Ciclo Formativo de Grado Superior

Desarrollo de Aplicaciones Multiplataforma

Autor

Roberto Campo

Tutor

Luis Manuel Moreno Lara

Coordinadores

José Luis López Álvarez

I.E.S Clara del Rey

ÍNDICE

Índice de Ilustraciones

1. INTRODUCCIÓN

1.1 Descripción

El proyecto surge de la necesidad de mejorar la comunicación y colaboración dentro del ámbito educativo, un área en la que las herramientas existentes, como las plataformas oficiales de comunidades autónomas por ejemplo EducaMadrid, no siempre son utilizadas de manera efectiva, comprensible por los usuarios. Además, aunque existen alternativas como grupos de WhatsApp o Facebook, estas plataformas suelen distraer con contenido ajeno al entorno académico.

Este proyecto propone el desarrollo de una red social educativa que se enfoque exclusivamente en la comunicación académica. La plataforma permite organizar la interacción por grupos o cursos, asegurando la privacidad de cada grupo, y está diseñada para ser gestionada por los delegados de curso con permisos proporcionados por el instituto. Entre las funcionalidades destacan la gestión de horarios interactivos, publicación de contenido, encuestas, y el intercambio de ficheros, promoviendo una comunicación colectiva sin la necesidad de mensajería individual, lo que facilita el enfoque en los objetivos educativos.

1.2 Objetivo

Consolidar una plataforma web educativa que optimiza la comunicación y organización académica, ofreciendo una herramienta funcional y práctica que se ajusta a las necesidades específicas de cada grupo, asegurando privacidad, eficiencia y facilidad de uso.

Garantizar que la plataforma permita la gestión autónoma de grupos, publicaciones y horarios por parte de los delegados, cumpliendo con los estándares de interacción en tiempo real mediante el uso de WebSockets.

Proveer un entorno seguro y privado mediante mecanismos de autenticación robustos basados en tokens y contraseñas encriptadas, cumpliendo con los requisitos de seguridad y privacidad educativa.

2. ALCANCE DEL PROYECTO

2.1 Funcionalidades principales

El proyecto incluye las siguientes funcionalidades principales, diseñadas para optimizar la organización y comunicación académica en los institutos:

Publicaciones:

- Los usuarios pueden publicar archivos (Word, PDF, DOC), imágenes, comunicados y encuestas.
- Cada publicación está asociada al usuario gracias al uso de un token de autenticación.
- Se utiliza WebSocket en las publicaciones para actualizarlas en tiempo real en el muro de cada estudiante.

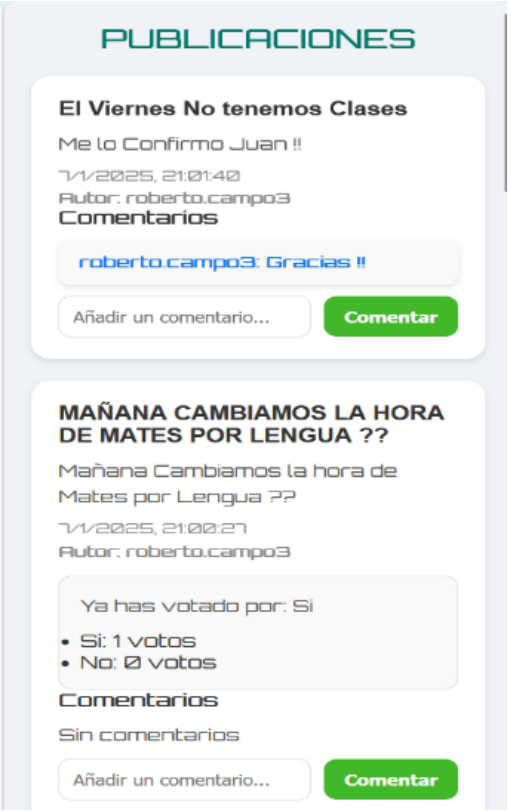

Ilustración 1 Captura Publicaciones
Fuente: Elaboración propia

Gestión de permisos:

- Los delegados de curso cuentan con un perfil que incluye una tabla de permisos. Esta tabla les permite gestionar y controlar qué usuarios tienen autorización para realizar publicaciones.
- En caso de que un alumno pierda el permiso para publicar, se le notificará mediante una alerta y se reflejará visualmente en su perfil, que cambiará a un color rojo como indicador de su estado.

Ilustración 2 Captura Tabla de Integrantes del Grupo – Perfil Delegando
Fuente: Elaboración propia

Gestión de horarios:

- Los delegados tienen la capacidad de modificar los horarios académicos directamente desde su perfil.
- Los cambios realizados por los delegados se actualizan automáticamente en los perfiles de los demás alumnos, cuyos horarios son de solo lectura.

Ilustración 3 Captura Modal Horario de Clases – Perfil Delegando
Fuente: Elaboración propia

Tabla de asignaturas:

Los delegados tienen la capacidad de añadir, modificar o eliminar asignaturas directamente desde su perfil.

Los cambios realizados en la tabla de asignaturas se aplican automáticamente a las opciones disponibles en el horario, ya que están vinculadas.

Gracias a WebSocket, estas actualizaciones se reflejan en tiempo real para todos los usuarios correspondientes.

Ilustración 4 Captura Tabla de Asignaturas – Perfil Delegando
Fuente: Elaboración propia

Registro de alumnos:

- El registro de alumnos se realiza mediante dos formularios: uno para alumnos y otro para delegados.

Ilustración 5 Captura Registro de formularios
Fuente: Elaboración propia

- Para acceder al formulario del delegado, es necesario introducir un código exclusivo proporcionado por el administrador del instituto.

Ilustración 6 Captura Pasarela de Registro
Fuente: Elaboración propia

- Al registrarse, se envía un correo de verificación al usuario. Una vez verificado, se muestra un mensaje con el estatus asignado (alumno o delegado).

```
String subject = "Verificación de correo electrónico";
String body = "<p>Gracias por registrarte en InstiConnect. Para completar tu registro y verificar tu correo electrónico, por favor haz clic en el siguiente botón:</p>" +
        "<a href=\"" + verificationLink + "\" style=\"display:inline-block;padding:10px 20px;background-color:#31EC56;color:#fff;text-decoration:none;border-radius:5px;\">Verificar mi Correo Electrónico</a>";

helper.setFrom(fromEmail);
helper.setTo(toEmail);
helper.setSubject(subject);
helper.setText(body, html: true);

mailSender.send(message);
log.info("Correo de verificación enviado a: {}", toEmail);
```

Ilustración 7 Captura Línea de Código Mensaje de Verificación
Fuente: Elaboración propia

- Se generan un token de autenticación y una contraseña encriptada mediante las herramientas de seguridad de Spring Boot.

email	clave
robertocampo@theserjs.es	$2a$10$V.SgFPJlAEZQ3EJOpaivV.Zn55zzgw…

Ilustración 8 Captura Ejemplo de Clave Encriptada en la base de datos
Fuente: Elaboración propia

2.2 Planificación previa

La planificación del proyecto se estructuró en cuatro fases principales, cada una diseñada para garantizar un desarrollo progresivo y enfocado en los objetivos del sistema.

	Lista de tareas	Fecha de inicio	Fecha final	Duración (días)
1	Planificación Inicial y Diseño de Requisitos	2024-10-01	2024-10-25	24
2	Diseño de Interfaces y Registro de Sesión	2024-10-25	2024-11-08	14
3	Desarrollo de Grupos y Roles	2024-11-08	2024-11-22	14
4	Pruebas de Usabilidad y Optimización	2024-11-22	2024-12-06	14

Ilustración 9 Tabla De Tareas Del Proyecto
Fuente: Elaboración propia

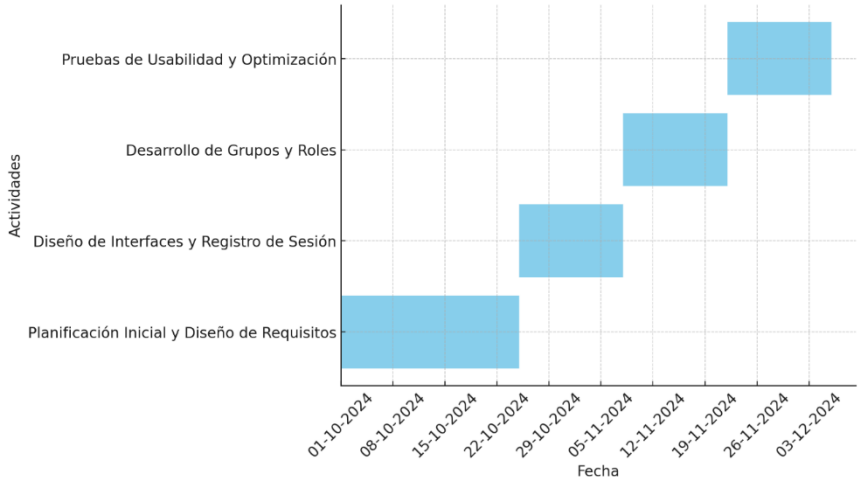

Ilustración 10 Diagrama de Gannt
Fuente: Elaboración propia

- Aprendizaje y búsqueda de información: Se investigaron las tecnologías claves que se utilizarían en el proyecto, para el Front Html, Css y JavaScript para el Back utilizaremos Spring Boot, WebSocket, JWT y la base de datos MySQL. Además, se revisaron prácticas recomendadas para la implementación de sistemas web académicos.

- Diseño y desarrollo de la aplicación: Incluyó la creación de la arquitectura del sistema, el desarrollo de las funcionalidades principales como publicaciones, gestión de horarios y permisos y la integración de tecnologías avanzadas para lograr un sistema seguro y eficiente.
- Pruebas y validación: Se realizarán pruebas funcionales básicas, unitarias y validaciones en tiempo real utilizando herramientas como Mockito con Spring Boot para garantizar que cada módulo cumpliera con los requerimientos.
- Implantación en entorno simulado: El sistema será desplegado en un entorno local simulado para verificar la integración de todos los módulos y la interacción con la base de datos.

3. JUSTIFICACIÓN

3.1 Importancia de la gestión y colaboración académica

Facilita la organización de recursos y cumplimiento de metas académicas esto fomenta el trabajo en equipo entre el grupo de estudiantes promoviendo la integridad y haciendo más dinámica el entorno donde conviven ayudando a superar las barreras de la comunicación siendo más directo y ayudado a no tener mucha información duplicada centralizado el propósito de la plataforma.

3.2 Beneficios de la plataforma para la comunidad educativa

- Accesibilidad y centralización de información: Todos los miembros del grupo de clases pueden acceder a recursos académicos de interés del grupo desde un único punto, mejorando la eficiencia y reduciendo la pérdida de información.

- Fomento de la interacción y el aprendizaje colaborativo: La plataforma puede integrar herramientas como compartir documentos, espacios para la divulgación de comunicados encuestas para una mejor democracia en el grupo

- Optimización de procesos: Automatiza tareas administrativas como la modificación de horarios de clases por cambios de horas

- Adaptabilidad a contextos diversos: Diseñada para ser flexible, la plataforma se puede ajustar a diferentes niveles educativos, desde primaria hasta superior.

4. ANÁLISIS Y DISEÑO

4.1 Arquitectura de la aplicación Web

La aplicación está diseñada bajo una arquitectura basada en capas, asegurando modularidad y facilidad de mantenimiento.

- Capa de presentación (Frontend): Desarrollada con HTML5, CSS3 y JavaScript, esta capa está diseñada para proporcionar una interfaz intuitiva y accesible a los usuarios. Su principal función es consumir los servicios del backend mediante peticiones HTTP a la API REST.
- Capa de negocio (Backend): Implementada en Spring Boot, esta capa contiene la lógica de la aplicación. Proporciona una API REST que gestiona funcionalidades como el registro de usuarios, autenticación basada en tokens JWT, publicación de contenido y la configuración de horarios interactivos. Además, se utiliza WebSockets para actualizaciones en tiempo real en las publicaciones, actualizaciones de horario, permisos y comentarios.
- Capa de Base de Datos: La información es almacenada en una base de datos relacional MySQL. Esta capa incluye el modelo de datos para usuarios, publicaciones, encuestas, comentarios y horarios, siguiendo un diseño normalizado que evita redundancias y optimiza las consultas.
- Servicios SMTP: La aplicación incorpora el envío de correos electrónicos, utilizado en la recuperación de contraseñas y verificación de cuenta.

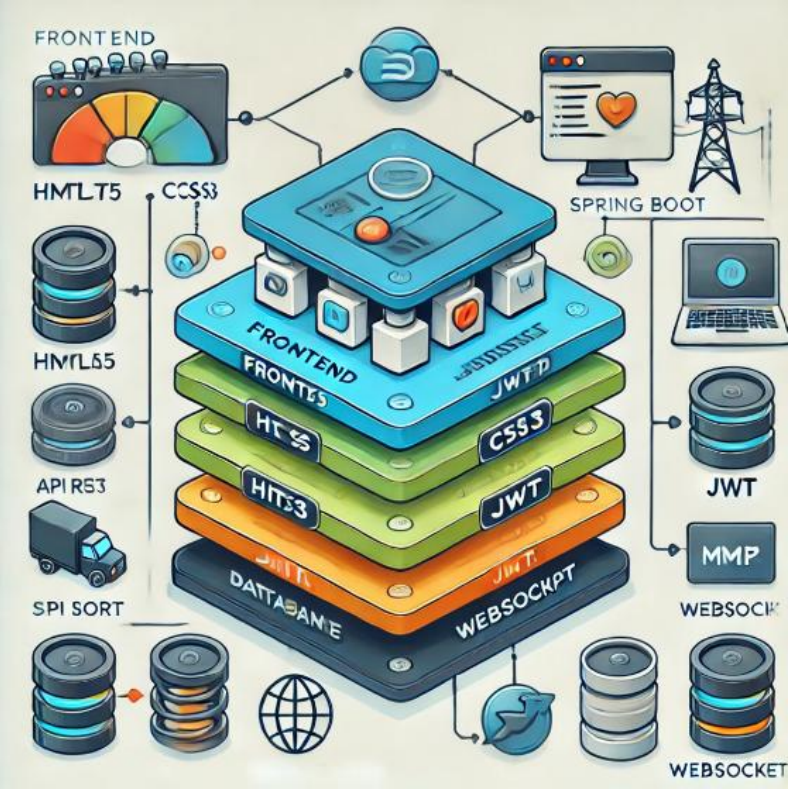

Ilustración 11 Diagrama Arquitectura de Aplicaciones Web
Fuente: aitor-medrano.github.io

4.2 Diagrama de componentes

- Cliente - Frontend: Maneja la interacción con el usuario final y se comunica con el backend mediante peticiones HTTP RESTful.
- Servidor API - Backend: Expuesto como un conjunto de endpoints para manejar operaciones CRUD Create, Read, Update, Delete sobre las entidades del sistema. Implementa servicios de seguridad, como autenticación y autorización mediante JWT, y un sistema de WebSockets para comunicaciones en tiempo real.
- Base de datos MySQL: Gestiona las entidades, como usuarios, publicaciones, comentarios, encuestas, y horarios, asegurando la integridad mediante relaciones definidas.
- Servidor de despliegue: Aunque en el proyecto actual se está ejecutando de forma local, la aplicación está preparada para ser desplegada en un servidor remoto o utilizando contenedores Docker para garantizar portabilidad.

4.3 Modelado de datos (ERD)

El modelo de datos se basa en un diseño relacional que incluye las principales entidades y relaciones necesarias para la aplicación. El diagrama de entidad-relación (ERD) presentado incluye las siguientes entidades:

- Usuarios: Incluye datos como id, nombre, correo electrónico, contraseña (encriptada) y rol (administrador, delegado, alumno).
- Grupos: Define los grupos académicos, asociados a cursos y ciclos formativos, e incluye información sobre delegados y asignaturas.
- Publicaciones: Almacena contenido compartido en el grupo, como textos, imágenes, documentos y encuestas. Además, permite comentarios por parte de los usuarios.
- Comentarios: Registra las respuestas o interacciones en las publicaciones.
- Horarios: Incluye datos sobre las clases, como día, hora de inicio, hora de fin y asignaturas asociadas.
- Encuestas: Permite realizar votaciones dentro de un grupo académico, almacenando las opciones y resultados de estas.

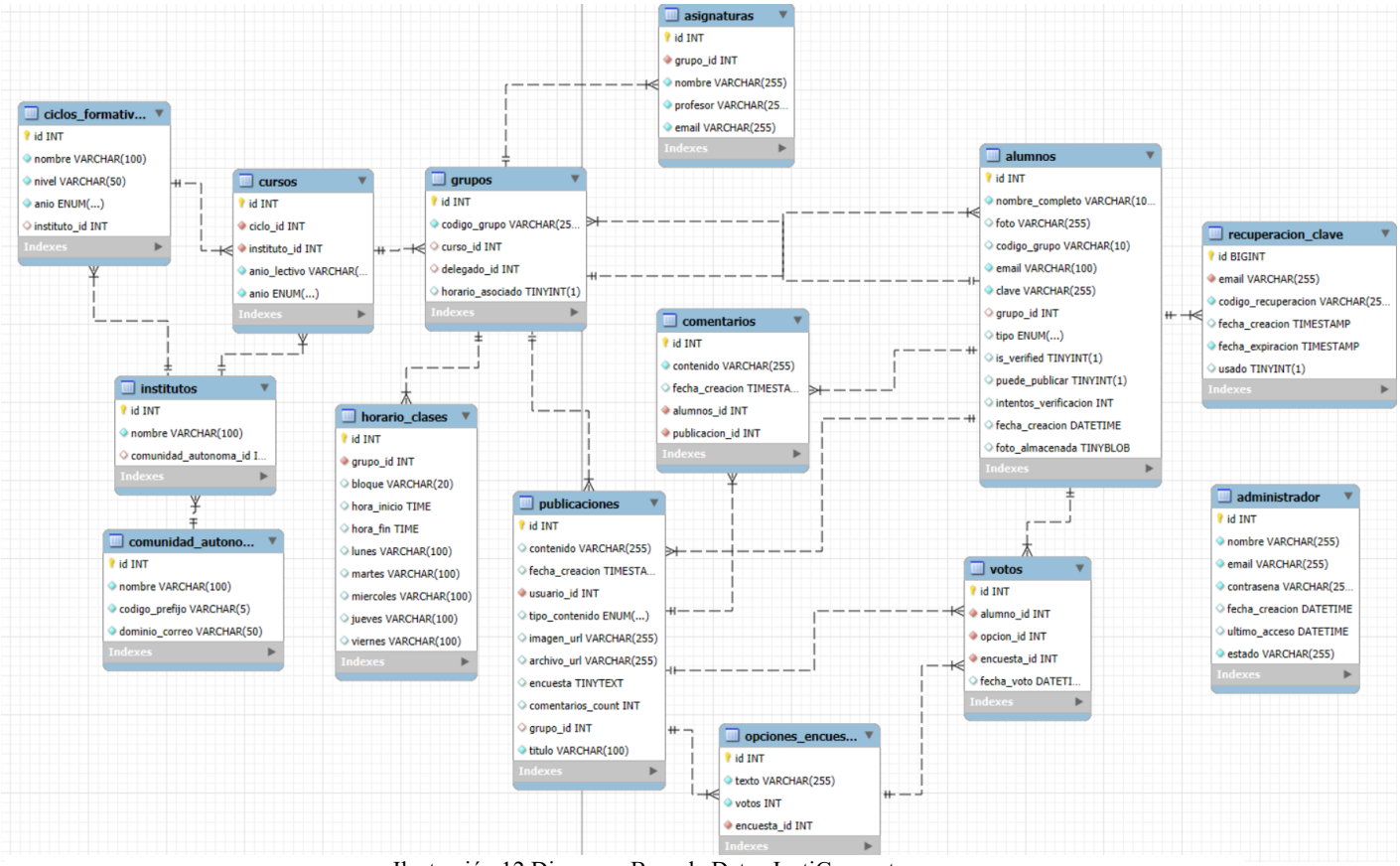

Ilustración 12 Diagrama Base de Datos InstiConnect
Fuente: Elaboración propia

5. TECNOLOGÍAS Y HERRAMIENTAS

5.1 Backend

5.1.1 Spring Boot

Spring Boot es el framework elegido para el desarrollo del backend debido a su capacidad para simplificar la creación de aplicaciones Java empresariales. Este framework permite una configuración mínima gracias a su enfoque dogmático. Esto significa que está diseñado con una serie de configuraciones y decisiones predefinidas basadas en las mejores prácticas, además de proporcionar soporte integrado para dependencias, inyección de código y una amplia variedad de bibliotecas.

El desarrollo de la aplicación ha requerido una selección de dependencias clave para Spring Boot, enfocadas en la funcionalidad y la seguridad del sistema:

- Spring Boot Starter Web: Para implementar la API REST.
- Spring Boot Starter Data JPA: Para la interacción con la base de datos MySQL mediante JPA.
- Spring Boot Starter Security: Para la gestión de autenticación y autorización.
- Spring Boot Starter WebSocket: Para implementar actualizaciones en tiempo real.
- JWT (JSON Web Token): Para la autenticación basada en tokens.
- MySQL Connector: Para la conexión con la base de datos.
- Spring Boot Starter Mail: Para el envío de correos electrónicos.

5.1.2 WebSockets

Se utiliza WebSockets para establecer comunicación en tiempo real entre el cliente y el servidor. Esta tecnología permite una conexión persistente, ideal para casos donde se requiere el intercambio constante de datos, como sistemas de chat o actualizaciones en tiempo real.

5.1.3 JWT para autenticación

JSON Web Tokens (JWT) se emplea como mecanismo de autenticación y autorización. Su uso permite la gestión segura de sesiones y accesos, evitando la necesidad de mantener estados en el servidor y ofreciendo una solución escalable.

5.2 Frontend

5.2.1 HTML, CSS, JavaScript

El desarrollo del frontend se realiza con tecnologías estándar como HTML para la estructura del contenido, CSS para el diseño visual y JavaScript para la lógica del cliente. Esta combinación permite crear una interfaz de usuario interactiva y visualmente atractiva.

5.2.2 Conexión WebSocket y Fetch API

Se integra la conexión mediante WebSockets en el frontend para la comunicación en tiempo real con el servidor. Además, se utiliza la Fetch API para realizar peticiones HTTP asíncronas, lo que facilita el consumo de las APIs expuestas por el backend.

5.3 Base de datos

5.3.1 MySQL

Para el almacenamiento de datos se utiliza MySQL, un sistema de gestión de bases de datos relacional (RDBMS) ampliamente adoptado. Se selecciona esta herramienta por su robustez, confiabilidad y facilidad de integración con Spring Boot.

```
DESCRIBE alumnos;
```

Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nombre_completo	varchar(100)	NO		NULL	
foto	varchar(255)	YES		NULL	
codigo_grupo	varchar(10)	YES		NULL	
email	varchar(100)	NO	UNI	NULL	
clave	varchar(255)	NO		NULL	
grupo_id	int	YES	MUL	NULL	
tipo	enum('delegado','alumno')	YES		alumno	
is_verified	tinyint(1)	YES		0	
puede_publicar	tinyint(1)	YES		0	
intentos_verifica...	int	YES		0	
fecha_creacion	datetime	YES		CURR...	DEFAULT_GEN...
foto_almacenada	tinyblob	YES		NULL	

Ilustración 13 Tabla de alumnos
Fuente: Elaboración propia

```
DESCRIBE grupos;
```

Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
codigo_grupo	varchar(255)	NO	UNI	NULL	
curso_id	int	YES	UNI	NULL	
delegado_id	int	YES	MUL	NULL	
horario_asociado	tinyint(1)	YES		0	

Ilustración 14 Tabla de grupos
Fuente: Elaboración propia

6. FUNCIONAMIENTO DE LA APLICACIÓN

6.1 Registro e inicio de sesión

El sistema de registro e inicio de sesión permite a los usuarios crear cuentas y acceder a la plataforma. Este apartado describe la implementación de las funcionalidades de registro para los roles de alumno y delegado, así como el inicio de sesión para todos los usuarios registrados.

Registro

1. Selección de Rol:

- Los usuarios eligen entre alumno y delegado. Dependiendo del rol seleccionado, se muestra un formulario con campos adaptados a sus necesidades.
- Los delegados deben validar una clave de acceso, lo que añade una capa adicional de seguridad.

Ilustración 16 Captura Rol de Alumno
Fuente: Elaboración propia

Ilustración 15 Captura Clave de Acceso - delegado
Fuente: Elaboración propia

2. Validación de Datos:

- Se utiliza JavaScript para validar los campos del formulario en tiempo real.
- En el backend, los datos se procesan y se verifica su integridad antes de almacenarlos.

```javascript
delegateForm.addEventListener('submit', function (event) {
    event.preventDefault();

    // Mostrar el overlay de carga
    document.getElementById('loadingOverlay').style.display = 'flex';

    // Obtiene el ciclo formativo y el año seleccionado
    const cicloFormativoData = JSON.parse(document.getElementById('cicloFormativo').value || '{}');
    const cicloFormativoNombre = cicloFormativoData.nombre || ""; // Nombre del ciclo formativo
    const anio = cicloFormativoData.anio || ""; // Año del ciclo formativo (primero o segundo)

    // Captura los datos del formulario
    const data = {
        institutoNombre: document.getElementById('instituteName').value || "",
        cicloFormativo: cicloFormativoNombre,
        anioLectivo: document.getElementById('academicYear').value || "",
        anio: anio,
        nombreCompleto: document.getElementById('fullNameDelegate').value || "",
        foto: document.getElementById('photoDelegate').files[0] ? document.getElementById('photoDelegate')
        correoEducativo: document.getElementById('emailDelegate').value || "",
        clave: document.getElementById('passwordDelegate').value || ""
    };

    // Realiza el fetch para enviar los datos
    fetch('http://localhost:8080/api/alumnos/delegado', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    })
    .then(response => {
        if (!response.ok) {
            throw new Error('Error en la solicitud de registro de delegado');
        }
        return response.json();
    })
    .then(result => {
        console.log('Registro exitoso:', result);
```

Ilustración 17 Captura validación de Formulario
Fuente: Elaboración propia

3. Conexión con la Base de Datos:

- Los datos de registro se almacenan en una tabla de MySQL estructurada con los siguientes campos principales:
- nombre_completo, email y clave para la información básica.
- tipo para identificar si el usuario es "Alumno" o "Delegado".
- is_verified y intentos_verificacion para gestionar la verificación del correo.
- Relación con otros datos a través de grupo_id.

```
6 •  DESCRIBE alumnos;
7
```

sult Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
nombre_completo	varchar(100)	NO		NULL	
foto	varchar(255)	YES		NULL	
codigo_grupo	varchar(10)	YES		NULL	
email	varchar(100)	NO	UNI	NULL	
clave	varchar(255)	NO		NULL	
grupo_id	int	YES	MUL	NULL	
tipo	enum('delegado','alumno')	YES		alumno	
is_verified	tinyint(1)	YES		0	
puede_publicar	tinyint(1)	YES		0	
intentos_verifica...	int	YES		0	
fecha_creacion	datetime	YES		CURR...	DEFAULT_GEN...
foto_almacenada	tinyblob	YES		NULL	

Ilustración 18 Base de Datos - Alumnos
Fuente: Elaboración propia

4. Carga Dinámica de Datos:

- Algunas opciones, como las comunidades autónomas, institutos y ciclos formativos, se cargan dinámicamente desde el backend mediante endpoints diseñados para este propósito.

Ilustración 19 Captura Carga dinámica Madrid
Fuente: Elaboración propia

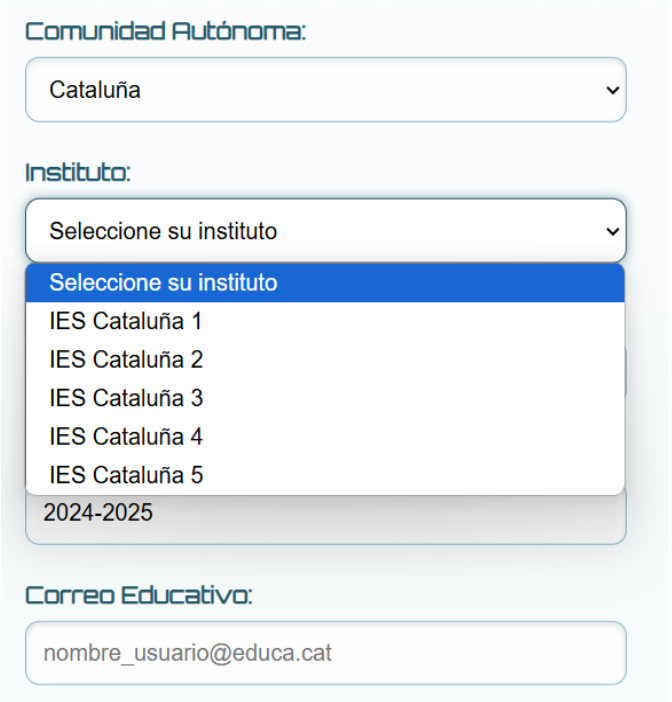

Ilustración 20 Captura Carga dinámica Cataluña
Fuente: Elaboración propia

6.2 Gestión de roles y grupos

La gestión de grupos y roles es un componente clave en el sistema, diseñado para estructurar a los usuarios en grupos y delegar responsabilidades según sus roles. El sistema distingue tres roles principales: Alumno, Delegado y Administrador General, cada uno con privilegios y responsabilidades específicas.

Roles

1. Alumno: Miembro regular de un grupo, con permisos básicos, como acceder a contenido del grupo y participar en encuestas o publicaciones si tiene los permisos habilitados.
2. Delegado: Usuario con funciones de liderazgo dentro de un grupo. Puede gestionar asignaturas, aprobar permisos de publicación y representar al grupo.
3. Administrador General: Usuario con privilegios para gestionar globalmente la plataforma, incluyendo la administración de usuarios y supervisión de actividades.

id	nombre_completo	foto	codigo_grupo	email	clave	grupo_id	tipo	is_verified	puede_publicar
84	verne	uploa...	99-5821	robertoc...	$2a$10$...	49	delegado	1	1
86	juanperez	iconD...	99-5821	juanper...	$2a$10$...	49	alumno	1	1
87	mariafernandez	iconD...	99-5821	mariafer...	$2a$10$...	49	alumno	1	1
88	pedrogomez	iconD...	99-5821	pedrogo...	$2a$10$...	49	alumno	1	1
94	roberto.campo3	uploa...	91-1760	roberto....	$2a$10$...	52	delegado	1	1
95	Robert Scoot	uploa...	91-8861	alexserj...	$2a$10$f...	53	delegado	1	1
96	Usuario Eliminado	NULL	NULL	usuario_...	clave_eli...	NULL	alumno	1	0
97	Romeo	uploa...	91-1002	admin@t...	$2a$10$...	54	delegado	1	1
98	noen	iconD...	91-1002	default...	$2a$10$i...	54	alumno	1	1
99	Ale	uploa...	91-8861	acua.ale...	$2a$10$...	53	alumno	1	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 21 Captura Roles Alumnos - id Grupo
Fuente: Elaboración propia

Grupos

El sistema organiza a los alumnos en grupos asociados a un código único. Los grupos se vinculan a un curso, un delegado y un conjunto de asignaturas.

Ilustración 22 Captura Perfil delegado - Código de grupo
Fuente: Elaboración propia

1. **Visualización de Grupos y Miembros**:

- Los delegados pueden consultar detalles de su grupo, como la lista de integrantes, utilizando APIs dedicadas. Por ejemplo, el endpoint /api/grupo/integrantes devuelve una lista de alumnos con sus datos clave.
- Los alumnos pueden consultar su información personal y estado en el grupo mediante el endpoint /api/alumnos/info.

2. **Gestión de Permisos**:

- Los delegados pueden otorgar o revocar permisos de publicación a los miembros del grupo. Esta acción se realiza a través de la API /api/grupo/integrantes/{id}/publicar.
- Los cambios de permisos se notifican en tiempo real a través de WebSockets, mejorando la experiencia del usuario.

Ilustración 23 Captura Integrantes del Grupo - Roles - Permisos
Fuente: Elaboración propia

3. **Gestión de Asignaturas**:

- Los delegados pueden agregar, actualizar o eliminar asignaturas asociadas al grupo mediante APIs específicas.
- Las asignaturas se vinculan al grupo, y los cambios se notifican a los integrantes mediante WebSockets para garantizar sincronización.

Ilustración 24 Captura Gestión de Asignaturas
Fuente: Elaboración propia

4. **Autenticación y Seguridad**:

- Todas las operaciones están protegidas mediante autenticación JWT, asegurando que solo los usuarios autorizados puedan acceder o modificar información sensible.
- Los administradores generales tienen acceso a herramientas adicionales para gestionar la plataforma globalmente.

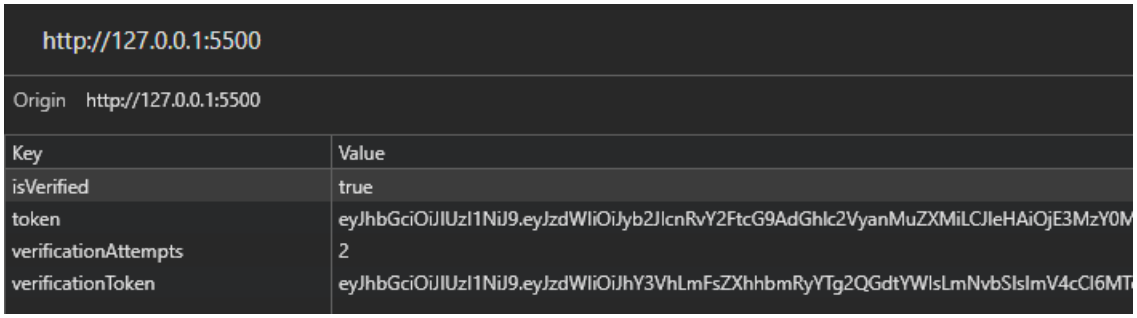

Ilustración 25 Captura Local Storage - token JWT
Fuente: Elaboración propia

6.3 Funcionalidades de publicaciones

El sistema incluye un módulo avanzado de publicaciones y encuestas que fomenta la interacción entre los integrantes de un grupo. Este módulo permite crear publicaciones con diferentes tipos de contenido, realizar encuestas interactivas y gestionar comentarios.

Publicaciones

Los usuarios pueden crear publicaciones con diferentes tipos de contenido:

- Texto: Publicaciones simples de texto.
- Imágenes: Con opción de cargar y mostrar imágenes en el muro.
- Archivos: Posibilidad de subir y descargar archivos.
- Encuestas: Permiten a los usuarios votar en opciones predefinidas.
- Comunicados: Información oficial enviada por delegados.

```
select * from publicaciones;
```

lt Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Con

id	contenido	fecha_creacion	usuario_id	tipo_contenido
53	Fichero subido	2024-11-19 18:36:24	94	archivo
54	Imagen subida	2024-11-19 18:37:40	94	imagen
55	Imagen subida	2024-11-19 18:41:46	94	imagen
56	Fichero subido	2024-11-19 18:42:07	94	archivo
57	Fichero subido	2024-11-19 18:42:21	94	archivo
58	Fichero subido	2024-11-19 18:42:48	94	archivo
102	Imagen subida	2024-11-19 19:04:59	94	imagen
103	Fichero subido	2024-11-19 19:05:09	94	archivo
152	Imagen subida	2024-11-19 19:24:32	94	imagen
202	Fichero subido	2024-11-20 17:14:13	84	archivo
252	Fichero subido	2024-11-20 22:17:05	94	archivo
253	Imagen subida	2024-11-20 22:17:16	94	imagen
254	Imagen subida	2024-11-20 22:17:49	94	imagen
255	Imagen subida	2024-11-20 22:18:16	94	imagen

Ilustración 26 Tabla de Publicaciones - Tipo Contenido
Fuente: Elaboración propia

Creación de encuestas:

- Los delegados pueden crear encuestas para su grupo. Cada encuesta incluye una pregunta y un conjunto de opciones.
- La API valida que el usuario tenga permisos para crear encuestas y esté asignado a un grupo.

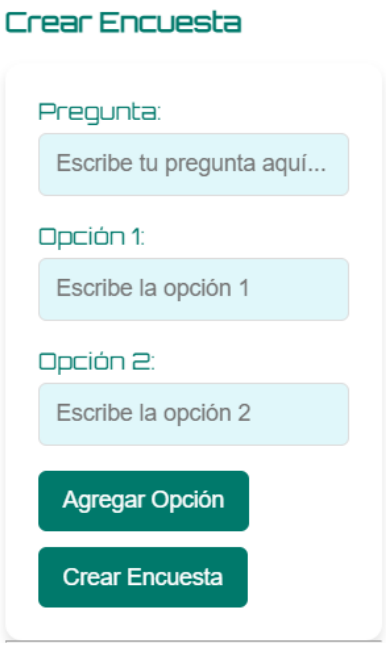

Ilustración 27 Formulario de Encuesta
Fuente: Elaboración propia

Votación:

- Los alumnos pueden votar en encuestas activas dentro de su grupo.
- El sistema asegura que cada alumno pueda votar solo una vez por encuesta.
- Las votaciones se gestionan mediante WebSockets para actualizar los resultados en tiempo real.

Visualización de resultados:

- Los resultados de la encuesta se presentan en tiempo real tras cada voto.
- Los alumnos pueden ver las opciones disponibles y los votos acumulados.

Ilustración 28 Encuesta-Resultados -Comentarios
Fuente: Elaboración propia

6.4 Gestión de horarios

La gestión de horarios en el proyecto es una funcionalidad importante que permite al delegado de un grupo organizar y administrar los horarios de clases. Este sistema incluye la creación, edición y sincronización de horarios, conectándose con la base de datos de asignaturas y actualizándose en tiempo real mediante WebSocket.

```
describe horario_clases;
```

Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
grupo_id	int	NO	MUL	NULL	
bloque	varchar(20)	YES		NULL	
hora_inicio	time	YES		12:00:00	
hora_fin	time	YES		12:00:00	
lunes	varchar(100)	YES		Libre	
martes	varchar(100)	YES		Libre	
miercoles	varchar(100)	YES		Libre	
jueves	varchar(100)	YES		Libre	
viernes	varchar(100)	YES		Libre	

Ilustración 29 Tabla Horario Clases
Fuente: Elaboración propia

```
select * from asignaturas;
```

Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	grupo_id	nombre	profesor	email
6	49	PROGRAMACION	JULIO	juliopañi@gmail.com
7	49	BASE DE DATOS	MATEO	mateo@gbaiojs.com
11	49	PYTON	JUAN	juanhuk@gmail.com
15	49	RECURSOS HUMANOS	NEREA	nerea_rrhhh@gmail.com
19	49	WORD	TRILINE	triline@gmail.com
22	52	TEST	TEST PROFESOR	test@test.com
23	49	PROCESO DE ACTIVIDAD COMERCIAL	MARIA VAZQUEZ	ana.vazquez@educamadrid.com
25	52	MATEMATICAS	MARIA	eli15@mates.com
26	49	LENGUAJE	ELI	eleiciuabd.@gmail.com
27	53	BBDD	ALEX	alex@edu.madrid.org
33	54	MATE	JULIO	julio@mate.org
34	53	MATEMATICAS	GUSTAVO	gustavo.matematicas@gob.es
35	53	LENGUAJE	LUCAS	lucas.lenguaje@gob.es

Ilustración 30 Ejemplo tabla de Asignaturas guardadas
Fuente: Elaboración propia

Cuando un grupo no tiene un horario asignado, al crear el grupo cuando se registra el delegado se genera un horario predeterminado. Este horario inicializa el sistema con bloques horarios predefinidos, como primera hora y segunda hora, y asigna automáticamente el estado "Libre" a todas las franjas horarias.

```
private HorarioClases crearBloqueHorario(Grupo grupo, String bloque) {
    HorarioClases horario = new HorarioClases();
    horario.setGrupo(grupo);
    horario.setBloque(bloque);
    horario.setHoraInicio(Time.valueOf( s: "12:00:00"));
    horario.setHoraFin(Time.valueOf( s: "12:00:00"));
    horario.setLunes("Libre");
    horario.setMartes("Libre");
    horario.setMiercoles("Libre");
    horario.setJueves("Libre");
    horario.setViernes("Libre");
    return horario;
}
```

Ilustración 31 Horario Default
Fuente: Elaboración propia

Después de crear el horario, el delegado puede modificarlo desde su perfil. Esto se hace mediante un modal que permite ajustar las horas de inicio y fin de cada bloque, seleccionar asignaturas disponibles o establecer franjas como "Libre". Los cambios realizados se guardan en la base de datos y se notifican a los alumnos del grupo en tiempo real a través de WebSocket.

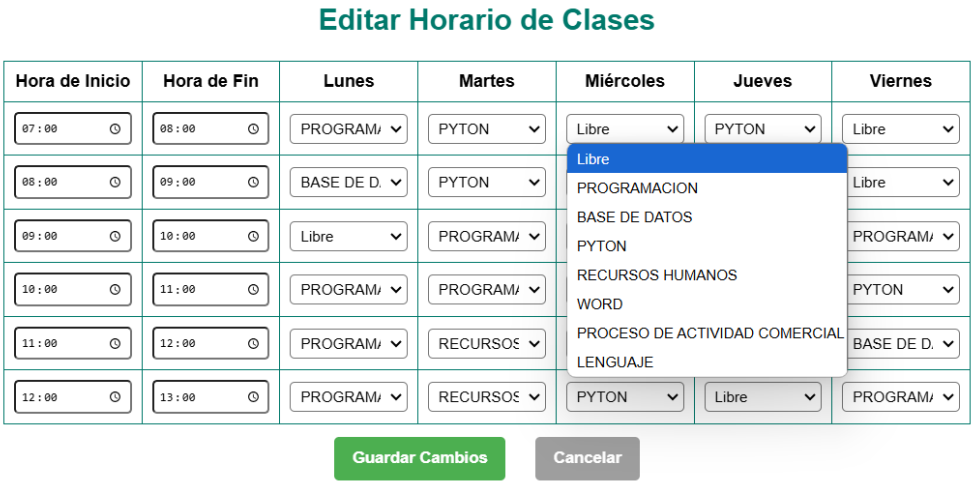

Ilustración 32 Modal Horario
Fuente: Elaboración propia

La sincronización en tiempo real es una parte fundamental del sistema. Cualquier cambio hecho por el delegado, como la modificación de franjas horarias o asignaturas, se comunica de inmediato a los alumnos mediante WebSocket. Esto asegura que los alumnos siempre tengan la información más actualizada en sus perfiles.

El horario está vinculado a la base de datos de asignaturas. Esto significa que cualquier cambio en las asignaturas, como el nombre del profesor, eliminación o adición de una asignatura, se reflejará automáticamente en el horario del grupo.

```
public void notifyHorarioUpdate(String mensaje) {  3 usages  Roberto Campo
    Map<String, String> payload = new HashMap<>();
    payload.put("tipo", "ACTUALIZACION_HORARIO");
    payload.put("mensaje", mensaje);

    try {
        String jsonMensaje = new ObjectMapper().writeValueAsString(payload);
        for (WebSocketSession session : sessions) {
            if (session.isOpen()) {
                session.sendMessage(new TextMessage(jsonMensaje));
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Ilustración 33 Extracto de código Socket notificación
Fuente: Elaboración propia

6.5 Verificación de correo electrónico

La verificación de correo electrónico es una funcionalidad para garantizar la autenticidad de los usuarios en el sistema. Este proceso permite validar las direcciones de correo proporcionadas por los alumnos al registrarse, asegurando que solo usuarios legítimos puedan acceder a las funcionalidades completas de la plataforma.

Ilustración 34 Correo de Verificación
Fuente: Elaboración propia

Cuando un alumno se registra, se genera un token único vinculado a su dirección de correo electrónico. Este token es enviado al usuario mediante un enlace de verificación que, al ser accedido, activa el endpoint correspondiente para validar el token. Si el token es válido, se actualiza el estado del usuario a "verificado". Este proceso está diseñado para ser rápido y seguro, y los endpoints involucrados en esta tarea son:

- **GET /api/test-verify**: Valida el token proporcionado en el enlace de verificación. Si es correcto, el estado del usuario cambia a "verificado".

```
boolean isValidToken = tokenService.validateToken(token);
if (isValidToken) {
    String email = tokenService.extraerCorreo(token);
    log.info("Token válido. Correo extraído: {}", email);
    Optional<Alumno> alumnoOptional = alumnoService.findByEmail(email);
    if (alumnoOptional.isPresent()) {
        Alumno alumno = alumnoOptional.get();
        alumno.setIsVerified(true);  // Marca al alumno como verificado
        alumnoService.saveAlumno(alumno);
        String tipo = alumno.getTipo().name();
        log.info("Alumno con correo {} verificado exitosamente.", email);
        return ResponseEntity.ok().body(new VerificacionResponse( status: "Tu correo ya está verificado, cierra la ventana", tipo));
    } else {
        log.warn("No se encontró el alumno con el correo: {}", email);
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
                .body(new ConcreteErrorResponse( status: "error",  message: "No se encontró el usuario con el correo: " + email));
    }
```

Ilustración 35 Extracto código de Api Verificación
Fuente: Elaboración propia

- **POST /api/resend-verification**: Permite reenviar un correo de verificación en caso de que el usuario no haya completado el proceso inicial.

```
@PostMapping("/resend-verification")  Roberto Campo
public ResponseEntity<?> reenviarCorreoVerificacion(@RequestBody ReenviarCorreoRequest request) {
    log.info("Recibiendo solicitud para reenviar correo de verificación.");
    try {
        // Extraer el correo a partir del token proporcionado en la solicitud
        String email;
        try {
            email = tokenService.extraerCorreo(request.getToken());
        } catch (Exception e) {
            log.error("Error al extraer el correo del token: ", e);
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                    .body(new ConcreteErrorResponse( status: "error",  message: "Token inválido o expirado."));
        }

        if (email == null || email.isEmpty()) {
            log.warn("El token es inválido o ha expirado. No se pudo extraer el correo.");
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                    .body(new ConcreteErrorResponse( status: "error",  message: "Token inválido o expirado."));
        }

        // Buscar al alumno utilizando el correo extraído del token
        Optional<Alumno> alumnoOptional = alumnoService.findByEmail(email);
        if (alumnoOptional.isPresent()) {
            Alumno alumno = alumnoOptional.get();
```

Ilustración 36 Extracto código de renvió de verificación
Fuente: Elaboración propia

En caso de que el usuario no verifique su correo en el primer intento, el sistema permite hasta tres reintentos para completar el proceso. Si se alcanza este límite, se bloquea temporalmente la posibilidad de reenviar el correo. Durante cada intento de reenvío, el sistema genera un nuevo token y envía un enlace actualizado al correo del usuario.

Para garantizar que el sistema funcione correctamente, se implementa un servicio de correo electrónico que envía los mensajes de verificación utilizando un formato claro y sencillo. Este correo incluye un enlace único que redirige al usuario al endpoint de verificación.

6.6 Recuperación de contraseñas

La recuperación de contraseñas se implementa para garantizar que los usuarios puedan restablecer su acceso en caso de olvidar su contraseña. Este proceso se compone de tres pasos principales: envío de un código de recuperación, validación del código recibido y actualización de la contraseña.

El primer paso consiste en que el usuario proporcione su correo electrónico mediante un formulario. El sistema valida que el correo exista en la base de datos y, si es así, genera un código de recuperación de seis dígitos. Este código tiene una vigencia de 10 minutos y se almacena junto con la dirección de correo en la tabla de recuperación de contraseñas. Una vez generado, el código se envía al usuario mediante un correo electrónico. Esta funcionalidad asegura que solo las direcciones de correo válidas puedan iniciar el proceso.

Ilustración 37 Captura Recuperación Contraseña
Fuente: Elaboración propia

Cuando el usuario recibe el código, pasa al segundo paso, que consiste en verificarlo. En esta etapa, el usuario introduce el código recibido en otro formulario. El sistema busca el código en la base de datos, valida que coincida con el correo y comprueba que no haya expirado ni haya sido usado previamente. Si el código es válido, el sistema permite al usuario avanzar al siguiente paso; en caso contrario, notifica que el código es incorrecto o que ya no es válido.

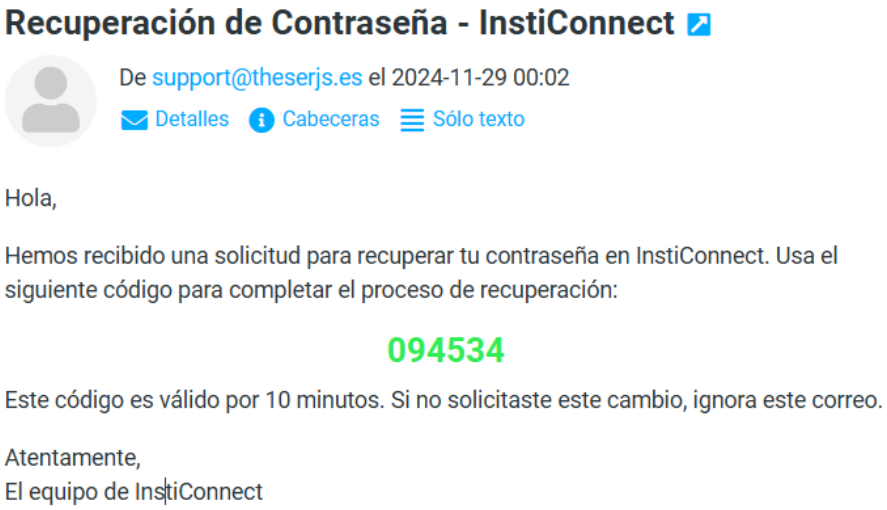

Ilustración 38 Correo Electrónico Recuperación de Contraseña
Fuente: Elaboración propia

El último paso es el cambio de contraseña. En esta etapa, el usuario proporciona una nueva contraseña y la confirma.

Para garantizar la seguridad, la contraseña es cifrada utilizando el algoritmo BCrypt antes de almacenarla en la base de datos. Una vez actualizada la contraseña, el sistema marca el código de recuperación como usado, lo que asegura que no pueda reutilizarse para otro intento.

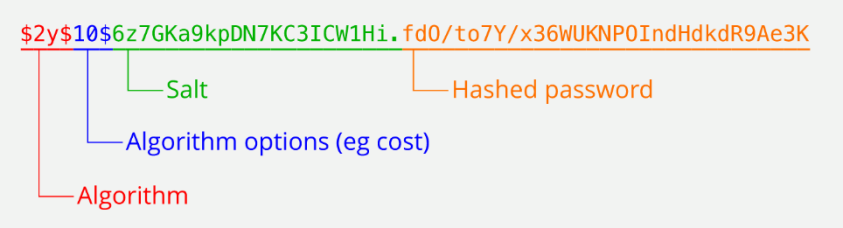

Ilustración 39 Estructura de un hash generado con Bcrypt
Fuente: blog.segu-info.com.ar

Todo el proceso se complementa con medidas de seguridad adicionales, como la limitación del tiempo de validez del código y la protección contra el uso de códigos expirados. Esto asegura que el flujo sea seguro y robusto, minimizando riesgos de accesos no autorizados.

```
// Generar código y tiempo de expiración
String recoveryCode = String.format("%06d", new Random().nextInt( bound: 1000000));
LocalDateTime expirationTime = LocalDateTime.now().plusMinutes(10);
```

Ilustración 40 Extracto de código de Recuperación
Fuente: Elaboración propia

Además, para mejorar la experiencia del usuario, se emplean mensajes visuales y un indicador de carga en cada paso del proceso. Esto ayuda al usuario a comprender lo que está sucediendo y cuándo debe esperar una respuesta del sistema. En el caso de errores, como un correo no registrado o un código incorrecto, el sistema muestra mensajes claros y específicos.

Finalmente, esta funcionalidad garantiza que el usuario pueda recuperar el acceso de manera efectiva, sencilla y segura, proporcionando una experiencia confiable y alineada con las mejores prácticas de seguridad.

7. WEBSOCKETS

Los WebSockets son un protocolo de comunicación en tiempo real diseñado para establecer una conexión bidireccional persistente entre un cliente y un servidor. Este protocolo se diferencia de las peticiones HTTP tradicionales por permitir una interacción continua y eficiente, lo que lo convierte en una herramienta clave para aplicaciones que requieren actualizaciones instantáneas y comunicación en tiempo real.

7.1 Comunicación en tiempo real

La comunicación en tiempo real es uno de los principales desafíos en el desarrollo de aplicaciones modernas. WebSockets resuelve este problema permitiendo que el servidor y el cliente intercambien mensajes directamente, sin necesidad de realizar múltiples peticiones HTTP.

Características técnicas de WebSockets:

- Conexión persistente: Una vez establecida, la conexión se mantiene activa, eliminando la necesidad de abrir y cerrar múltiples conexiones.
- Bajo consumo de recursos: Reduce significativamente la latencia y el tráfico al evitar el "overhead" de las cabeceras HTTP en cada mensaje.
- Soporte nativo: Integrado en la mayoría de los navegadores modernos y frameworks de backend, lo que facilita su implementación.
- Bidireccionalidad: Permite que tanto el cliente como el servidor envíen y reciban datos de manera simultánea, ideal para casos de sincronización en tiempo real.

Proceso técnico:

a) Handshake inicial: La comunicación comienza con una petición HTTP estándar, que se transforma en una conexión WebSocket a través del proceso inicial.

b) Envío de mensajes: Una vez establecida la conexión, el cliente y el servidor pueden enviar mensajes en formato binario o de texto.

c) Eventos asociados:

Los WebSockets tienen eventos clave que permiten manejar la comunicación de manera eficiente:

- onopen: Se activa cuando la conexión WebSocket se establece correctamente.
- onmessage: Se activa cuando el cliente recibe un mensaje desde el servidor.
- onerror: Se activa cuando ocurre un error en la comunicación WebSocket.
- onclose: Se activa cuando la conexión WebSocket se cierra, ya sea de forma intencional o por un problema.

7.2 Uso en las funcionalidades clave

En este proyecto, los WebSockets han sido esenciales para garantizar que los usuarios reciban actualizaciones en tiempo real en varias funcionalidades.

7.2.1 Horario

La gestión de horarios utiliza WebSockets para sincronizar cambios realizados por el delegado con los perfiles de los alumnos. Esto garantiza que cualquier modificación en el horario (como cambios en las franjas horarias o asignaturas) se refleje instantáneamente en todos los dispositivos conectados.

a) Implementación técnica:

- Canal WebSocket: ws://localhost:8080/ws/horario

b) Eventos manejados:

- ACTUALIZACION_HORARIO: Notifica a los clientes sobre cambios en el horario, actualizando la tabla de horarios en tiempo real.
- CONFIRMACION: Envía un mensaje de confirmación al delegado tras guardar cambios.

c) Ejemplo técnico:

El servidor emite un mensaje JSON con la estructura del nuevo horario, que es recibido y procesado por los clientes para actualizar la interfaz de usuario sin recargar la página.

```
socketHorario.onmessage = (event) => {
    try {
        const mensaje = JSON.parse(event.data);
        if (typeof mensajeCallback === 'function') {
            mensajeCallback(mensaje);
        } else {
            switch (mensaje.tipo) {
                case 'ACTUALIZACION_HORARIO':
                    console.log('Horario actualizado:', mensaje.mensaje);
                    cargarHorarioAlumno(token); // Recargar horario
                    break;
                case 'CONFIRMACION':
                    console.log('Mensaje de confirmación recibido:', mensaje.mensaje);
                    break;
                case 'ELIMINAR_ASIGNATURA': // Manejar el mensaje de eliminar asignatura
                    console.log('Se eliminó una asignatura:', mensaje.mensaje);
                    cargarAsignaturasDelGrupo(token); // Actualiza las asignaturas en la interfaz
                    break;
                case 'NUEVA_ASIGNATURA': // Manejar el mensaje de nueva asignatura
                    console.log('Nueva asignatura añadida:', mensaje.mensaje);
                    cargarAsignaturasDelGrupo(token); // Refresca la lista de asignaturas
                    break;
                default:
                    console.warn('Tipo de mensaje no reconocido:', mensaje.tipo);
            }
        }
```

Ilustración 41 Gestión de Mensajes en WebSocket
Fuente: Elaboración propia

7.2.2 Publicaciones

El muro de publicaciones y encuestas se mantiene sincronizado utilizando WebSockets. Esto permite que los usuarios vean nuevas publicaciones, actualizaciones de encuestas o comentarios sin necesidad de refrescar manualmente la página.

a) Implementación técnica:

- Canal WebSocket: ws://localhost:8080/ws/notificaciones

b) Eventos manejados:

- NUEVA_PUBLICACION: Notifica sobre nuevas publicaciones en el grupo.

- ACTUALIZACION_ENCUESTA: Informa sobre resultados de encuestas en tiempo real.
- NUEVO_COMENTARIO: Actualiza la lista de comentarios en publicaciones específicas.

c) Ventaja principal: La experiencia del usuario mejora significativamente al eliminar la necesidad de realizar peticiones repetidas al servidor para buscar actualizaciones.

```
// Función para conectar al WebSocket
function conectarWebSocket(token) {
    const socket = new WebSocket("ws://localhost:8080/ws/notificaciones");

    socket.onopen = () => {
        console.log("Conexión WebSocket establecida.");
        // Si es necesario, autentica al WebSocket enviando el token
        socket.send(JSON.stringify({ tipo: "AUTENTICACION", token }));
    };

    socket.onerror = (error) => {
        console.error("Error en el WebSocket:", error);
    };

    socket.onclose = () => {
        console.warn("WebSocket cerrado. Reintentando en 5 segundos...");
        setTimeout(() => conectarWebSocket(token), 5000); // Reintentar conexión
    };

    return socket;
}
```

Ilustración 42Evento de Conexión y Reintento en WebSocket
Fuente: Elaboración propia

7.2.3 Permisos

El sistema de permisos utiliza WebSockets para garantizar que los cambios realizados por los administradores (como la revocación o asignación de permisos) se reflejen de inmediato en los usuarios afectados. Esto es crucial para mantener la seguridad y el control de acceso en tiempo real.

a) Implementación técnica:
   - Canal WebSocket: ws://localhost:8080/ws/permisos

b) Eventos manejados:
   - CAMBIO_PERMISOS: Informa a un usuario sobre la actualización de sus permisos.

- NOTIFICACION_ADMIN: Notifica a los administradores sobre acciones realizadas en el sistema de permisos.

c) Sincronización en tiempo real: Asegura que los usuarios afectados no puedan acceder a funcionalidades restringidas en cuanto se revocan sus permisos.

```
public class PermisoWebSocketHandler extends TextWebSocketHandler {

    private final List<WebSocketSession> sessions = new ArrayList<>();  3 usages

    @Override  no usages  Roberto Campo
    public void afterConnectionEstablished(WebSocketSession session) { sessions.add(session); }

    @Override  no usages  Roberto Campo
    public void afterConnectionClosed(WebSocketSession session, CloseStatus status) { sessions.remove(session);

    public void notifyClients(String mensajeJson) {  Roberto Campo
        for (WebSocketSession session : sessions) {
            if (session.isOpen()) {
                try {
                    session.sendMessage(new TextMessage(mensajeJson));
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Ilustración 43 Gestión de Sesiones en WebSocket - Permisos
Fuente: Elaboración propia

8. JSON WEB TOKENS JWT

Los JSON Web Tokens (JWT) son un estándar utilizado para la autenticación y transmisión de información de forma segura entre dos partes. Este sistema permite verificar la identidad de un usuario sin necesidad de almacenar información en el servidor, asegurando la integridad de los datos mediante una firma digital.

8.1 ¿Qué es un Token JWT y cómo funciona?

Un JWT es un token que consta de tres partes principales:

- Header (Encabezado): Contiene el tipo de token y el algoritmo de firma.
- Payload (Carga útil): Incluye datos como el nombre de usuario y permisos.
- Signature (Firma): Garantiza la autenticidad del token combinando el encabezado, el payload y una clave secreta.

Los JWT se utilizan para autenticar usuarios y proteger recursos, integrándolos en todas las solicitudes mediante el encabezado Authorization: Bearer <token>.

Ilustración 44 Ciclo de vida de un Token JWT
Fuente: openwebinars.net

8.2 Implementación de JWT en el Proyecto

El sistema utiliza un enfoque basado en la configuración, generación, validación y uso de tokens. A continuación, se describen los pasos clave:

- Configuración y Generación

  La configuración se define en el archivo JwtConfig, donde se especifica la clave secreta (jwt.secret) y la duración del token (jwt.expirationMs). Mediante el método generateToken, se crea un token que incluye datos como el usuario y la fecha de expiración, firmado con el algoritmo HS256.

```
private String createToken(Map<String, Object> claims, String subject) {  1 usage  Roberto Campo

    return Jwts.builder()
            .setClaims(claims)
            .setSubject(subject)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + jwtConfig.getExpirationMs()))
            .signWith(SignatureAlgorithm.HS256, jwtConfig.getSecretKey().getBytes())
            .compact();
```

Ilustración 45 Generación de Tokens JWT
Fuente: Elaboración propia

- Validación y Extracción de Información

  El token se válida para asegurar que:

  a) El usuario extraído del token coincida con el esperado.

  b) El token no haya expirado.

  Además, se pueden extraer datos específicos, como el nombre de usuario o la fecha de expiración, utilizando el método extractClaim.

```
public String extractUsername(String token) { return extractClaim(token, Claims::getSubject); }

public Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); }

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {  2 usages  Roberto Campo
    final Claims claims = extractAllClaims(token);
    return claimsResolver.apply(claims);
}

public Claims extractAllClaims(String token) {  1 usage  Roberto Campo
    return Jwts.parser()  JwtParser
            .setSigningKey(jwtConfig.getSecretKey().getBytes())
            .parseClaimsJws(token)  Jws<Claims>
            .getBody();
}
```

Ilustración 46 Extracto de Código Extracción de Claims en JWT
Fuente: Elaboración propia

8.3 Uso y Seguridad de JWT en el

Uso en el Proyecto

- Autenticación: Al iniciar sesión, el servidor genera un JWT que se envía al cliente. Este token debe ser incluido en todas las solicitudes protegidas.
- Verificación de Acceso: El servidor valida el token en cada solicitud, permitiendo el acceso solo si es válido.

Medidas de Seguridad

- Para garantizar la seguridad del sistema:
- Claves seguras: La clave secreta se mantiene privada.
- Expiración limitada: Los tokens tienen un tiempo de validez corto para reducir riesgos.
- HTTPS: Toda comunicación entre cliente y servidor se realiza de forma cifrada.
- Revocación: En caso de compromiso, se puede invalidar el token actualizando la clave secreta o implementando listas negras.

```
try {
    // Validar token y obtener alumno
    String email = jwtUtil.extractUsername(token.replace( target: "Bearer ",  replacement: ""));
    Optional<Alumno> alumnoOptional = alumnoRepository.findByEmail(email);

    if (alumnoOptional.isEmpty()) {
        return new ResponseEntity<>( body: "Alumno no autorizado", HttpStatus.UNAUTHORIZED);
    }
```

Ilustración 47 Validación de Token JWT y Autorización de Usuario
Fuente: Elaboración propia

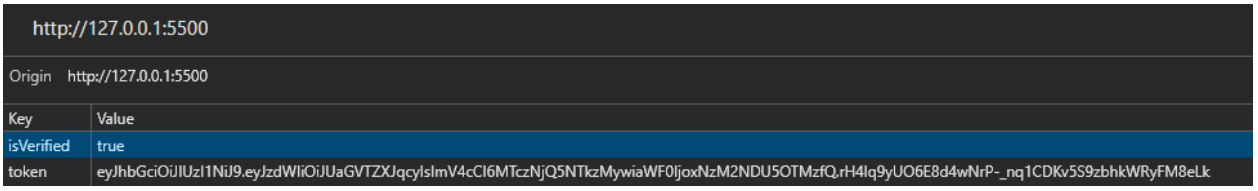

Ilustración 48 Cabeceras de Respuesta con Verificación y Token JWT - Local Storage
Fuente: Elaboración propia

9. PRUEBAS DE SEGURIDAD Y COMUNICACIÓN

las pruebas realizadas para validar los componentes clave del sistema relacionados con la seguridad y la comunicación en tiempo real. Dado que la autenticación mediante tokens JWT y la funcionalidad de WebSocket son fundamentales para garantizar la protección de los datos y la interacción eficiente entre los usuarios y el servidor, estas pruebas se enfocan en asegurar su correcto funcionamiento.

9.1 Pruebas unitarias

Para realizar las pruebas unitarias, utilizamos Mockito, una biblioteca de pruebas ampliamente utilizada en proyectos Java. Mockito es una herramienta diseñada específicamente para facilitar la creación de objetos simulados, lo que permite probar componentes individuales de un sistema sin depender de sus entornos o dependencias externas.

¿Por qué utilizamos Mockito?

- Aislamiento: Mockito nos permite simular el comportamiento de servicios, repositorios u otras dependencias, aislando el código que queremos probar.

- Eficiencia: Al eliminar la necesidad de acceder a recursos externos como bases de datos o APIs reales, las pruebas son más rápidas y manejables.
- Verificación: Podemos comprobar que los métodos de nuestras clases interactúan correctamente con sus dependencias.

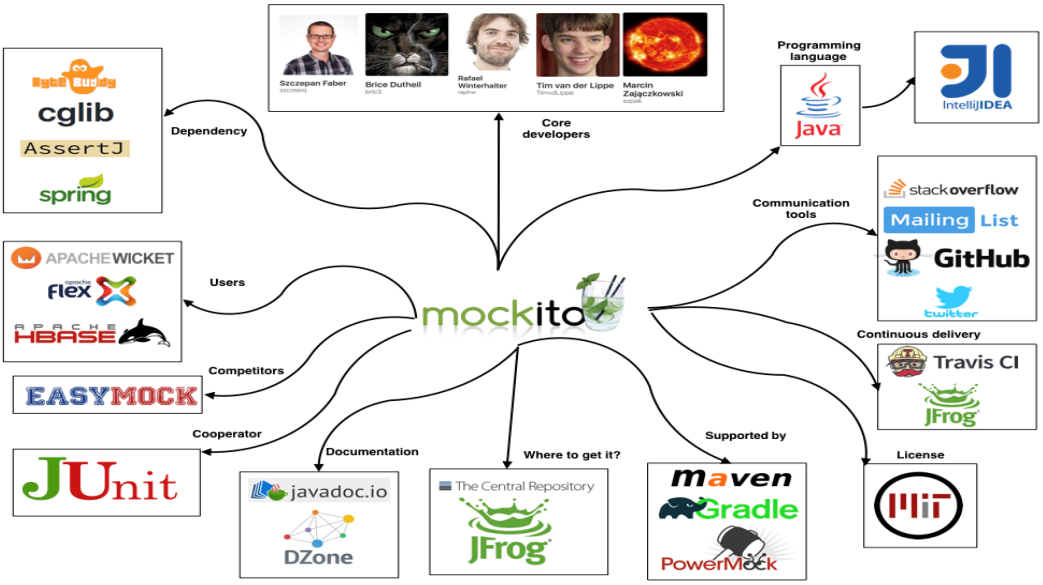

Ilustración 49 Vista de contexto de Mockito
Fuente: delftswa.gitbooks.io

9.1.1 Prueba de Autenticación con Token

Esta prueba tiene como objetivo validar la generación y verificación de tokens de autenticación (JWT) en el sistema. Se simula la autenticación de un usuario mediante credenciales válidas e inválidas y se asegura que el sistema genere o rechace tokens correctamente.

Estas pruebas aseguran que el sistema cumple con los estándares de seguridad y funcionalidad requeridos para gestionar autenticaciones.

Casos de prueba implementados

1. Generación de token: Validar que el token generado sea no nulo y contenga información codificada correctamente.

```
@Test  new *
public void generarToken() {
    String token = jwtUtil.generateToken(username);
    System.out.println("Token generado: " + token);
    assertNotNull(token,  message: "El token generado no debe ser nulo");
    assertTrue( condition: token.length() > 0,  message: "El token debe tener contenido");
}
```

Ilustración 50 Extracto de Código generarToken
Fuente: Elaboración propia

2. Extracción de nombre de usuario: Asegurar que el nombre de usuario extraído del token coincida con el utilizado durante la generación.

```
@Test  new *
public void extraerNombreDeUsuario() {
    String token = jwtUtil.generateToken(username);
    String nombreExtraido = jwtUtil.extractUsername(token);
```

Ilustración 51 Extracto de Código extracción de nombre de usuario
Fuente: Elaboración propia

3. Extracción de fecha de expiración: Verificar que el token contenga una fecha de expiración válida y posterior a la fecha actual.

```
@Test  new *
public void extraerNombreDeUsuario() {
    String token = jwtUtil.generateToken(username);
    String nombreExtraido = jwtUtil.extractUsername(token);
    assertEquals(username, nombreExtraido);}
```

Ilustración 52 Extracto de Código Extracción de fecha de expiración
Fuente: Elaboración propia

4. Validación de token válido: Confirmar que el sistema reconozca como válido un token generado correctamente.

```
@Test  new *
public void validarTokenValido() {
    String token = jwtUtil.generateToken(username);
    boolean esValido = jwtUtil.validateToken(token, username);
    assertTrue(esValido);}
```

Ilustración 53 Extracto de Código Validación de token válido
Fuente: Elaboración propia

5. Validación de token expirado: Simular la expiración del token y comprobar que sea rechazado.

```
@Test  new *
public void validarTokenExpirado() throws InterruptedException {
    Mockito.when(jwtConfig.getExpirationMs()).thenReturn( t: 1L);
    String token = jwtUtil.generateToken(username);
    Thread.sleep( millis: 2);
    boolean esValido = jwtUtil.validateToken(token, username);
    assertFalse(esValido);}
```

Ilustración 54 Extracto de Código Validación de token expirado
Fuente: Elaboración propia

6. Validación de token con nombre de usuario incorrecto: Asegurar que el sistema rechace un token cuando el nombre de usuario no coincide.

```
@Test  new *
public void validarTokenConNombreDeUsuarioInvalido() {
    String token = jwtUtil.generateToken(username);
    boolean esValido = jwtUtil.validateToken(token,  username: "otrouser");
    assertFalse(esValido);}
```

Ilustración 55 Extracto de Código Validación de token con nombre de usuario incorrecto
Fuente: Elaboración propia

- Resultados esperados
  a) El token generado debe ser válido para el usuario proporcionado.
  b) Los tokens inválidos o manipulados deben ser rechazados por el sistema.
  c) La información extraída del token (nombre de usuario y fecha de expiración) debe ser correcta y coherente con los datos utilizados en su generación.
  d) Los tokens expirados deben ser reconocidos como inválidos.

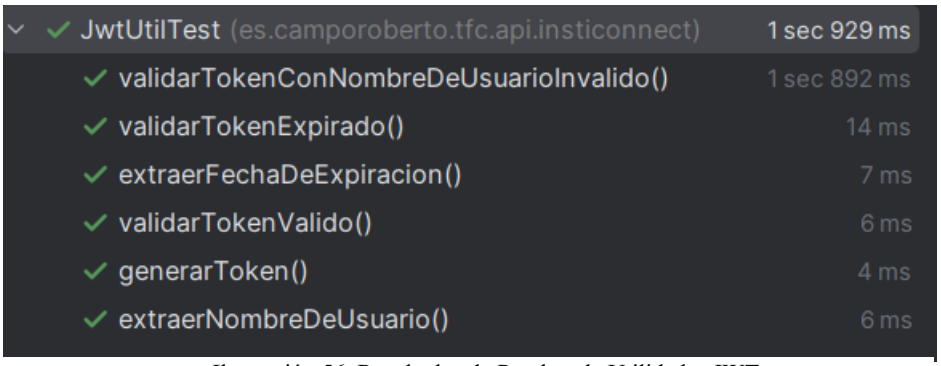

Ilustración 56 Resultados de Pruebas de Utilidades JWT
Fuente: Elaboración propia

Las pruebas realizadas en el sistema de autenticación con JWT garantizan que se cumplen los requisitos de seguridad, integridad y funcionalidad en la gestión de tokens. Esto asegura que solo los usuarios autenticados puedan acceder a los recursos protegidos del sistema.

9.1.2 Prueba de WebSocket

Esta prueba tiene como objetivo validar la funcionalidad de la comunicación en tiempo real utilizando WebSocket. Se evalúan diferentes aspectos del flujo de comunicación, como la

conexión, el manejo de mensajes, el cierre de conexiones y la notificación a los clientes conectados.

Casos de prueba implementados

1. Conexión establecida: Verificar que después de establecerse la conexión, las propiedades de la sesión, como el nombre de usuario, se gestionen correctamente.

```
@Test  new *
public void despuesDeConexionEstablecida() throws Exception {
    when(sesion.getAttributes()).thenReturn(Collections.singletonMap("username", "testuser"));
    handler.afterConnectionEstablished(sesion);
    verify(sesion, times( wantedNumberOfInvocations: 1)).getAttributes();
}
```

Ilustración 57 Extracto de Código Conexión establecida
Fuente: Elaboración propia

2. Manejo de mensajes: Validar que el sistema procese correctamente los mensajes de texto recibidos desde los clientes.

```
@Test  new *
public void manejarMensajeDeTexto() throws Exception {
    TextMessage mensaje = new TextMessage( payload: "Mensaje de prueba");
    handler.handleTextMessage(sesion, mensaje);
}
```

Ilustración 58 Extracto de Código Manejo de mensajes
Fuente: Elaboración propia

3. Cierre de conexión: Comprobar que las conexiones se cierren correctamente y que las sesiones se eliminen del sistema.

```
@Test  new *
public void despuesDeConexionCerrada() throws Exception {
    handler.afterConnectionEstablished(sesion);
    handler.afterConnectionClosed(sesion,  status: null);
    assertEquals( expected: 0, handler.sessions.size());
}
```

Ilustración 59 Extracto de Código Cierre de conexión
Fuente: Elaboración propia

4. Notificación a clientes: Asegurar que los clientes conectados reciban mensajes enviados desde el servidor.

```
@Test  new *
public void notificarClientes() throws Exception {
    handler.afterConnectionEstablished(sesion);
    when(sesion.isOpen()).thenReturn( t: true);
    handler.notifyClients("Hola clientes");
    verify(sesion, times( wantedNumberOfInvocations: 1)).sendMessage(new TextMessage( payload: "Hola clientes"));
}
```

Ilustración 60 Extracto de Código Notificación a clientes
Fuente: Elaboración propia

Las pruebas realizadas en el sistema de WebSocket validan que las interacciones en tiempo real entre el cliente y el servidor se gestionan de manera eficiente. Estas pruebas aseguran que el servidor responde correctamente a los eventos de conexión, manejo de mensajes y cierre de sesiones, proporcionando una experiencia estable para los usuarios.

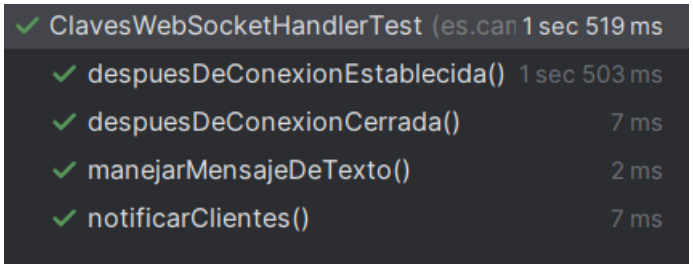

Ilustración 61 Pruebas de ClavesWebSocketHandle
Fuente: Elaboración propia

10. CONCLUSIONES

10.1 Conclusiones personales

Personalmente, creo que este proyecto me permitió consolidar y aplicar conocimientos técnicos y habilidades prácticas adquiridas durante el grado. A nivel técnico, fortalecí mi comprensión del acceso a datos con Spring Boot, logrando extraer información de la base de datos de manera eficiente y práctica. Adquirí experiencia en la implementación de medidas de seguridad como el uso de tokens, que previamente conocía de manera teórica pero que ahora comprendo su utilidad y aplicación.

Exploré tecnologías como WebSocket, que me ayudaron a implementar actualizaciones en tiempo real, lo cual fue un desafío que me permitió comprender mejor cómo funcionan las

aplicaciones web modernas. En el ámbito personal, aprendí a investigar de manera eficiente, a buscar soluciones en diferentes fuentes y a gestionar un proyecto completo que integra backend, frontend y base de datos.

Los principales retos enfrentados fueron la falta de experiencia previa en proyectos integrales y la organización del código. Antes, trabajaba por separado en frontend y backend, pero este proyecto me exigió combinarlos y darles funcionalidad. Para superar estos desafíos, investigué, consulté soluciones de otros desarrolladores, leí la documentación de las tecnologías utilizadas y conté con el apoyo de mi tutor. Además, aprendí a estructurar mejor mi código, separando funciones en diferentes ficheros para mejorar su legibilidad y facilitar su mantenimiento.

10.2 Posibles aplicaciones futuras

Una mejora clave sería ampliar la base de datos para incluir nombres reales de institutos y grados de todas las comunidades autónomas, convirtiéndolo en un recurso útil para más estudiantes.

Otra mejora sería desplegar la página web en un servidor, permitiendo que los estudiantes accedan desde cualquier lugar. Además, me gustaría explorar la posibilidad de desarrollar el frontend utilizando React para aprovechar su agilidad y modernidad en este tipo de proyectos.

10.3 Valoración general

En términos educativos, considero que este proyecto es una herramienta útil y funcional, ya que actúa como una red social enfocada en los grupos de estudiantes. Su diseño busca minimizar distracciones, centrándose en publicaciones relevantes como tareas, comunicados y horarios, lo que lo hace práctico y eficiente.

Profesionalmente, este proyecto refleja el uso de tecnologías actuales como HTML, CSS y JavaScript para el frontend, Spring Boot y Java para el backend, y MySQL para la base de datos. Estas herramientas son ampliamente utilizadas en el mercado laboral y han sido fundamentales para desarrollar una base sólida que me permitirá aprender otros frameworks y tecnologías más avanzadas en el futuro.

El mayor logro de este trabajo ha sido integrar conocimientos adquiridos de manera aislada y aplicarlos en un proyecto funcional. Logré comprender cómo se complementan distintas tecnologías para crear una aplicación web completa, lo que me ha dado una visión más amplia y práctica del desarrollo de software.

11. WEBGRAFÍA

- Auth0. (s. f.). Introducción a los tokens web JSON. https://jwt.io/introduction
- Fette, I., & Melnikov, A. (2011). El protocolo WebSocket (RFC 6455). Grupo de Trabajo de Ingeniería de Internet (IETF). https://tools.ietf.org/html/rfc6455
- MySQL. (s. f.). MySQL 8.0 Manual de Referencia. Oracle Corporation. https://dev.mysql.com/doc/
- OpenWebinars. (s. f.). Cómo implementar autenticación con JSON Web Tokens. https://openwebinars.net/
- Pivotal Software. (s. f.). Spring Boot Documentación de referencia. https://spring.io/projects/spring-boot
- W3C. (s. f.). HTML5 - Un vocabulario y asociados APIs para HTML y XHTML. https://html.spec.whatwg.org/
- W3C. (s. f.). CSS: Cascading Style Sheets. Recuperado de https://www.w3.org/Style/CSS/
- Mockito. (s. f.). Mockito User Guide. https://site.mockito.org/
- Hale, C. (s. f.). bcrypt. https://github.com/jeremyh/jBCrypt
- Stack Overflow. (2017). En un nivel bajo, ¿cómo detecta el protocolo WebSocket el estado de una conexión? https://stackoverflow.com/questions/45700602/at-a-low-level-how-does-the-websocket-protocol-detect-the-status-of-a-connectio
- Programando en JAVA. (2020). WEBSOCKETS en SPRING BOOT. [Video]. YouTube. https://www.youtube.com/watch?v=KAm6I_iLXOI
- Rojas Córsico, I. S. (2023*).* ¿Cómo crear el login? Spring Boot 3 + Spring Security 6 + JWT Authentication #backend*.* [Video]. YouTube. https://www.youtube.com/watch?v=nwqQYCM4YT8&t=7s
- Stack Overflow. (2023). CORS in Spring Security (Spring Boot 3). https://stackoverflow.com/questions/76987080/cors-in-spring-security-spring-boot-3
- Yo Androide. (2023). SPRING BOOT: Como solucionar y configurar problemas de cors*.* [Video]. YouTube. https://www.youtube.com/watch?v=jJTfuiyEqb8

12. ANEXOS

Enlace al repositorio: https://github.com/AlexSerjs/RobertoCampoInstiConnect.git