteración 5	3	150€
<b>Total</b>		<b>1.000€</b>

Tabla 6.4: Detalle del coste del personal humano.

<b>Recurso</b>	<b>Coste (€)</b>
Material	0
Analista-programador	10.620
Jefe de proyecto	1.000
<b>Coste Total</b>	<b>11.620</b>

# Diseño y Arquitectura del Sistema

---

UNA vez explicada la planificación, se describe el diseño de la aplicación. Este capítulo está dividido en dos apartados: modelo de datos y diseño de la arquitectura.

## 7.1 Modelo de datos

El modelado de los datos es la primera parte del apartado de diseño, en ella se define la estructura de la base de datos que persistirá la información del sistema.

Este apartado consistió en dotar a la aplicación la capacidad de gestionar los distintos roles en el gimnasio, pero el mayor reto en esta aplicación consistió en generar un horario para los diversos usuarios. Un ejemplo es el caso de los clientes que al inscribirse en diversos cursos, es necesario generar un horario para él y luego conseguirlo en formato físico (PDF). Además la cadena de gimnasios también necesita obtener los horarios que tiene cada sucursal, que es donde radicaría la mayor dificultad en esta aplicación.

Las entidades diseñadas para lograr el desarrollo de este proyecto son las siguientes. En la figura 7.1 podemos apreciar el modelo Entidad Relación de las mismas.

- **Empleado:** representa al Encargado que está relacionado con un gimnasio, y al monitor que es el empleado que impartirá los cursos.
- **Actividad:** entidad que representa a la actividad que será impartida en en una clase. Está relacionada con los cursos.
- **Clase:** entidad que representa a la lección que el monitor explica a los clientes.
- **Cliente:** entidad que representa al usuario que disfruta de las clases.
- **Curso:** entidad que representa al tiempo (generalmente cada año) asignado a las clases.
- **Gimnasio:** entidad que representa a una sucursal física dentro de una cadena de gimnasio.

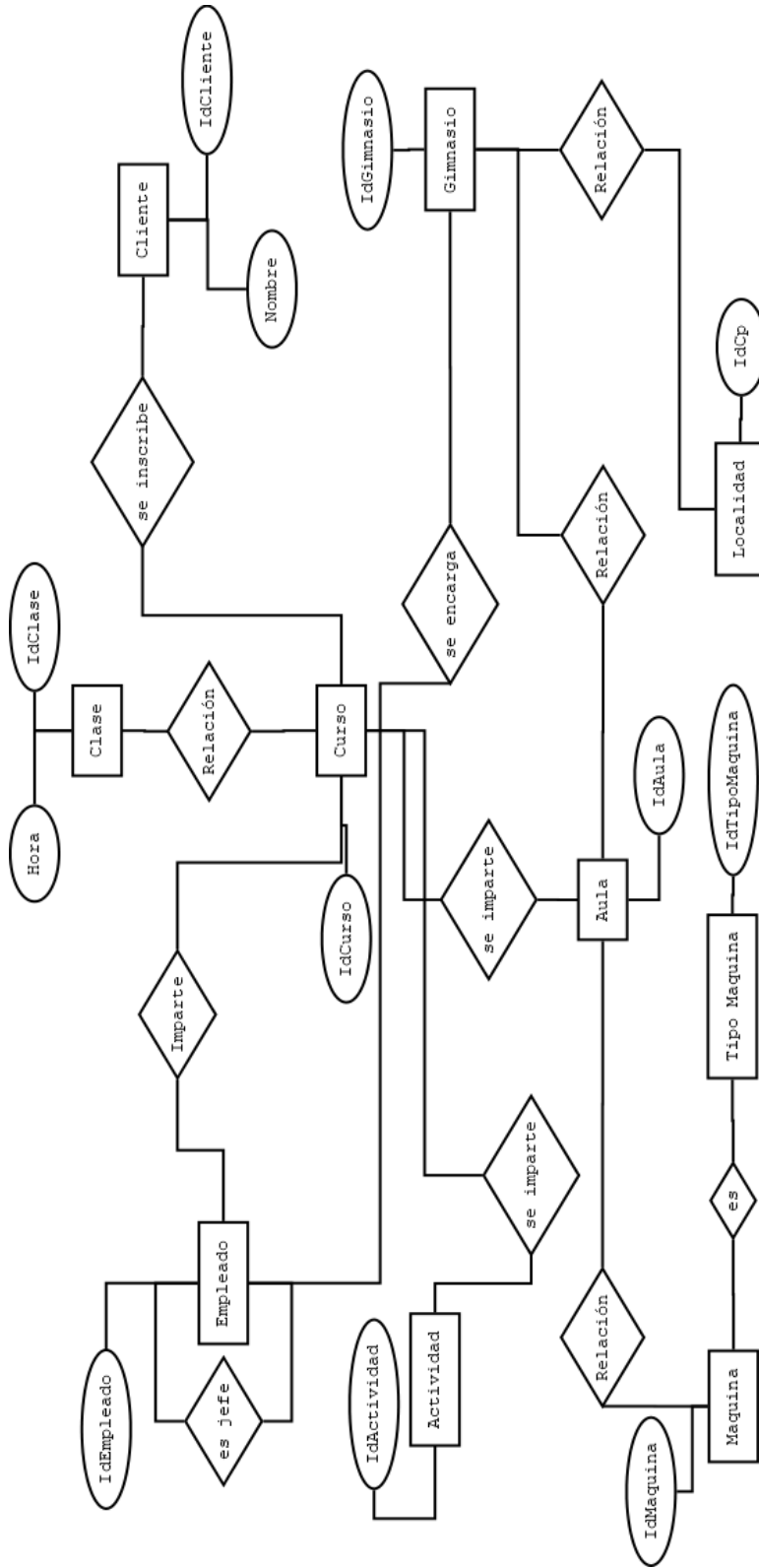


Figura 7.1: Diagrama Entidad-Relación

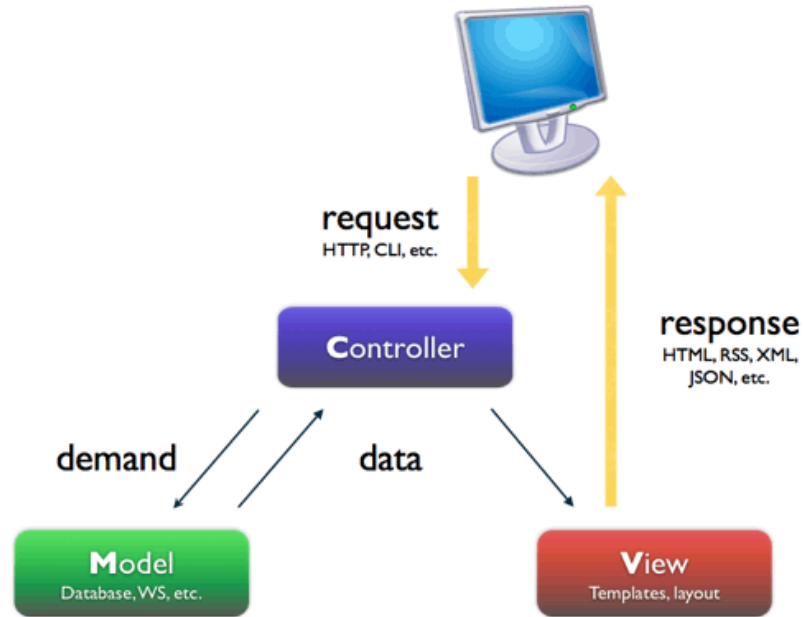


Figura 7.2: Funcionamiento del patrón MVC.

- **Aula:** entidad que representa al lugar físico donde se impartirán los cursos.
- **Localidad:** entidad que representa al lugar físico donde se encuentran los gimnasios.
- **Máquina:** entidad que representa a los aparatos físicos que se utilizan en los cursos, está asociado a las aulas y a los tipos de máquina.
- **Tipo de Máquina:** entidad que representa a los tipos de aparatos físicos clasificados según su utilidad asociándose luego a las distintas máquinas.

## 7.2 Diseño de la arquitectura del sistema

Esta aplicación se ha desarrollado siguiendo las directrices del patrón arquitectónico **Modelo-Vista-Controlador (MVC)** (figura 7.2) combinado con la arquitectura **Cliente-Servidor**. El patrón MVC separa la lógica de la aplicación de la lógica de la interfaz en tres componentes bien definidos:

- **Modelo:** es el encargado de la lógica de negocio y el acceso a la capa de datos.
- **Vista:** se trata de la representación visual de los datos en un formato adecuado para que el usuario pueda interactuar con ellos.

- **Controlador:** procesa los distintos eventos realizados por el usuario en la vista y le envía peticiones al modelo.

En el caso de la arquitectura **Cliente-Servidor**, en esta arquitectura la computadora de cada uno de los usuarios, llamada cliente, produce una demanda de información a cualquiera de las computadoras que proporcionan información, conocidas como servidores estos últimos responden a la demanda del cliente que la produjo. Bajo este modelo cada usuario tiene la libertad de obtener la información que requiera en un momento dado proveniente de una o varias fuentes locales o distantes y de procesarla como según le convenga.

Las partes que componen esta arquitectura son:

- **Cliente:** Programa ejecutable que participa activamente en el establecimiento de las conexiones. Envía una petición al servidor y se queda esperando por una respuesta.
- **Servidor:** Es un programa que ofrece un servicio que se puede obtener en una red. Acepta la petición desde la red, realiza el servicio y devuelve el resultado al solicitante.

En la figura 7.3 se indica como se ha combinado la arquitectura **Cliente-Servidor** con el ya mencionado patrón **MVC** que básicamente explica como nuestro cliente en este caso **AngularJs** envía una petición **Http** solicitando un recurso, nuestro controlador, en este caso **Spring** procesa la petición y solicita los datos a través de **JPA** a nuestra **BD PostgreSQL**.



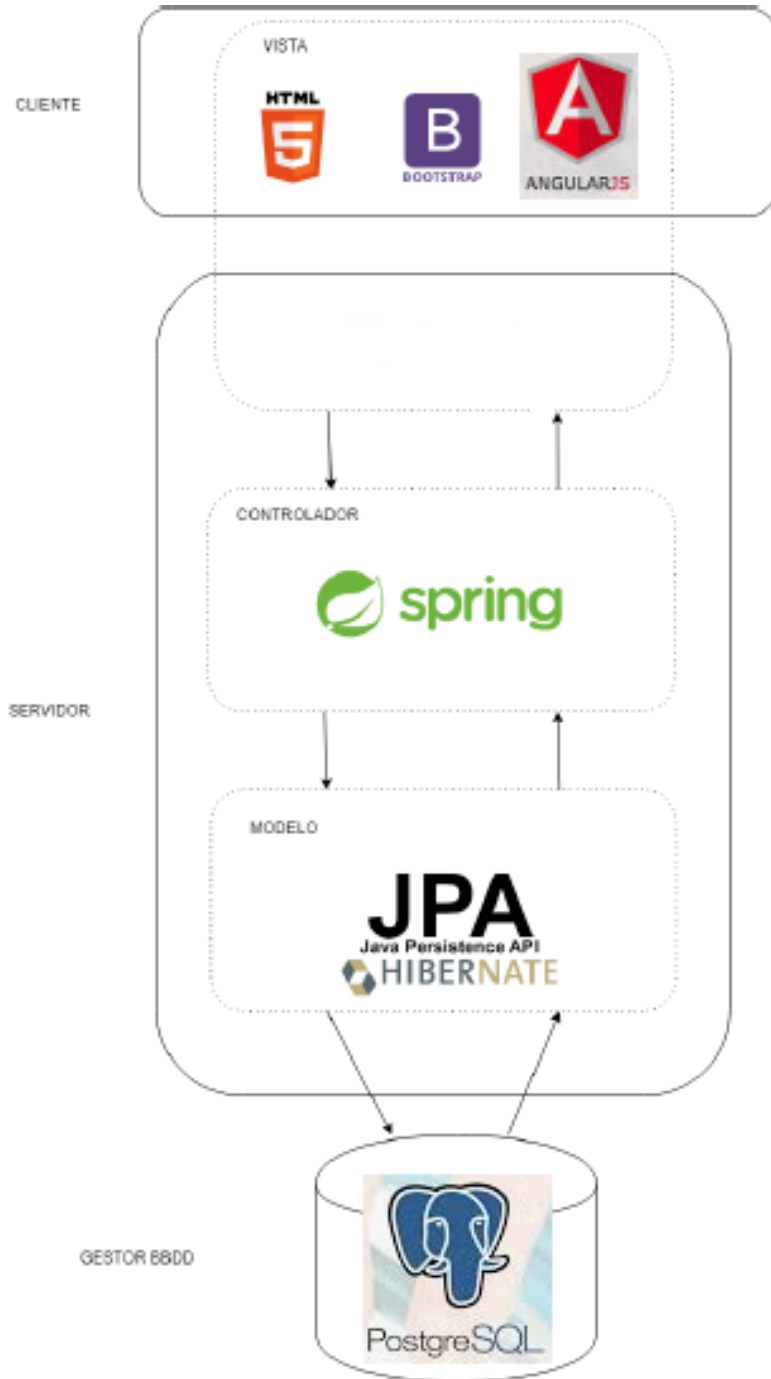


Figura 7.3: Arquitectura general de nuestro sistema.



# Implementación

---

EN este capítulo se explica cómo se ha llevado a cabo el desarrollo de la aplicación, consta de dos grandes apartados: la implementación del servidor y la del cliente. Ahora bien, por su importancia, se ha descrito en un apartado diferente la interfaz gráfica de usuario del cliente.

## 8.1 Implementación del servidor

Una de las grandes ventajas de JPA [6] es que nos permite manipular la base de datos a través de objetos, estos objetos son conocidos como Entity, las cuales son clases comunes y corrientes también llamada POJO's (Plain Old Java Objects), estas clases tiene la particularidad de ser clases que están mapeadas contra una tabla de la base de datos, dicho mapeo se lleva a cabo generalmente mediante Anotaciones.

Dichas anotaciones brindan los suficientes metadatos como para poder relacionar las clases contra las tablas y las propiedades contra las columnas. Es de esta forma que JPA es capaz de interactuar con la base de datos a través de las clases.

En la aplicación se ha mapeado las clases que representan a las distintas entidades como se muestra en el código de la figura 8.1.

En la figura 8.2 se muestra el diagrama de clases de las entidades persistentes existentes en la aplicación.

La parte servidor se ha desarrollado siguiendo la estructura **Controller-Service-DAO**, tal y como se muestra en el diagrama de paquetes de la figura 8.3. Esta implementación se basa en el uso de tres patrones: el **patrón DAO** (*Data Access Object*), el **patrón DTO** (*Data Transfer Object*) y el **patrón Facade**. En las siguientes secciones explicamos cómo se ha implementado cada uno.

```
1 @Entity
2 @Table(name = "empleado")
3 public class Empleado implements Serializable {
4     private static final long serialVersionUID =
5         -4440816780062224013L;
6     @Id
7     @GeneratedValue(strategy=GenerationType.IDENTITY)
8     @Column(name = "id_empleado")
9     private Integer empleadoId;
10
11     @Column(name = "nombre_empleado")
12     private String nombreEmpleado;
13
14     @Column(name = "apellido1_empleado")
15     private String apellido1Empleado;
16
17     @Column(name = "apellido2_empleado")
18     private String apellido2Empleado;
19
20     @Column(name = "email_empleado")
21     private String emailEmpleado;
22
23     @Column(name="dni_empleado")
24     private String dniEmpleado;
25
26     @Column(name="telefono")
27     private String telefono;
28
29     @Column(name="password")
30     private String password;
31     ...

```

Figura 8.1: Mapeado de las clases que representan las distintas entidades de la aplicación

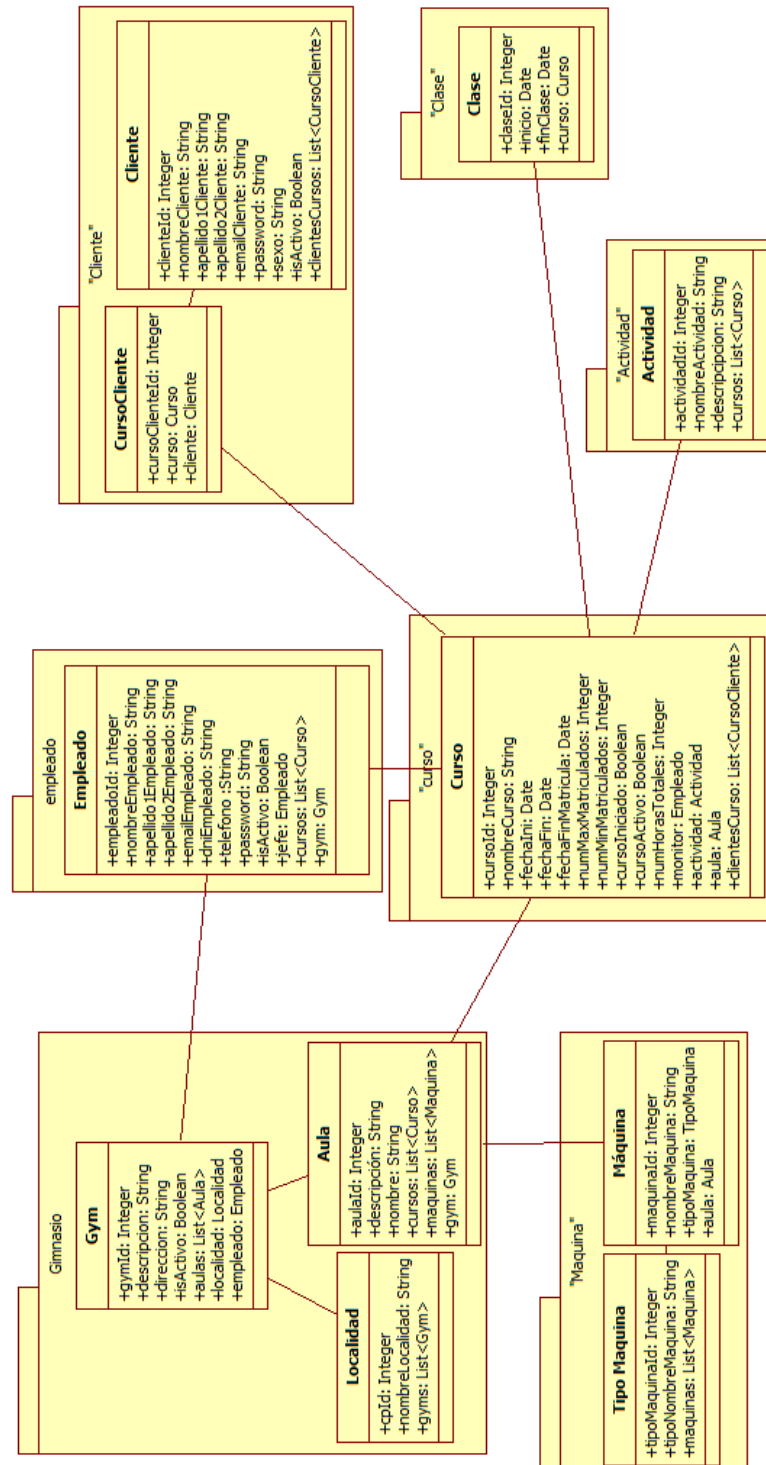


Figura 8.2: Diagrama de paquetes de la parte "Entities".

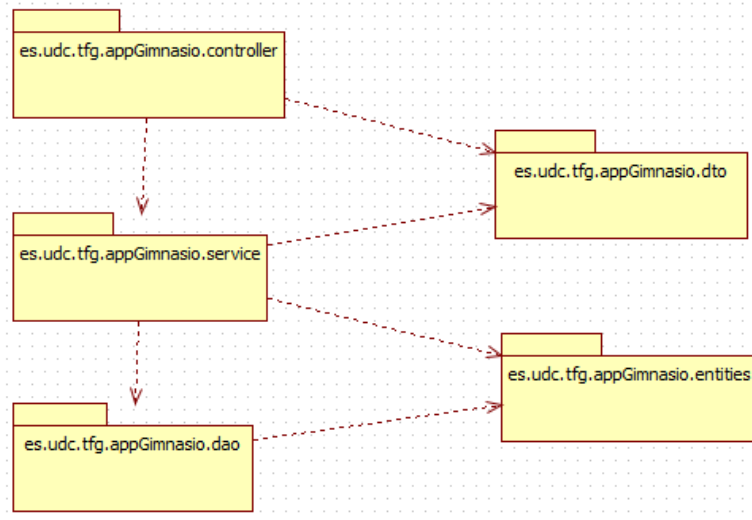


Figura 8.3: Diagrama de paquetes de la parte servidor.

```

1 @Repository
2 public interface EmpleadoDAO extends JpaRepository<Empleado,
   Integer>{
3
4     @Query(value="SELECT * FROM empleado emp where
5         (emp.id_jefe)=:empleadoId", nativeQuery = true)
6     List<Empleado> findSubordinadosByIdEmpleado(@Param("empleadoId")
7         final Integer empleadoId);
8
9     @Query(value="SELECT * FROM empleado emp join gym gym on
10        emp.id_empleado != gym.id_empleado", nativeQuery = true)
11    List<Empleado> findNoEncargados();
12 }

```

Figura 8.4: Ejemplo de la interfaz *EmpleadoDAO* con dos de las búsquedas implementadas.

### 8.1.1 El patrón DAO

El patrón DAO es un patrón que nos permite encapsular el acceso a la base de datos a través de la aplicación, es decir, proporciona los métodos necesarios para insertar, actualizar, borrar y consultar información desde la aplicación a la base de datos de una manera transparente, lo que permite aislar la capa de lógica de negocio de la capa de acceso a datos.

A su vez, este patrón permite una abstracción que oculta el gestor de base de datos a la aplicación, por lo que sería posible cambiar entre distintos gestores como MySQL y Oracle sin necesidad de cambios en el código. En este proyecto se ha utilizado el framework Spring Data para facilitar el desarrollo de este apartado.

En las clases DAO se ha implementado la interfaz *JpaRepository* (ver figura 8.4), la cual

```
1 import java.io.Serializable;
2 import es.udc.tfg.appGimnasio.entities.empleado.Empleado;
3
4 public class EmpleadoDto implements Serializable {
5     private static final long serialVersionUID = 1L;
6     private Integer empleadoId;
7     private String nombreEmpleado;
8     private String apellido1Empleado;
9     private String apellido2Empleado;
10    private String emailEmpleado;
11    private String dniEmpleado;
12    private String telefono;
13    private String password;
14    private Empleado jefe;
15    private Boolean isActivo;
16    ...
}
```

Figura 8.5: Ejemplo del patrón DTO en la clase *EmpleadoDto*.

nos proporciona los métodos CRUD relativos a la entidad relacionada y nos simplifica enormemente la codificación.

### 8.1.2 El patrón DTO

El patrón DTO permite optimizar la transferencia de datos a través de las distintas capas de la aplicación. Su finalidad es la de crear un objeto plano con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de una única tabla o de varias, representadas en una clase simple que únicamente cuenta con atributos, getters y setters.

Es importante destacar que las clases DTO deben implementar la interfaz *Serializable*, lo que permite que las instancias de dicha clase se puedan convertir en una serie de bytes para enviarlas a través de la red y volver a reconstruirlas una vez recibidas, tal y como se muestra en el siguiente ejemplo de la figura 8.5.

En la figura 8.6 se indican las clases existentes en el paquete "dto" del proyecto.

### 8.1.3 El patrón *Facade*

El patrón *Facade* o Fachada se utiliza para estructurar un sistema en capas, utilizando las fachadas como punto de entrada a cada nivel para agrupar funcionalidades. En esta aplicación, los distintos casos de uso están implementados en clases *Service* que siguen el comportamiento establecido por este patrón: las distintas clases *Service* utilizan los DAOs para acceder a la información y los DTOs para transferirla.

En este proyecto, las clases que implementan las interfaces *Service* cuentan con las ano-

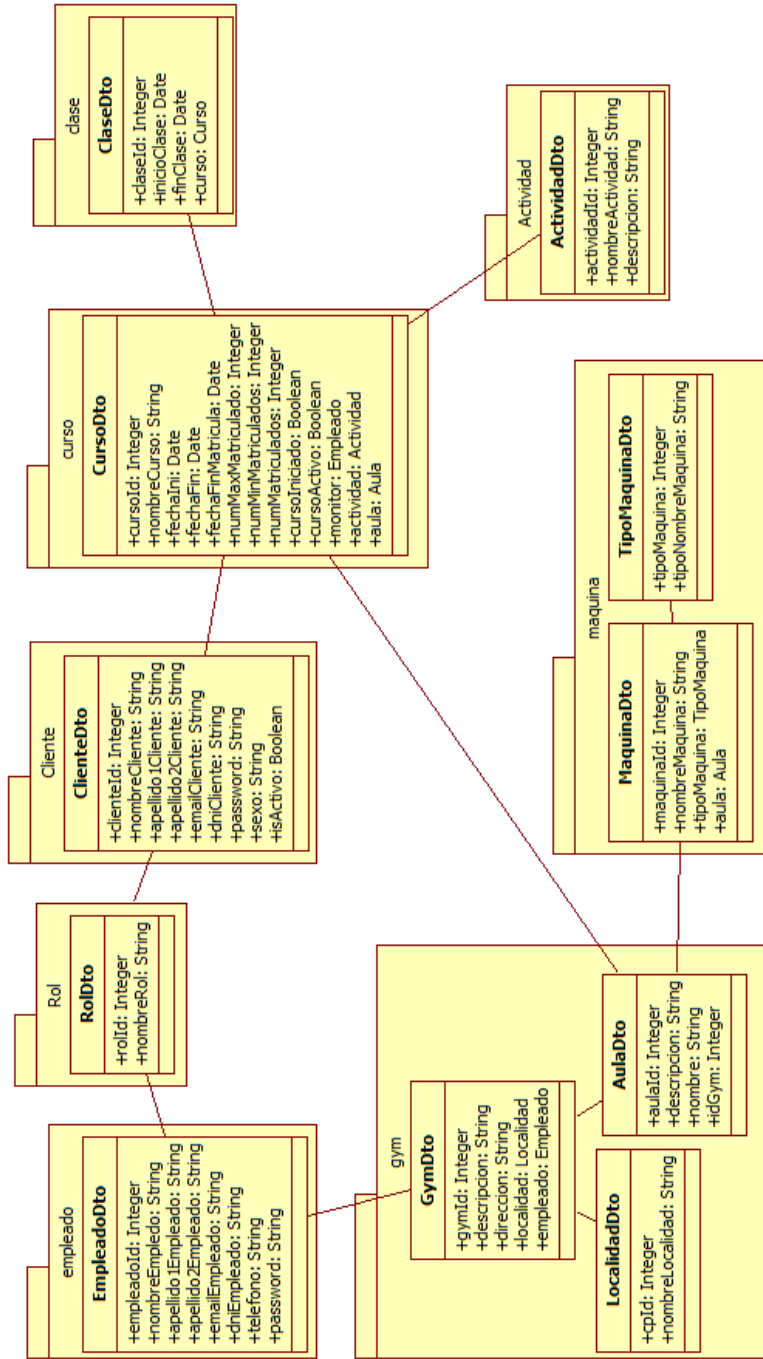


Figura 8.6: Diagrama de clases del paquete "dto".



```
1 @Service
2 @Transactional
3 public class EmpleadoServiceImpl implements EmpleadoService {
4     @Autowired
5     EmpleadoDAO empleadoRepository;
6
7     @Autowired
8     CursoDAO cursoRepository;
9
10    @Autowired
11    GymDAO gymRepository;
12
13    @Override
14    public EmpleadoDto createEmp(final EmpleadoInt empleadoInt) {
15        empleadoInt.setIsActive(true);
16        Empleado emp = new Empleado();
17        if(empleadoInt.getIdJefe() != null) {
18            Empleado empJefe = this.empleadoRepository.
19                findById(empleadoInt.getIdJefe()).get();
20            emp = EntityToDtoConverter.
21                convertEmpleadoDtoToEmpleado(empleadoInt, empJefe);
22        } else {
23            emp = EntityToDtoConverter.
24                convertEmpleadoDtoToEmpleado(empleadoInt, null);
25        }
26        final Empleado empSaved = this.empleadoRepository.save(emp);
27        return
28        EntityToDtoConverter.convertEmpleadoToEmpleadoDto(empSaved);
29    }
}
```

Figura 8.7: Ejemplo del patrón FACADE del servicio *EmpleadoServiceImpl*.

```

1 @RestController
2 @CrossOrigin
3 public class UserController {
4     @Autowired
5     ControllerImpl controller;
6
7     @RequestMapping("/login")
8     public String login(@RequestBody User user) {
9         return "login";
10    }
11
12    @RequestMapping("/user")
13    public Principal user(HttpServletRequest request) {
14        String authToken = request.getHeader("Authorization")
15            .substring("Basic".length()).trim();
16        return () -> new String(Base64.getDecoder()
17            .decode(authToken)).split(":")[0];
18    }

```

Figura 8.8: Extracto del código controlador UserController.

taciones `@Service` y `@Transactional`. La primera anotación le indica a Spring que esta clase es un objeto disponible para ser usado por el controlador, mientras que la segunda anotación indica que cada método de la clase ha de ser ejecutado como si fuera una transacción de base de datos. Los DAOs utilizados en las clases service se anotan con `@Autowired` para que Spring los inicialice automáticamente.

En la figura 8.7 se muestra un ejemplo de las anotaciones utilizadas en el patrón FACADE del servicio *EmpleadoServiceImpl*. En las figuras B.2, B.1, B.3, B.4, B.5, B.6, B.7 podemos ver los diagramas de clases de los distintos service.

#### 8.1.4 Controladores del servidor

Para finalizar el apartado sobre la implementación del servidor, en el paquete controller se han creado dos controladores “User-Controller” y “HomeController”. El primero de ellos cuenta con los endPoint. Uno para el inicio de sesión y otro para obtener los datos del usuario tal y como se puede ver en el siguiente extracto del código de la figura 8.8. Por el contrario, el “HomeController” se encarga de cargar el modelo de datos e invoca a la vista necesaria (figura 8.9).

## 8.2 Implementación del cliente

Para la implementación del cliente se utilizó AngularJS por las mejoras introducidas en la semántica del HTML que en los últimos años han facilitado a los programadores el desarrollo

```
1 @RestController
2 @RequestMapping("/app")
3 public class HomeController implements Controller {
4     @Autowired
5     GymService gymService;
6     @Autowired
7     EmpleadoService empService;
8     @Autowired
9     MaquinaService maquinaService;
10    @Autowired
11    ActividadService actividadService;
12    @Autowired
13    ClienteService clienteService;
14    @Autowired
15    CursoService cursoService;
16    @Autowired
17    ClaseService claseService;
18    @Autowired
19    RolService rolService;
20
21    @Override
22    @RequestMapping(value = "/createEmp", method =
23        RequestMethod.POST, produces = MediaType.APPLICATION_JSON_VALUE)
24    public EmpleadoDto createEmp(@RequestBody EmpleadoDto emp) {
25        return this.empService.createEmp(emp);
26    }
27 }
```

Figura 8.9: Extracto del código del controlador HomeController.

de aplicaciones más fáciles de entenderse e integrarse con otros sistemas. En este sentido, AngularJS dispone de numerosas herramientas para crear el HTML enriquecido que necesita este tipo de proyectos.

AngularJS es una tecnología que, por su utilidad y características, ha sido muy útil en este proyecto porque promueve patrones de diseño adecuados para aplicaciones web y que, básicamente, nos marcan la separación del código de los programas dependiendo de su responsabilidad, repariendo la lógica de la aplicación por capas.

### 8.2.1 Elementos y conceptos de AngularJS

- **Vistas:** Será el HTML y todo lo que sea información y datos.
- **Controladores:** Son los encargados de la parte lógica y de las llamadas Factorías y Servicios, que mueven datos de cara al servidor o memoria local en HTML5.
- **Modelo de la vista:** Son los datos, más los datos adicionales que necesitas para mostrarlos adecuadamente.
- **Módulos:** Nos propone que seamos ordenados con el código y evitar el código espagueti, ficheros grandes con interminables líneas de código, etc.

### 8.2.2 División de AngularJS en dos áreas

- **Parte del HTML:** Es la parte declarativa, con las vistas, así como las directivas y filtros que nos proporciona AngularJS, los que hagamos nosotros mismos o los facilitados por terceros.
- **Parte javascript puro:** Serán los controladores, factorías y servicios.

### 8.2.3 Aplicación de página única (SPA)

Cuando se eligió AngularJS se hizo siguiendo el modelo *Singlepage Application (SPA)* o aplicación de página única. Consiste en una aplicación web o sitio web que cabe en una sola página. En otras palabras, la página web sólo se “carga” una vez con el objetivo de proporcionar una experiencia más fluida a los usuarios. Es una herramienta bastante flexible para resolver el problema de transmisión de grandes cantidades de información en una single page application.

En una SPA todos los códigos de HTML, JavaScript, y CSS se carga de una vez. Podemos conseguir una SPA mediante AngularJS y su sistema de rutas o routes. El enrutado (*routing*) de Angular indica donde está nuestro contenido y cuando un usuario lo pida, debería de encontrarlo y cargarlo.

```
1 var app = angular.module('AppTfg', ['ngRoute', 'ui.bootstrap',  
2   'ngAnimate', 'ngSanitize'])  
3   .config(function ($routeProvider) {  
4     $routeProvider  
5       .when("/", {  
6         controller: "homeController",  
7         templateUrl: "home/home.view.html",  
8         controllerAs: 'vm'  
9       }).when("/login", {  
10        controller: "LoginController",  
11        templateUrl: "home/login.view.html",  
12        controllerAs: 'vm'  
13      })  
14      .when("/empleado", {  
15        controller: "EmpleadoController",  
16        templateUrl: "template/empleado.html"  
17      })  
18    })
```

Figura 8.10: Ejemplo de implementación del servicio *routing* en nuestra aplicación.

Para poder usar el sistema de *routes* de Angular precisaremos el servicio *route* y el módulo *ngRoute*. Para definir rutas usaremos el servicio *route*. Este servicio lo que hace es mapear entre las URLs y los archivos de las vistas. Cuando el valor de éste cambie, y coincida con alguna de las rutas que definamos, se cargará la correspondiente vista. En el ejemplo de la figura 8.10 se muestra como hemos utilizado el *routing* en nuestra aplicación.

## 8.2.4 Servicios en AngularJS

Los servicios son objetos singleton, inyectables por Dependency Injection, donde definimos la lógica de negocio de la aplicación, con el objetivo de que sea reutilizable e independiente de las vistas. Un servicio en AngularJS mantiene los datos durante el tiempo de vida de una aplicación y también se comunican entre diferentes controladores de una manera consistente. En el siguiente apartado explicaremos 2 de los servicios más usados en esta aplicación así como la forma en como hemos utilizado estos 2 servicios.

### Servicio *http*

La interacción de datos entre el servidor y el cliente se ha realizado utilizando el servicio **”http”** integrado en angularJs como servicio que permite hacer peticiones AJAX al servidor. En el ejemplo de la figura 8.11 muestra como crear un empleado en nuestra aplicación utilizando el servicio *http*.

```

1 angular.module("AppTfg")
2   .factory('gymModel', ['$http', function ($http) {
3     var gymModel = {};
4     gymModel.crearEmp = function (emp_IN) {
5       var config = {
6         headers: {
7           'Content-Type': 'application/json'
8         }
9       }
10      var resultadoPetición =
11      $http.post('http://localhost:8080/app/createEmp', emp_IN, config)
12      return resultadoPetición;
13    };

```

Figura 8.11: Ejemplo de aplicación del servicio *http* en nuestra aplicación para crear un empleado.

### Servicio *Scope*

En medio de la parte HTML y JavaScript tendríamos *Scope*, que no es más que un objeto Javascript que puedes extender creando propiedades que pueden ser datos o funciones, y que nos sirve para comunicar desde la parte del HTML a la parte del Javascript y viceversa. Es un servicio importante en nuestra aplicación porque es el servicio *Scope* el que enlaza un elemento de la vista con un modelo JSON disponible en la lógica de negocio de nuestra aplicación. Con este enlazado lo que conseguimos es que cuando nuestro modelo cambie, la vista actualice los datos en el HTML de una forma transparente, sin que nosotros, como desarrolladores, tengamos que indicarle nada más al framework sobre el enlazado. De igual forma ocurre cuando el cambio del modelo se realiza desde la vista. Cuando el usuario modifica un dato de un modelo enlazado, este se actualiza transparentemente. El siguiente ejemplo muestra como usamos el servicio *Scope* en nuestra aplicación.

### Código en Html:

En el siguiente código se aprecia como utilizamos la propiedad "responses" que no es más que un atributo del **Scope** declarado en el Controlador.

```

1 <tbody>
2   <tr ng-repeat="response in responses"
3     <td>{{response.nombreEmpleado}}</td>
4     <td>{{response.apellido1Empleado}}</td>
5     <td>{{response.telefono}}</td>
6     <td>...

```

**Código en JavaScript:**

En el siguiente código se aprecia como hemos creado en el **Scope** una propiedad "responses" inyectándole el valor devuelto en la petición del servicio. La propiedad "responses" es la variable que se usará en archivo Html del código anterior.

```
1 gymService.getEmp($scope.email).then(function (data) {  
2     $scope.responses = angular.copy(data);  
3     ...
```

**8.2.5 Componentes de la aplicación en AngularJS**

La aplicación que se desarrolló en AngularJS está dividida en componentes, estos componentes se han encapsulados en controladores que a través de un servicio se podrán comunicar distintos componentes.

Es una buena práctica que cada archivo del proyecto tiene que tener una única responsabilidad, sin importar el número de líneas que pueda tener. Cada responsabilidad hay que encapsularla y aislarla para conseguir código ravioli.

Aunque, en un principio, tengamos la tentación de unificar varias funcionalidades con la excusa de que son pocas líneas, es muy importante empezar a separar el código desde el principio. A medida de que el proyecto vaya creciendo, aumentarán las líneas de código y si no seguimos esta buena práctica, nos encontraremos con código espagueti.

La diferencia entre un código espagueti y uno ravioli se ve al hacer cambios en el código. En el espagueti, esa parte del código puede estar "enredado" con otras funcionalidades y provocar mal funcionamiento o conflictos indeseados en otras partes del código. Sin embargo, en un código ravioli al estar todo empaquetado individualmente, cada cambio afectaría únicamente a esa parte; sin provocar daños colaterales.

Es por ello que hemos implementado "**factorías**" que son como contenedores de código que podemos usar en nuestra app desarrollado con AngularJS. Son un tipo de servicio, "service" en Angular, con el que podemos implementar librerías de funciones o almacenar datos.

Cuando las usamos tienen la particularidad de devolvernos un dato, de cualquier tipo. Lo común es que nos devuelvan un objeto de Javascript donde podremos encontrar datos (propiedades) y operaciones (métodos). Con diferencia de los controladores, las factorías tienen la característica de ser instanciados una única vez dentro de las aplicaciones, por lo que no pierden su estado. Por tanto, son un buen candidato para almacenar datos en nuestra aplicación que queramos usar a lo largo de varios controladores, sin que se inicialicen de nuevo cuando se cambia de vista.

Angular consigue ese comportamiento usando el patrón "Singleton" que básicamente

quiere decir que, cada vez que se necesite un objeto de ese tipo, se enviará la misma instancia de ese objeto en lugar de volver a instanciar un ejemplar. A continuación se muestra como en nuestra aplicación se crea una factoría que es inyectada en el controlador como un servicio,este servicio inyectado en el controlador será el encargado de gestionar la información a compartir.

### Código del controlador:

En el siguiente código de nuestra aplicación se aprecia como se inyecta un servicio en el controlador,el valor devuelto por el servicio (en el ejemplo sería "data") es inyectado a una propiedad del Scope ("reponses").

```

1 app.controller('EmpleadoController', function ($scope, gymService){
2   gymService.getEmp($scope.email).then(function (data) {
3     $scope.responses = angular.copy(data);
4     ...

```

### Código del servicio:

En el siguiente código se aprecia como se crea una factory que será inyectada como un servicio en el controlador.Este factory será el encargado de resolver las peticiones asíncronas realizadas por el servicio **gymModel** a través del metodo **.then()**.

```

1 angular.module("AppTfg")
2   .factory('gymService', ['gymModel', function (gymModel) {
3     var gymService = {};
4     gymService.getEmp = function (email) {
5       return gymModel.getEmp(email).then(function (data) {
6         return data
7       }, function (error) {
8         return $q.reject(gymService.crearObjetoError(error))
9       });
10    };
11    ...
12    return gymService

```

### Código del modelo:

En el siguiente código de nuestra aplicación se aprecia como se crea una factoría encargada se hacer las peticiones **Rest** usando el servicio **http** que se inyecta como un servicio llamado **gymModel** en la factory del código anterior.



```
1 angular.module("AppTfg")
2   .factory('gymModel', ['$http', function ($http) {
3     var gymModel = {};
4     gymModel.getEmp = function (email) {
5       var config = {
6         headers: {
7           'Content-Type': 'application/json'
8         }
9       }
10      var resultadoPeticion =
11      $http.get('http://localhost:8080/app/emp/' + email, config)
12      return resultadoPeticion;
13    };
14    return gymModel;
15  })
```

### 8.3 Interfaz Gráfica para el usuario

El diseño de la página web está conformado por un *footer* común para las páginas de login e index y un cuerpo que en la página principal está conformado por cinco pestañas. Cada una de estas pestañas contiene uno de los distintos módulos ya citados. El rol del usuario logueado determinará que pestañas aparecerán visibles.

A continuación se explica cada uno de los módulos y pestañas de la aplicación así como las secciones que están contenidas en cada pestaña.

Para seguir el ejemplo se ha designado los usuarios y los roles para cada usuario:

- jeremy@app.com con el rol "Administrador"
- jenny@app.com con el rol "Encargado"
- cesar@app.com con el rol "Encargado"
- lorena@app.com con el rol "Monitor"
- adriana@app.com con el rol "Cliente"

#### 8.3.1 Gestión Empleados

Esta pestaña está conformada por un botón que permite registrar altas de empleados en el sistema y una tabla mostrará información de todos los empleados si el usuario logueado es el administrador, para el resto de usuarios sólo se mostrará la información del usuario logueado.

Name	Apellido	Telefono	Actualizar	Eliminar	
Juan	Ortiz	658779585	<a href="#">Details</a>		
Jeremy	Ortiz	660338023	<a href="#">Details</a>		
Cesar	Ortega	655998542	<a href="#">Details</a>		
Jenny	Ortega	685275970	<a href="#">Details</a>		
Johanna	Ortiz	651364632	<a href="#">Details</a>		
Jhonatan	Bocanegra	659876321	<a href="#">Details</a>		
Lorena	Torres	674802350	<a href="#">Details</a>		

Figura 8.12: Captura conforme se muestra que el administrador puede crear empleados y visualizar todos los empleados en la tabla.

Cada usuario puede cambiar la información de sus datos pero solo el administrador podrá modificar el dato jefe y rol para todos los demás empleados, es decir, solo el usuario con el rol administrador podrá asignar y designar tanto el campo jefe como los roles.

En la siguiente figura 8.12 se muestra una captura para el usuario "jeremy@app.com" con el rol **Administrador** en la que se puede apreciar que es el único usuario que puede dar de alta a los empleados y visualizar en la tabla a todos los empleados, además en la parte superior se observa que tiene disponible todas las pestañas para la gestión del gimnasio.

En la figura 8.13 se muestra una captura para el usuario "jenny@app.com" con el rol

Name	Apellido	Telefono	Actualizar	Eliminar	
Jenny	Ortega	685275970	<a href="#">Details</a>		

« < 1 > »

Figura 8.13: Captura conforme se muestra que el encargado no puede crear empleados y solo puede visualizar sus datos como empleado.

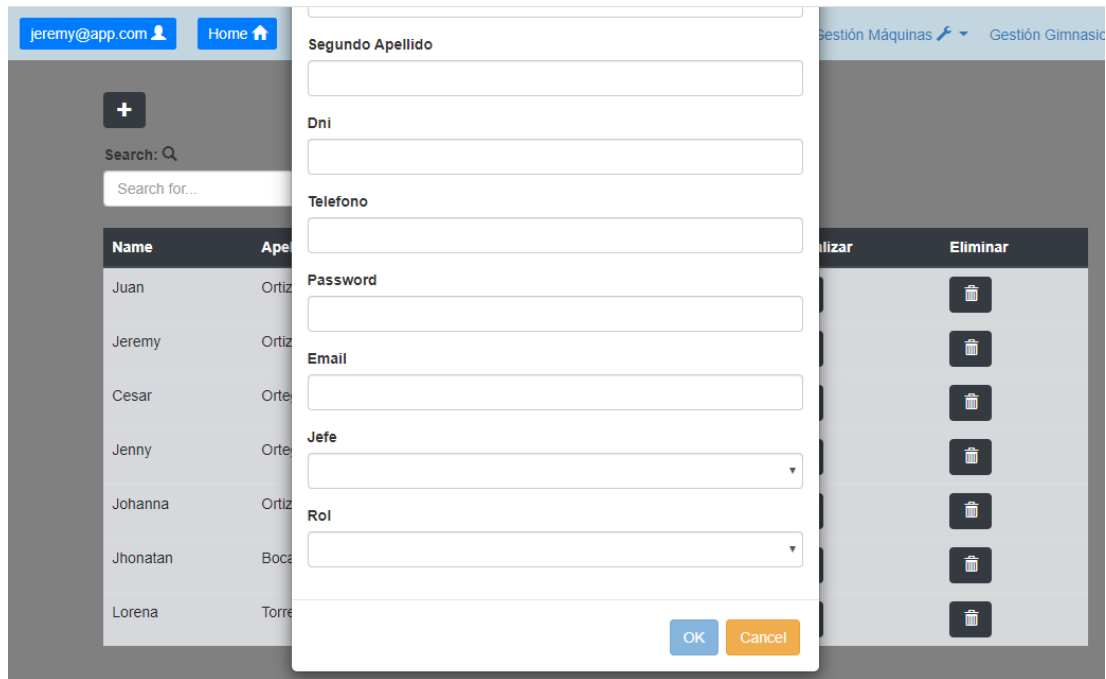


Figura 8.14: Captura conforme se muestra que el administrador puede modificar los datos rol y jefe de cada empleado.

**Encargado** en la que se puede apreciar que no tiene habilitado el botón para dar de alta a los empleados y además solo puede visualizar sus datos como empleado, ya que como se explicó anteriormente sólo el usuario con rol **Admin** puede visualizar todos los empleados.

En la figura 8.14 se puede apreciar que el usuario **"jeremy@app.com"** es el único que puede modificar los datos **"jefe"** y **"rol"** para cualquier empleado.

### 8.3.2 Gestión Cursos

Esta pestaña mostrará un menú desplegable con 3 Secciones: **Actividades, Cursos** y **Clases**. Dependiendo del rol del usuario logueado se tendrá acceso a cada Sección. Para el usuario con el rol **"Administrador"** y **"Encargado"** serán accesibles las **3 secciones**, pero con la diferencia que el usuario con el rol **Encargado** solo visualizará los cursos y clases pertenecientes al centro donde es encargado.

Para el usuario con el rol **"Monitor"** sólo serán accesibles las secciones Cursos y clases, pudiendo visualizar solo los cursos y clases que él impartira.

Dependiendo de cada rol, se mostrará diferentes datos de cada Sección. Para el rol **Administrador** se podrá dar de alta a una **Actividad** y se podrá mostrar todos los cursos y clases disponibles en toda la cadena del Gimnasio.

Para el rol **Encargado**, se podrá dar de alta a una **Actividad**, y sólo se podrá visualizar los

The screenshot shows a web application interface for a gym management system. At the top, there is a navigation bar with the user 'jeremy@app.com' and several menu items: 'Home', 'Gestión Empleados', 'Gestión Cursos' (which is selected and has a dropdown menu open showing 'Actividades', 'Cursos', and 'Clases'), 'Gestión Clientes', 'Gestión Máquinas', and 'Gestión Gimnasio'. Below the navigation bar is a search bar with the text 'Search for...'. The main content area displays a table of courses with the following data:

curso	Fecha de Inicio	Fecha de Fin	Fecha Final Matricula	Monitor	Actividad	Actualizar	Eliminar	Listar alumnos	
curso zumba	2020-03-28	2020-06-22	2020-02-12	Lorena	Zumba	<a href="#">Details</a>			
Curso Pilates	2020-03-08	2020-03-28	2020-02-20	Jhonatan	Pilates	<a href="#">Details</a>			
Curso Body Combat	2020-03-28	2020-06-07	2020-02-20	Jhonatan	Body combat	<a href="#">Details</a>			

At the bottom of the table, there is a pagination control showing '1' in a blue box, indicating the first page of results.

Figura 8.15: Captura conforme se muestra que el administrador tiene acceso a las secciones sobre la gestión Cursos, además visualiza todos los cursos disponibles.

cursos y clases disponibles del gimnasio en el que es encargado como se explicó anteriormente.

Para el rol **Monitor**, se podrá visualizar sólo los cursos y clases que impartirá dicho monitor.

En la figura 8.15 se muestra una captura en la que se puede apreciar como el usuario "jeremy@app.com" tiene acceso a las 3 secciones, además se observa como visualiza todos los cursos disponibles.

En la figura 8.16 se muestra una captura de pantalla en la que se puede apreciar como el usuario "jenny@app.com" sólo puede ver los cursos que tiene disponible en el gimnasio del que es encargado.

En la figura 8.17 se muestra una captura como el usuario "lorena@app.com" visualiza los cursos que impartirá como monitor y en la figura 8.18 las clases que impartirá de dichos cursos.

### 8.3.3 Gestión Clientes

Esta pestaña está conformada por un botón que permite registrar **alta** de **clientes** en el sistema y una tabla que mostrará información de todos los clientes si el usuario logueado tiene rol **Administrador** o el rol **Encargado**., Si el usuario logueado tiene el rol **cliente**, mostrará en la tabla sólo los datos del cliente logueado.

En la figura 8.19 se muestra una captura de como el administrador "jeremy@app.com" tiene acceso a todos clientes así como poder dar de alta a nuevos clientes.

Para cada cliente mostrado en la tabla se visualizará 3 botones: **Actualizar**, **Inscripción** y **Cursos**. Si el usuario tiene el rol **Administrador** ó **Encargado** se mostrará un botón a ma-

The screenshot shows a user interface for 'Gestión Cursos'. The user is logged in as 'jenny@app.com'. The navigation bar includes 'Home', 'Gestión Empleados', 'Gestión Cursos', 'Gestión Máquinas', and 'Gestión Gimnasio'. A dropdown menu is open over 'Gestión Cursos', showing 'Actividades', 'Cursos', and 'Clases'. Below the navigation is a search bar and a table of courses.

curso	Fecha de Inicio	Fecha de Fin	Fecha Final Matricula	Monitor	Actividad	Actualizar	Eliminar	Listar alumnos	
Curso Pilates	2020-03-08	2020-03-28	2020-02-20	Jhonatan	Pilates	Details			
Curso Body Combat	2020-03-28	2020-06-07	2020-02-20	Jhonatan	Body combat	Details			

Below the table is a pagination control showing '1' as the current page.

Figura 8.16: Captura conforme se muestra que el encargado tiene acceso a las secciones sobre la gestión Cursos, pero solo visualiza los cursos disponibles en el gimnasio del que es encargado.

The screenshot shows a user interface for 'Gestión Cursos'. The user is logged in as 'lorena@app.com'. The navigation bar includes 'Home', 'Gestión Empleados', 'Gestión Cursos', 'Gestión Máquinas', and 'Gestión Gimnasio'. A dropdown menu is open over 'Gestión Cursos', showing 'Cursos' and 'Clases'. Below the navigation is a search bar and a table of courses.

curso	Fecha de Inicio	Fecha de Fin	Fecha Final Matricula	Actividad
curso zumba	2020-03-28	2020-06-22	2020-02-12	Zumba

Below the table is a pagination control showing '1' as the current page.

Figura 8.17: Captura conforme se muestra que el Monitor tiene acceso a las secciones sobre la gestión Cursos pero solo visualiza los cursos que impartirá.

The screenshot shows the user interface for 'lorena@app.com'. The navigation bar includes 'Home', 'Gestión Empleados', and 'Gestión Cursos'. A search bar is present with the text 'Search: Q'. Below the search bar is a table with the following data:

Inicio	Fin	Curso		Actualizar	Eliminar
2020-03-28	10:02AM	curso zumba	<a href="#">Details</a>		
2020-03-29	10:14AM	curso zumba	<a href="#">Details</a>		
2020-03-30	10:44AM	curso zumba	<a href="#">Details</a>		
2020-03-31	10:16AM	curso zumba	<a href="#">Details</a>		

Below the table is a pagination control showing '1' in a blue box, indicating the first page of results.

Figura 8.18: Captura conforme se muestra que el Monitor **lorena@app.com** tiene acceso a las secciones sobre la gestión clases, pero solo visualiza clases que impartirá.

The screenshot shows the user interface for 'jeremy@app.com'. The navigation bar includes 'Home', 'Gestión Empleados', 'Gestión Cursos', 'Gestión Clientes', 'Gestión Máquinas', and 'Gestión Gimnasio'. A search bar is present with the text 'Search: Q'. Below the search bar is a table with the following data:

Name	Apellido		Actualizar	Inscripción	Eliminar	Cursos
Jair	Bulai	<a href="#">Details</a>				
Alejandro	Perez	<a href="#">Details</a>				
Adriana	Gervasini	<a href="#">Details</a>				

Below the table is a pagination control showing '1' in a blue box, indicating the first page of results.

Figura 8.19: Captura conforme se muestra que el Admin puede dar de alta a nuevos clientes y visualizar a todos los clientes.

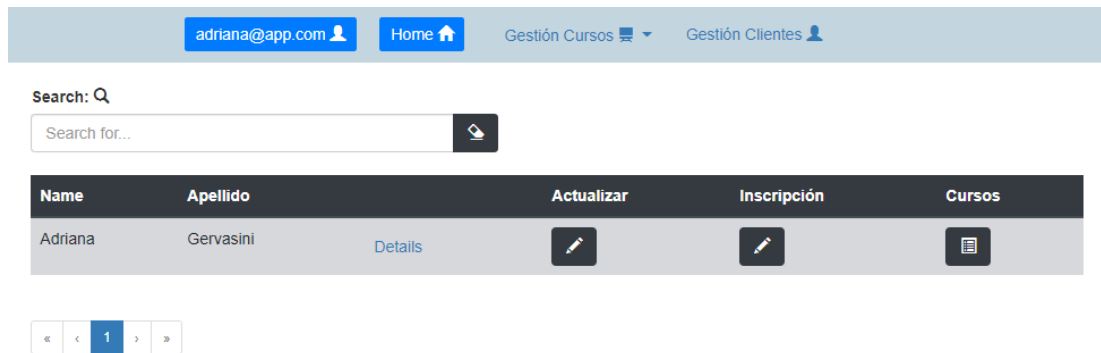


Figura 8.20: Captura conforme se muestra que el cliente "adriana@app.com" puede visualizar solo sus datos y además los botones para, poder modificar sus datos, poder inscribirse y mostrar los cursos del que está inscrito.

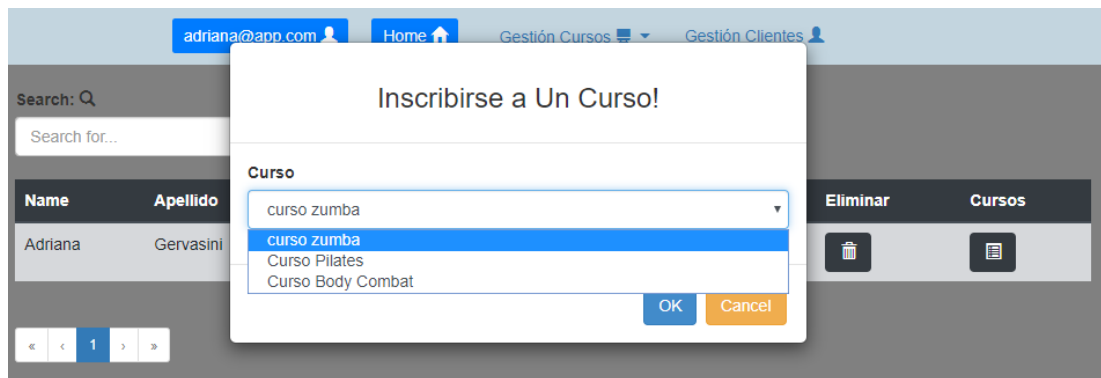


Figura 8.21: Captura conforme se muestra como el usuario "adriana@app.com" se inscribe a un curso.

yores que será **Dar de Baja** El primero de ellos, **Actualizar**, modificará los datos personales del cliente.

El siguiente botón, **Inscripción**, mostrará una nueva ventana y cargará una lista de cursos disponibles(cursos que estén activos y haya plazas libres) en un desplegable y que los clientes podrán elegir e inscribirse .

El siguiente botón, **Dar de Baja**, solo estará disponible para el usuario con el rol de "Administrador" y "Encargado" y será para borrar al cliente de la aplicación.

El último botón, **Cursos**, será para visualizar todos los cursos en los que está inscrito el cliente.

En la figura 8.20 se muestra una captura visualizando los datos del cliente "adriana@app.com" y los botones Actualizar, Inscripción y Cursos.

En la figura 8.21 se muestra una captura de como el cliente "adriana@app.com" se inscribe a un curso cargando en una lista desplegable los cursos disponibles.

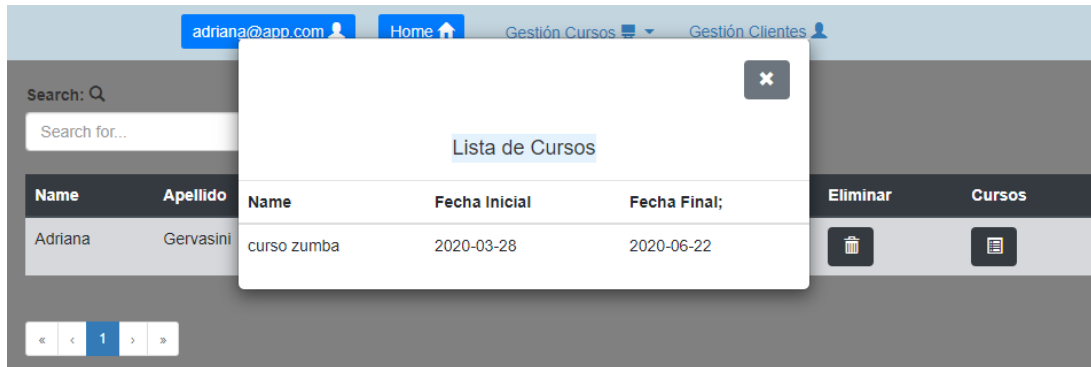


Figura 8.22: Lista de los cursos al que está inscrito el cliente "adriana@app.com".

En la figura 8.22 se muestra una captura de como el cliente "adriana@app.com" se inscribe a un curso cargando en una lista los cursos disponibles.

#### 8.3.4 Gestión Máquina

Esta pestaña está conformada por 2 secciones: **Tipo de Máquina y Máquina**. Los usuarios que tienen acceso a estas secciones son los que tienen el rol "Administrador" y "Encargado". Estos 2 usuarios podrán crear Máquinas así como sus tipos pero cada encargado podrá crear máquinas solo para el gimnasio donde es encargado mientras que el administrador podrá hacerlo en cualquier centro. Cada máquina distinta estará presente en las aulas a las que fue seleccionada.

En la figura 8.23 se muestra una captura en la que se visualiza las máquinas para el gimnasio del que es encargado "jenny@app.com" y en la figura 8.24 se muestra una captura en la que se visualiza las máquinas para el gimnasio del que es encargado "cesar@app.com".

#### 8.3.5 Gestión Gimnasio

Esta pestaña consta de 3 secciones: **Localidad, Gimnasio y Aula**. la sección **Localidad y gimnasio** solo es accesible por el usuario con el rol "Administrador", mientras que la sección **aula** es accesible por el administrador y el encargado, pero aquellos con rol "Encargado" solo podrán visualizar las aulas del gimnasio en el que es encargado.

En la figura 8.25 se muestra una captura donde el usuario "jeremy@app.com" con rol administrador da de alta a una localidad.

En la figura 8.26 se muestra captura donde el usuario "jeremy@app.com" con rol "Administrador" da de alta a un Gimnasio, destacar que a la hora de elegir un encargado, solo se puede elegir aquellos usuarios que no tienen el rol de encargado anteriormente, es por eso que en la figura en el desplegable no se visualizan empleados que ya son encargados, puesto que un



## CAPÍTULO 8. IMPLEMENTACIÓN

The screenshot shows a web application interface for user 'jenny@app.com'. The navigation bar includes 'Home', 'Gestión Empleados', 'Gestión Cursos', 'Gestión Máquinas', and 'Gestión Gimnasio'. A search bar is present with the text 'Search for...'. Below the search bar is a table with the following data:

Nombre	Tipo	Aula	Eliminar
MaquinaAula 71A	10	71	<a href="#">Details</a>
MaquinaAula 71B	10	71	<a href="#">Details</a>
Banco press.	10	71	<a href="#">Details</a>
Prensa de piernas.	10	72	<a href="#">Details</a>
Máquina de dorsales.	10	73	<a href="#">Details</a>
Bicicletas estáticas.	10	71	<a href="#">Details</a>

At the bottom of the table, there is a pagination control showing '1' in a blue box, indicating the current page.

Figura 8.23: Lista de las máquinas para el encargado "jenny@app.com".

The screenshot shows a web application interface for user 'cesar@app.com'. The navigation bar includes 'Home', 'Gestión Empleados', 'Gestión Cursos', 'Gestión Máquinas', and 'Gestión Gimnasio'. A search bar is present with the text 'Search for...'. Below the search bar is a table with the following data:

Nombre	Tipo	Aula	Eliminar
MaquinaAula 61	10	61	<a href="#">Details</a>
Ciclo Indoor	10	62	<a href="#">Details</a>
Elípticas	10	61	<a href="#">Details</a>
Cintas de correr	10	62	<a href="#">Details</a>
Máquinas de remo	10	62	<a href="#">Details</a>

At the bottom of the table, there is a pagination control showing '1' in a blue box, indicating the current page.

Figura 8.24: Lista de las máquinas para el encargado "cesar@app.com".

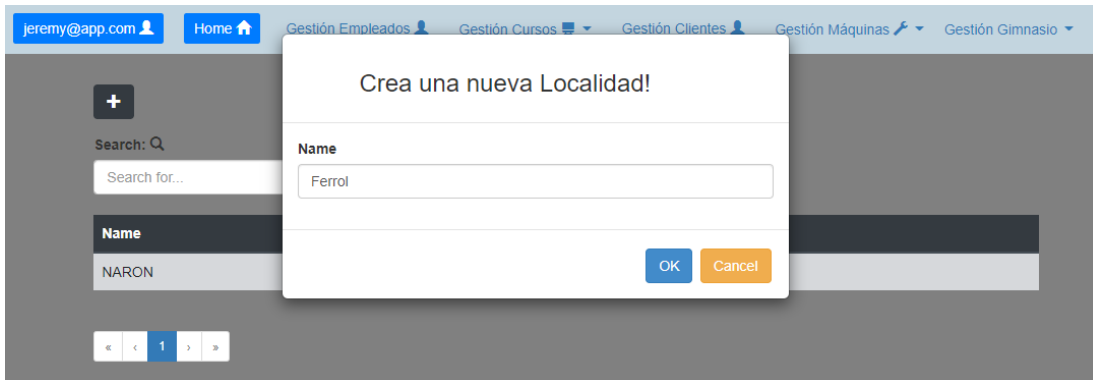


Figura 8.25: Captura donde el usuario "jeremy@app.com" da de alta a una localidad

usuario solo puede ser encargado de un centro y recordar que el usuario **lorena@app.com** tiene el rol Monitor y por eso puede ser candidata a ser Encargada.



Figura 8.26: Captura donde el usuario "jeremy@app.com" da de alta a un responsable del Gimnasio.

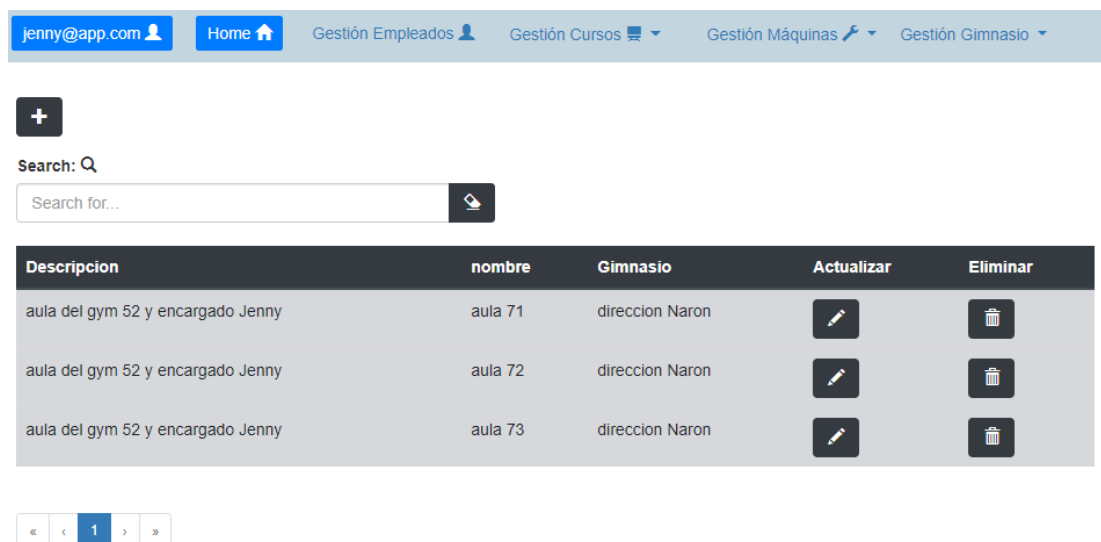


Figura 8.27: Captura donde el usuario "jenny@app.com" muestra las aulas del Gimnasio del que es encargada.



Figura 8.28: Captura donde el usuario "cesar@app.com" muestra las aulas del Gimnasio del que es encargado.

# Pruebas

---

**E**N este capítulo se indican las pruebas que se han realizado a lo largo del desarrollo de la aplicación. En un proyecto de desarrollo de software pueden aparecer errores en cualquiera de las etapas del ciclo de vida, algunos de ellos incluso permanecen sin ser descubiertos, de ahí la importancia de las pruebas.

### 9.1 Pruebas unitarias

En esta aplicación se han realizado pruebas unitarias en dos de las tres capas existentes de la parte servidor (DAOs y controlador), abarcando la mayoría de métodos implementados, centrándose especialmente en aquellos de mayor complejidad puesto que son propensos a errores. Se han utilizado las herramientas JUnit, para las pruebas unitarias del código relativo a los DAOs y Postman para las peticiones publicadas en la API Rest. A continuación se incluyen ejemplos de una prueba de cada capa. Ejemplo de pruebas de métodos en un dao con la herramienta JUnit se puede ver en la figura 9.1.

Un ejemplo de prueba de métodos del controlador REST con la herramienta Postman se puede ver en la figura 9.2.

### 9.2 Pruebas de aceptación

Al final de cada una de las tres iteraciones se realizaron pruebas de aceptación por parte del alumno y el director de proyecto sobre los módulos finalizados. Estas pruebas se realizan sobre el sistema de manera manual abarcando todos los requisitos establecidos.

```

1 @RunWith(SpringRunner.class)
2 @DataJpaTest
3 @AutoConfigureTestDatabase(replace = Replace.NONE)
4 public class GimnasioTest {
5
6     @Autowired
7     GymDAO gymDao;
8     @Autowired
9     EmpleadoDAO empleadoDAO;
10
11     @Test
12     public void updateGymDAO() {
13         final Empleado empleado
14             =this.empleadoRepository.findById(1).get();
15         final Gym gym =this.empleadoRepository.findById(1).get();
16         gym.setEmpleado(empleado);
17         this.gymDAO.save(gym)
18         assertEquals(this.gymDAO.findById(1).get(), gym)
19     }
20     ...
21 }

```

Figura 9.1: Ejemplo de la interfaz *GymDAO* con dos de las búsquedas implementadas.

The screenshot shows a web browser interface for a REST client. The request is a GET to localhost:8080/app/gym. The response is a JSON object with the following structure:

```

28 {
29   "gymId": 52,
30   "descripcion": "encargado es Jenny",
31   "direccion": "direccion NARON",
32   "localidad": {
33     "cpId": 5,
34     "nombreLocalidad": "NARON"
35   },
36   "empleado": {
37     "empleadoId": 22,
38     "nombreEmpleado": "Jenny",
39     "apellido1Empleado": "Ortega",
40     "apellido2Empleado": "Cuba",
41     "emailEmpleado": "jenny@app.com",
42     "dniEmpleado": "327850990",
43     "telefono": "685275970",
44     "password": "jjjj",
45     "isActivo": true,

```

Figura 9.2: Captura de pantalla de una petición GET a la url /gym.

### **9.3 Pruebas de integración**

Se han realizado pruebas de integración entre el servidor y el cliente de la aplicación en cada una de las iteraciones, para comprobar que la comunicación entre ellos era la correcta.

### **9.4 Otras pruebas**

No se estimó oportuno la realización de otro tipo de pruebas, como podrían ser pruebas de carga, puesto que la aplicación no ha sido desplegada en un servidor público.





# Conclusiones y trabajo futuro

---

Este capítulo resume las principales conclusiones y el trabajo futuro del proyecto realizado.

## 10.1 Conclusiones

El objetivo de este proyecto consistía en desarrollar una aplicación web para la gestión de una cadena de gimnasio. Se ha desarrollado un software que provee a los usuarios de las herramientas necesarias para la gestión de empleados, cursos, clientes, máquinas y los centros del gimnasio. Todo ello con una interfaz amigable e intuitiva que ha reducido considerablemente la curva de aprendizaje media de una aplicación de este tipo.

El diseño realizado ha sido modular y la aplicación final se puede ejecutar en prácticamente cualquier sistema operativo y con una amplia variedad de navegadores web. De forma más específica, se han cumplido todos los objetivos especificados en la sección 1.2, desarrollando una aplicación web que permite a los usuarios gestionar los procesos relativos a:

- **Gestión de Empleados:** se ha desarrollado un módulo que dota al usuario de las funcionalidades necesarias para la gestión de sus empleados. Permite dar de baja, modificar sus datos personales así como el rol dentro del gimnasio o dar de alta a cualquier empleado bajo una jerarquía de mando.
- **Gestión de Cursos:** se ha desarrollado un módulo que dota al usuario las funcionalidades necesarias para la gestión de sus cursos, tanto para los clientes, monitores y encargados que gestionarán los cursos del centro en que es encargado.
- **Gestión de Gimnasio:** se ha implementado un módulo especialmente para el encargado con el fin de que pueda gestionar las aulas, cursos y clases ocurridas en su centro.
- **Gestión de Máquinas:** se ha implementado un módulo para la gestión de las máquinas para cada centro donde el encargado será el que tome las decisiones como eliminación, reubicación, asignación acerca de dichas máquina en un centro.

- **Gestión de Clientes:** se ha desarrollado un módulo que permitirá a los clientes poder visualizar su datos, inscribirse y consultar tanto los cursos como las clases a las que está inscrito.

En cuanto a los conocimientos adquiridos a lo largo del desarrollo de este trabajo, destacan la planificación de un proyecto, la estimación de costes del mismo, el aprendizaje de nuevas tecnologías y la aplicación de una metodología de desarrollo. En resumen, la experiencia ha resultado de lo más satisfactoria, no sólo por los conocimientos adquiridos a lo largo del desarrollo de este proyecto, sino por el esfuerzo y la dedicación durante estos meses que se ven recompensados al alcanzar los objetivos que se marcaron en un principio.

## 10.2 Trabajo futuro

En esta última sección se exponen algunas de líneas de trabajo futuras que podrían llegar a incluirse en la aplicación para dotarla de un mayor número de funcionalidades:

- Internacionalización de la aplicación, puesto que la versión entregada tan sólo está disponible en castellano
- Añadir geolocalización a los lugares y permitir visualizarlos en visores de mapas interactivos (por ej. Google Maps).

# Material adicional

---

EN este apéndice se explica brevemente el proceso de instalación y ejecución de la aplicación.

## A.1 Instalación y ejecución de la aplicación

### A.1.1 Despliegue del servidor Web

Una vez terminado la parte servidor de la aplicación se procederá a su despliegue, para ejecutar una aplicación de spring-boot usando Maven, simplemente ejecutamos por línea de comandos: **mvn spring-boot:run**. En la figura A.1 se muestra el despliegue de la aplicación del servidor.

### A.1.2 Instalación Node Package Manager (NPM)

*npm* es un gestor de paquetes, que gracias a él podremos tener cualquier librería disponible con solo una línea de código, npm nos ayudará a administrar nuestros módulos, distribuir paquetes y agregar dependencias de una manera sencilla.

Antes de empezar deberíamos instalar el gestor de paquetes NPM disponible en la página oficial [13], una vez instalado ejecutaríamos el comando **npm init** para crear un archivo llamado **package.js** que es un archivo para administrar las dependencias del proyecto. En este archivo se definen y manejan características como:

- Nombre del Proyecto
- versión
- Dependencias
- Repositorio

```

C:\Users\51991\eclipse-workspace\appGimnasio>mvn spring-boot:run
[INFO] Scanning for projects...
[INFO] ----- es.udc.tfg:appGimnasio -----
[INFO] Building appGimnasio 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.2.1.RELEASE:run (default-cli) > test-compile @ appGimnasio >>>
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ appGimnasio ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 2 resources
[INFO] Copying 12698 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ appGimnasio ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 105 source files to C:\Users\51991\eclipse-workspace\appGimnasio\target\classes
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ appGimnasio ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\51991\eclipse-workspace\appGimnasio\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ appGimnasio ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\51991\eclipse-workspace\appGimnasio\target\test-classes
[INFO]
[INFO] <<< spring-boot-maven-plugin:2.2.1.RELEASE:run (default-cli) < test-compile @ appGimnasio <<<
[INFO]
[INFO]
[INFO] --- spring-boot-maven-plugin:2.2.1.RELEASE:run (default-cli) @ appGimnasio ---
[INFO] Attaching agents: []

  ____  __  __
 / ___/  / /  / /
/ /   / /_/ / /_/ /
/ /___/ __/ / __/ /
/____/_/ /_/ /_/ /

:: Spring Boot ::      (v2.2.1.RELEASE)

```

Figura A.1: Ejemplo de como se despliega la aplicación servidor con el comando Maven.

- Autores
- Licencia

En el código de la figura A.2 se explica el *package.js* con la configuración de esta aplicación. Del archivo creado (*package.js*) se explicarán 3 secciones:

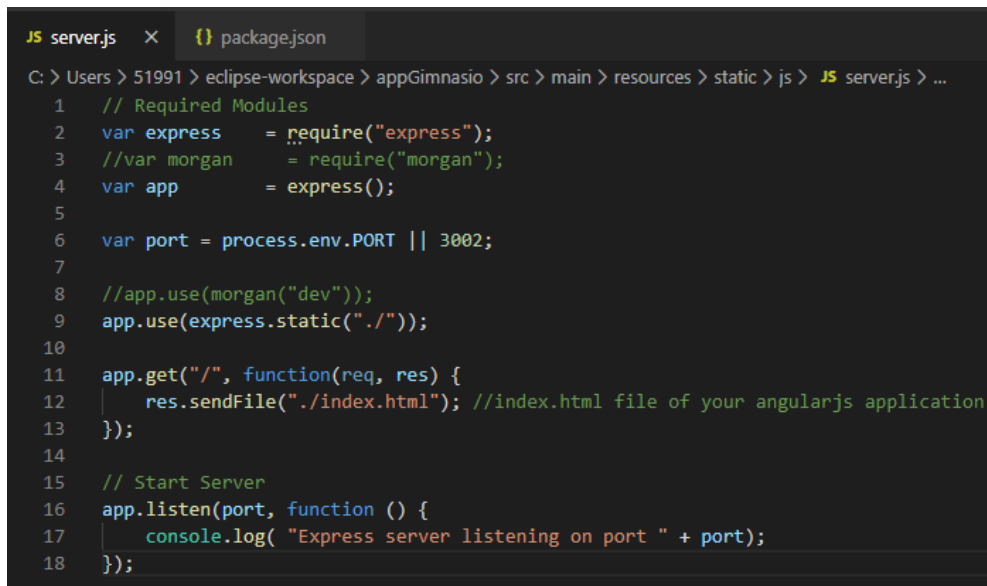
- **main:** Un string que se ha nombrado *server.js* que define la ruta del archivo principal o punto de entrada del proyecto. Este es el archivo que una persona recibirá si importa el proyecto al suyo. En la figura A.3 se muestra el código del archivo que hemos utilizado para el main. Un ejemplo rápido es que no está diciendo que la aplicación cliente está escuchando en el puerto 3002.
- **scripts:** Es un objeto que indica comandos que podemos correr dentro de nuestro proyecto, asociándolos a una palabra clave para que npm los reconozca cuando queramos ejecutarlos. En la aplicación ejecutaremos el comando **npm start** tal como se muestra en la figura A.4.
- **dependencies:** Un objeto que guarda los nombres y versiones de cada dependencia que hemos instalado dentro del proyecto. De esta forma, cada vez que alguien obtenga una

```

1 {
2   "name": "package",
3   "version": "1.0.0",
4   "description": "",
5   "main": "server.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1",
8     "start": "node server.js"
9   },
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "angular": "^1.7.9",
14    "angular-datatables": "^8.0.0",
15    "angular-ui-grid": "^4.8.3",
16    "bootstrap": "^4.4.1",
17    "datatables.net": "^1.10.20",
18    "datatables.net-dt": "^1.10.20",
19    "express": "^4.17.1",
20    "http-server": "^0.12.1",
21    "jquery": "^3.4.1"
22  }
23 }

```

Figura A.2: Ejemplo que explica el *package.js* de esta aplicación



```

JS server.js  X  {} package.json
C: > Users > 51991 > eclipse-workspace > appGimnasio > src > main > resources > static > js > JS server.js > ...
1  // Required Modules
2  var express = require("express");
3  //var morgan = require("morgan");
4  var app     = express();
5
6  var port = process.env.PORT || 3002;
7
8  //app.use(morgan("dev"));
9  app.use(express.static("./"));
10
11 app.get("/", function(req, res) {
12   res.sendFile("./index.html"); //index.html file of your angularjs application
13 });
14
15 // Start Server
16 app.listen(port, function () {
17   console.log( "Express server listening on port " + port);
18 });

```

Figura A.3: Ejemplo del archivo declarado en la sección **main** en el archivo *package.js*.

```
<28/01/2020 16:22          98.544 package-lock.json
26/01/2020 18:11          511 package.json
08/01/2020 12:22 <DIR>      nanacacunar
05/01/2020 17:23          470 server.js
11/01/2020 12:34          277 style.css
01/02/2020 23:22 <DIR>      template
          17 archivos          204.676 bytes
          9 dirs 150.513.856.512 bytes libres

C:\Users\51991\eclipse-workspace\appGimnasio\src\main\resources\static\j>npm start
> package@1.0.0 start C:\Users\51991\eclipse-workspace\appGimnasio\src\main\resources\static\js
> node server.js

Express server listening on port 3002
```

Figura A.4: Ejemplo del comando **npm start** para desplegar la aplicación cliente.

copia de este proyecto, y corra el comando `npm install`, se instalarán todas las dependencias que aquí estén definidas y por ende, no habrán problemas de compatibilidad al correr el proyecto. En nuestra aplicación, en la figura A.2 se observa las dependencias utilizadas para esta aplicación.

## Apéndice B

# Servicios

---

EN este apéndice se muestran los diagramas de clase de los servicios implementados en el servidor. Se referencian en el apartado 8.1.4.

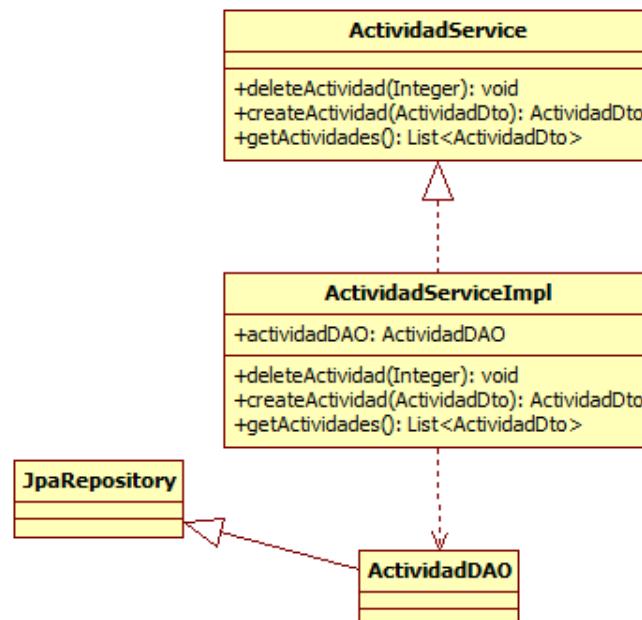


Figura B.1: Diagrama de clases de ActividadService.

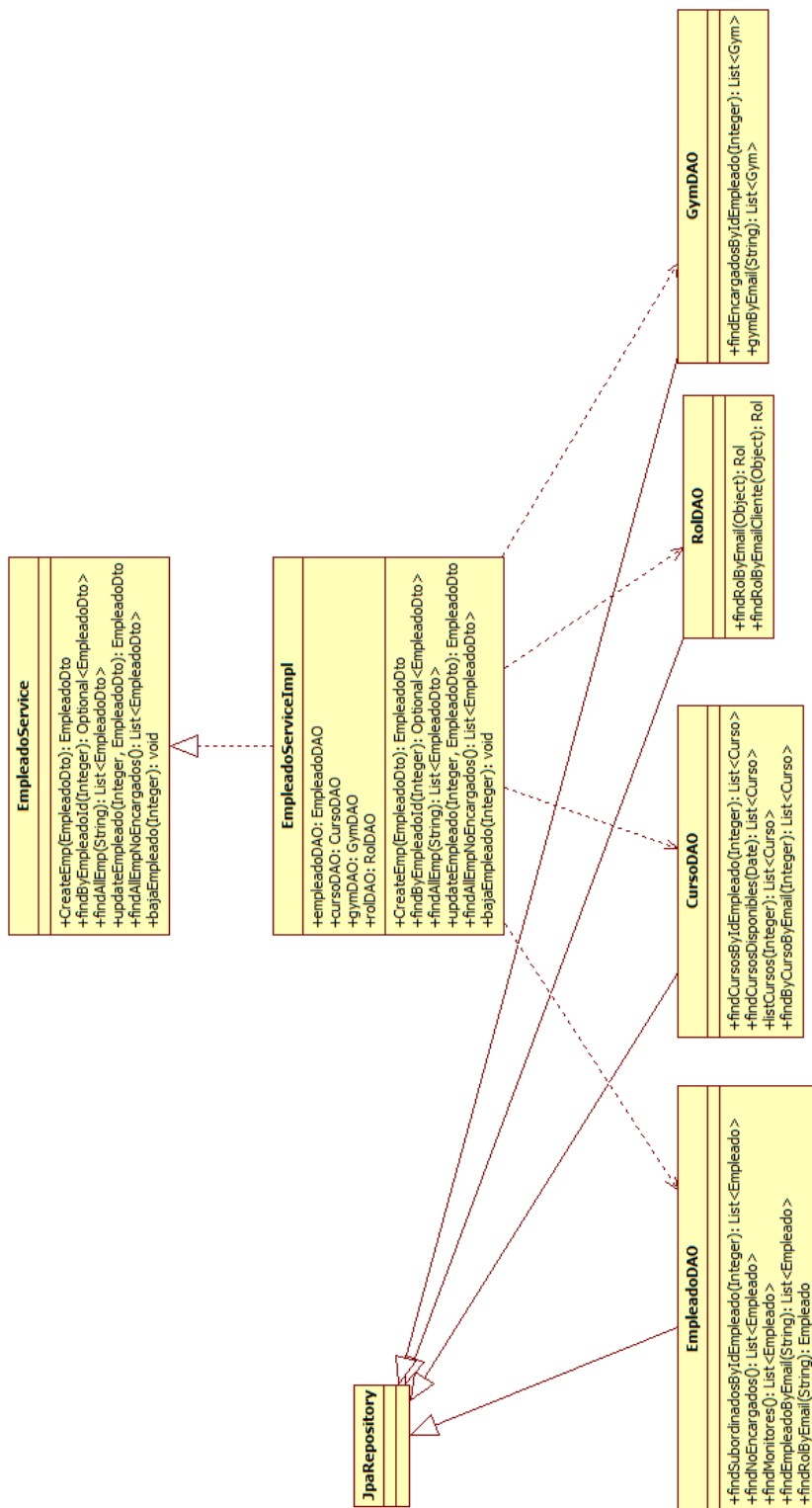


Figura B.2: Diagrama de clases de EmpleadoService.



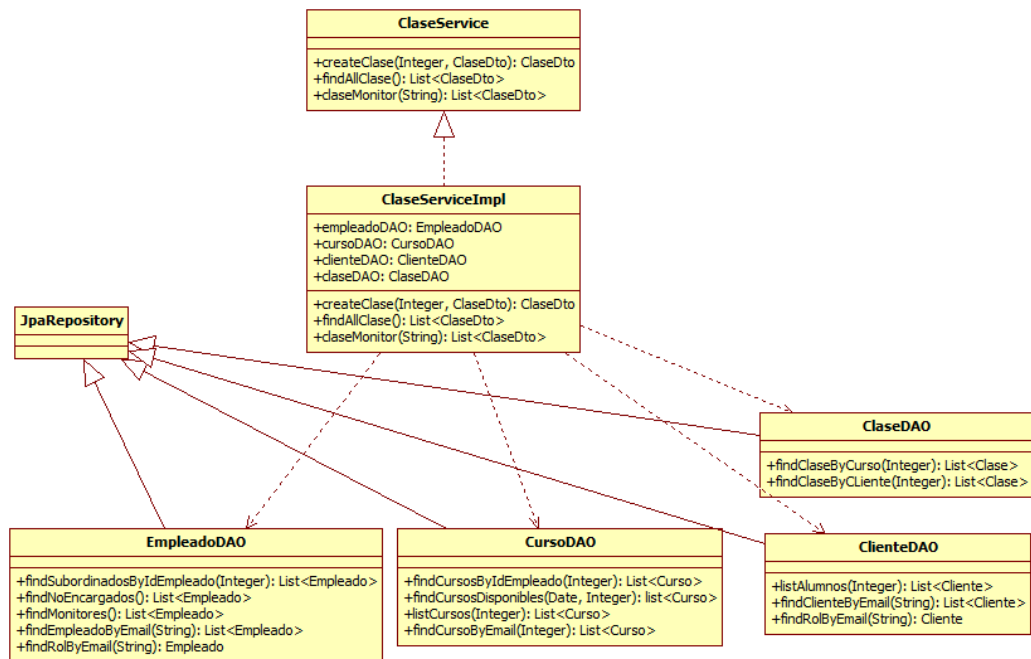


Figura B.3: Diagrama de clases de ClaseService.

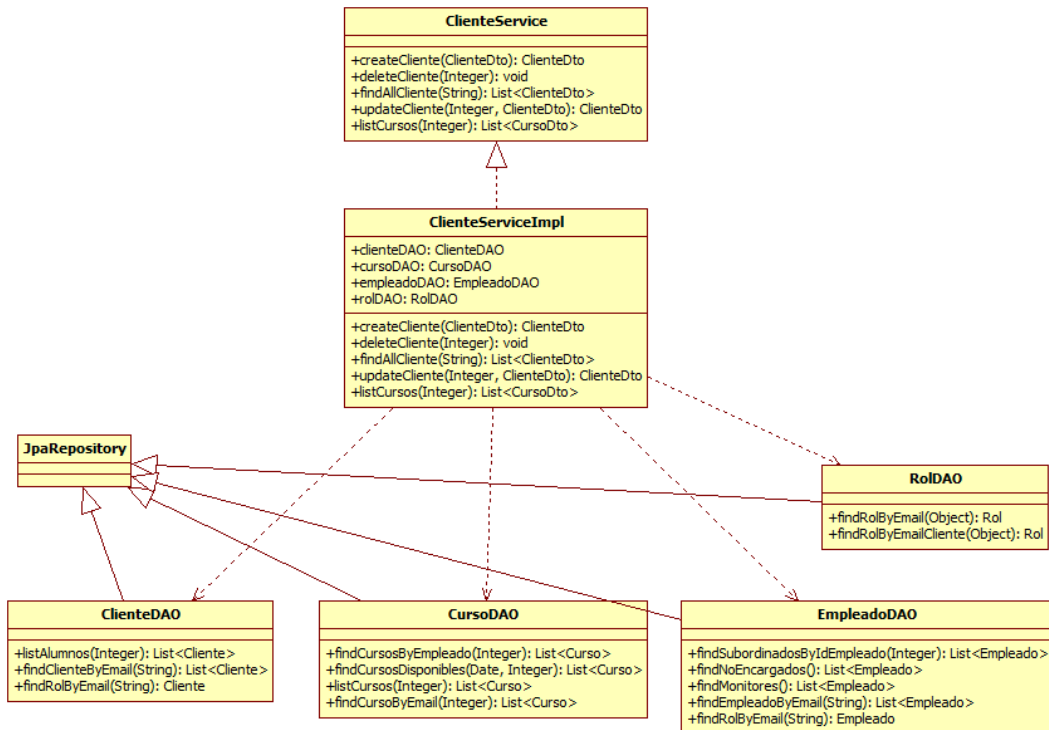


Figura B.4: Diagrama de clases de ClienteService.

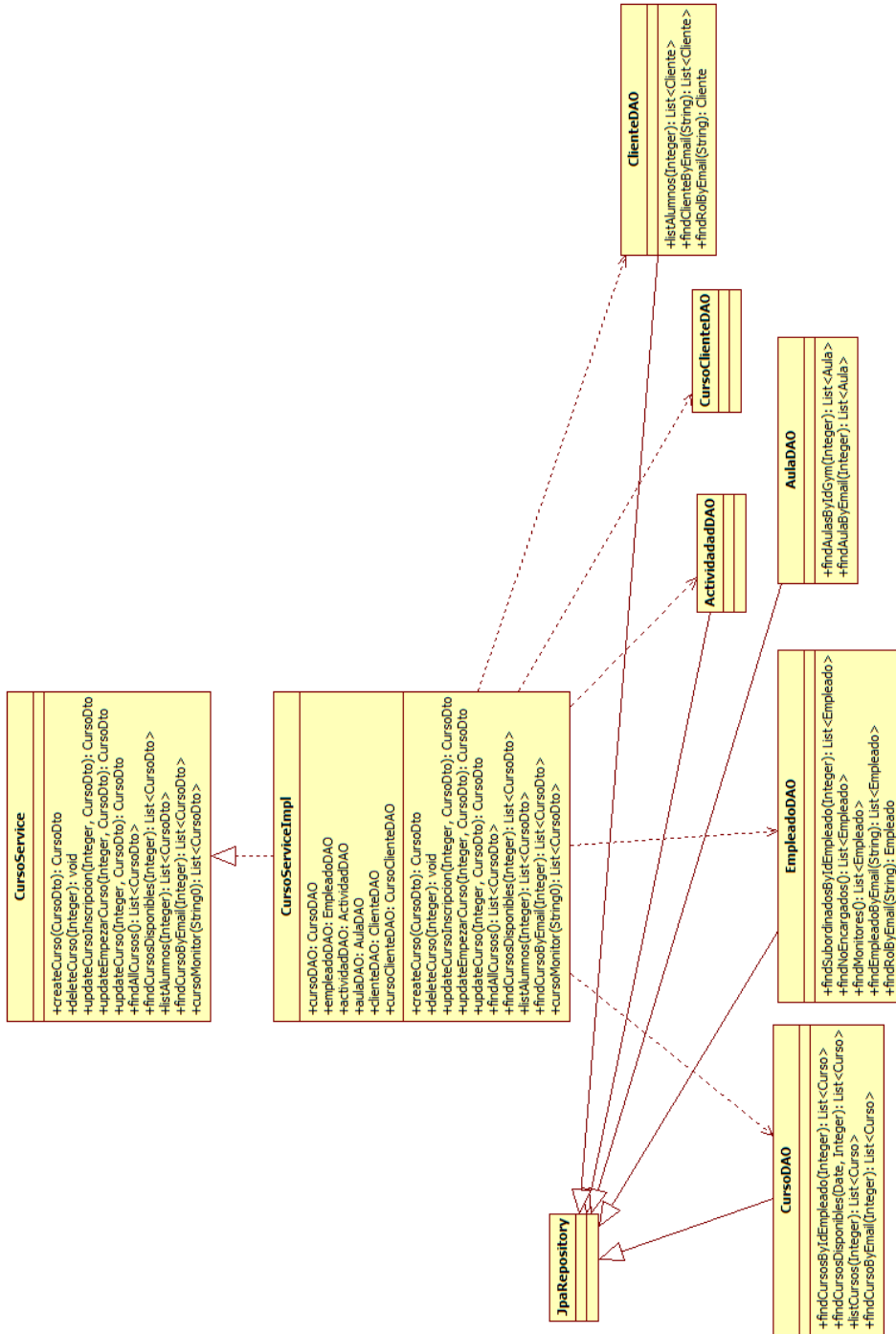


Figura B.5: Diagrama de clases de CursoService.

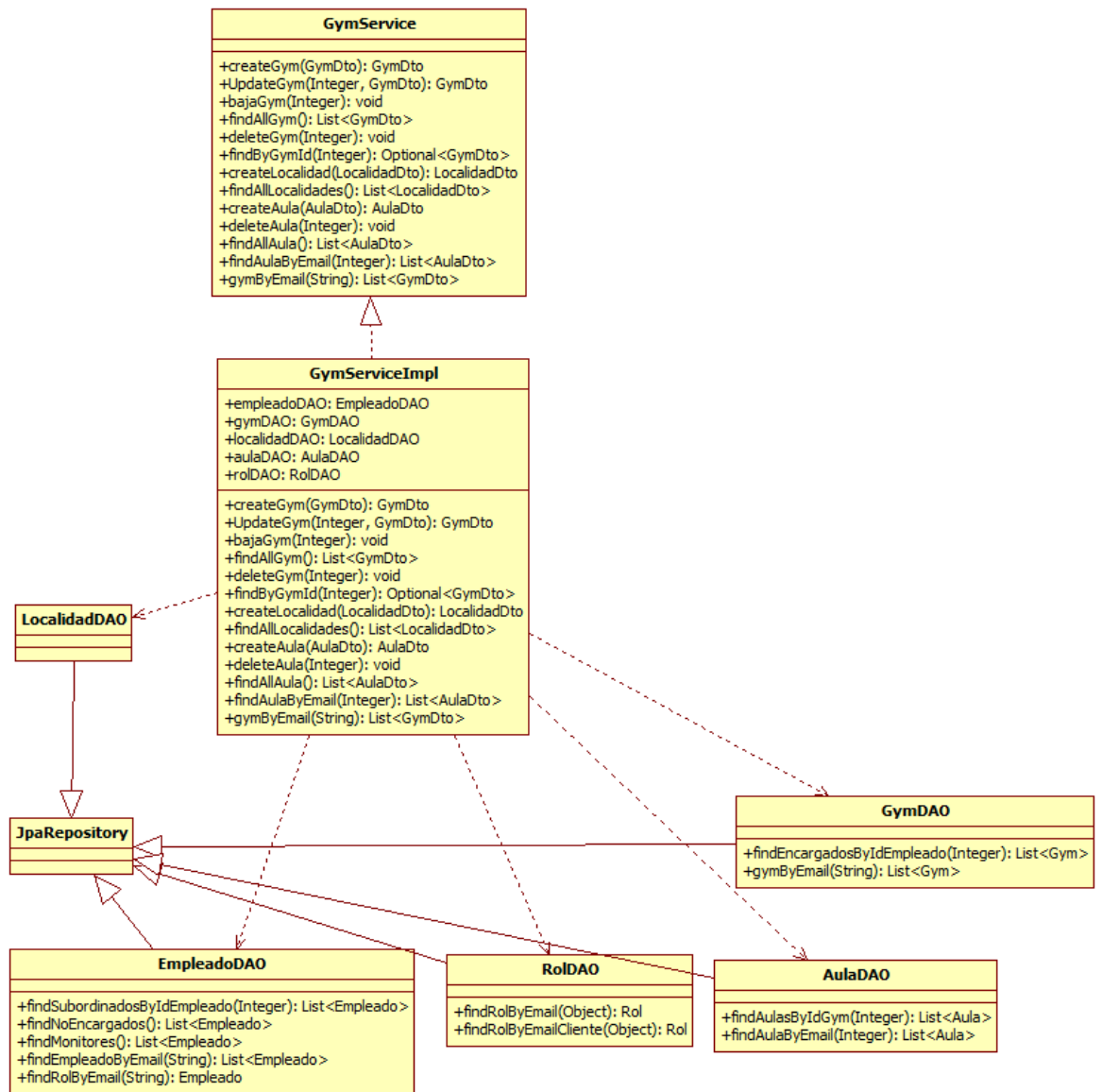


Figura B.6: Diagrama de clases de GimnasioService.

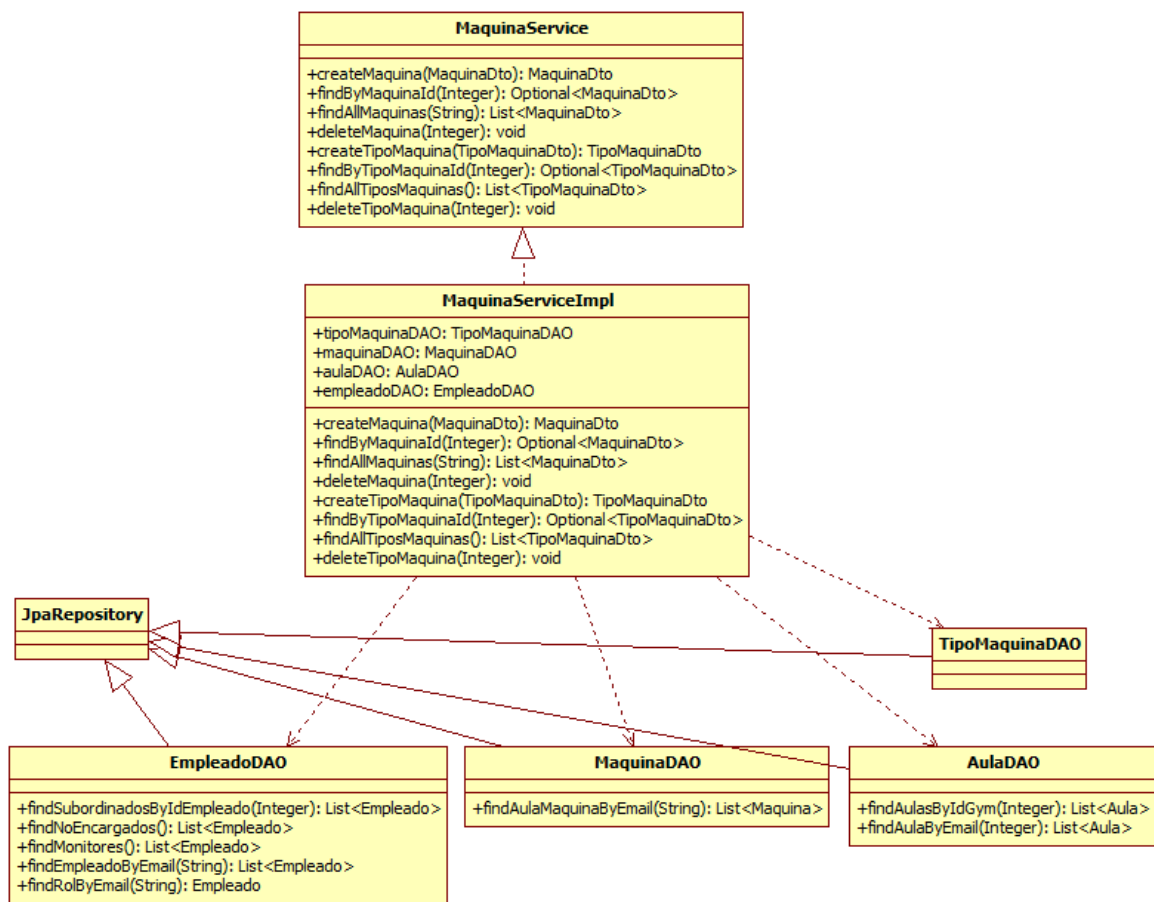


Figura B.7: Diagrama de clases de MaquinaService.

# Bibliografía

---

- [1] I. Jacobson, G. Booch, and J. Rumbaugh, *El proceso unificado de desarrollo de software*, ser. Fuera de colección Out of series. Pearson Educación, 2000. [En línea]. Disponible en: <https://books.google.es/books?id=zHKbQgAACAAJ>
- [2] M. Stonebraker, *PostgreSQL Global Development Group*, 1996. [En línea]. Disponible en: <https://https://www.postgresql.org/download/>
- [3] R. Cadenhead, *Java 8*. Anaya, 2000.
- [4] C. Walls, *Spring in Action*. Manning, 2018.
- [5] —, *Spring Boot in Action*. Manning, 2015.
- [6] V. autores, “Baeldung,” 2019. [En línea]. Disponible en: <https://www.baeldung.com/>
- [7] L. Atencio., *The Joy of JavaScript*. Manning, 2019.
- [8] V. autores., “W3schools. the world’s largest web developer site,” ser. En línea, 2019. [En línea]. Disponible en: <https://w3schools.com//>
- [9] A. Buscaglia, *Introducción a AngularJS*, ser. Manual de referencia, 2018. [En línea]. Disponible en: <https://www.belatrixsf.com/>
- [10] —, “Bootstrap. the most popular html, css and js library in the web,” ser. En línea, 2019. [En línea]. Disponible en: <https://getbootstrap.com/>
- [11] V. Massol, *jUnit in Action*. Manning, 2004.
- [12] I. J. J. Rumbaugh and G. Boock, *El lenguaje unificado de modelado*, ser. Manual de referencia. Addison Wesley, 2006.
- [13] I. Z. Schlueter., *sistema de gestión de paquetes software libre,”* ser. En línea. [En línea]. Disponible en: <https://www.npmjs.com/>

