2023

# Memoria proyecto CFGS

DESARROLLO DE APLICACIONES WEB

JONATHAN DÍAZ MENÉNDEZ

# Índice

# 1 Introducción

## 1.1 Objetivos

A través de esta memoria se tiene como propósito mostrar de una forma clara y más concisa la estructura del proyecto de fin de curso para el ciclo formativo de Desarrollo de Aplicaciones Web.

La página tratará sobre la gestión de autoescuelas, que servirá como un punto para centralizar la información necesaria para llevar el negocio, desde los alumnos que están suscritos al servicio, a los profesores que allí trabajan y los coches que se utilizan, pudiendo visualizar información necesaria y modificable sobre cada uno de ellos y estableciendo relaciones entre estos.

El objetivo final es crear una aplicación funcional que permita gestionar las autoescuelas con comodidad y facilidad, con interfaces intuitivas y agradables para la vista y que permita guardar información de una forma más segura y precisa.

## 1.2 Contexto

Hace algo menos de dos años entré a una autoescuela en mi villa para sacarme el carné de conducir de tipo B. Estuve todo el verano y varios meses más hasta mayo del siguiente año en aquella autoescuela entre los estudios para el examen teórico y el examen práctico.

La gestión del examen teórico estaba bien, principalmente porque era casi todo online con aplicaciones que estaban a disposición de los conductores de España para realizar test y aprender de forma progresiva, pero una vez que empecé a dar las clases prácticas ya entré en conocimiento de cómo se gestionaban las cosas en una autoescuela.

Por lo general, las operaciones se escribían en papel, tanto para las clases de los alumnos, como las notas, información sobre los coches… etc.

No era extraño ver a mi profesor recibiendo llamadas o llamando a su secretaria para informarse de la clase que tenía después, o de si un alumno estaba disponible para cualquier cosa, incluso preguntando sobre revisiones de los coches.; o a su secretaria tardando algo más de lo usual al buscar las clases que tenían programadas los alumnos de la autoescuela para encontrar huecos. En ese momento no veía nada mejorable.

Una vez que en el segundo curso del ciclo se nos propuso crear un proyecto, al pensar en una buena idea se me vino a la cabeza todo aquello y de esta forma propuse crear una aplicación que mejorase todas esas situaciones para que todo se realice de forma rápida, y de esta forma el profesor puede estar más centrado en sus clases y la información quedará mejor guardada.

## 1.3 Planteamiento del proyecto

El proyecto se divide en dos partes perfectamente distinguibles, el front-end, lo que el usuario ve; y el back-end, lo que el usuario ignora.

### 1.3.1 Front-end

Para la parte de front-end del proyecto, se ha empleado HTML, Thymeleaf, CSS, JavaScript, jQuery y Bootstrap, con el objetivo de crear interfaces de usuario bonitas, intuitivas y eficientes, mediante las cuales se pueda informar al usuario de lo que realiza y los errores que puedan ocurrir, además por supuesto de mostrarle toda la información, que es la principal función de la aplicación.

### 1.3.2 Back-end

En el back-end nos encontramos con MySQL como sistema gestor de Bases de Datos en la que almacenar la información que será mostrada por el front-end, Java como lenguaje de programación para construir todos los métodos, clases, controladores… en el IDE Spring Tool Suite.

También JavaScript, que se puede utilizar tanto en front-end como en back-end, para el manejo de algunas solicitudes en el lado del servidor.

Por último, Docker, para permitir la mayor portabilidad posible del programa a casi cualquier equipo informático.

# 2 Requisitos de la aplicación

## 2.1 Introducción

Este apartado constará de una descripción detallada del funcionamiento del sistema, con requisitos que especifican qué debe hacer la aplicación en determinadas situaciones.

## 2.2 Requisitos

### 2.2.1 Requisitos funcionales

La aplicación tendrá tres tipos de usuarios, distinguidos entre sí por la asignación de roles a cada uno de ellos:

Administrador

Obtiene sus privilegios del rol "ROLE_ADMIN". Es el usuario con mayores capacidades del sistema, pudiendo ejecutar absolutamente todas las funciones disponibles en él. Es capaz de borrar, editar, insertar y ver todas las entidades, desde alumnos y coches, hasta los propios usuarios. Este usuario está pensado para que sea el dueño o la persona que gestione la autoescuela.

Tiene un panel exclusivo para su uso como una forma de centralizar las operaciones de todas las entidades en una sola página, para facilitar la gestión de su autoescuela, también puede crear usuarios y asignarles roles, y ver todas las clases en el calendario, por lo que su capacidad es total.

Profesor

Distinguido de los demás por el rol "ROLE_TEACHER", está en el segundo lugar en la jerarquía del sistema. No tendrá ninguna capacidad de creación, eliminación y edición de registros, no obstante, podrá verlos para informarse sobre los alumnos a los que imparte clase y los vehículos que utiliza.

El profesor estará obligatoriamente vinculado a uno de los profesores presentes en la base de datos mediante los identificadores.

En el calendario, la visualización de las clases se limitará a aquellas en las que el profesor al que el usuario está vinculado participe, pudiendo ver la ahora y el alumno al que imparte clase.

También tendrá una ventana exclusiva para él llamada "Mis alumnos", en la que aparecerán solo los alumnos a los que imparta clase, de esta manera podrá acceder a la información que necesite con rapidez, especialmente útil al añadir notas al alumno al que esté impartiendo.

Alumno

El tercero en la jerarquía del sistema posee el rol "ROLE_USER". Es el usuario registrado con mayores limitaciones de la aplicación. Al igual que el profesor, no podrá crear, eliminar ni editar registros, ni tampoco podrá ver la gran mayoría de ellos, ya que la información estará limitada sólo a aquella con la que él se encuentre relacionado, como las clases, o su perfil.

De esta forma, tendrá acceso al calendario para poder ver sus clases, pero no las de los demás ni tampoco podrá editarla.

Usuario no registrado

No tiene ningún privilegio y solo podrá ver la pantalla de inicio. Tendrá la opción de registrarse e iniciar sesión solamente como un usuario normal.

### 2.2.2 Requisitos no funcionales

Para mejorar la carga de datos, se implementarán DTOs donde sea posible, evitando así la sobrecarga de información innecesaria. Esta optimización no solo incrementará la eficiencia, velocidad y rendimiento del sistema, sino que también mejorará la experiencia del usuario al acceder a los datos de una forma más rápida.

Se realizará una carga inicial de datos, que incluirá profesores, alumnos, coches, clases, usuarios y roles para el correcto funcionamiento del sistema sin inicializar. Esta se producirá al visitar la sección de inicio.

La aplicación podrá ser ejecutada en cualquier ordenador, teniendo como únicos requisitos indispensables la conexión a Internet y un navegador desde el que acceder a ella, si bien el comportamiento podría variar en los navegadores menos utilizados.

### 2.2.3 Requisitos de Interfaces Externas

#### *2.2.3.1 Interfaces de los usuarios*

Los usuarios compartirán las mismas interfaces, la única diferencia que presentarán será las limitaciones en la visualización, por ejemplo, el administrador tendrá un panel exclusivo o el usuario no registrado no podrá ver ningún perfil o calendario.

El header de la aplicación será un elemento común para todos los usuarios, desde el administrador hasta el usuario no registrado.

#### *2.2.3.2 Interfaces hardware*

Para el correcto uso de la aplicación se necesitarán ciertos elementos hardware. Para introducir la información será necesario contar con un teclado y para navegación y visualización de datos, un ratón y una pantalla respectivamente.

#### *2.2.3.3 Interfaces software*

Como ya fue mencionado en el punto 2.2.2, a pesar de que la aplicación se ejecute de forma local para su presentación, será necesario contar con un navegador y una conexión a Internet para acceder a la aplicación cuando se despliegue de forma pública.

#### *2.2.3.4 Interfaces de comunicaciones*

En caso de ser desplegada de forma pública, la aplicación utilizará el protocolo HTTPS para mantener la seguridad de los datos junto con una conexión TCP/IP.

### 2.2.4 Requisitos de rendimiento

Para garantizar un buen rendimiento en la aplicación, se procurará utilizar un código limpio y sencillo, que no realice instrucciones innecesarias, además de realizar la carga solo de los datos que sean necesarios.

La base de datos debe estar optimizada para poder responder con rapidez, para lo cual se han empleado prácticas tales como guardar en ellas las rutas de las imágenes, y que las propias imágenes se encuentren en el servidor, de esta forma la carga se realiza más velozmente.

Las imágenes también cuentan con limitaciones de tamaño, como una medida adicional para optimizar el rendimiento.

### 2.2.5 Atributos

#### *2.2.5.1 Seguridad*

Para mantener la seguridad de los datos, es esencial mantener la jerarquía en el sistema, y limitar las acciones de acuerdo con ella, además de asegurarse de que los datos se entregan de la forma correcta, para que no surjan errores a la hora de manipularlos.

Para el primer caso, se realizan comprobaciones a nivel de código sobre la autoridad de un usuario al realizar una determinada acción, una vez comprobada, dependiendo de si se cumple o no esa autoridad, ejecuta una u otra acción. Además, se incluye una clase de Seguridad que limita el acceso a determinadas URL en función de los roles que tenga el usuario autenticado actual.

En el segundo caso, se realizan validaciones en el lado de cliente, como los vistos en Desarrollo Web Entorno Cliente, mediante el uso de clases de validación de Bootstrap y JavaScript, con los cuales no se permitirá enviar información que no cumpla el formato establecido.

En el lado del servidor, visto en Desarrollo Web Entorno Servidor, se realizan también validaciones de formatos, conversiones (véase la matrícula o el DNI, se pueden introducir con letras minúsculas, pero serán convertidas a mayúsculas) y comprobaciones de campos vacíos.

No solo eso, si no que las contraseñas del usuario son encriptadas y desencriptadas en la capa de negocio, para poder mantener la privacidad de los usuarios.

De esta manera, se facilita la interacción al usuario, a la vez que se evitan una gran cantidad de errores poco importantes que impedirían al usuario realizar las acciones que necesita.

#### *2.2.5.2 Facilidades de mantenimiento*

Para asegurar las correctas relaciones entre datos y su persistencia, se ha creado un panel para el administrador, donde se centralizan casi todas las operaciones en un solo lugar, facilitándole la gestión y actualización de todos los datos.

El panel del administrador es una parte importante del sistema al proporcionar una interfaz intuitiva y de fácil uso, diseñado específicamente para satisfacer las necesidades diarias del administrador. Desde este punto de control centralizado, el administrador puede realizar diversas tareas, como ingresar, visualizar y eliminar datos.

Sin embargo, carece de la posibilidad de editarlos, esto es así porque esta función se encuentra en los apartados de alumnos, coches, profesores… etc. Por lo que si el administrador quisiera editar una de estas entidades, muy probablemente acudirá a su sección para buscar su identificador y en esa misma sección se encuentra la función de edición, liberando carga de elementos visuales en el panel del administrador.

#### *2.2.5.3 Portabilidad*

La aplicación será ejecutable en cualquier ordenador que cumpla con los requisitos no funcionales especificados en el apartado 2.2.2. Sin embargo, la portabilidad entre estos dispositivos sería tediosa de hacerlo simplemente transfiriendo los archivos, ya que de esta manera habría que cambiar muchas configuraciones de la comunicación entre la base de datos y la aplicación puesto que esta configuración puede variar mucho entre ordenadores, o incluso puede no existir.

Para solucionar esto está Docker, mediante el cual podremos crear dos contenedores, uno para la aplicación y otro para la base de datos, utilizando Docker Compose para orquestar estos servicios.

Así, el único requisito para que la aplicación funcione en cualquier ordenador será contar con la presencia de Docker y Docker Compose en el ordenador anfitrión.

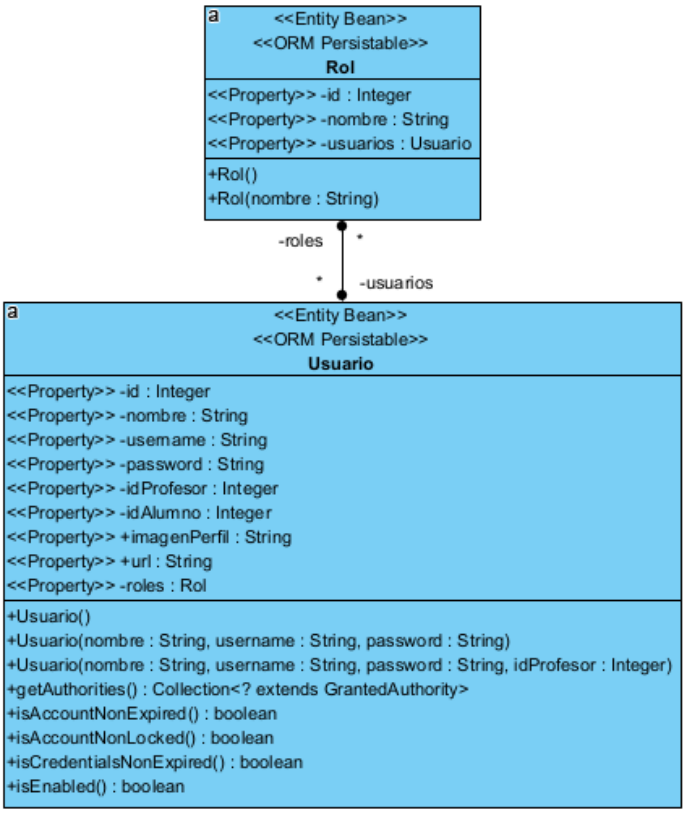
# 3 Análisis

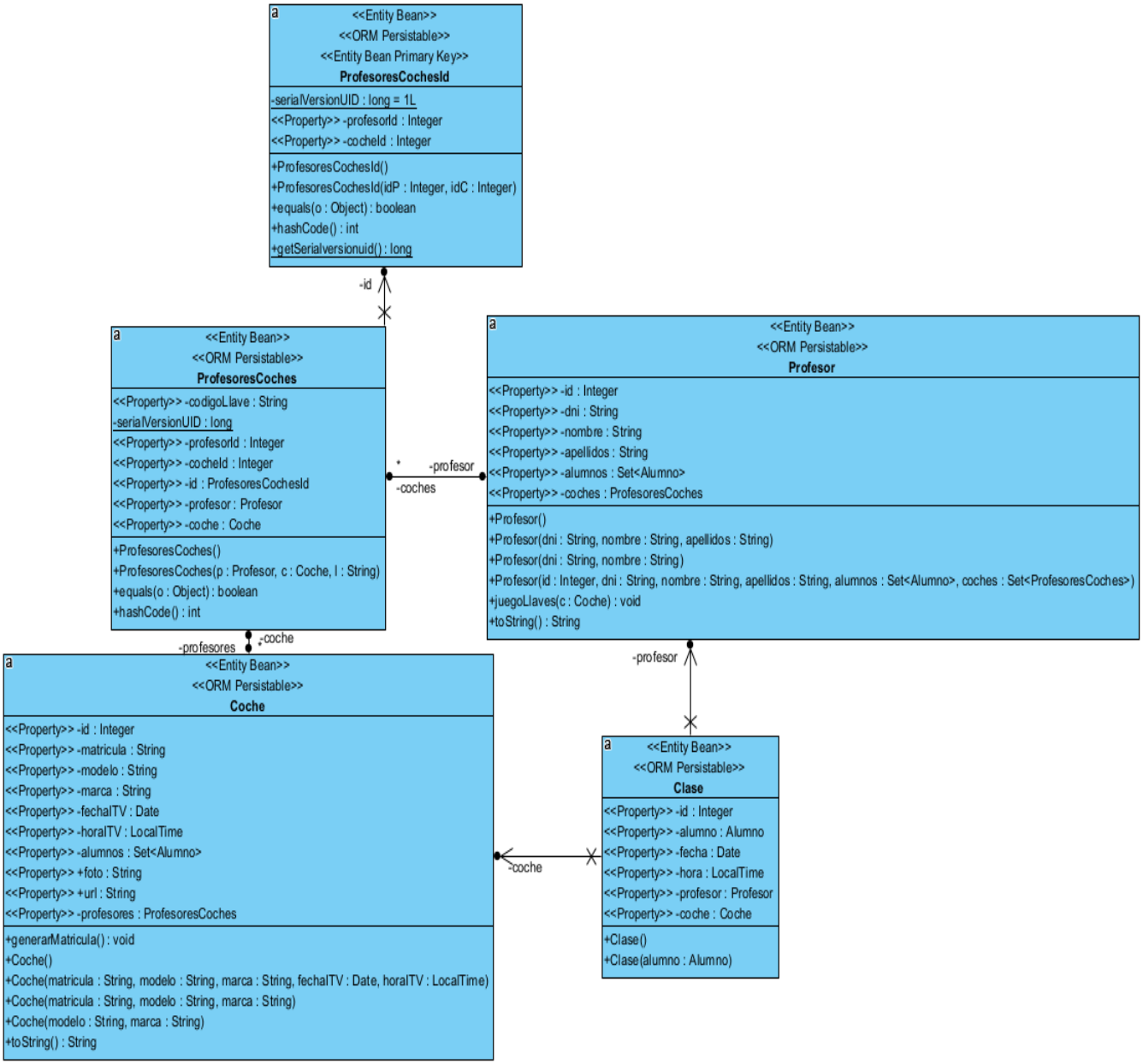
## 3.1 Introducción

En el apartado de análisis se presentarán distintos diagramas que muestren el funcionamiento y el flujo de información del sistema desde diferentes perspectivas

## 3.2 Diagrama de clases

En el siguiente diagrama de clases se mostrarán las relaciones entre las clases más esenciales, es decir, las clases modelo, para una mejor comprensión de cómo se relacionan unas con otras:

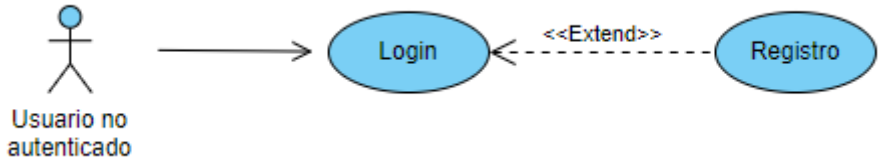




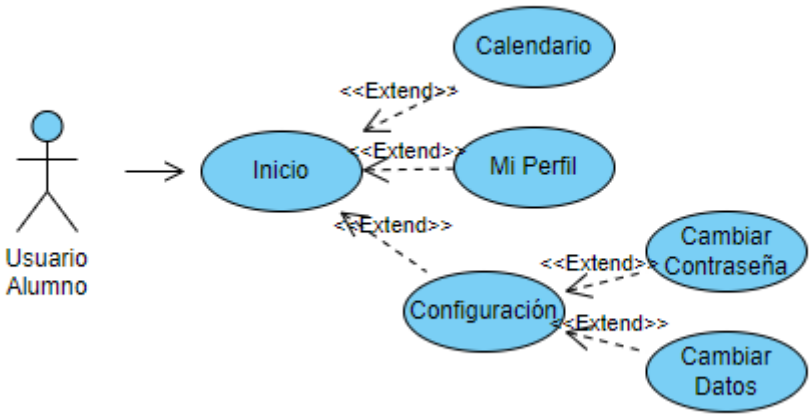

## 3.3 Diagrama de Casos de Uso

En este apartado se presenta un diagrama de casos de uso, mostrando las principales situaciones que se van a producir a la hora de utilizar la aplicación.

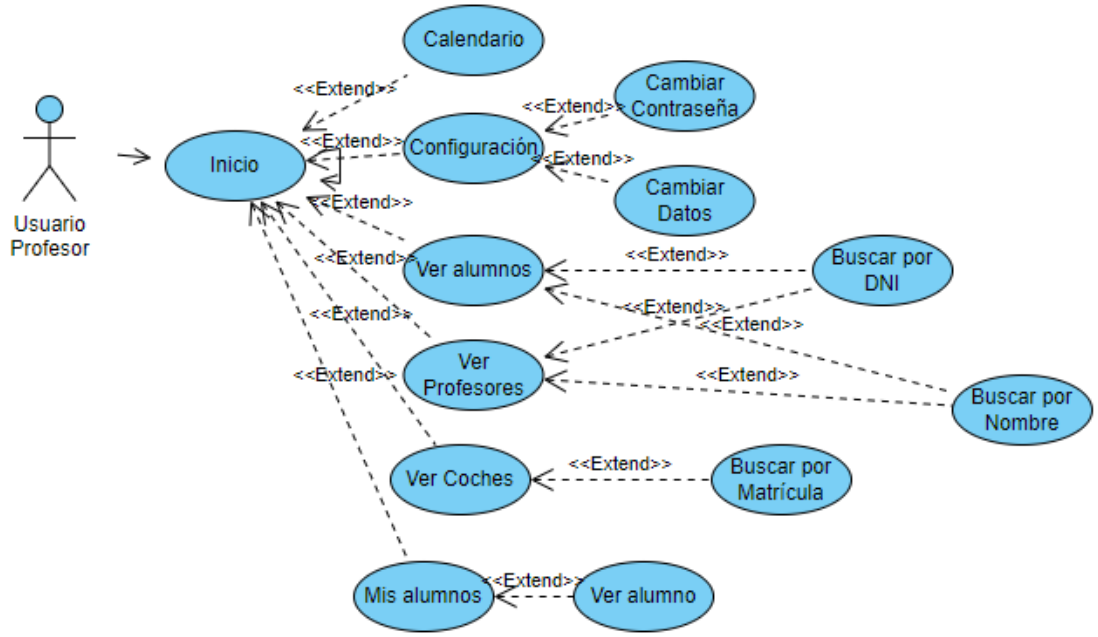
Para el usuario no registrado:



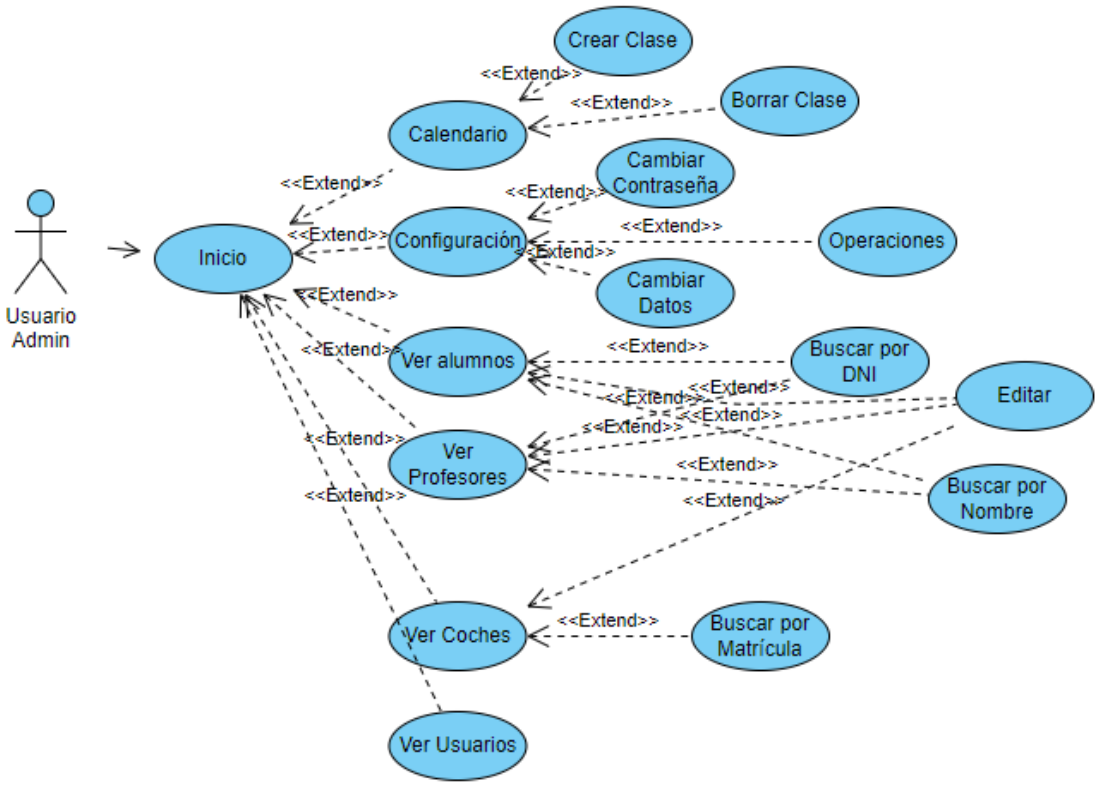

Para el usuario alumno:




Para el usuario profesor:




Para el usuario administrador:


Usuario
Admin
Inicio
Calendario
Crear Clase
Borrar Clase
Configuración
Cambiar Contraseña
Operaciones
Cambiar Datos
Ver alumnos
Buscar por DNI
Editar
Buscar por Nombre
Ver Profesores
Ver Coches
Buscar por Matrícula
Ver Usuarios
<<Extend>>

# 4 DISEÑO

## 4.1 Introducción

El diseño de la aplicación está formado por tres capas: la capa de presentación, la capa de negocio y la capa de persistencia de datos. Cada una de ellas está diferenciada de las demás, asumiendo sus propias responsabilidades y se coordinan para dar forma a todo el flujo de datos.

Mantener las tres capas independientes entre sí ayuda a realizar un correcto mantenimiento, a la escalabilidad de la aplicación y a la reutilización de código.

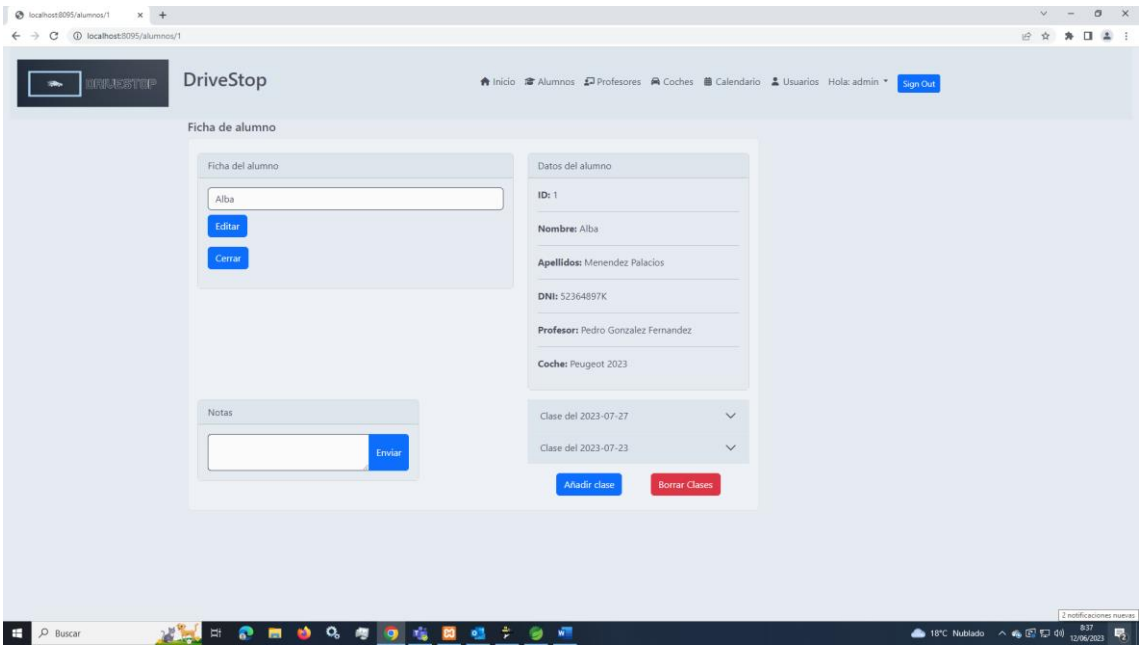
## 4.2 Capa de presentación

La capa de presentación es aquella que el usuario puede visualizar, formada por las interfaces y páginas mediante las que interactúa con el sistema.

Esta capa está formada por los archivos HTML, CSS y en parte, JavaScript para algunas tareas de generación de contenido dinámico.

JavaScript se emplea, por ejemplo, en la página inicial, mandando una solicitud GET a la URL de una API, la cual devuelve resultados que se encarga de convertir en elementos del DOM. También para mostrar mensajes de alerta o incluso configurar todo el calendario y su envío de solicitudes al controlador.

En el HTML se utiliza, además de JavaScript, la sintaxis de Thymeleaf, que permite que el código realice ciertas acciones o no dependiendo de lo que se evalúa, tales como la creación de determinados elementos por cada elemento de una lista, mostrar u ocultar elementos según el valor de una variable, o el almacenamiento de componentes de código para reutilizarlos fácilmente.




Ejemplo de una de las pantallas de la aplicación

## 4.3 Capa de negocio

La capa de negocio es la capa que el usuario no ve, es decir, el código Java en este caso.

Contiene todas las clases de la aplicación, tanto las de modelo como las de los controladores, pasando por los DTO, los servicios… etc. Estas clases están formadas por una serie de atributos y métodos, coordinándose unas con otras y estableciendo relaciones entre sí.

Se encuentra entre la capa de presentación y la capa de persistencia, interactuando con ambas y ejerciendo de mediador.

Con la capa de presentación se comunica recibiendo la información que se le envía, ya sea a través de formularios o de las URL. También se encarga de enviarle información a través de los modelAttributes, los cuales serán utilizados por la sintaxis de Thymeleaf; o de las imágenes, las cuales se envían a través de controladores.

También interactúa con la capa de persistencia, entregándole instrucciones para insertar, borrar, editar o recuperar registros y convirtiéndolos en datos interpretables por la capa de presentación cuando es necesario.

```
53      @Autowired
54      private CocheServiceImpl cocheService;
55      @Autowired
56      private UsuarioServiceImpl usuarioService;
57
58      @GetMapping({"","/"})
59      String homealumnos(Model model) {//lista de alumnos
60          ArrayList<Coche> misCoches=(ArrayList<Coche>) cocheService.listarCoches();
61          ArrayList<Alumno> misAlumnos= (ArrayList<Alumno>) alumnoService.listarAlumnos();
62          ArrayList<Profesor> misProfesores= (ArrayList<Profesor>) profeService.listarProfesores();
63
64          model.addAttribute("listaCoches", misCoches);
65          model.addAttribute("listaAlumnos", misAlumnos);
66          model.addAttribute("listaProfesores", misProfesores);
67
68          model.addAttribute("alumnoNuevo", new Alumno());
69          model.addAttribute("alumnoaEditar", new Alumno());
70          model.addAttribute("alumnoaBuscar", new Alumno());//carga de recursos
71          return "alumnos";
72      }
73
74      @SuppressWarnings("unlikely-arg-type")
75      @PostMapping("/add")
76      public String addAlumno(@ModelAttribute("alumnoNuevo") Alumno alumnoNew,RedirectAttributes redirectAttributes,Model model, BindingResult bidingresult) throws SQLExc
77          //añadir alumno
78          try {
79          Profesor profeNuevo=profeService.obtenerProfesorPorId(alumnoNew.getProfesor().getId()).get();//profe del alumno
80          Coche cocheNuevo=cocheService.obtenerCochePorId(alumnoNew.getCoche().getId()).get();//coche del alumno
81          String dni=alumnoNew.getDni();
82          char letra=dni.charAt(8);
83          letra=Character.toUpperCase(letra);
84          dni=dni.substring(0,8)+letra;
85          alumnoNew.setDni(dni);//conversion de letra minuscula a mayuscula
86          alumnoNew.setProfesor(profeNuevo);
87          profeNuevo.getAlumnos().add(alumnoNew);
88          alumnoNew.setCoche(cocheNuevo);
89          cocheNuevo.getAlumnos().add(alumnoNew);
90          if(!profeNuevo.getCoches().contains(cocheNuevo)) {//si el profesor aun no usa el coche nuevo
91              cocheService.insertarCoche(cocheNuevo);
92              profeService.insertarProfesor(profeNuevo);
93              profeNuevo.juegoLlaves(cocheNuevo);//crea la combinacion y altera las tablas de la BDD
94          }
95          alumnoService.insertarAlumno(alumnoNew);
96          }catch (DataIntegrityViolationException e) {
97              redirectAttributes.addFlashAttribute("error", "El DNI ya existe.");//controlar DNI duplicado
98          }
99          return "redirect:/alumnos";
100     }
101
102     @PostMapping("/edit/{id}")
103     public String editarAlumno(@PathVariable Integer id, @ModelAttribute("alumnoaEditar") AlumnoEditarNotasNombreApellidoDTO alumnoEditado, BindingResult bidingresult)
104         //editar alumno
105         Alumno alumnoaEditar=alumnoService.obtenerAlumnoPorId(id).get();
```

Ejemplo de uno de los documentos .java que conforman la aplicación.

## 4.4 Capa de persistencia

Constituida por la base de Datos, almacena toda la información de la aplicación, usuarios, coches, alumnos, clases… Esto lo consigue a través de dos ORM, Hibernate, el cual mapea los objetos de Java en tablas SQL, y con JPA, ahorrando el tener que ejecutar las consultas SQL manualmente en la gran mayoría de los casos

```java
package principal.modelo;

import java.time.LocalTime;

@Entity
@Table(name="Coches")
public class Coche {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name="Matricula", unique=true)
    private String matricula;

    @Column(name="Modelo")
    private String modelo;

    @Column(name="Marca")
    private String marca;

    @Temporal(TemporalType.DATE)
    @DateTimeFormat(iso = ISO.DATE)
    @Column(name="FechaITV")
    private Date fechaITV; //fecha adaptada

    @Column(name="HoraITV")
    private LocalTime horaITV;

    @OneToMany(mappedBy = "coche", fetch = FetchType.EAGER)
    private Set<Alumno> alumnos;

    @OneToMany(mappedBy = "profesor", cascade=CascadeType.ALL,orphanRemoval=true)
    private Set<ProfesoresCoches> profesores;

    @Column(name="foto")
    public String foto; //ruta de la que se extraerá la foto

    public String url; //url generada dinámicamente para visualizarlas


    public void generarMatricula() { //generador de matrículas para los coches genéricos
        Random random = new Random();

        String numeros = "";
        for (int i = 0; i < 4; i++) {
            int numero = random.nextInt(10); // generar un número aleatorio entre 0 y 9
            numeros += numero;
        }

        String letras = "";
        for (int i = 0; i < 3; i++) {
            char letra = (char) (random.nextInt(26) + 'A'); // generar una letra aleatoria entre A y Z
```

Clase Coche.java con anotaciones Hibernate para su posterior mapeo en la base de datos.

# 5 Implementación

## 5.1 Tecnologías utilizadas

### 5.1.1 Java

Java es un lenguaje de programación de alto nivel, orientado a objetos y ampliamente utilizado en el desarrollo de aplicaciones empresariales y software en general. Es utilizado para crear todas las clases del proyecto y es conocido por su portabilidad, lo que significa que los programas escritos en Java pueden ejecutarse en diferentes plataformas sin necesidad de reescribir el código.

### 5.1.2 HTML

HTML (HyperText Markup Language) es el lenguaje más utilizado para crear páginas web. Se compone de etiquetas y elementos que estructuran y presentan el contenido en un navegador web. Permite definir encabezados, párrafos, imágenes, enlaces y otros elementos interactivos. Se suele utilizar con CSS y JavaScript para mejorar su diseño y comportamiento.

En el proyecto, es el tipo de archivo utilizado para visualizar las páginas, y el que devuelven casi todos los controladores.

### 5.1.3 CSS

CSS (Cascading Style Sheets) es un lenguaje de estilo utilizado junto con HTML para darle estilo y diseño a las páginas web. Permite controlar la apariencia de los elementos HTML, como el color, la fuente, el tamaño y el diseño. Se puede utilizar para crear elementos responsive que se adapten al tamaño de diferentes dispositivos.

Todos los HTML del proyecto utilizan algún CSS, ya sean personalizados o de Bootstrap.

### 5.1.4 JavaScript

JavaScript es un lenguaje de programación utilizado principalmente en el desarrollo web. Con JavaScript, se pueden realizar acciones como manipular el contenido de una página, responder a eventos del usuario, enviar y recibir datos del servidor, y crear efectos visuales y animaciones como si fuese CSS. Es un lenguaje versátil y ampliamente utilizado en el desarrollo web.

En la creación se utiliza para mostrar u ocultar elementos y generar eventos de click en los HTML. Incluso se encarga de la creación de algunas partes importantes, como el calendario.

### 5.1.5 Bootstrap

Bootstrap es un framework de desarrollo web de código abierto que proporciona herramientas y componentes preestablecidos para agilizar y facilitar la creación de sitios web responsivos y visualmente atractivos. Con Bootstrap, se puede utilizar una variedad de clases predefinidas para establecer rápidamente la estructura, el diseño y los estilos de una página web. También ofrece características como grillas flexibles, navegación receptiva, estilos de tipografía y elementos interactivos, lo que hace que el diseño y la implementación de sitios web sean más eficientes y consistentes.

Casi todas las páginas de la aplicación utilizan estilos de Bootstrap para facilitar el diseño responsive a la vez que generan elementos atractivos visualmente.

### 5.1.6 MySQL

MySQL es un sistema de gestión de bases de datos relacional de código abierto ampliamente utilizado. Proporciona una forma eficiente y confiable de almacenar, organizar y administrar grandes cantidades de datos. MySQL utiliza un lenguaje de consulta estructurado (SQL) para interactuar con la base de datos, permitiendo la creación, modificación y eliminación de tablas, así como la consulta y manipulación de los datos almacenados.

Es la base de datos que se emplea para almacenar todos los datos de la aplicación.

### 5.1.7 Hibernate

Hibernate es un framework de mapeo objeto-relacional (ORM) para el desarrollo de aplicaciones en Java. Proporciona una capa de abstracción entre la base de datos y el código de la aplicación, permitiendo el mapeo automático de objetos Java a tablas de bases de datos y simplificando las operaciones de persistencia. Hibernate se encarga de manejar las consultas SQL y la manipulación de datos, lo que reduce la cantidad de código manual que debe escribirse en el proyecto, ahorrando una gran cantidad de líneas de código.

### 5.1.8 JPA

JPA (Java Persistence API) es una especificación de Java para el mapeo objeto-relacional. Permite interactuar con bases de datos utilizando objetos Java. Abstrae la complejidad de las consultas SQL y proporciona portabilidad entre diferentes implementaciones ORM, entre ellos Hibernate, trabajando en conjunto dentro de la aplicación y simplificando las operaciones de persistencia de datos en aplicaciones Java.

### 5.1.9 Thymeleaf

Thymeleaf es un motor de plantillas Java que se utiliza para generar vistas dinámicas en aplicaciones web. Permite la creación de plantillas HTML enriquecidas con expresiones y atributos específicos que se procesan en el servidor para generar contenido dinámico. Además, proporciona una integración fluida con datos del lado del servidor, lo que facilita la manipulación y presentación de información en las vistas.

Se emplea en casi todos los HTML del proyecto, para interpretar información subida desde el lado servidor y generar elementos dependiendo de evaluaciones que realizamos en su código.

### 5.1.10 Spring Tool Suite

Spring Tool Suite (STS) es un entorno de desarrollo integrado (IDE) basado en Eclipse y diseñado específicamente para facilitar el desarrollo de aplicaciones basadas en el framework Spring. Proporciona un conjunto de herramientas y características que agilizan el proceso de desarrollo de aplicaciones Spring, incluyendo soporte para la configuración, autocompletado de código, navegación entre clases, depuración y generación de proyectos.

Es el IDE utilizado para escribir la totalidad del código del proyecto.

### 5.1.11 Docker

Docker es una plataforma de virtualización a nivel de sistema operativo que permite empaquetar y distribuir aplicaciones junto con todas sus dependencias en contenedores. Los contenedores son unidades de software autónomas que encapsulan todo lo necesario para que una aplicación se ejecute de manera consistente en cualquier entorno, independientemente del sistema operativo subyacente.

Es el método de despliegue utilizado para aumentar la portabilidad de la aplicación y evitar fallos relacionados con la configuración del ordenador anfitrión.

### 5.1.12 Git

Git es un sistema de control de versiones que permite realizar un seguimiento de cambios en archivos a lo largo del tiempo. Utiliza commits para capturar instantáneas del estado de los archivos. Permite trabajar en ramas separadas para desarrollar nuevas funcionalidades o solucionar problemas, las cuales he utilizado para desarrollar diferentes partes del proyecto.

Una herramienta muy útil que me ha permitido desarrollar la aplicación en cualquier lugar con conexión a Internet.

## 5.2 Descripción del proyecto

### 5.2.1 Capa de Presentación

La capa de presentación en esta aplicación se encuentra formada en su totalidad por ficheros HTML, que serán mostrados por los controladores dependiendo de las acciones que se ejecuten.

Estos HTML se combinan con CSS, Bootstrap, Thymeleaf y JavaScript para generar contenido dinámico que responda a las solicitudes de datos por parte del usuario.

Al utilizar Bootstrap en el HTML, conseguimos un diseño responsive sencillo y funcional, que se ve moderno y es bonito de ver. Esto nos permite ahorrar tiempo en el diseño y poder dedicarlo a otras funciones importantes, como el contenido dinámico o el Back-end.

JavaScript y Thymeleaf son empleados para la generación de contenido dinámico, además de para controlar eventos activados por el usuario de forma sencilla, lo que ayuda a crear una página fácil de usar e intuitiva.

```
<div class="col-8 d-flex justify-content-around w-100 flex-wrap">
    <div class="card exter border border primary col-3 user-card" th:each="user:${listaUsuarios}">
        <div class="card-body row pt-4 pb-4">
            <div class="col-12">
                <div class="card inter">
                    <div class="card-header text-dark bg-light">Datos del usuario

                    </div>
                    <div class="card-body">
                        <div class="mb-2 d-flex justify-content-between w-75">
                            <h5 th:text="${user.nombre}+', '+${user.username}" class="mb-2 card-text">Usuario
                            </h5>
                            <div th:if="${user.url!=null}">
                                <div class="d-flex justify-content-center" th:if="${user.url!=null}">
                                    <img th:src="@{${user.url}}" alt="${image.name}" width="60px" height="60px"
                                        class="ml-3 rounded-circle" />
                                </div>
                            </div>
                        </div>
                        <p><a th:href="@{/usuarios/{id}(id=${user.id})}" class="btn btn-primary mt-3">Ver
                            más</a></p>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Ejemplo de plantilla HTML. Los fragmentos marcados en amarillo corresponden a sintaxis de Thymeleaf.

Además, se muestran imágenes en el HTML, las cuales están gestionadas por su propio controlador, que se encarga de guardarlas y recuperarlas del servidor.

### 5.2.2 Capa de negocio

En nuestro caso, la capa de negocio se encuentra formada en su gran mayoría por el código Java.

Se definen distintas clases y atributos, además de métodos, los cuales en su conjunto determinan cómo se comportará una clase y cómo se relacionará con las demás

Además, se emplean anotaciones de Spring, las cuales simplifican el proceso de ajuste del comportamiento de una clase, especialmente para la capa de persistencia.

```java
@Controller
@RequestMapping("/profesores")
public class ProfesoresController {

    /*AlumnoDAO alumnoDAO=new AlumnoDAO();
    CocheDAO cocheDAO=new CocheDAO();
    ProfesorDAO profeDAO=new ProfesorDAO();*/

    @Autowired
    private AlumnoServiceImpl alumnoService;
    @Autowired
    private ProfesorServiceImpl profeService;
    @Autowired
    private CocheServiceImpl cocheService;


    @GetMapping(value={"","/"})
    String homealumnos(Model model) {
        ArrayList<Coche> misCoches=(ArrayList<Coche>) cocheService.listarCoches();
        ArrayList<Alumno> misAlumnos= (ArrayList<Alumno>) alumnoService.listarAlumnos();
        ArrayList<Profesor> misProfesores= (ArrayList<Profesor>) profeService.listarProfesores();

        model.addAttribute("listaCoches", misCoches);
        model.addAttribute("listaAlumnos", misAlumnos);
        model.addAttribute("listaProfesores", misProfesores);

        model.addAttribute("profeNuevo", new Profesor());
        model.addAttribute("profeaEditar", new Profesor());
        model.addAttribute("profeaBuscar", new Profesor());
        boolean vacio = (Boolean) model.asMap().getOrDefault("vacio", false);
        model.addAttribute("vacio", vacio);
        return "profesores";
    }
```

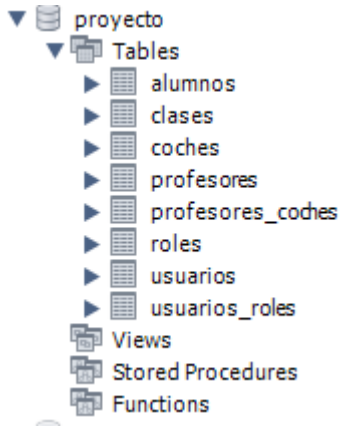
En la imagen superior, se puede apreciar un fragmento de código de un controlador, el cual controla la URL “/profesores” y todos sus derivados. Utiliza varios servicios para gestionar las entidades y se puede contemplar el uso de anotaciones de Spring, tales como “@Controller”, para indicar que la clase es un controlador; y “@GetMapping”, para especificar que controlará las solicitudes GET de la URL “/profesores” y “profesores/”.

### 5.2.3 Capa de persistencia

Constituida por la base de Datos, almacena toda la información de la aplicación, usuarios, coches, alumnos, clases… Esto lo consigue a través de un ORM, Hibernate, el cual mapea los objetos de Java en tablas SQL, ahorrando el tener que ejecutar las consultas SQL manualmente en la gran mayoría de los casos.

La Base de Datos está formada por 8 tablas generadas por Hibernate y JPA automáticamente mediante el uso de anotaciones. Las tablas pueden representar las entidades del proyecto (Alumno, Profesor, Coche, Clase, Usuario…) o las relaciones entre estas (usuarios_roles, profesores_coches).

La información contenida es enviada o manipulada por las acciones que realiza el usuario en la capa de presentación, utilizando la capa de negocio para interpretarlas y ejecutarlas.




Estructura de la base de datos.

```java
@Entity
@Table(name="Alumnos")
public class Alumno {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    //@Column(name="id")
    private Integer id;

    @Column(name="DNI", unique=true)
    private String dni;

    @Column(name="Nombre", nullable = false)
    private String nombre;

    @Column(name="Apellidos", nullable = false)
    private String apellidos;

    @Column (name="Notas")
    private String notas;

    @JsonIgnore
    @ManyToOne
    @JoinColumn(name ="id_profesor", nullable = false)
    private Profesor profesor;

    @JsonIgnore
    @ManyToOne
    @JoinColumn(name = "id_Coche", nullable = false)
    private Coche coche;

    @JsonIgnore
    @OneToMany(mappedBy = "alumno", cascade=CascadeType.REMOVE)
    private Set<Clase> clases;
```

Clase Alumno con sus atributos. El código marcado son anotaciones de JPA para definir la estructura de la tabla.

```
Hibernate: create table alumnos (id integer not null auto_increment, apellidos varchar(255) not null, dni varchar(255), nombre varchar(
Hibernate: create table clases (id integer not null auto_increment, fecha date not null, hora time not null, id_alumno integer not null
Hibernate: create table coches (id integer not null auto_increment, fechaitv date, foto varchar(255), horaitv time, marca varchar(255),
Hibernate: create table profesores (id integer not null auto_increment, apellidos varchar(255), dni varchar(255), nombre varchar(255),
Hibernate: create table profesores_coches (coche_id integer not null, profesor_id integer not null, codigo_llave varchar(255), primary
Hibernate: create table roles (id integer not null auto_increment, nombre varchar(255), primary key (id)) engine=InnoDB
Hibernate: create table usuarios (id integer not null auto_increment, id_alumno integer, id_profesor integer, imagen_perfil varchar(255
Hibernate: create table usuarios_roles (id_usuario integer not null, id_rol integer not null, primary key (id_usuario, id_rol)) engine=
Hibernate: alter table alumnos add constraint UK_f2c9i14wnebiikl4lldh5yam2 unique (dni)
Hibernate: alter table coches add constraint UK_t8qtix7g3natypy6cgca60yfd unique (matricula)
Hibernate: alter table profesores add constraint UK_tlaenyc9ynoyn45ueko23ivgk unique (dni)
```

Log de la creación de tablas con Hibernate.

# 6 Evaluación

## 6.1 Introducción

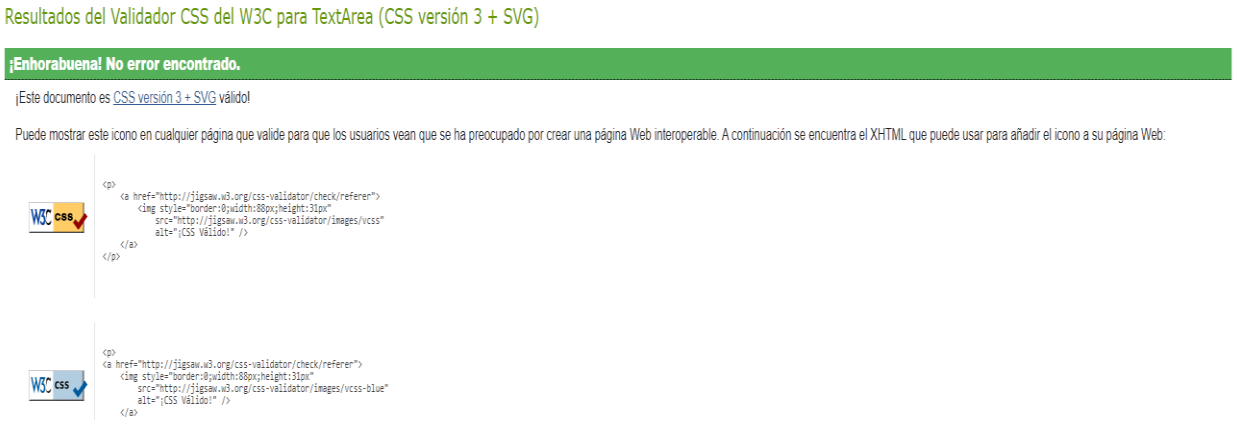
En general, los documentos HTML y CSS se pueden validar utilizando herramientas en línea como el Validador de W3C Markup Validation Service, que verifica el código con los estándares establecidos. Sin embargo, cuando se utiliza sintaxis específica de Thymeleaf en el código HTML, estas herramientas no reconocen dicha sintaxis y pueden generar errores sin sentido.

Por desgracia, no existen alternativas directas para validar específicamente la sintaxis de Thymeleaf en línea. En estos casos, la validación se vuelve más manual y depende del desarrollador para verificar que la sintaxis de Thymeleaf se utilice correctamente.

Entonces la validación se realiza a simple vista, realizando pruebas y ayudándose de las funciones que ofrece el IDE en el que se está desarrollando.

## 6.2 Validaciones de Páginas de Estilo

Para las páginas de estilo, ya que estas sí han empleado su sintaxis original, utilizamos W3C para su validación.




Resultado positivo de una de las validaciones de los archivos CSS.

## 6.3 Validaciones de los enlaces

Normalmente en los sitios web se suelen utilizar varios enlaces externos que conducen a distintas páginas que pueden ser de interés para el usuario. Por ello es importante asegurarse de que estos permanecen estables y no se encuentran caídos para poder acceder a esa información

En nuestra página web, los enlaces utilizados corresponden a enlaces dentro de la propia aplicación, por lo que se sobreentiende que, al estar ejecutándose la aplicación, estos enlaces permanecen estables en todo momento.

Hay una excepción en los enlaces de las noticias generadas por la API en el menú de Inicio. Estos enlaces sí son externos, pero al ser buscados por la propia API en Internet, es casi seguro que estarán en funcionamiento ya que de otra forma no serían rastreados por ella.

## 6.4 Documentación

En este proyecto, se ha realizado un esfuerzo adicional para mejorar la documentación y comprensión del código Java, agregando comentarios en el código para facilitar la comprensión de lo que este hace.

Además de los comentarios en el código, se ha utilizado Swagger para automatizar aún más la documentación. De esta forma podemos describir, visualizar y probar APIs de manera sencilla. Se han usado anotaciones Swagger en las clases y métodos del proyecto para generar la documentación de forma automática.

# 7 Conclusión

## 7.1 Valoración Personal

Siendo el primer proyecto al ser el primer ciclo que hago, al comenzar tenía muchas dudas en cuanto a si conseguiría terminarlo, o si quizás estaba teniendo muchas expectativas en la versión final de la aplicación, a lo mejor no conseguía hacer un calendario o tenía problemas con la persistencia que en ese momento era un mundo.

Han sido unos cinco meses y medio de trabajo bastante intenso, considero que empecé en enero porque hasta ese momento lo único que había desarrollado era a partir de tareas de la clase de Desarrollo Web en Entorno Servidor y puedo decir que el haber trabajado en esto me ha hecho mejorar enormemente mi capacidad de programación y diseño web desde aquellos días en los que recibía whitelabels por escribir mal la URL y me tiraba 10 minutos sin ver el fallo, o los que el crear varias imágenes alineadas horizontalmente con HTML y CSS suponían un quebradero de cabeza para mí.

El haberme tenido que enfrentar a errores de todo tipo, tanto de Back-end como de Front-end me ha ayudado no solo a aprender a resolverlos de forma rápida, sino también a comprender de verdad cómo funcionan muchas de las cosas que utilizamos en clase, ya que por lo general no se suele tener ese conocimiento profundo cuando estás todavía aprendiendo, solo tienes un concepto de cómo se ejecutan y la verdad que resulta muy útil conocerlos puesto que te ayuda a codificar mejor y resolver los problemas más rápidos.

En mi opinión, el resultado final de la aplicación me ha dejado satisfecho, he podido desarrollar todo o casi todo lo que me propuse en un momento inicial, si bien hay algunas ideas que han surgido durante la fase final que tendrán que quedarse atrás.

Me gustaría también destacar la ayuda de todos los profesores, tanto en el proyecto como en el ciclo en general, especialmente de Juanjo, al que le debo una buena parte de errores que no conseguía resolver y del que aprendí que en programación es tan importante el saber codificar como el buscar la información correcta cuando no sabes hacerlo.

## 7.2 Posibles ampliaciones

En el camino mientras iba programando la aplicación, se me fueron ocurriendo algunas funcionalidades que podrían implementarse para hacer que la página fuera más práctica.

Por ejemplo, introducir un sistema de mensajes, en el que el profesor o el administrador le pueda enviar al alumno algún mensaje informándole de lo que necesiten.

Que los coches puedan contener más información, como fechas de revisiones, cambios de aceite…etc.

Una sección en la que puedas apuntar pedidos para la autoescuela, tales como nuevas ruedas para los coches, piezas mecánicas o simplemente mobiliario; apuntar el precio y calcular el coste del conjunto.

Añadir más vehículos, no solo coches, ya que, aunque son el principal tipo de vehículo que se escoge en las autoescuelas, también pueden ser motos o incluso camiones, los cuales tienen otras características.

Mejoras en el apartado de diseño, añadir más temas y sobre todo perfeccionar el responsive ya que considero que me ha quedado algo justo.

# 8 Bibliografía

Documentación oficial de Java: https://docs.oracle.com/en/java/

Introducción a HTML en Mozilla Developer Network: https://developer.mozilla.org/es/docs/Learn/HTML/Introduccion_a_HTML

Introducción a CSS en Mozilla Developer Network: https://developer.mozilla.org/es/docs/Web/CSS

Introducción a JavaScript en Mozilla Developer Network: https://developer.mozilla.org/es/docs/Web/JavaScript

Página oficial de Bootstrap: https://getbootstrap.com/

Página oficial de MySQL: https://www.mysql.com/

Página oficial de Hibernate: https://hibernate.org/

Oracle JPA: https://www.oracle.com/java/technologies/persistence-jsp.html

Página oficial de Thymeleaf: https://www.thymeleaf.org/

Información sobre el protocolo TCP/IP: https://www.ibm.com/docs/es/aix/7.2?topic=protocol-tcpip-protocols

Validador oficial de W3C: https://validator.w3.org/