



UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro



PROYECTO DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA **GEARHUB**

CICLO FORMATIVO DE GRADO SUPERIOR
DESARROLLO DE APLICACIONES MULTIPLATAFORMA (IFCS02)

CURSO 2024-2025

Autores:

José María García Sánchez
Rodrigo Tapiador Cano

Tutor:

Federico Banda Sierra



DEPARTAMENTO DE INFORMÁTICA Y COMUNICACIONES
IES LUIS VIVES

Agradecimientos

En primer lugar, nuestro agradecimiento a nuestro tutor de TFG, Federico Banda Sierra, por su apoyo, dedicación y comprensión durante el desarrollo

Asimismo, agradezco al centro I.E.S Luis Vives y al Departamento de informática por haberme brindado los recursos académicos y el ambiente propicio para llevar a cabo este proyecto.

Nuestra gratitud se extiende también a José Manuel Pérez Lobato por su inestimable guía durante los primeros pasos en el mundo de la programación y los buenos momentos que nos han dado sus tutorías e icónicas frases.

Sin olvidar a Javier Palacios González por su dedicación, su empatía y su guía en temas profesionales y personales. Jamás olvidaremos tu amor por Docker y por tus alumnos. Lo que has marcado nuestro futuro no lo sabe nadie.

José María

A mi compañero Rodrigo, por aguantar que me pusiera pesado y pesimista en puntos críticos del proyecto. Sin su capacidad para resolver problemas y su optimismo nato no hubiera sido posible terminar el proyecto a tiempo.

Finalmente, y de manera muy especial, deseo agradecer a mi familia, en particular a mi madre María Jo por su amor incondicional, su comprensión y su constante motivación. A mi hermana Silvia que me ha escuchado rabiar del estrés, y sobre todo a mi pareja, Cris, que tantas noches me ha soportado hablando de Docker y redes en sueños. Su apoyo ha sido el pilar fundamental que me ha permitido alcanzar este logro académico y personal.

Rodrigo:

A mi compañero José María (Chema) por aguantar mi desorden durante la planificación del proyecto, sin su organización y sus habilidades de programación, además de su creatividad este proyecto no habría sido posible.

Especialmente me gustaría agradecer a mi familia, en particular a mi madre Eva por su amor y por aguantarme las noches en vela, a mi hermana por permitirme desentenderme de las tareas del hogar para poder dedicar el tiempo al proyecto, y a mi padre por ofrecerme montar el servidor en casa. También a todos mis amigos por permitirme rechazar los planes que me proponían, y esperar a que respondiera sus mensajes días tarde. De igual forma agradecerles que me sacaran de casa cuando el ordenador empezaba a quemarme los ojos. Sin el apoyo de mi círculo cercano todo esto no habría sido posible, no por falta de habilidad, sino por falta de motivación.

Finalmente, mi más sincero y especial agradecimiento a mis abuelos. Sin su sonrisa, cariño y amor incondicional, absolutamente nada de lo que soy o hago sería posible.

Resumen

Red social para usuarios y talleres donde los usuarios pueden registrar coches y gestionar su historial de mantenimiento en talleres. Incluye mensajería entre cuentas, un sistema de foros con hilos y respuestas moderados por roles, y gestión de órdenes de mantenimiento.

Abstract

Social network for users and workshops where users can register cars and manage their maintenance history in workshops. It includes messaging between accounts, a forum system with threads and replies moderated by roles, and maintenance order management.

Índice

1.	<u>INTRODUCCIÓN</u>	4
1.1.	OBJETIVO	5
1.2.	ALCANCE	6
1.2.1.	ALCANCE DEL PROYECTO	6
1.3.	JUSTIFICACIÓN	7
2.	<u>IMPLEMENTACIÓN</u>	8
2.1.	ANÁLISIS DE LA APLICACIÓN	8
2.1.1.	REQUISITOS FUNCIONALES DE LA APLICACIÓN	8
2.1.2.	ANÁLISIS Y SELECCIÓN DE LAS TECNOLOGÍAS	9
2.1.3.	PLANIFICACIÓN DE LA REALIZACIÓN DEL PROYECTO	10
2.2.	DISEÑO	11
2.2.1.	PROTOTIPADO	11
2.2.2.	DIAGRAMAS E/R Y DIAGRAMAS DE CLASE	13
2.2.3.	ARQUITECTURA	14
2.3.	IMPLEMENTACIÓN	17
2.3.1.	BACKEND – API AUTORIZACIÓN Y CORREO	17
2.3.2.	BACKEND – API DE COMUNIDADES	23
2.3.3.	BACKEND – API DE PERFILES	30
2.3.4.	BACKEND – API DE REVIEWS	40
2.3.5.	BACKEND – API DE TALLERES	47
2.3.6.	BACKEND – API DE MENSAJERÍA	59
2.3.7.	BACKEND – API GATEWAY	68
2.3.8.	FRONTEND WEB	70
2.3.9.	FRONTEND MÓVIL	76
2.4.	IMPLANTACIÓN	83
2.4.1.	PLATAFORMAS UTILIZADAS	83
2.4.2.	SERVICIOS DESPLEGADOS	83
2.4.3.	VARIABLES DE ENTORNO	84
2.4.4.	PERSISTENCIA DE DATOS	85
2.4.5.	REDES	85
2.4.6.	PROCEDIMIENTO DE DESPLIEGUE	86
	EJECUTAR EL DESPLIEGUE	86
2.5.	DOCUMENTACIÓN	87
3.	<u>CONCLUSIONES</u>	88
4.	<u>BIBLIOGRAFIA</u>	91
	<u>ANEXOS</u>	94

1. INTRODUCCIÓN

¿En qué consiste el proyecto?

Este proyecto consiste en el desarrollo de una **red social para usuarios y talleres**, en la que los usuarios pueden registrar y gestionar sus coches, llevar un historial de mantenimiento realizado en talleres y comunicarse con otros usuarios y talleres mediante mensajería y foros. La plataforma facilita la búsqueda de talleres según distintos criterios, como ubicación y especialización, y ofrece recomendaciones mediante técnicas de análisis de datos avanzadas.

Desarrollo

El proyecto se ha desarrollado como parte de la formación del **Ciclo Formativo de Grado Superior (CFGs)**, donde se han aplicado conocimientos en **arquitecturas de software, gestión de bases de datos, seguridad informática y metodologías ágiles** para garantizar un desarrollo estructurado y eficiente de la plataforma.

Módulos implicados

- **Gestión del Backend:** Este módulo se encarga de gestionar los datos de los usuarios, coches, talleres y las interacciones entre ellos. Además, maneja la autenticación de los usuarios y asegura el acceso a la plataforma de manera segura.
- **Aplicación para Usuarios:** Este módulo permite a los usuarios gestionar sus vehículos, consultar talleres, llevar un historial de mantenimiento y participar en la comunidad de la plataforma.
- **Aplicación para Talleres:** Este módulo está dirigido a los propietarios de talleres, quienes podrán gestionar sus servicios.
- **Sistema de Foros:** Permite a los usuarios crear foros, iniciar hilos de discusión y responder a otros usuarios.
- **Mensajería entre Cuentas:** Facilita la comunicación privada entre usuarios y talleres, permitiendo que puedan intercambiar mensajes sobre servicios, precios y otros temas relevantes.
- **Recomendaciones Inteligentes:** Mediante el análisis de datos históricos y el comportamiento de los usuarios, este módulo proporciona recomendaciones personalizadas de vehículos y piezas, mejorando la experiencia de la plataforma.

1.1. OBJETIVO

El objetivo principal de este proyecto es desarrollar una plataforma integral que facilite la interacción entre usuarios y talleres, permitiendo a los usuarios gestionar sus vehículos, acceder a un historial de mantenimiento detallado y realizar consultas o recibir recomendaciones personalizadas basadas en sus necesidades específicas. A través de esta plataforma, también se busca optimizar la gestión de talleres, permitiendo una conexión eficiente con los usuarios y mejorando la visibilidad de sus servicios.

A lo largo de este proyecto, se busca **profundizar en el uso de tecnologías y herramientas clave** que son fundamentales en el desarrollo de aplicaciones de gran escala, tales como **bases de datos relacionales**, que permiten una gestión eficiente de grandes volúmenes de datos, y **arquitecturas de software escalables**, que aseguran que el sistema pueda crecer y adaptarse a medida que aumentan los usuarios y las operaciones.

El proyecto permitirá adquirir experiencia práctica en la creación de **APIs RESTful**, mediante las cuales los diferentes módulos de la aplicación puedan interactuar de forma eficiente y segura. Se trabajará también en la implementación de **sistemas de autenticación y autorización**, lo que permitirá a los usuarios acceder a la plataforma de manera segura y personalizada, protegiendo su información y facilitando una experiencia de usuario optimizada.

El desarrollo de **sistemas inteligentes de recomendación**, basados en Big Data y técnicas de análisis predictivo, será otro de los puntos clave de aprendizaje. Esto permitirá no solo mejorar la experiencia del usuario final, sino también integrar capacidades de análisis que optimicen las decisiones tanto para los usuarios como para los talleres.

En resumen, este proyecto no solo tiene como objetivo la creación de una plataforma funcional, sino que también proporciona una valiosa oportunidad para adquirir una comprensión más profunda de áreas críticas del desarrollo de software, como la **gestión de datos, la seguridad, la integración de sistemas inteligentes y el desarrollo de aplicaciones escalables**, los cuales son esenciales para proyectos tecnológicos de mayor envergadura en el futuro.

1.2.ALCANCE

1.2.1. Alcance del Proyecto

El proyecto tiene como objetivo desarrollar una plataforma integral que permita la interacción entre usuarios y talleres, facilitando la gestión de vehículos y el acceso a servicios de mantenimiento. La plataforma incluirá varias funcionalidades esenciales para la comunicación, gestión de la información y la interacción entre los usuarios.

Aplicación Cliente

- **Plataforma Móvil:** Permitirá a los usuarios gestionar sus vehículos, consultar talleres cercanos y acceder al historial de mantenimiento. Además, podrán participar en foros y enviar mensajes a otros usuarios y talleres.
- **Plataforma web:** Estará dirigida a los talleres para gestionar sus servicios, consultas de mantenimiento y comunicaciones con los usuarios.

Aplicación Servidor

- **API RESTful:** Gestionará la comunicación entre las aplicaciones cliente y servidor, permitiendo la interacción fluida de datos.
- **Base de Datos:** Almacenará la información de usuarios, vehículos, talleres, mantenimiento, mensajes y foros.
- **Autenticación y Autorización de Usuarios:** Implementará un sistema seguro para la gestión de las credenciales de los usuarios.

Puntos Opcionales

- **Sistema de Notificaciones:** Alertas en tiempo real para mantener informados a los usuarios sobre actividades relevantes
- **Sistema de Recomendación:** Sugerencias personalizadas basadas en el comportamiento del usuario y el historial de mantenimiento de sus vehículos.
- **Geolocalización:** Permite a los usuarios encontrar talleres cercanos a su ubicación.
- **Servicios con Contenedores:** Mejorará la comunicación entre los diferentes módulos de la aplicación a través de una arquitectura modular.

1.3.JUSTIFICACIÓN

Hoy en día, las personas que necesitan gestionar el mantenimiento de sus coches suelen recurrir a varias aplicaciones para controlar el historial de reparaciones, localizar talleres o incluso pedir recomendaciones. Sin embargo, ninguna de estas herramientas combina todas esas funcionalidades de manera fácil y centralizada. El proceso de mantener un vehículo en buen estado se vuelve mucho más complicado cuando tienes que utilizar diferentes aplicaciones y servicios para cada tarea, sin una forma sencilla de conectar todos esos aspectos.

Aunque existen aplicaciones que ayudan a localizar talleres o llevar un registro de los servicios que se han hecho al coche, la mayoría no permiten una interacción fluida entre usuarios y talleres. Además, la mayoría no ofrecen recomendaciones personalizadas o inteligentes basadas en el comportamiento del usuario o el historial de mantenimiento del vehículo. Esto significa que los usuarios a menudo toman decisiones sin contar con información suficiente o recomendaciones que les puedan facilitar la vida.

Este proyecto nace con la idea de ofrecer una solución completa y sencilla para estos problemas. Queremos crear una plataforma donde los usuarios puedan gestionar el mantenimiento de sus coches, comunicarse directamente con talleres y recibir recomendaciones personalizadas basadas en el uso de su vehículo. Todo eso desde un solo lugar, sin tener que saltar de una aplicación a otra. Además, a través de herramientas como Big Data e Inteligencia Artificial, buscamos hacer que la experiencia sea aún más útil, al ofrecer sugerencias inteligentes sobre talleres y piezas de repuesto.

De esta forma, la plataforma no solo ayudará a los usuarios a llevar un mejor control de sus vehículos, sino que también les facilitará la tarea de encontrar talleres y servicios adecuados para sus necesidades, mejorando la forma en la que interactúan con los profesionales del sector y creando una experiencia más personalizada y eficiente.

2. IMPLEMENTACIÓN

2.1. ANÁLISIS DE LA APLICACIÓN

La aplicación tiene como objetivo principal ofrecer una plataforma integral para la gestión de vehículos y la interacción entre usuarios y talleres. Permitirá la gestión del historial de mantenimiento de coches, la búsqueda de talleres según varios criterios y la comunicación directa entre usuarios y talleres. Además, contará con un sistema de recomendaciones basadas en Big Data e Inteligencia Artificial.

2.1.1. Requisitos funcionales de la aplicación

Los requisitos funcionales definen las características y funcionalidades que debe cumplir la aplicación para satisfacer las necesidades de los usuarios. Estos incluyen:

- **Gestión de vehículos:** Los usuarios podrán registrar sus vehículos, incluir información relevante como marca, modelo, año, y mantener un historial detallado de mantenimiento. Además, podrán consultar el historial de mantenimiento de sus vehículos, incluyendo reparaciones previas, piezas cambiadas, etc.
- **Búsqueda de talleres:** La plataforma permitirá a los usuarios buscar talleres de reparación y mantenimiento en función de diversos criterios, como la ubicación, especialización, valoraciones de otros usuarios y disponibilidad. Los usuarios podrán ver los detalles de cada taller y realizar citas para reparaciones.
- **Comunicación entre usuarios y talleres:** La aplicación incluirá un sistema de mensajería para que los usuarios puedan comunicarse con los talleres, facilitando la resolución de dudas o consultas sobre los servicios de mantenimiento. Los talleres podrán responder a los usuarios de manera directa.
- **Interacción entre usuarios:** Se habilitará un sistema de foros donde los usuarios podrán crear hilos, responder preguntas y participar en discusiones sobre temas relacionados con los vehículos y los talleres. Las respuestas en los foros podrán generar sobreuestas, creando una estructura de conversación en forma de árbol.
- **Recomendación personalizada(opcional):** La plataforma incluirá un sistema de recomendación que utilizará datos históricos y algoritmos inteligentes para sugerir talleres, piezas de repuesto o servicios basados en el comportamiento y las preferencias de los usuarios, así como en las necesidades de sus vehículos.
- **Autenticación y seguridad:** Los usuarios deberán poder crear una cuenta personal y acceder de forma segura. La aplicación implementará medidas de seguridad para proteger la privacidad de los datos y asegurar que solo los usuarios autenticados tengan acceso a sus vehículos y datos personales.
- **Interfaz amigable y accesible:** La aplicación debe ser fácil de usar y accesible tanto en dispositivos móviles como de escritorio.

2.1.2. Análisis y selección de las tecnologías

La elección de las tecnologías a utilizar para implementar la aplicación se basa en su capacidad para cubrir los requisitos funcionales y no funcionales, así como en su facilidad de uso, soporte y escalabilidad.

Análisis de tecnologías:

- **Backend:** Se evaluaron varias opciones para la implementación del servidor de la aplicación. Tecnologías como Node.js, Django y FastAPI fueron consideradas. Finalmente, se eligió **ASP .Net** por su rendimiento, facilidad de uso y excelente integración con bases de datos relacionales.

Para mantener el entorno .Net en el back se decidió usar Ocelot para la orquestación del API Gateway y SignalR para la comunicación en tiempo real.

- **Frontend (Aplicación móvil):** Para el desarrollo móvil, se consideraron opciones como Flutter, React Native y MAUI. Se optó por **Jetpack Compose** por implementar una tecnología que no habíamos usado, pero con un lenguaje que sí conocíamos, además de por ser la nueva tendencia en las empresas de desarrollo móvil.
- **Frontend (Aplicación web):** Para el desarrollo web, se consideraron opciones como React o angular. Se optó por **React** y tailwind por la reusabilidad de componentes y las etiquetas de componentes.
- **Base de Datos:** Se eligió **PostgreSQL** debido a su robustez, flexibilidad y capacidad para manejar grandes volúmenes de datos, lo que resulta ideal para gestionar la información de vehículos, usuarios y talleres.
- **Recomendación y Big Data:** Se planea implementar algoritmos de recomendación utilizando técnicas de Big Data e Inteligencia Artificial para ofrecer recomendaciones personalizadas sobre talleres y piezas de repuesto. Se evaluarán tecnologías como Python con librerías de Machine Learning.
- **Seguridad:** Para la gestión de usuarios, se empleará un sistema de autenticación JWT (JSON Web Token) que permitirá una autenticación segura y escalable.

2.1.3. Planificación de la realización del proyecto

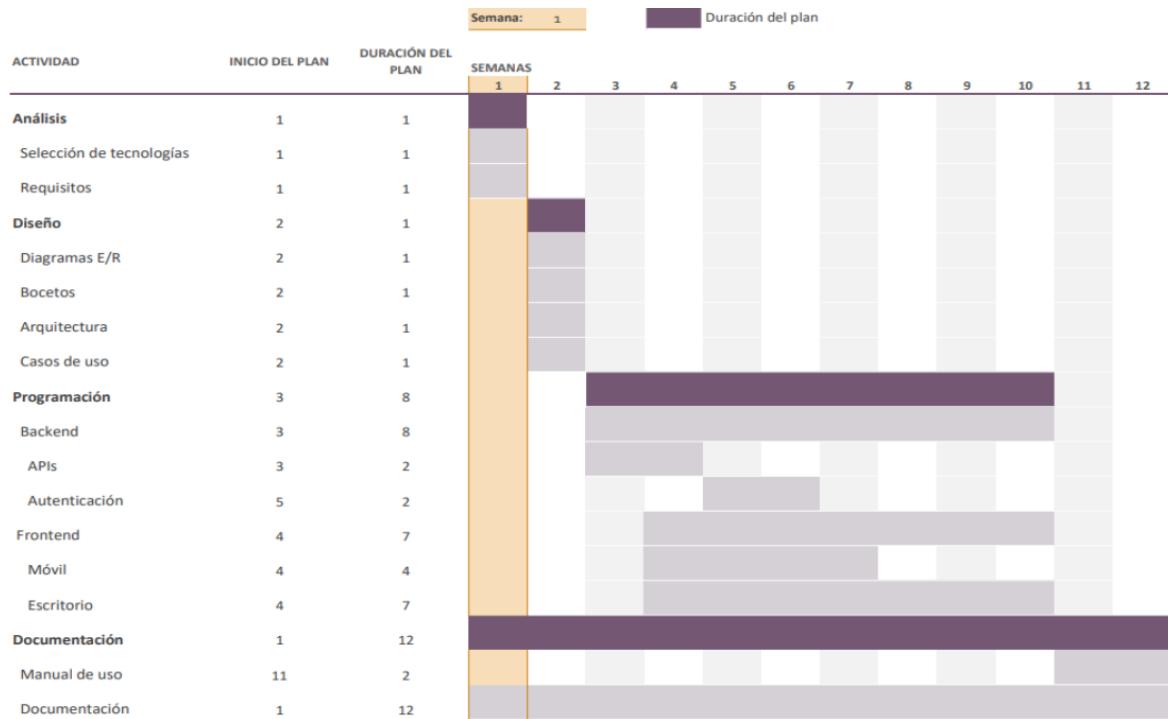


Ilustración 2-1: Diagrama de Gantt

2.2. DISEÑO

El diseño de esta aplicación ha pasado por muchas fases, sufriendo cambios continuos y adaptaciones a ideas más portentosas. El diseño inicial fue realizado en Figma.

2.2.1. Prototipado

- Diseño inicial de aplicación web.

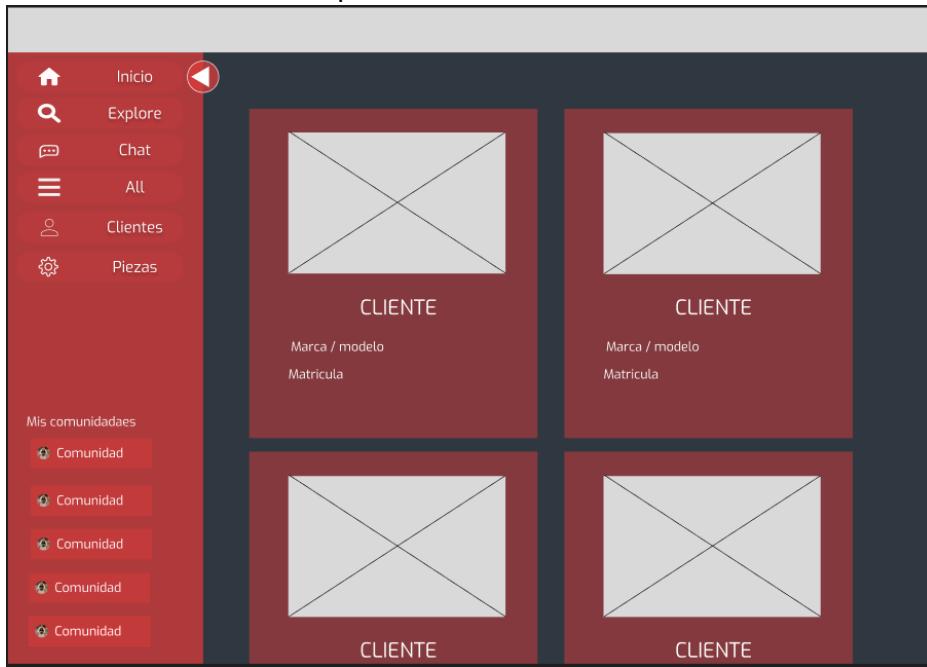


Imagen 2-2: Vista 1 de aplicación web.

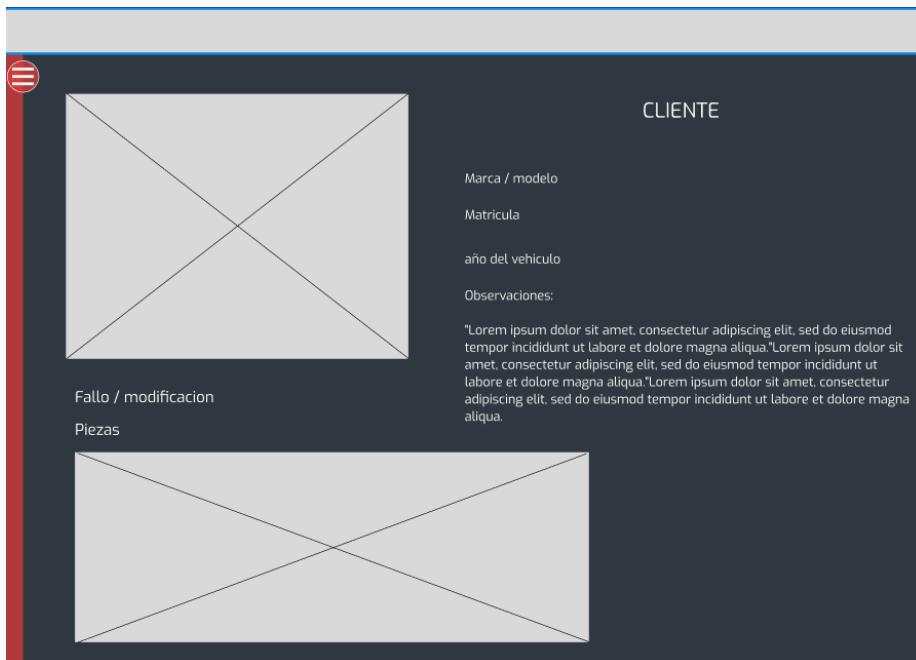


Imagen 2-3: Vista 1 de aplicación web.

- Diseño inicial de aplicación móvil.

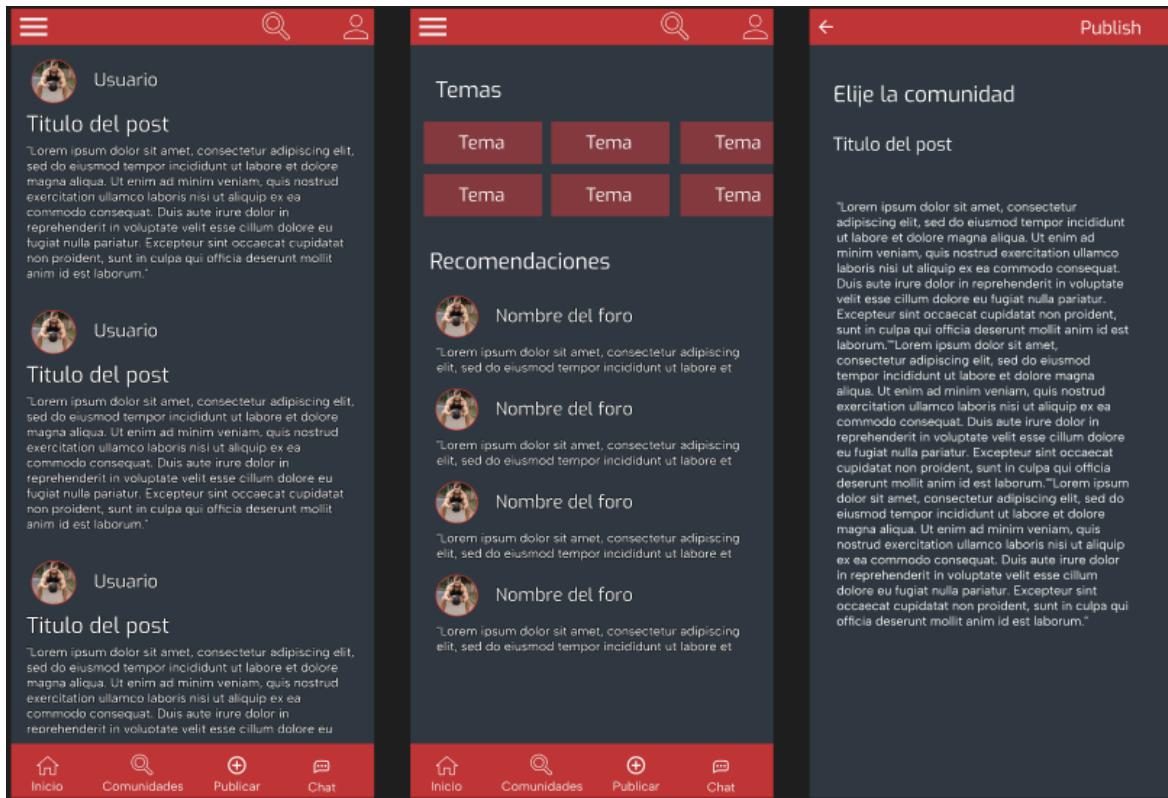


Imagen 2-4: Vistas iniciales de aplicación móvil

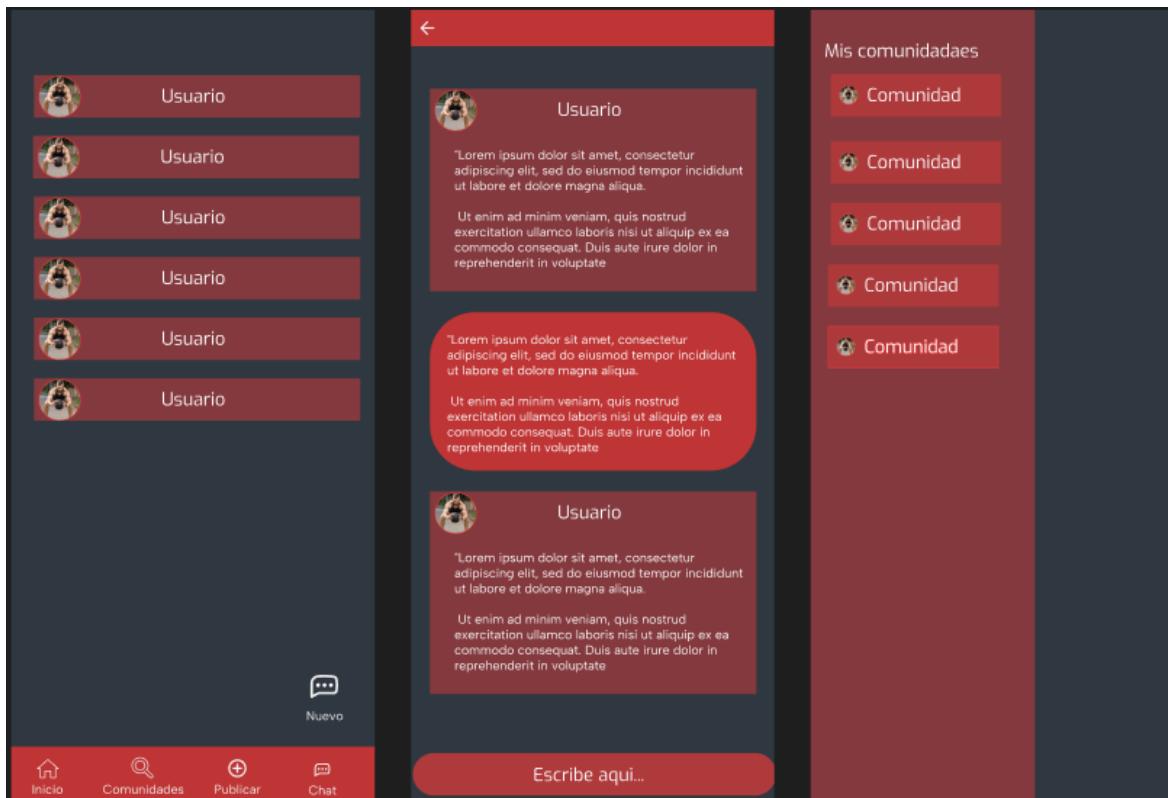


Imagen 2-5: Vistas iniciales de aplicación móvil

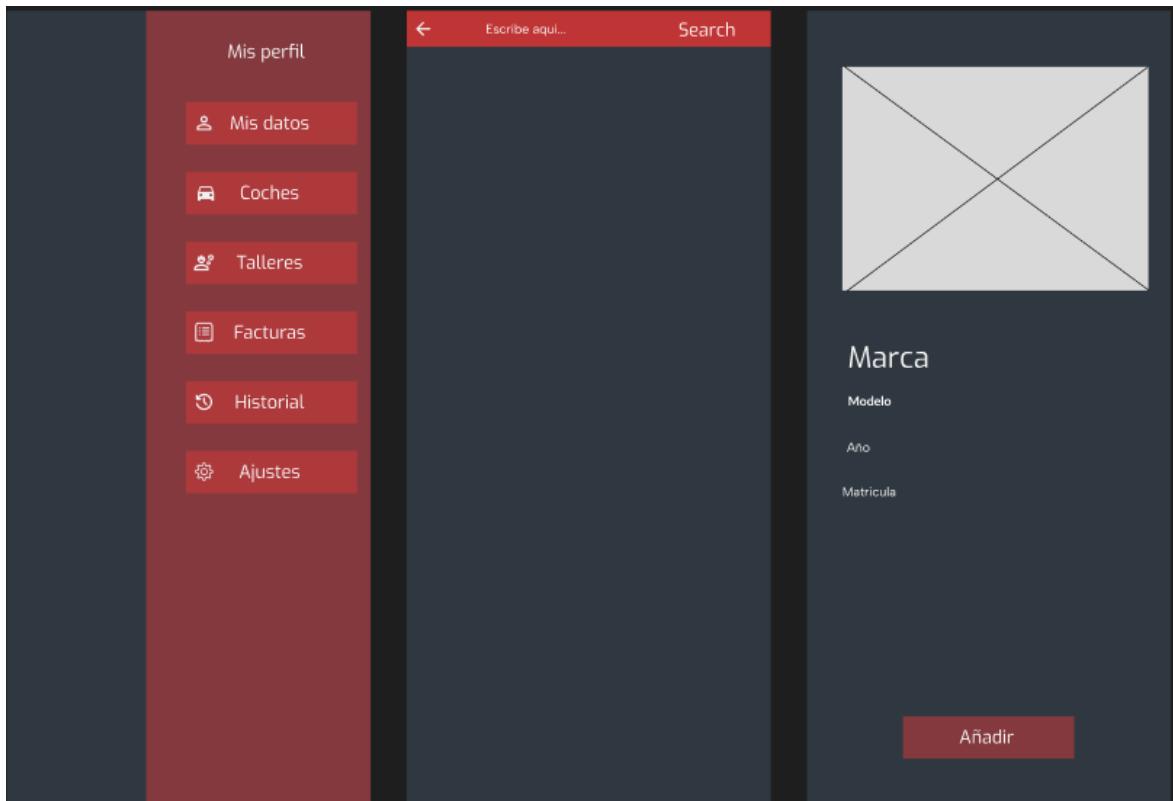


Imagen 2-6: Vistas iniciales de aplicación móvil

2.2.2. Diagramas E/R y Diagramas de clase

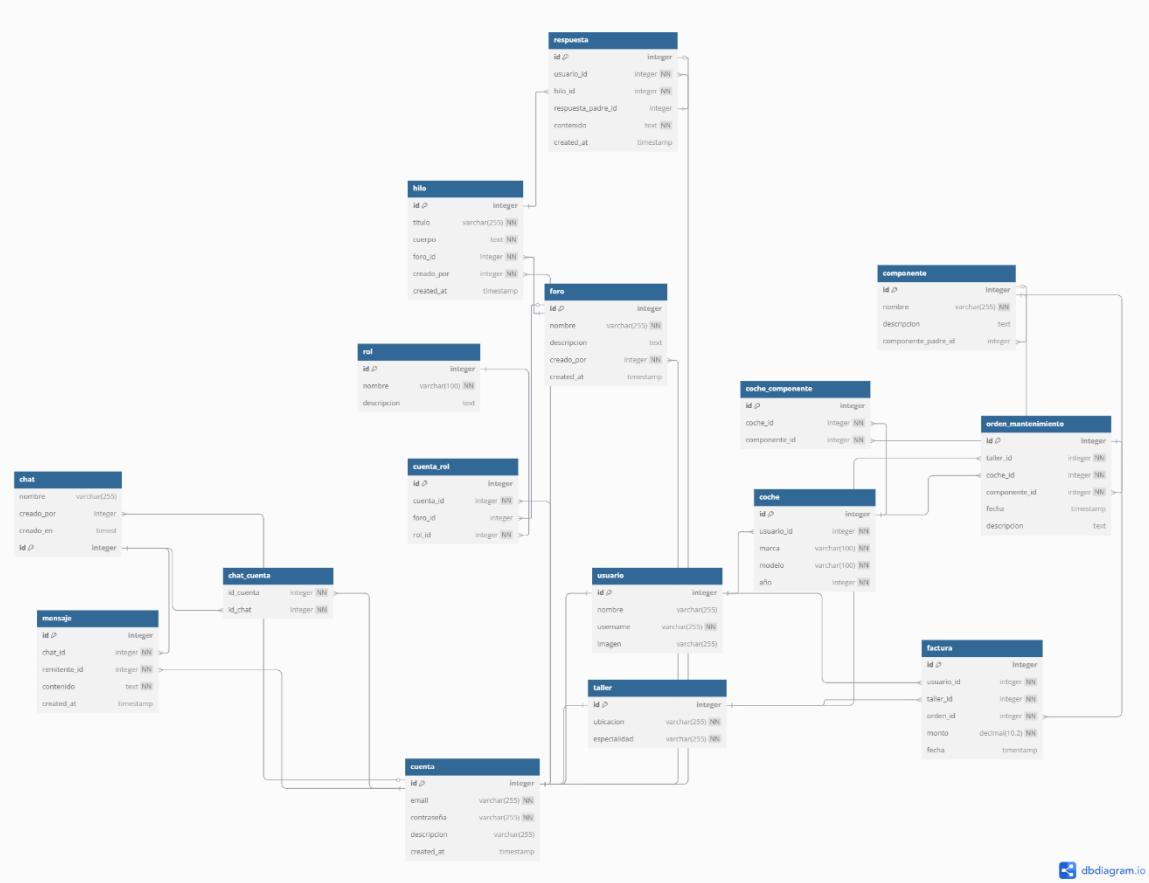
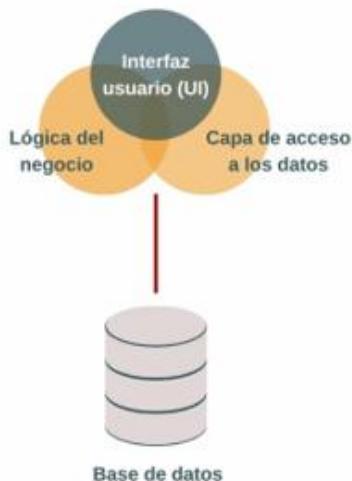


Imagen 2-7: Diagrama E/R

2.2.3. Arquitectura

Al diseñar la estructura quisimos que esta tuviera dos características principalmente: Modularidad y escalabilidad. Es por ello por lo que se decidió renunciar a la idea de una arquitectura monolítica a favor de una estructura de microservicios.

Arquitectura monolítica



Arquitectura microservicios

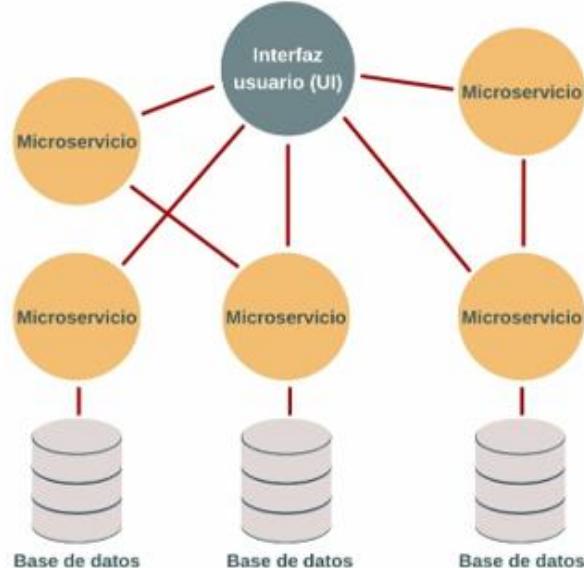


Imagen 2-8: Tipos de estructura de aplicaciones

Una vez se estableció el tipo de arquitectura que tuviera la aplicación se decidió establecer las tecnologías a usar tecnologías.

Se decidió que el backend estaría conformado por **6 microservicios**, cada uno implementado en **ASP.NET Core** para mantener la consistencia de tecnologías, facilitar el mantenimiento y la integración sencilla con Docker.

Microservicios

- **5 microservicios** estarán respaldados por **bases de datos relacionales** utilizando **PostgreSQL**.
- **1 microservicio** utilizará **MongoDB** como base de datos **no relacional**.

Cada microservicio es **autónomo** y sigue el principio de *Database per Service*. Esto garantiza un alto grado de encapsulamiento.

Para facilitar el despliegue, aislamiento y escalabilidad de cada componente, se tomó la decisión de utilizar **Docker** para hacer contenedores tanto con los microservicios como con sus respectivas bases de datos.

Cada microservicio tendrá su propio Dockerfile, y todo el entorno será coordinado con un archivo docker-compose.yml que:

- Levanta simultáneamente:
 - Los 6 microservicios (ASP.NET Core).
 - 5 instancias de PostgreSQL (una por microservicio que lo requiere).
 - 1 instancia de MongoDB.
 - El **API Gateway**.
 - El Front Web.
- Define las redes internas para que los servicios puedan comunicarse sin exponer innecesariamente sus puertos.
- Incluye configuraciones para volúmenes persistentes y variables de entorno.

API Gateway

El **API Gateway** de Ocelot también se ejecuta dentro de Docker, y centraliza el acceso externo a los microservicios. Este componente:

- Expone una única entrada pública para las aplicaciones web y móviles.
- Realiza enrutamiento a los microservicios según los endpoints.
- Aplica políticas de autenticación (por ejemplo, verificación de JWT).

SignalR

Para la funcionalidad de **mensajería en tiempo real**, el backend tiene implementada la notificación de websockets mediante **SignalR**, la biblioteca de ASP.NET Core que facilita la comunicación bidireccional en tiempo real entre el cliente y el servidor.

Frontend Web

La aplicación web fue desarrollada utilizando el stack **React + JavaScript**, estilizada con **Tailwind** para crear una interfaz moderna, adaptable y fácilmente mantenible. Para levantarla se usó NGinx

Características principales

- **Framework:** Se usó **React** como librería base por su flexibilidad.
- **Lenguaje:** **JavaScript**.
- **Estilos:** **TailwindCSS** permitió implementar estilos que favorecen un desarrollo visual rápido, coherente y responsive.
- **Routing:** Navegación basada en react-router-dom.
- **Estado global:** Se manejan contextos de uso.

Frontend Móvil – Jetpack Compose (Kotlin)

La aplicación móvil fue desarrollada utilizando Jetpack compose, el framework más moderno para el desarrollo de interfaces nativas en Android. En cuanto al patrón de diseño de la aplicación optamos por el Modelo-Vista-Vistamodelo (MVVM) por su versatilidad y por ser el estándar en Android, para la correcta implementación de este patrón se utilizó el contenedor de dependencias Hilt, eliminando los artefactos factory para la creación de viewmodels y permitiendo la inyección de dependencias.

Características principales

- **Framework:** Se usó **Jetpack Compose** como librería base por su flexibilidad.
- **Lenguaje:** **Kotlin**.
- **Routing:** Se utilizó la navegación de androidx adaptada a Jetpack Compose.
- **Seguridad:** Al iniciar sesión en la aplicación se almacena el Token en el dispositivo haciendo uso de datastore-preferences y permitiendo que se mantenga abierta la sesión.

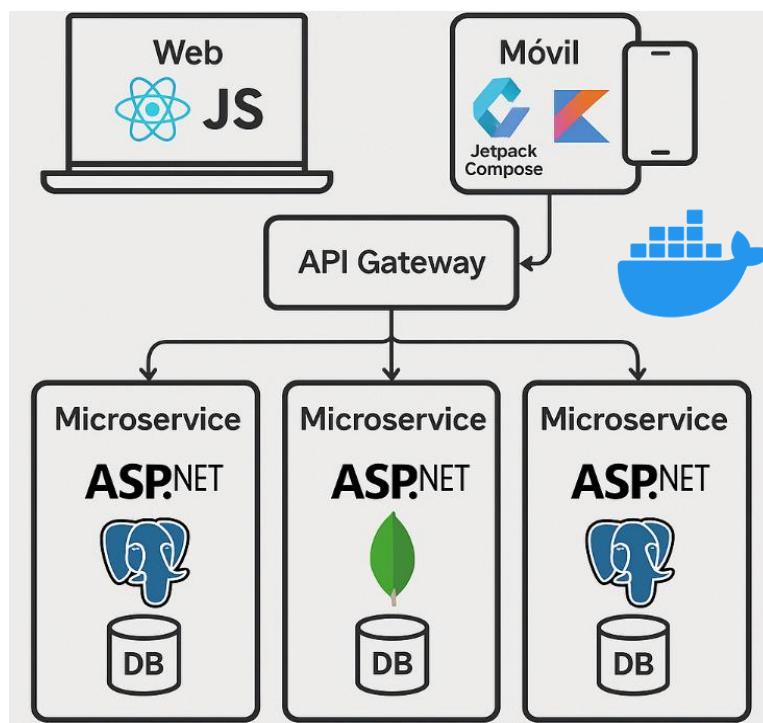


Imagen 2-4: Esquema básico de la aplicación.

2.3. IMPLEMENTACIÓN

En esta sección se detalla la implementación técnica del sistema, abarcando tanto el desarrollo del backend —que incluye control de acceso, gestión de comunidades, perfiles, reseñas, chats y mantenimientos— como el Frontend web y móvil, encargado de la interacción con el usuario.

2.3.1. Backend – API Autorización y correo.

Esta API permite el registro, inicio de sesión, confirmación de correo y gestión de contraseñas para usuarios.

Endpoints destacables

1. Registro de Usuario

POST /User/RegisterUsers

Registra un nuevo usuario y envía un correo de confirmación.

```
[HttpPost("RegisterUsers", Name = "RegUser")]
public async Task<IActionResult> Postuser([FromBody] RegisterDTO credentials)
{
    if (credentials.Password != credentials.PasswordRepeat)
    {
        return BadRequest(new { message = "Las contraseñas no coinciden." });
    }

    var usuarioExistente = await _context.Users.FirstOrDefaultAsync(u => u.Email == credentials.Email);

    if (usuarioExistente != null)
    {
        if (usuarioExistente.Confirmed)
        {
            return BadRequest(new { message = "Este correo ya está en uso." });
        }
        else
        {
            string newToken = _jwtService.GenerateJwtToken(usuarioExistente, "RegisteredUser", TimeSpan.FromHours(1));
            bool reenviado = _emailService.CreateAndSendConfirmacionEmail(usuarioExistente.Email, newToken);

            if (reenviado)
            {
                return Ok(new { message = "Usuario creado. Por favor, confirme su email." });
            }
            else
            {
                return BadRequest(new { message = "Error al reenviar el correo de confirmación." });
            }
        }
    }

    User nuevo = new();
    nuevo.Name = credentials.Name;
    nuevo.Email = credentials.Email;
    nuevo.PasswordHash = _encryptService.HashPassword(credentials.Password);
    nuevo.UserType = credentials.UserType;

    string token = _jwtService.GenerateJwtToken(nuevo, "RegisteredUser", TimeSpan.FromHours(1));
    bool enviado = _emailService.CreateAndSendConfirmacionEmail(nuevo.Email, token);

    if (!enviado)
    {
        return BadRequest(new { message = "Hubo un problema al enviar el correo de confirmación." });
    }

    _context.Users.Add(nuevo);
    await _context.SaveChangesAsync();

    return Ok(new { message = "Usuario creado. Por favor, confirme su email." });
}
```

2. Inicio de Sesión

POST /User/LogUser

Autentica a un usuario y devuelve un JWT.

```
[HttpPost("LogUser", Name = "LogUser")]
public async Task<IActionResult> Getuser([FromBody] LoginDTO credentials)
{
    var confirmacion = await _context.Users.FirstOrDefaultAsync(u => u.Email == credentials.Email);
    if (confirmacion != null)
    {
        if (confirmacion.Confirmed == true)
        {
            if (_encryptService.VerifyPassword(credentials.Password, confirmacion.PasswordHash))
            {
                string loginToken = _jwtService.GenerateJwtToken(confirmacion, "LoginUser", TimeSpan.FromDays(30));
                return Ok(new { token = loginToken });
            }
            else return Unauthorized(new { message = "Email o contraseña no válidos" });
        }
        else return BadRequest(new { message = "Por favor, confirma tu email" });
    }
    else return NotFound(new { message = "Usuario no encontrado" });
}
```

3. Confirmar Email

GET /User/ConfirmEmail/{token}

Valida el email del usuario usando un token JWT.

```
[HttpGet("ConfirmEmail/{token}", Name = "ConfirmUser")]
public async Task<IActionResult> PostEmail(string token)
{
    try
    {
        ClaimsPrincipal? confirm = _jwtService.ValidateToken(token, "RegisteredUser");
        if (confirm != null)
        {
            string? email = confirm.FindFirst(ClaimTypes.Email)?.Value;
            if (email != null)
            {
                User? confirmado = await _context.Users.FirstOrDefaultAsync(u => u.Email == email);
                if (confirmado != null && confirmado.Confirmed == false)
                {
                    confirmado.Confirmed = true;
                    _context.Users.Update(confirmado);
                    await _context.SaveChangesAsync();
                    return Ok(new { message = "Email confirmado" });
                }
                else return StatusCode(403, new { message = "Ya validado" });
            }
            else return BadRequest(new { message = "Token inválido: no contiene un email" });
        }
        else return Unauthorized(new { message = "Validación incorrecta" });
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

4. Solicitar Cambio de Contraseña

GET /User/RequestChangePassword/{email}

Envía un correo con token para cambiar la contraseña.

```
[HttpGet("RequestChangePassword/{email}", Name = "Request")]
public async Task<IActionResult> PostPassword(string email)
{
    try
    {
        if (email != null)
        {
            User? confirmado = await _context.Users.FirstOrDefaultAsync(u => u.Email == email);
            if (confirmado != null)
            {
                string tokenReset = _jwtService.GenerateJwtToken(confirmado, "UserChangePassword", TimeSpan.FromMinutes(30));
                bool enviado = _emailService.CreateAndSendPasswordResetEmail(email, tokenReset);
                if (enviado)
                {
                    return Ok(new { message = "Email enviado" });
                }
                else return BadRequest(new { message = "No ha sido posible enviar el email" });
            }
            else return NotFound(new { message = "Usuario no encontrado" });
        }
        else return BadRequest(new { message = "Email no válido" });
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

5. Cambiar Contraseña

PATCH /User/ChangePassword

Cambia la contraseña del usuario autenticado con token.

```
[HttpPatch("ChangePassword", Name = "ChangePassword")]
public async Task<IActionResult> PatchNew([FromBody] ChangePasswordDTO change)
{
    try
    {
        if (change.Password == change.PasswordRepeat)
        {
            ClaimsPrincipal? confirm = _jwtService.ValidateToken(change.Token, "UserChangePassword");
            if (confirm != null)
            {
                string? email = confirm.FindFirst(ClaimTypes.Email)?.Value;
                if (email != null)
                {
                    User? confirmado = await _context.Users.FirstOrDefaultAsync(u => u.Email == email);
                    if (confirmado != null)
                    {
                        confirmado.PasswordHash = _encryptService.HashPassword(change.Password);
                        _context.Users.Update(confirmado);
                        await _context.SaveChangesAsync();
                        return Ok(new { message = "Contraseña cambiada" });
                    }
                    else return NotFound(new { message = "Usuario no encontrado" });
                }
                else return BadRequest(new { message = "Token inválido: no contiene un email" });
            }
            else return Unauthorized(new { message = "Validación incorrecta" });
        }
        else return BadRequest(new { message = "Contraseñas no coinciden" });
    }
    catch (Exception e)
    {
        return BadRequest(new { message = e.Message });
    }
}
```

Servicios Internos

1. EmailService

Usa SendGrid para enviar correos. Se configura mediante variables de entorno:

```
public EmailService(IConfiguration configuration)
{
    var emailSettings = configuration.GetSection("EmailSettings");
    Sender = emailSettings.GetValue<string>("UserEmail");
    UserName = emailSettings.GetValue<string>("UserName");
    Password = emailSettings.GetValue<string>("UserApiKey");
    Host = emailSettings.GetValue<string>("Host");
}
```

En esta clase tenemos 2 métodos principales:

- Confirmación de registro

```
public async Task<bool> SendConfirmEmail(string toEmail, string token)
{
    string confirmationLink = $"http://[{Host}]/confirm-email/{token}";
    var client = new SendGridClient(Password);
    var from = new EmailAddress(Sender, UserName);
    var to = new EmailAddress(toEmail);
    var msg = MailHelper.CreateSingleEmail(from, to, "Confirmation email", confirmationLink, htmlContent: null);
    var response = await client.SendEmailAsync(msg);
    return response.IsSuccessStatusCode;
}
```

- Cambio de contraseña

```
public async Task<bool> SendResetEmail(string toEmail, string token)
{
    string confirmationLink = $"http://[{Host}]/pass-change/{token}";
    var client = new SendGridClient(Password);
    var from = new EmailAddress(Sender, UserName);
    var to = new EmailAddress(toEmail);
    var msg = MailHelper.CreateSingleEmail(from, to, "Change password", confirmationLink, htmlContent: null);
    var response = await client.SendEmailAsync(msg);
    return response.IsSuccessStatusCode;
}
```

2. EncryptService

Usa BCrypt para hashes y verificación de contraseñas

```
public class EncryptService
{
    public string HashPassword(string password)
    {
        return BCrypt.Net.BCrypt.HashPassword(password);
    }

    public bool VerifyPassword(string password, string hashedPassword)
    {
        return BCrypt.Net.BCrypt.Verify(password, hashedPassword);
    }
}
```

3. JWTService

Se encarga de generar y validar tokens. Se configura mediante variables de entorno

```
public JWTService(IConfiguration configuration)
{
    var jwtSettings = configuration.GetSection("JwtSettings");
    _jwtSecret = jwtSettings.GetValue<string>("SecretKey");
    _key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtSecret));
    _creds = new SigningCredentials(_key, SecurityAlgorithms.HmacSha256);
    _tokenHandler = new JwtSecurityTokenHandler();
}
```

Los métodos principales son:

- Genera tokens JWT con claim de Email, ID y Tipo de Usuario.

```
public string GenerateJwtToken(User user, string audience, TimeSpan expiration)
{
    var claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
        new Claim(ClaimTypes.Email, user.Email),
        new Claim("UserType", ((int)user.UserType).ToString())
    };

    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(claims),
        Issuer = "RegisterSystem",
        Audience = audience,
        Expires = DateTime.UtcNow.Add(expiration),
        SigningCredentials = _creds
    };

    SecurityToken token = _tokenHandler.CreateToken(tokenDescriptor);
    return _tokenHandler.WriteToken(token);
}
```

- Genera parámetros de validación con audiencias diferentes para registro, login y cambio de contraseña.

```
private TokenValidationParameters CreatevalidationParameters(string audience)
{
    return new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidIssuer = "RegisterSystem",
        ValidateAudience = true,
        ValidAudience = audience,
        ValidateLifetime = true,
        ClockSkew = TimeSpan.Zero,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = _creds.Key
    };
}
```

- Validación de tokens

```
public ClaimsPrincipal ValidateToken(string token, string audience)
{
    try
    {
        TokenValidationParameters validationParameters = CreatevalidationParameters(audience);
        return _tokenHandler.ValidateToken(token, validationParameters, out var validatedToken);
    }
    catch (SecurityTokenExpiredException)
    {
        throw new UnauthorizedAccessException("El token ha expirado.");
    }
    catch (Exception ex)
    {
        throw new UnauthorizedAccessException("El token no es válido." + " " + ex.Message);
    }
}
```

2.3.2. Backend – API de comunidades.

La API de comunidades está compuesta por aproximadamente 40 endpoints, organizados en torno a entidades como comunidades, threads, respuestas y roles. La mayoría de estos endpoints corresponden a operaciones de creación, algunas de ellas con manejo de archivos adjuntos, y de lectura, filtradas por usuario, “likes” o por comunidad/hilo. Con el objetivo de evitar redundancias, en esta sección se describen únicamente los casos más representativos.

1. Update thread

El método updateThread requiere de revisiones y borrado múltiple de imágenes. Primero hace comprobaciones básicas sobre si existe el thread, si eres el propietario. Luego, comprueba que las imágenes sean válidas en formato y tamaño. La guarda en una lista temporal

```
[HttpPost()]
[Authorize]
public async Task<IActionResult> UpdateThread([FromForm] ThreadUpdateDTO ThreadDTO)
{
    var thread = await _context.Threads
        .Include(t => t.Images)
        .FirstOrDefaultAsync(t => t.Id == ThreadDTO.Id);
    long maxSize = 5 * 1024 * 1024;
    if (thread == null)
    {
        _logger.LogWarning("No se encontró el thread");
        return NotFound(new { error = "No se encuentra el thread" });
    }
    var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
    if (thread.CreatorId != userId)
    {
        _logger.LogWarning("No eres el creador del thread");
        return Forbid("No tienes permisos para actualizar este thread");
    }
    thread.Title = ThreadDTO.Title;
    thread.Content = ThreadDTO.Content;
    thread.UpdatedAt = DateTime.UtcNow;
    List<string> imagePaths = new List<string>();
    if (ThreadDTO.Images != null && ThreadDTO.Images.Count > 0)
    {
        string picUID = $"{Guid.NewGuid()}";
        for (int i = 0; i < ThreadDTO.Images.Count; i++)
        {
            var file = ThreadDTO.Images[i];
            if (file != null && file.Length > 0)
            {
                string extension = Path.GetExtension(file.FileName).ToLower();
                if (!_fileFolderService.IsValidExtension(extension))
                    return BadRequest(new { error = "Solo se permiten archivos JPG o PNG para las imágenes." });
                if (!_fileFolderService.IsValidFileSize(file.Length, maxSize))
                    return BadRequest(new { error = "Cada imagen no debe exceder los 5 MB." });

                string fileName = $"{picUID}_{i}";
                string savedPath = await _fileFolderService.SaveFileAsync(file, fileName, "community_images/Threads");

                if (savedPath == null)
                    return BadRequest(new { error = "Error al guardar una de las imágenes." });
                imagePaths.Add(savedPath);
            }
        }
    }
}
```

A continuación, compara las imágenes antiguas con las que se mantienen y las que no coinciden se borran del servidor. Por último, añade las nuevas imágenes a la base de datos.

```
var onServer = thread.Images.ToList();
var imagesToKeep = ThreadDTO.ImagesToKeep ?? new List<string>();
var imagesToDelete = onServer
    .Where(img => !imagesToKeep.Contains(img.ImageUrl))
    .ToList();

foreach (var image in imagesToDelete)
{
    var fullPath = Path.Combine(Directory.GetCurrentDirectory(), image.ImageUrl.TrimStart('/'));
    if (_fileFolderService.DeleteFile(fullPath))
    {
        _logger.LogInformation($"Imagen eliminada: {image.ImageUrl}");
    }
    else
    {
        _logger.LogWarning($"No se pudo eliminar: {image.ImageUrl}");
    }
}
_context.ThreadImages.RemoveRange(imagesToDelete);

foreach (var path in [REDACTED])
{
    _context.ThreadImages.Add(new ThreadImage
    {
        ThreadId = thread.Id,
        ImageUrl = path
    });
}

await _context.SaveChangesAsync();
_logger.LogInformation("Thread actualizado correctamente");
return Ok(new { message = "Updateo correcto" });
}
```

2. Delete thread

El método Delete Thread se encarga de realizar el borrado de los threads de la BBDD. Sin embargo, el borrado no se realiza solamente en función de si es propietario o no. Los administradores, moderadores y usuarios que tengan la posibilidad de borrar threads gracias a su rol de comunidad podrán borrarlos también.

Al principio se realizan comprobaciones básicas sobre si el thread existe, si el usuario es el propietario del thread o aquel que borra puede borrar threads entonces se procederá primero a eliminar todas las imágenes del directorio. Por último, se borrará el thread de la base de datos.

```
[HttpDelete("community/{idCom}/Thread/{idThread}")]
[Authorize]
public async Task<IActionResult> DeleteThread(int idThread, int idCom)
{
    bool canDeleteThread = false;
    var thread = await _context.Threads.FindAsync(idThread);
    if (thread != null)
    {
        var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
        var userRol = await _context.CommunityRoles
            .Where(ucr => ucr.UserId == userId && ucr.CommunityId == idCom)
            .Include(ucr => ucr.Role)
            .Select(ucr => ucr.Role)
            .FirstOrDefaultAsync();
        if (userRol != null)
        {
            if (userRol.CanDeleteThreads)
            {
                canDeleteThread = userRol.CanDeleteThreads;
            }
        }
        if (thread.CreatorId == userId || canDeleteThread)
        {
            var images = await _context.ThreadImages
                .Where(img => img.ThreadId == thread.Id)
                .Select(img => img.ImageUrl)
                .ToListAsync();
            if (images.Count > 0)
            {
                foreach (string path in images)
                {
                    var imagepath = Path.Combine(Directory.GetCurrentDirectory(), path.TrimStart('/'));
                    if (!_fileFolderService.DeleteFile(imagepath))
                    {
                        _logger.LogWarning("No se pudo eliminar la imagen.");
                    }
                    else
                    {
                        _logger.LogInformation("Imagen de thread eliminada.");
                    }
                }
            }
            _context.Threads.Remove(thread);
            await _context.SaveChangesAsync();
            _logger.LogInformation("Thread eliminado");
            return Ok(new { error = "Thread eliminado" });
        }
        else
        {
            _logger.LogWarning("No es su thread");
            return Forbid("No es su thread");
        }
    }
    else
    {
        _logger.LogWarning("No existe este thread");
        return NotFound(new { error = "No existe este thread" });
    }
}
```

class System.String
Represents text as a sequence of characters.

3. Assign Role

Se encarga de asignar un rol específico a un usuario dentro de una comunidad. Sin embargo, esta operación no puede ser realizada por cualquier usuario: solo el creador de la comunidad tiene permiso para asignar roles.

Primero, se comprueba si la comunidad existe en la base de datos. Luego, se verifica si el usuario ya tiene un rol asignado en dicha comunidad; en ese caso, la operación se cancela con un error, ya que un usuario no puede tener múltiples roles simultáneamente en una misma comunidad.

Si el usuario autenticado es efectivamente el creador de la comunidad, se valida que el rol especificado exista y pertenezca a esa comunidad. Si todo es correcto, se guarda en la base de datos.

En caso de que el rol no exista, la comunidad no sea válida o el usuario autenticado no sea el creador, se devuelve el error correspondiente.

```
[HttpPost("assignRole")]
[Authorize]
public async Task<IActionResult> AssignRoleToUser([FromBody] AssReRoleDTO dto)
{
    var community = await _context.Communities.FindAsync(dto.CommunityId);
    if (community != null)
    {
        var userRole = await _context.CommunityRoles
            .FirstOrDefaultAsync(ur => ur.UserId == dto.UserId && ur.CommunityId == dto.CommunityId);

        if (userRole != null)
            return BadRequest(new { error = "El usuario ya tiene un rol asignado en esta comunidad." });

        var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
        if (userId == community.CreatorId)
        {
            var role = await _context.Roles
                .FirstOrDefaultAsync(r => r.Id == dto.RoleId && r.CommunityId == dto.CommunityId);

            if (role != null)
            {
                var newUserRole = new UserCommunityRole
                {
                    UserId = dto.UserId,
                    CommunityId = dto.CommunityId,
                   RoleId = dto.RoleId
                };

                _context.CommunityRoles.Add(newUserRole);
                await _context.SaveChangesAsync();
                return Ok(new { error = "Rol asignado correctamente." });
            }
            else
                return NotFound(new { error = "El rol no existe para esta comunidad." });
        }
        else
            return Unauthorized(new { error = "Solo el creador de la comunidad puede asignar roles." });
    }
    else
        return NotFound(new { error = "Comunidad no encontrada." });
}
```

4. GetThreadsByLike

Permite obtener una lista paginada de threads que han sido marcados con "like" por un usuario específico.

Primero, se consulta la tabla de "likes" (ThreadLikes) filtrando por el UserId correspondiente al creatorId recibido como parámetro. Los datos se proyectan en un DTO que contiene el ID, título, lista de URLs de imágenes y número de likes del thread.

Si no se encuentran resultados, se devuelve un error indicando que el usuario no ha dado like a ningún thread. En caso contrario, se devuelve un objeto paginado con la información solicitada y metadatos de la paginación.

```
[HttpGet("byLike/{creatorId}")]
public async Task<IActionResult> GetThreadsByLikeId(int creatorId, [FromQuery] int pageNumber = 1)
{
    const int PageSize = 10;

    var threads = _context.ThreadLikes
        .Where(cs => cs.UserId == creatorId)
        .Include(cs => cs.thread)
        .ThenInclude(t => t.Images);

    var total = await threads.CountAsync();

    var threadLikes = await threads
        .OrderBy(cs => cs.thread.Id)
        .Skip((pageNumber - 1) * PageSize)
        .Take(PageSize)
        .Select(cs => new ThreadListDTO
    {
        Id = cs.thread.Id,
        Title = cs.thread.Title,
        Images = cs.thread.Images.Select(i => i.ImageUrl).ToList(),
        Likes = cs.thread.CountLikes
    })
    .ToListAsync();

    if (!threadLikes.Any())
    {
        _logger.LogInformation("Usuario sin likes");
        return NotFound(new { error = "No se encontraron likes a threads." });
    }

    var result = new PageDTO<ThreadListDTO>
    {
        Data = threadLikes,
        PageNumber = pageNumber,
        PageSize = PageSize,
        TotalRecords = total
    };

    _logger.LogInformation("Threads con likes del usuario");
    return Ok(result);
}
```

5. HasLike

Se encarga de comprobar si el usuario actualmente autenticado ha dado "like" a un thread específico. Esta verificación es sencilla pero muy útil, ya que permite adaptar dinámicamente la interfaz de usuario en el Frontend.

Primero, se obtiene el ID del usuario a partir del token JWT mediante el sistema de autenticación. Luego, se consulta la base de datos para verificar si existe una relación en la tabla de likes que coincida con el threadId recibido por parámetro y el UserId autenticado.

```
[HttpGet("HasLike/{threadId}")]
[Authorize]
public async Task<IActionResult> HasUserLikedThread(int threadId)
{
    int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

    bool hasLike = await _context.ThreadLikes
        .AnyAsync(tl => tl.ThreadId == threadId && tl.UserId == userId);

    return Ok(hasLike);
}
```

6. LikeThread

Permite a un usuario autenticado dar "like" a un thread determinado.

Primero, se comprueba si el thread indicado existe en la base de datos. Luego, se obtiene el ID del usuario autenticado y se verifica si ya ha dado like a ese thread. Si el like ya existe, se retorna un error informando que no se puede dar like más de una vez al mismo hilo.

Si el usuario no ha dado like previamente, se crea una nueva entrada en la base de datos, se incrementa el contador de likes del thread y se guardan los cambios.

```
[HttpPost("LikeThread/{threadId}")]
[Authorize]
public async Task<IActionResult> LikeThread(int threadId)
{
    var thread = await _context.Threads.FindAsync(threadId);
    if (thread != null)
    {
        var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

        var existingLike = await _context.ThreadLikes
            .FirstOrDefaultAsync(tl => tl.UserId == userId && tl.ThreadId == thread.Id);

        if (existingLike != null)
        {
            _logger.LogWarning("El usuario ya ha dado like a este hilo");
            return BadRequest(new { error = "Ya has dado like a este hilo" });
        }
        else
        {
            ThreadLikes newLike = new()
            {
                ThreadId = thread.Id,
                UserId = userId
            };
            thread.CountLikes += 1;
            _context.ThreadLikes.Add(newLike);
            _context.Threads.Update(thread);
            await _context.SaveChangesAsync();
            _logger.LogInformation("Like al hilo realizado");
            return Ok();
        }
    }
    else
    {
        _logger.LogWarning("No se encontró el hilo para dar like");
        return NotFound(new { error = "No se encuentra el hilo" });
    }
}
```

7. Create role

Se encarga de crear un nuevo rol dentro de una comunidad específica. Esta funcionalidad permite que el creador de una comunidad defina roles personalizados con distintos niveles de permisos.

Primero, se verifica si la comunidad indicada existe en la base de datos. A continuación, se comprueba que el usuario autenticado sea el creador de dicha comunidad, ya que solo él tiene autorización para crear nuevos roles.

Luego, se revisa que no exista ya un rol con el mismo nombre dentro de esa comunidad. Si no hay duplicados, se crea una nueva instancia del rol con los permisos definidos en el DTO, como la posibilidad de borrar threads, respuestas o banear usuarios.

Una vez creado, el nuevo rol se guarda en la base de datos. En caso de error, como comunidad inexistente, duplicado de rol o falta de autorización, se devuelve un mensaje informativo correspondiente.

```
[HttpPost("createRole")]
[Authorize]
public async Task<IActionResult> CreateRole([FromBody] RoleCreateDTO dto)
{
    var community = await _context.Communities.FindAsync(dto.CommunityId);
    if (community != null)
    {
        var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
        if (userId == community.CreatorId)
        {
            var existingRole = await _context.Roles
                .FirstOrDefaultAsync(r => r.CommunityId == dto.CommunityId && r.RoleName == dto.RoleName);

            if (existingRole != null)
                return BadRequest(new { error = "El rol ya existe en esta comunidad." });

            var newRole = new Role
            {
                CommunityId = dto.CommunityId,
                RoleName = dto.RoleName,
                CanDeleteThreads = dto.CanDeleteThreads,
                CanDeleteResponses = dto.CanDeleteResponses,
                CanBanUsers = dto.CanBanUsers
            };

            _context.Roles.Add(newRole);
            await _context.SaveChangesAsync();
            return Ok(new { error = "Rol creado correctamente." });
        }
        else
            return Unauthorized(new { error = "Solo el creador de la comunidad puede crear roles." });
    }
    else
        return NotFound(new { error = "Comunidad no encontrada." });
}
```

2.3.3. Backend – API de perfiles

Esta API permite la creación o edición de un perfil se usuario, así como manejar los seguidores y los seguidos. Esto está dividido en 2 controladores. Además, en este microservicio se pueden guardar imágenes.

Para poder guardar imágenes se añade en el program.cs la opción de usar archivos

```
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "profile_images")),
    RequestPath = "/profile_images"
});
```

estáticos:

Controlador User

En este controlador se tratan todos los endpoints correspondientes a la obtención de perfiles, su creación y su edición.

1. Listar usuarios

GET /api/UserProfile/Users

Devuelve un listado paginado de usuarios.

```
[Authorize]
[HttpGet("Users")]
public async Task<IActionResult> GetUsers([FromQuery] int lastUserId = 0, [FromQuery] string name = "")
{
    int pageSize = 10;

    var userList = _context.Users
        .Where(user => (lastUserId == 0 || user.UserId > lastUserId) &&
            (string.IsNullOrEmpty(name) || user.UserName.Contains(name)))
        .Select(user => new UserListDTO
        {
            UserId = user.UserId,
            UserName = user.UserName,
            ProfilePicture = user.ProfilePicture,
        });
    
    var totalRecords = await userList.CountAsync();

    var users = await userList
        .Take(pageSize)
        .ToListAsync();

    bool hasMore = users.Count == pageSize;

    int lastId = users.Any() ? users.Last().UserId : 0;

    var result = new ScrollDTO<UserListDTO>
    {
        Data = users,
        TotalRecords = totalRecords,
        LastId = lastId,
        HasMore = hasMore
    };

    _logger.LogInformation("Listado paginado de usuarios");

    return Ok(result);
}
```

2. Listar talleres

GET /api/UserProfile/Workshop

Devuelve un listado paginado de usuarios con rol de taller.

```
[Authorize]
[HttpGet("Workshop")]
public async Task<IActionResult> GetWorkshop([FromQuery] int lastUserId = 0, [FromQuery] string name = "")
{
    int pageSize = 10;

    var shopList = _context.Users
        .Where(user => (lastUserId == 0 || user.UserId > lastUserId) &&
            (string.IsNullOrEmpty(name) || user.UserName.Contains(name)) &&
            user.Type == UserType.Workshop)
        .Select(user => new UserListDTO
    {
        UserId = user.UserId,
        UserName = user.UserName,
        ProfilePicture = user.ProfilePicture
    });
}

var totalRecords = await shopList.CountAsync();

var shops = await shopList
    .Take(pageSize)
    .ToListAsync();

bool hasMore = shops.Count == pageSize;

int lastId = shops.Any() ? shops.Last().UserId : 0;

var result = new ScrollDTO<UserListDTO>
{
    Data = shops,
    TotalRecords = totalRecords,
    LastId = lastId,
    HasMore = hasMore
};

_logger.LogInformation("Listado paginado de talleres");

return Ok(result);
}
```

3. Obtener usuario por ID

GET /api/UserProfile/Users/GetUserById/{UserId}

Devuelve los detalles de un usuario específico.

```
[Authorize]
[HttpGet("Users/GetUserById/{userId}")]
public async Task<IActionResult> GetUserById(int userId)
{
    var user = await _context.Users
        .Where(u => u.UserId == userId)
        .FirstOrDefaultAsync();

    if (user == null)
    {
        _logger.LogWarning("Usuario no encontrado");
        return NotFound("Usuario no encontrado.");
    }
    else
    {
        var userDetail = new UserDetailDTO
        {
            UserId = user.UserId,
            UserName = user.UserName,
            ProfilePicture = user.ProfilePicture,
            Name = user.Name,
            Description = user.Description,
            Adress = user.Adress,
            CountFollowers = user.CountFollowers,
            CountFollowing = user.CountFollowing,
            Type = user.Type
        };

        _logger.LogInformation("Detalles del usuario");

        return Ok(userDetail);
    }
}
```

4. Crear usuario

POST /api/UserProfile/UserCreate

Crea un nuevo usuario con los datos proporcionados y una imagen de perfil.

```
[Authorize]
[HttpPost("UserCreate")]
public async Task<IActionResult> CreateUser([FromForm] UserCreateDTO userCreateDto)
{
    int userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
    int userTypeClaim = int.Parse(User.FindFirst("UserType")?.Value);
    UserType userType = (UserType)userTypeClaim;

    long maxSize = 5 * 1024 * 1024;
    string targetFolder = "profile_images";

    if (string.IsNullOrEmpty(userCreateDto.Name) || string.IsNullOrEmpty(userCreateDto.UserName))
        return BadRequest("El nombre y el nombre de usuario son obligatorios.");

    var existingUser = await _context.Users
        .FirstOrDefaultAsync(u => u.UserName == userCreateDto.UserName);

    if (existingUser != null)
        return Conflict("El nombre de usuario ya está en uso.");

    if (userCreateDto.profilePicture != null && userCreateDto.profilePicture.Length > 0)
    {
        var fileExtension = Path.GetExtension(userCreateDto.profilePicture.FileName);

        if (!fileFolderService.IsValidFileExtension(fileExtension))
            return BadRequest("Solo se permiten archivos JPG o PNG.");

        if (!fileFolderService.IsValidFileSize(userCreateDto.profilePicture.Length, maxSize))
            return BadRequest("El tamaño del archivo no debe exceder los 5 MB.");

        string picUID = $"{Guid.NewGuid()}";
        var profilePicturePath = await _fileFolderService.SaveFileAsync(userCreateDto.profilePicture, picUID, targetFolder);

        if (profilePicturePath == null)
        {
            _logger.LogError("No se pudo guardar la imagen del perfil.");
            return BadRequest("Hubo un error al guardar la imagen.");
        }

        var newUser = new User
        {
            UserId = userId,
            ProfilePicture = profilePicturePath,
            Name = userCreateDto.Name,
            UserName = userCreateDto.UserName,
            Description = userCreateDto.Description,
            Adress = userCreateDto.Address,
            CreatedAt = DateTime.UtcNow,
            UpdatedAt = DateTime.UtcNow,
            Type = userType
        };

        _context.Users.Add(newUser);
        await _context.SaveChangesAsync();
        _logger.LogInformation("Nuevo usuario creado");

        return Ok();
    }
    else
    {
        _logger.LogWarning("No ha subido imagen.");
        return BadRequest("No ha subido imagen.");
    }
}
```

5. Actualizar usuario

PATCH /api/UserProfile/UserUpdate

Actualiza los datos del usuario autenticado. Permite subir una nueva imagen de perfil.

```
[Authorize]
[HttpPatch("UserUpdate")]
public async Task<IActionResult> UpdateUser([FromForm] UserCreateDTO userDTO)
{
    var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
    var user = await _context.Users.FindAsync(userId);

    if (user != null)
    {
        if (userDTO.profilePicture != null && userDTO.profilePicture.Length > 0)
        {
            var fileExtension = Path.GetExtension(userDTO.profilePicture.FileName).ToLower();
            long maxSize = 5 * 1024 * 1024; // 5 MB

            if (!_fileFolderService.IsValidFileExtension(fileExtension))
                return BadRequest("Solo se permiten archivos JPG o PNG.");

            if (!_fileFolderService.IsValidFileSize(userDTO.profilePicture.Length, maxSize))
                return BadRequest("El tamaño del archivo no debe exceder los 5 MB.");

            if (!string.IsNullOrEmpty(user.ProfilePicture))
            {
                var oldImagePath = Path.Combine(Directory.GetCurrentDirectory(), user.ProfilePicture.TrimStart('/'));
                if (!_fileFolderService.DeleteFile(oldImagePath))
                    return BadRequest("Hubo un problema al intentar eliminar la imagen anterior.");
            }
            string picUID = $"{Guid.NewGuid()}";

            var newFilePath = await _fileFolderService.SaveFileAsync(userDTO.profilePicture, picUID, "profile_images");
            if (newFilePath == null)
                return BadRequest("Hubo un error al guardar la imagen.");
            user.ProfilePicture = newFilePath;
        }

        user.Name = userDTO.Name;
        user.UserName = userDTO.UserName;
        user.Description = userDTO.Description;
        user.Adress = userDTO.Address;
        user.UpdatedAt = DateTime.UtcNow;

        _logger.LogInformation("Perfil actualizado");
        _context.Users.Update(user);
        await _context.SaveChangesAsync();

        return Ok("Perfil actualizado");
    }
    else
    {
        _logger.LogWarning("El perfil no existe");
        return NotFound();
    }
}
```

Controlador followers

En este controlador se tratan todos los endpoints correspondientes a la visualización de seguidores, seguidos y las acciones que puede realizar el usuario.

1. Obtener seguidores

GET /api/UserProfile/followers/{UserId}

Devuelve una lista paginada de usuarios que siguen al usuario especificado.

```
[HttpGet("followers/{userId}")]
public async Task<IActionResult> GetFollowers(int userId, [FromQuery] int lastUserId = 0, [FromQuery] string name = null)
{
    int pageSize = 10;

    var followersUser = _context.Followers
        .Where(f => f.FollowingId == userId)
        .Select(f => f.Follower);

    if (!string.IsNullOrEmpty(name))
    {
        followersUser = followersUser.Where(u => u.UserName.Contains(name));
    }

    followersUser = followersUser.OrderBy(u => u.UserId);

    var follows = followersUser
        .Where(u => lastUserId == 0 || u.UserId > lastUserId)
        .Select(u => new UserListDTO
    {
        UserId = u.UserId,
        ProfilePicture = u.ProfilePicture,
        UserName = u.UserName
    });
}

var followers = await follows
    .Take(pageSize + 1)
    .ToListAsync();

bool hasMore = followers.Count > pageSize;

if (hasMore)
{
    followers.RemoveAt(pageSize);
}

int lastId = followers.LastOrDefault()?.UserId ?? 0;

ScrollDTO<UserListDTO> result = new()
{
    Data = followers,
    TotalRecords = followers.Count,
    LastId = lastId,
    HasMore = hasMore
};

_logger.LogInformation("Mostrar seguidores");

return Ok(result);
}
```

2. Obtener seguidos

GET /api/UserProfile/following/{UserId}

Devuelve una lista paginada de usuarios que el usuario especificado sigue.

```
[HttpGet("following/{userId}")]
public async Task<IActionResult> GetFollowing(int userId, [FromQuery] int lastUserId = 0, [FromQuery] string name = null)
{
    int pageSize = 10;

    var followingUser = _context.Followers
        .Where(f => f.FollowerId == userId)
        .Select(f => f.Following);

    if (!string.IsNullOrEmpty(name))
    {
        followingUser = followingUser.Where(u => u.UserName.Contains(name));
    }

    followingUser = followingUser.OrderBy(u => u.UserId);

    var following = followingUser
        .Where(u => lastUserId == 0 || u.UserId > lastUserId)
        .Select(u => new UserListDTO
    {
        UserId = u.UserId,
        ProfilePicture = u.ProfilePicture,
        UserName = u.UserName
    });

    var followingList = await following
        .Take(pageSize + 1)
        .ToListAsync();

    bool hasMore = followingList.Count > pageSize;

    if (hasMore)
    {
        followingList.RemoveAt(pageSize);
    }

    int lastId = followingList.LastOrDefault()?.UserId ?? 0;

    ScrollDTO<UserListDTO> result = new()
    {
        Data = followingList,
        TotalRecords = followingList.Count,
        LastId = lastId,
        HasMore = hasMore
    };

    _logger.LogInformation("Mostrar siguiendo");

    return Ok(result);
}
```

3. Comprobar si un usuario sigue a otro

GET /api/UserProfile/IsFollowing/{UserId}/{otherUserId}

Devuelve true o false indicando si un usuario sigue a otro

```
[HttpGet("IsFollowing/{userId}/{otherUserId}")]
public async Task<IActionResult> IsFollowing(int userId, int otherUserId)
{
    bool isFollowing = await _context.Followers
        .AnyAsync(f => f.FollowerId == userId && f.FollowingId == otherUserId);

    return Ok(isFollowing);
}
```

4. Seguir a un usuario

POST /api/UserProfile/StartFollowing

Permite al usuario autenticado comenzar a seguir a otro usuario

```
[Authorize]
[HttpPost("StartFollowing")]
public async Task<IActionResult> StartFollowing(int profileId) {
    var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
    if (userId != profileId)
    {
        var userFollowing = await _context.Users.FindAsync(profileId);
        if (userFollowing != null)
        {
            var follows = await _context.Followers.FirstOrDefaultAsync(r => r.FollowerId == userId && r.FollowingId == profileId);
            if (follows == null)
            {
                var user = await _context.Users.FindAsync(userId);
                Follower follower = new()
                {
                    FollowerId = userId,
                    FollowingId = profileId
                };
                _logger.LogWarning("Following");
                user.CountFollowing += 1;
                userFollowing.CountFollowers += 1;

                _context.Followers.Add(follower);
                _context.Users.Update(user);
                _context.Users.Update(userFollowing);
                await _context.SaveChangesAsync();
                return Ok();
            }
            else
            {
                _logger.LogWarning("Ya sigue a este usuario");
                return NotFound("Ya sigue al usuario");
            }
        }
        else
        {
            _logger.LogWarning("Usuario no encontrado");
            return NotFound("Usuario no encontrado");
        }
    }
    else {
        _logger.LogWarning("No puedes seguirte a ti mismo");
        return BadRequest("No puedes seguirte a ti mismo");
    }
}
```

5. Dejar de seguir a un usuario

DELETE /api/UserProfile/StopFollowing

Permite al usuario autenticado dejar de seguir a otro usuario

```
[Authorize]
[HttpDelete("StopFollowing")]
public async Task<IActionResult> StopFollowing(int profileId) {
    var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
    var userFollowing = await _context.Users.FindAsync(profileId);
    if (userFollowing != null)
    {
        var follows = await _context.Followers.FirstOrDefaultAsync(r => r.FollowerId == userId && r.FollowingId == profileId);
        if (follows != null)
        {
            var user = await _context.Users.FindAsync(userId);

            user.CountFollowing -= 1;
            userFollowing.CountFollowers -= 1;

            _logger.LogWarning("Paró de seguir");
            _context.Followers.Remove(follows);
            _context.Users.Update(user);
            _context.Users.Update(userFollowing);
            await _context.SaveChangesAsync();
            return Ok();
        }
        else
        {
            _logger.LogWarning("No sigue a este usuario");
            return NotFound("No sigue al usuario");
        }
    }
    else
    {
        _logger.LogWarning("Usuario no encontrado");
        return NotFound("Usuario no encontrado");
    }
}
```

6. Eliminar un seguidor

DELETE /api/UserProfile/DropFollower

Permite al usuario autenticado eliminar a un seguidor.

```
[Authorize]
[HttpDelete("DropFollower")]
public async Task<IActionResult> DropFollower(int profileId) {
    var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
    var userFollowing = await _context.Users.FindAsync(profileId);
    if (userFollowing != null)
    {
        var follows = await _context.Followers.FirstOrDefaultAsync(r => r.FollowerId == profileId && r.FollowingId == userId);
        if (follows != null)
        {
            var user = await _context.Users.FindAsync(userId);
            _logger.LogWarning("Paró de seguerte");
            user.CountFollowers -= 1;
            userFollowing.CountFollowing -= 1;

            _logger.LogWarning("Paró de seguir");
            _context.Followers.Remove(follows);
            _context.Users.Update(user);
            _context.Users.Update(userFollowing);
            await _context.SaveChangesAsync();
            return Ok();
        }
        else
        {
            _logger.LogWarning("Usuario no te sigue");
            return NotFound("No sigues al usuario");
        }
    }
    else
    {
        _logger.LogWarning("Usuario no encontrado");
        return NotFound("Usuario no encontrado");
    }
}
```

Servicio File Folder

La clase FileFolderService se encarga de gestionar operaciones relacionadas con archivos en el servidor, como validar extensiones y tamaños permitidos, guardar archivos en carpetas específicas con un nombre basado en el usuario, y eliminar archivos existentes.

En esta clase encontramos las distintas operaciones:

- Comprobar extensión

```
public bool IsValidFileExtension(string fileExtension)
{
    fileExtension = fileExtension.ToLower();
    if (!_allowedExtensions.Contains(fileExtension))
    {
        _logger.LogWarning("Extensión de archivo no permitida.");
        return false;
    }
    return true;
}
```

- Comprobar tamaño

```
public bool IsValidFileSize(long fileSize, long maxSize)
{
    if (fileSize > maxSize)
    {
        _logger.LogWarning("El archivo excede el tamaño máximo.");
        return false;
    }
    return true;
}
```

- Salvar archivo

```
public async Task<string> SaveFileAsync(IFormFile file, string userId, string targetFolder)
{
    try
    {
        var fileName = $"{userId}_{file.FileName}";
        var filePath = Path.Combine(Directory.GetCurrentDirectory(), targetFolder, fileName);
        var directoryPath = Path.GetDirectoryName(filePath);

        if (!Directory.Exists(directoryPath))
        {
            Directory.CreateDirectory(directoryPath);
        }

        using (var stream = new FileStream(filePath, FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }

        return $"{targetFolder}/{fileName}";
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error al guardar el archivo.");
        return null;
    }
}
```

- Eliminar archivo

```
public bool DeleteFile(string filePath)
{
    try
    {
        if (System.IO.File.Exists(filePath))
        {
            System.IO.File.Delete(filePath);
            _logger.LogInformation("Archivo eliminado");
            return true;
        }
        else {
            _logger.LogInformation("No existe la ruta");
            return false;
        }
    }
    catch (Exception ex)
    {
        _logger.LogError($"Error al eliminar archivo. Excepción: {ex.Message}");
        return false;
    }
}
```

2.3.4. Backend – Api de reviews

Esta API permite la creación y edición de un perfil de reviews y respuestas. Esto está dividido en 2 controladores. Además, en este microservicio se implementan políticas de uso en los endpoints para controlar quienes pueden realizar determinadas acciones.

Para las políticas de uso se añadió el siguiente código al programa principal:

```
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("ClientOnly", policy =>
        policy.RequireClaim("userType", ((int)UserType.Client).ToString()));

    options.AddPolicy("WorkshopOnly", policy =>
        policy.RequireClaim("userType", ((int)UserType.Workshop).ToString()));
});
```

Controlador reviews

1. Obtener reseñas por taller

GET /Reviews/GetReviewsByWorkshop/{workshopId}

Devuelve todas las reseñas (incluyendo respuesta si existe) asociadas a un taller.

```
[HttpGet("GetReviewsByWorkshop/{workshopId}")]
public async Task<IActionResult> GetReviewsByWorkshop(int workshopId)
{
    var reviews = await _context.Reviews
        .Include(r => r.Response)
        .Where(r => r.WorkshopId == workshopId)
        .Select(r => new ReviewDetailDTO
    {
        Id = r.Id,
        Rating = r.Rating,
        Comment = r.Comment,
        Response = r.Response != null ? new ResponseUpdateDTO()
        {
            Id = r.Response.Id,
            Message = r.Response.Message
        } : null,
        UserId = r.UserId
    })
    .ToListAsync();

    if (reviews == null || !reviews.Any())
    {
        return NotFound("Sin reviews");
    }
    return Ok(reviews);
}
```

2. Obtener reseña por id

GET /Reviews/GetReviewById/{id}

Devuelve los detalles de una reseña específica, incluyendo la respuesta si está presente.

```
[HttpGet("GetReviewById/{id}")]
public async Task<IActionResult> GetReviewById(int id)
{
    var review = await _context.Reviews
        .Include(r => r.Response)
        .FirstOrDefaultAsync(r => r.Id == id);

    if (review != null)
    {
        ReviewDetailDTO reviewDTO = new()
        {
            Id = review.Id,
            Rating = review.Rating,
            Comment = review.Comment,
            UserId = review.UserId,
            Response = review.Response != null ? new ResponseUpdateDTO
            {
                Id = review.Response.Id,
                Message = review.Response.Message
            } : null
        };

        return Ok(reviewDTO);
    }
    else
    {
        return NotFound("Review no encontrada");
    }
}
```

3. Contar reseñas de un taller

GET /Reviews/CountReviews/{workshopId}

Devuelve el número total de reseñas asociadas a un taller.

```
[HttpGet("CountReviews/{workshopId}")]
public async Task<IActionResult> CountReviews(int workshopId)
{
    var count = _context.Reviews.Where(r => r.WorkshopId == workshopId).Count();
    return Ok(count);
}
```

4. Obtener media de reseñas

GET /Reviews/GetMediaReview/{workshopId}

Devuelve la calificación promedio de todas las reseñas de un taller.

```
[HttpGet("GetMediaReview/{workshopId}")]
public async Task<IActionResult> GetMediaReviews(int workshopId)
{
    var ratings = await _context.Reviews
        .Where(r => r.WorkshopId == workshopId)
        .Select(r => (double?)r.Rating)
        .ToListAsync();

    var averageRating = ratings.Any() ? ratings.Average() : 0;

    return Ok(averageRating);
}
```

5. Obtener reseña de usuario para taller

GET /Reviews/GetReview/{UserId}/{workshopId}

Devuelve la reseña hecha por un usuario específico a un taller. Si no existe, devuelve NULL.

```
[HttpGet("GetReview/{userId}/{workshopId}")]
public async Task<IActionResult> GetReviewByUserAndWorkshop(int userId, int workshopId)
{
    var review = await _context.Reviews
        .Include(r => r.Response)
        .FirstOrDefaultAsync(r => r.UserId == userId && r.WorkshopId == workshopId);

    if (review != null)
    {
        var reviewDTO = new ReviewDetailDTO
        {
            Id = review.Id,
            Rating = review.Rating,
            Comment = review.Comment,
            UserId = review.UserId,
            Response = review.Response != null ? new ResponseUpdateDTO
            {
                Id = review.Response.Id,
                Message = review.Response.Message
            } : null
        };

        return Ok(reviewDTO);
    }

    return Ok(null);
}
```

6. Crear reseña

POST /Reviews

Crea una nueva reseña para un taller por parte del usuario autenticado. Solo puede existir una reseña por usuario-taller.

```
[Authorize(Policy = "ClientOnly")]
[HttpPost]
public async Task<IActionResult> Post([FromBody] ReviewCreateDTO review)
{
    var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
    var existingReview = _context.Reviews.FirstOrDefault(r => r.WorkshopId == review.WorkshopId && r.UserId == userId);

    if (existingReview == null)
    {
        Review rev = new() {
            UserId=userId,
            WorkshopId=review.WorkshopId,
            Rating = review.Rating,
            Comment=review.Comment,
            CreatedAt = DateTime.UtcNow,
            UpdatedAt = DateTime.UtcNow,
        };

        _context.Reviews.Add(rev);
        await _context.SaveChangesAsync();
        return Ok();
    }
    else {
        return Conflict("Ya tienes una review para este taller.");
    }
}
```

7. Actualizar reseña

PUT /Reviews/ReviewUpdate

Permite al usuario autenticado actualizar el comentario y calificación de una reseña existente.

```
[Authorize(Policy = "ClientOnly")]
[HttpPut("ReviewUpdate")]
public async Task<IActionResult> Put([FromBody] ReviewUpdateDTO review)
{
    var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

    var existingReview = _context.Reviews.FirstOrDefault(r => r.Id == review.Id && r.UserId==userId);
    if (existingReview == null)
    {
        return NotFound("Review no encontrada");
    }

    existingReview.Comment = review.Comment;
    existingReview.Rating = review.Rating;
    existingReview.UpdatedAt = DateTime.UtcNow;
    _context.SaveChanges();

    return Ok();
}
```

8. Eliminar reseña

DELETE /Reviews/{id}

Elimina una reseña realizada por el usuario autenticado si existe.

```
[Authorize(Policy = "ClientOnly")]
[HttpDelete("{id}")]
public async Task<IActionResult> Delete(int id)
{
    var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

    var review = _context.Reviews.FirstOrDefault(r => r.Id == id && r.UserId==userId);
    if (review == null)
    {
        return NotFound("Review no encontrada");
    }

    _context.Reviews.Remove(review);
    _context.SaveChanges();

    return Ok();
}
```

Controlador respuestas

1. Obtener respuesta por id

GET /Responses/GetResponseById/{id}

Devuelve una respuesta asociada a una reseña por su id.

```
[HttpGet("GetResponseById/{id}")]
public async Task<IActionResult> GetResponseById(int id)
{
    var response = await _context.Responses
        .FirstOrDefaultAsync(r => r.Id == id);

    if (response == null)
    {
        return NotFound("Response no encontrada");
    }
    else
    {
        ResponseUpdateDTO responseDTO = new()
        {
            Id = response.Id,
            Message = response.Message
        };

        return Ok(responseDTO);
    }
}
```

2. Crear respuesta

POST /Responses/CreateResponse

Permite al taller autenticado crear una respuesta a una reseña que le pertenece.

```
[Authorize(Policy = "WorkshopOnly")]
[HttpPost("CreateResponse")]
public async Task<IActionResult> CreateResponse([FromBody] ResponseCreateDTO dto)
{
    var workshopId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

    var review = await _context.Reviews.FirstOrDefaultAsync(r => r.Id == dto.ReviewId && r.WorkshopId == workshopId);

    if (review == null)
    {
        return NotFound("Review no encontrada o no te pertenece.");
    }
    else
    {
        if (review.Response != null)
        {
            return BadRequest("Ya existe una respuesta para esta review.");
        }
        else
        {
            _context.Responses.Add(new Response
            {
                ReviewId = dto.ReviewId,
                Message = dto.Message,
                RespondedAt = DateTime.UtcNow,
            });

            await _context.SaveChangesAsync();
            return Ok("Respuesta creada correctamente.");
        }
    }
}
```

3. Actualizar respuesta

PUT /Responses/UpdateResponse

Permite al taller autenticado actualizar el contenido de una respuesta existente

```
[Authorize(Policy = "WorkshopOnly")]
[HttpPut("UpdateResponse")]
public async Task<IActionResult> UpdateResponse([FromBody] ResponseUpdateDTO dto)
{
    var workshopId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

    var response = await _context.Responses
        .FirstOrDefaultAsync(r => r.Id == dto.Id && r.Review.WorkshopId == workshopId);

    if (response == null)
    {
        return NotFound("Respuesta no encontrada o no tienes permisos.");
    }
    else
    {
        response.Message = dto.Message;
        await _context.SaveChangesAsync();
        return Ok("Respuesta actualizada.");
    }
}
```

4. Eliminar respuesta

DELETE /Responses/DeleteResponse/{responseId}

Permite al taller autenticado eliminar una respuesta.

```
[Authorize(Policy = "WorkshopOnly")]
[HttpDelete("DeleteResponse/{responseId}")]
public async Task<IActionResult> DeleteResponse(int responseId)
{
    var workshopId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);

    var response = await _context.Responses.FirstOrDefaultAsync(r => r.Id == responseId && r.Review.WorkshopId == workshopId);

    if (response == null)
    {
        return NotFound("Respuesta no encontrada");
    }
    else
    {
        _context.Responses.Remove(response);
        await _context.SaveChangesAsync();

        return Ok("Respuesta eliminada.");
    }
}
```

2.3.5. Backend – Api de talleres

Esta API permite la administración de talleres y las peticiones de reparación que puedan recibir. Se divide en 7 controladores que permiten guardar en una base de datos MongoDB los datos sobre los vehículos de los usuarios, datos fiscales de los usuarios y de los talleres, componentes que tienen los vehículos, órdenes de mantenimiento en caso de que se inicie una reparación y facturas si la reparación ha sido completada.

Controlador Vehicles

En este controlador un usuario puede crear sus vehículos para así poder ser accedidos por los talleres, los talleres no pueden crear vehículos.

```
private bool IsAuthorized(out IActionResult unauthorizedResult)
{
    var userType = User.FindFirst("UserType")?.Value;

    if (userType != "1")
    {
        unauthorizedResult = Unauthorized(new { message = "Only users are allowed." });
        return false;
    }

    unauthorizedResult = null;
    return true;
}
```

De esta forma comprobamos que solo los usuarios y no los talleres sean capaces de crear, modificar o eliminar vehículos.

1. Crear Vehículo

Crea el vehículo y le asigna el id del usuario.

```
[Authorize]
[HttpPost]
O referencias
public async Task<IActionResult> Create(DTOVehiclePost dtoVehicle)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;
    var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0";

    var vehicle = new Vehicle
    {
        License = dtoVehicle.License,
        Vin = dtoVehicle.Vin,
        UserId = userId,
        Brand = dtoVehicle.Brand,
        Model = dtoVehicle.Model,
        Year = dtoVehicle.Year,
        ComponentIds = dtoVehicle.ComponentIds
    };

    await _vehicles.InsertOneAsync(vehicle);
    return CreatedAtAction(nameof(GetById), new { id = vehicle.Id }, vehicle);
}
```

2. Ver Vehículo

Tenemos un DTO diferente en caso de que se quieran ver todos los vehículos o uno específico, en caso de querer ver todos añadimos filtros de usuario, matrícula, etc. Si se quieren acceder a datos más sensibles es necesario acceder por el id del vehículo.

Este endpoint es accesible tanto por usuarios como por talleres.

```
[Authorize]
[HttpGet]
0 referencias
public async Task<IActionResult> GetAll([FromQuery] string? license, [FromQuery] string? userId)
{
    var filterBuilder = Builders<Vehicle>.Filter;
    var filter = filterBuilder.Empty;

    if (!string.IsNullOrEmpty(license))
        filter &= filterBuilder.Eq(v => v.License, license);

    if (!string.IsNullOrEmpty(userId))
        filter &= filterBuilder.Eq(v => v.UserId, userId);

    var vehicles = await _vehicles.Find(filter).ToListAsync();
    var dtoVehicles = vehicles.Select(v => new DTOVehicleReadAll
    {
        Id = v.Id,
        License = v.License,
        UserId = v.UserId
    });
}

return Ok(dtoVehicles);
}

[Authorize]
[HttpGet("{id}")]
1 referencia
public async Task<IActionResult> GetById(string id)
{
    var vehicle = await _vehicles.Find(v => v.Id == id).FirstOrDefaultAsync();
    if (vehicle == null)
        return NotFound();

    var dtoVehicle = new DTOVehicleRead
    {
        Id = vehicle.Id,
        License = vehicle.License,
        Vin = vehicle.Vin,
        UserId = vehicle.UserId,
        Brand = vehicle.Brand,
        Model = vehicle.Model,
        Year = vehicle.Year,
        ComponentIds = vehicle.ComponentIds
    };

    return Ok(dtoVehicle);
}
```

3. Modificar Vehículo

Permite modificar un vehículo si no eres taller y si además eres el propietario de este.

```
[Authorize]
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> Update(string id, DTOVehicleUpdate dtoVehicle)
{
    if (!IsAuthorized(out var unauthorizedResult) && User.FindFirst(ClaimTypes.NameIdentifier)?.Value == dtoVehicle.UserId)
        return unauthorizedResult;

    var vehicle = new Vehicle
    {
        Id = id,
        License = dtoVehicle.License,
        Vin = dtoVehicle.Vin,
        UserId = dtoVehicle.UserId,
        Brand = dtoVehicle.Brand,
        Model = dtoVehicle.Model,
        Year = dtoVehicle.Year,
        ComponentIds = dtoVehicle.ComponentIds
    };

    var result = await _vehicles.ReplaceOneAsync(v => v.Id == id, vehicle);
    return result.ModifiedCount > 0 ? NoContent() : NotFound();
}
```

4. Eliminar Vehículo

Te permite eliminar vehículos si eres el usuario propietario del vehículo y si el vehículo existe

```
[Authorize]
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> Delete(string id)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    var vehicle = await _vehicles.Find(v => v.Id == id).FirstOrDefaultAsync();

    if (vehicle == null)
        return NotFound();

    if (vehicle.UserId != userId)
        return Unauthorized(new { message = "You are not authorized to delete this vehicle." });

    var result = await _vehicles.DeleteOneAsync(v => v.Id == id);
    return result.DeletedCount > 0 ? NoContent() : NotFound();
}
```

Controlador Componentes

En este controlador un taller puede crear componentes para vehículos, en caso de que este componente forme parte de un componente mayor también se puede indicar

```
4 referencias
private bool IsAuthorized(out IActionResult unauthorizedResult)
{
    var userType = User.FindFirst("UserType")?.Value;

    if (userType != "2")
    {
        unauthorizedResult = Unauthorized(new { message = "Only workshops are allowed." });
        return false;
    }

    unauthorizedResult = null;
    return true;
}
```

Solo los talleres pueden modificar componentes, por lo tanto, se comprueba que quien hace la petición sea un taller

1. Crear componente

Permite crear componentes

```
[Authorize]
[HttpPost]
0 referencias
public async Task<IActionResult> Create(DTOComponentPost dtoComponent)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var component = new Component
    {
        Name = dtoComponent.Name,
        Description = dtoComponent.Description,
        ParentAssemblyId = dtoComponent.ParentAssemblyId
    };

    await _components.InsertOneAsync(component);
    return CreatedAtAction(nameof(GetAll), null, component);
}
```

2. Ver componente

Permite ver los componentes, siendo el mismo dto para getall y getbyid. Getall incluye filtros de nombre y de componente padre

```
[Authorize]
[HttpGet]
0 referencias
public async Task<IActionResult> GetAll(
    [FromQuery] string? name,
    [FromQuery] string? Parent)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var filter = Builders<Component>.Filter.Empty;

    if (!string.IsNullOrEmpty(name))
        filter &= Builders<Component>.Filter.Eq(o => o.Name, name);

    if (!string.IsNullOrEmpty(Parent))
        filter &= Builders<Component>.Filter.Eq(o => o.ParentAssemblyId, Parent);

    var components = await _components.Find(filter).ToListAsync();

    var dtoComponents = components.Select(c => new DTOComponentRead
    {
        Id = c.Id ?? string.Empty,
        Name = c.Name,
        Description = c.Description,
        ParentAssemblyId = c.ParentAssemblyId
    });

    return Ok(dtoComponents);
}
[Authorize]
[HttpGet("{id}")]
0 referencias
public async Task<IActionResult> GetById(string id)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var component = await _components.Find(c => c.Id == id).FirstOrDefaultAsync();

    if (component == null)
        return NotFound(new { message = "Component not found." });

    var dtoComponent = new DTOComponentRead
    {
        Id = component.Id ?? string.Empty,
        Name = component.Name,
        Description = component.Description,
        ParentAssemblyId = component.ParentAssemblyId
    };

    return Ok(dtoComponent);
}
```

3. Modificar componente

Permite modificar componentes

```
[Authorize]
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> Update(string id, DTOComponentPut dtoComponent)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var component = new Component
    {
        Id = id,
        Name = dtoComponent.Name,
        Description = dtoComponent.Description,
        ParentAssemblyId = dtoComponent.ParentAssemblyId
    };

    var result = await _components.ReplaceOneAsync(u => u.Id == id, component);
    return result.ModifiedCount > 0 ? NoContent() : NotFound();
}
```

Controlador VehicleComponent

En este controlador podemos añadirle componentes a un vehículo, de esta forma la próxima vez que el vehículo tenga que ir a un taller podrá saber exactamente qué componentes lleva y cuáles se cambiaron.

```
0 referencias
private bool IsAuthorized(out IActionResult unauthorizedResult)
{
    var userType = User.FindFirst("UserType")?.Value;

    if (userType != "2")
    {
        unauthorizedResult = Unauthorized(new { message = "Only workshops are allowed." });
        return false;
    }

    unauthorizedResult = null!;
    return true;
}
```

Este endpoint solo es accesible por talleres.

1. Añadir componente a vehículo

Te permite añadirle componentes a un vehículo.

```
[Authorize]
[HttpPost]
0 referencias
public async Task<IActionResult> AddComponent(string vehicleId, DTOVehicleComponentAdd dto)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var update = Builders<Vehicle>.Update.AddToSet(v => v.ComponentIds, dto.ComponentId);
    var result = await _vehicles.UpdateOneAsync(v => v.Id == vehicleId, update);
    return result.ModifiedCount > 0 ? NoContent() : NotFound();
}
```

2. Eliminar componente de vehículo

Te permite quitarle un componente a un vehículo

```
[Authorize]
[HttpDelete]
0 referencias
public async Task<IActionResult> RemoveComponent(string vehicleId, DTOVehicleComponentRemove dto)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var update = Builders<Vehicle>.Update.Pull(v => v.ComponentIds, dto.ComponentId);
    var result = await _vehicles.UpdateOneAsync(v => v.Id == vehicleId, update);
    return result.ModifiedCount > 0 ? NoContent() : NotFound();
}
```

Controlador Workshop

Este controlador nos permite acceder al nif, localización, especialidad y nombre de los talleres

```
2 referencias
private bool IsAuthorized(out IActionResult unauthorizedResult)
{
    var userType = User.FindFirst("UserType")?.Value;

    if (userType != "2")
    {
        unauthorizedResult = Unauthorized(new { message = "Only workshops are allowed." });
        return false;
    }

    unauthorizedResult = null!;
    return true;
}
```

Solo talleres tienen permitido acceder a este controlador así que se comprueba en su JWT

1. Crear Taller

Te permite añadirle datos a un taller en base a su id del JWT, controlando que no exista ya.

```
[Authorize]
[HttpPost]
0 referencias
public async Task<IActionResult> Create(DTOWorkshopPost dtoWorkshop)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (string.IsNullOrEmpty(userId))
        return BadRequest(new { message = "couldnt find jwtuserid." });

    var exists = await _workshops.FindAsync(w => w.UserId == userId).AnyAsync();
    if (exists)
        return Conflict(new { message = "already exists a workshop with that jwtuserid" });

    var workshop = new Workshop
    {
        UserId = userId,
        Nif = dtoWorkshop.Nif,
        Location = dtoWorkshop.Location,
        Speciality = dtoWorkshop.Speciality,
        Name = dtoWorkshop.Name
    };

    await _workshops.InsertOneAsync(workshop);
    return CreatedAtAction(nameof(GetAll), null, workshop);
}
```

2. Acceder a Taller

Te permite acceder a los talleres, el getall tiene filtros para localización, nombre y especialidad y el getbyid añade el nif en su dto, solo es accesible por el propio taller.

```
[Authorize]
[HttpGet]
0 referencias
public async Task<IActionResult> GetAll([FromQuery] string? userId, [FromQuery] string? location, [FromQuery] string? speciality, [FromQuery] string? name)
{
    var filterBuilder = Builders<Workshop>.Filter;
    var filter = filterBuilder.Empty;

    if (!string.IsNullOrEmpty(userId))
        filter &= filterBuilder.Eq(w => w.UserId, userId);

    if (!string.IsNullOrEmpty(location))
        filter &= filterBuilder.Eq(w => w.Location, location);

    if (!string.IsNullOrEmpty(speciality))
        filter &= filterBuilder.Eq(w => w.Speciality, speciality);

    if (!string.IsNullOrEmpty(name))
        filter &= filterBuilder.Eq(w => w.Name, name);

    var workshops = await _workshops.Find(filter).ToListAsync();
    var dtoWorkshops = workshops.Select(w => new DTOWorkshopReadAll
    {
        Id = w.Id,
        UserId = w.UserId,
        Location = w.Location,
        Speciality = w.Speciality,
        Name = w.Name
    });
}

return Ok(dtoWorkshops);
}

[Authorize]
[HttpGet("{JWTId}")]
0 referencias
public async Task<IActionResult> GetById(string JWTId)
{
    var workshop = await _workshops.Find(w => w.UserId == JWTId).FirstOrDefaultAsync();
    if (workshop == null)
        return NotFound(new { message = "Workshop not found." });

    var jwtUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (jwtUserId != workshop.UserId)
        return Forbid();

    var dtoWorkshop = new DTOWorkshopRead
    {
        Id = workshop.Id,
        UserId = workshop.UserId,
        Nif = workshop.Nif,
        Location = workshop.Location,
        Speciality = workshop.Speciality,
        Name = workshop.Name
    };

    return Ok(dtoWorkshop);
}
```

3. Borrar Taller

Permite borrar un taller en base a su id de JWT, solo te permite borrar tu propio taller.

```
[Authorize]
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> Delete(string id)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (userId != id)
        return Forbid("You can only delete your own workshop");

    var result = await _workshops.DeleteOneAsync(w => w.Id == id);
    if (result.DeletedCount == 0)
        return NotFound(new { message = "Workshop not found." });

    return NoContent();
}
```

Controlador User

Este controlador nos permite acceder al nombre, apellido y DNI de los usuarios para la posterior creación de facturas.

```
3 referencias
private bool IsAuthorized(out IActionResult unauthorizedResult)
{
    var userType = User.FindFirst("UserType")?.Value;
    if (userType != "1")
    {
        unauthorizedResult = Unauthorized(new { message = "Only users are allowed." });
        return false;
    }
    unauthorizedResult = null!;
    return true;
}
```

Solo los usuarios pueden acceder a post, put y Delete.

1. Crear Usuario

Te permite añadirle datos a un usuario en base a su id del JWT, controlando que no exista ya.

```
[Authorize]
[HttpPost]
0 referencias
public async Task<IActionResult> Create(DTOUserPost dtoUser)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var userid = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (string.IsNullOrEmpty(userid))
        return BadRequest(new { message = "couldnt find jwtuserid." });

    var exists = await _users.Find(u => u.UserId == userid).AnyAsync();
    if (exists)
        return Conflict(new { message = "Already exists an user with that JwtuserId." });

    var user = new User
    {
        UserId = userid,
        Name = dtoUser.Name,
        Surname = dtoUser.Surname,
        dni = dtoUser.dni
    };

    await _users.InsertOneAsync(user);
    return CreatedAtAction(nameof(GetById), new { id = user.Id }, user);
}
```

2. Modificar Usuario

Te permite modificar los datos del propio usuario.

```
[Authorize]
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> Update(string id, DTOUserUpdate dtoUser)
{
    if (!IsAuthorized(out var unauthorizedResult) && User.FindFirst(ClaimTypes.NameIdentifier)?.Value == id)
        return unauthorizedResult;

    var user = new User
    {
        Id = id,
        Name = dtoUser.Name,
        Surname = dtoUser.Surname
    };

    var result = await _users.ReplaceOneAsync(u => u.Id == id, user);
    return result.ModifiedCount > 0 ? NoContent() : NotFound();
}
```

3. Ver Usuario

Te permite ver los datos de todos los usuarios con filtros de nombre y apellido o te permite ver los datos de un usuario si es tu propio perfil, de esta forma incluye el DNI.

```
[Authorize]
[HttpGet]
0 referencias
public async Task<IActionResult> GetAll([FromQuery] string? name, [FromQuery] string? surname, [FromQuery] string? userId)
{
    var filterBuilder = Builders<User>.Filter;
    var filter = filterBuilder.Empty;

    if (!string.IsNullOrEmpty(name))
        filter &= filterBuilder.Eq(u => u.Name, name);

    if (!string.IsNullOrEmpty(surname))
        filter &= filterBuilder.Eq(u => u.Surname, surname);

    if (!string.IsNullOrEmpty(userId))
        filter &= filterBuilder.Eq(u => u.UserId, userId);

    var users = await _users.Find(filter).ToListAsync();
    var dtoUsers = users.Select(u => new DTOUserReadAll
    {
        Id = u.Id,
        UserId = u.UserId,
        Name = u.Name,
        Surname = u.Surname
    });
    return Ok(dtoUsers);
}

[Authorize]
[HttpGet("{JWTId}")]
1 referencia
public async Task<IActionResult> GetById(string JWTid)
{
    var user = await _users.Find(u => u.UserId == JWTid).FirstOrDefaultAsync();
    if (user == null)
        return NotFound(new { message = "User not found." });

    var jwtUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (jwtUserId != user.UserId)
        return Forbid();

    var dtoUser = new DTOUserRead
    {
        Id = user.Id,
        UserId = user.UserId,
        Name = user.Name,
        Surname = user.Surname,
        dni = user.dni
    };
    return Ok(dtoUser);
}
```

4. Borrar Usuario

Solo puede borrarse un usuario a sí mismo.

```
[Authorize]
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> Delete(string id)
{
    if (!IsAuthorized(out var unauthorizedResult) && User.FindFirst(ClaimTypes.NameIdentifier)?.Value == id)
        return unauthorizedResult;

    var result = await _users.DeleteOneAsync(u => u.Id == id);
    return result.DeletedCount > 0 ? NoContent() : NotFound();
}
```

Controlador Maintenance Order

En este controlador podemos iniciar una orden de mantenimiento para un vehículo y unos componentes.

```
4 referencias
private bool IsAuthorized(out IActionResult unauthorizedResult)
{
    var userType = User.FindFirst("UserType")?.Value;

    if (userType != "2")
    {
        unauthorizedResult = Unauthorized(new { message = "Only workshops are allowed." });
        return false;
    }

    unauthorizedResult = null!;
    return true;
}
```

Solo talleres pueden acceder a este controlador

1. Crear Orden de mantenimiento

Te permite crear una orden de mantenimiento asociada al taller que la crea

```
[Authorize]
[HttpPost]
0 referencias
public async Task<IActionResult> Create(DTOMaintenanceOrderPost dtoOrder)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var workshopId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0";

    var order = new MaintenanceOrder
    {
        WorkshopId = workshopId,
        VehicleId = dtoOrder.VehicleId,
        ComponentId = dtoOrder.ComponentIds,
        Date = dtoOrder.Date,
        Description = dtoOrder.Description
    };

    await _orders.InsertOneAsync(order);
    return CreatedAtAction(nameof(GetFiltered), null, order);
}
```

2. Ver Orden de mantenimiento

Te permite acceder a las órdenes de mantenimiento filtrando por taller, vehículo, componente y fecha, el dto de getbyid y de getall es el mismo.

```
[Authorize]
[HttpGet]
0 referencias
public async Task<IActionResult> GetFiltered(
    [FromQuery] string? vehicle,
    [FromQuery] string? component,
    [FromQuery] DateTime? startDate,
    [FromQuery] DateTime? endDate)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var workshopId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0";

    var filter = Builders<MaintenanceOrder>.Filter.Eq(o => o.WorkshopId, workshopId);

    if (vehicle != null)
        filter &= Builders<MaintenanceOrder>.Filter.Eq(o => o.VehicleId, vehicle);

    if (component != null)
        filter &= Builders<MaintenanceOrder>.Filter.AnyEq(o => o.ComponentId, component);

    if (startDate.HasValue)
        filter &= Builders<MaintenanceOrder>.Filter.Gte(o => o.Date, startDate.Value);

    if (endDate.HasValue)
        filter &= Builders<MaintenanceOrder>.Filter.Lte(o => o.Date, endDate.Value);

    var results = await _orders.Find(filter).ToListAsync();
    var dtoResults = results.Select(o => new DTOMaintenanceOrderRead
    {
        Id = o.Id,
        WorkshopId = o.WorkshopId,
        VehicleId = o.VehicleId,
        ComponentIds = o.ComponentId,
        Date = o.Date ?? DateTime.Now,
        Description = o.Description
    });
}

return Ok(dtoResults);
}

[Authorize]
[HttpGet("{id}")]
0 referencias
public async Task<IActionResult> GetById(string id)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var workshopId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0";

    var order = await _orders.Find(o => o.Id == id && o.WorkshopId == workshopId).FirstOrDefaultAsync();

    if (order == null)
        return NotFound(new { message = "Maintenance order not found." });

    var dtoOrder = new DTOMaintenanceOrderRead
    {
        Id = order.Id,
        WorkshopId = order.WorkshopId,
        VehicleId = order.VehicleId,
        ComponentIds = order.ComponentId,
        Date = order.Date ?? DateTime.Now,
        Description = order.Description
    };

    return Ok(dtoOrder);
}
```

3. Eliminar Orden de mantenimiento

Te permite eliminar una orden de mantenimiento.

```
[Authorize]
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> Delete(string id)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var result = await _orders.DeleteOneAsync(o => o.Id == id);

    if (result.DeletedCount == 0)
        return NotFound(new { message = "Maintenance order not found." });

    return NoContent();
}
```

Controlador Invoice

En este controlador podemos crear facturas a clientes asociadas a una orden de mantenimiento. Ni este controlador ni el de la orden de mantenimiento permiten la modificación, de esta forma aseguramos que los datos no sean modificados malintencionadamente. En caso de errores se debe crear una nueva factura.

```
4 referencias
private bool IsAuthorized(out IActionResult unauthorizedResult)
{
    var userType = User.FindFirst("UserType")?.Value;

    if (userType != "2")
    {
        unauthorizedResult = Unauthorized(new { message = "Only workshops are allowed." });
        return false;
    }

    unauthorizedResult = null!;
    return true;
}
```

Estos endpoints solo son accesibles por talleres por lo que se comprueba que el usertype en el JWT sea 2.

1. Crear facturas

Este endpoint le permite al taller crear facturas incluyendo la orden de mantenimiento correspondiente, el cliente a la que va dirigida, el monto total y la fecha

```
[Authorize]
[HttpPost]
0 referencias
public async Task<IActionResult> Create(DTOInvoicePost dtoInvoice)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;
    var workshopId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0";

    var invoice = new Invoice
    {
        ClientId = dtoInvoice.ClientId,
        WorkshopId = workshopId,
        MaintenanceId = dtoInvoice.MaintenanceId,
        Total = dtoInvoice.Total,
        Date = dtoInvoice.Date
    };

    await _invoices.InsertOneAsync(invoice);
    return CreatedAtAction(nameof(GetAll), null, invoice);
}
```

class System.Represents

2. Ver facturas

Este endpoint le permite a un taller ver sus facturas y solo sus facturas, el getall incluye filtros por cliente, orden de mantenimiento y fecha.

```
[Authorize]
[HttpGet]
0 referencias
public async Task<IActionResult> GetAll([FromQuery] string? clientId, [FromQuery] string? maintenanceId, [FromQuery] DateTime? startDate, [FromQuery] DateTime? endDate)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var workshopId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0";

    var filterBuilder = Builders<Invoice>.Filter;
    var filter = filterBuilder.Eq(i => i.WorkshopId, workshopId);

    if (!string.IsNullOrEmpty(clientId))
        filter &= filterBuilder.Eq(i => i.ClientId, clientId);

    if (!string.IsNullOrEmpty(maintenanceId))
        filter &= filterBuilder.Eq(i => i.MaintenanceId, maintenanceId);

    if (startDate.HasValue)
        filter &= filterBuilder.Gte(i => i.Date, startDate.Value);

    if (endDate.HasValue)
        filter &= filterBuilder.Lte(i => i.Date, endDate.Value);

    var invoices = await _invoices.Find(filter).ToListAsync();
    var dtoInvoices = invoices.Select(i => new DTOInvoiceRead
    {
        Id = i.Id,
        ClientId = i.ClientId,
        WorkshopId = i.WorkshopId,
        MaintenanceId = i.MaintenanceId,
        Total = i.Total,
        Date = i.Date
    });
}

return Ok(dtoInvoices);
}
[Authorize]
[HttpGet("{id}")]
0 referencias
public async Task<IActionResult> GetById(string id)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var workshopId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0";

    var invoice = await _invoices.Find(i => i.Id == id && i.WorkshopId == workshopId).FirstOrDefaultAsync();

    if (invoice == null)
        return NotFound(new { message = "Invoice not found." });

    var dtoInvoice = new DTOInvoiceRead
    {
        Id = invoice.Id,
        ClientId = invoice.ClientId,
        WorkshopId = invoice.WorkshopId,
        MaintenanceId = invoice.MaintenanceId,
        Total = invoice.Total,
        Date = invoice.Date
    };

    return Ok(dtoInvoice);
}
```

3. Eliminar facturas

Este endpoint le permite a un taller borrar una de sus facturas

```
[Authorize]
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> Delete(string id)
{
    if (!IsAuthorized(out var unauthorizedResult))
        return unauthorizedResult;

    var workshopId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0";

    var invoice = await _invoices.Find(i => i.Id == id && i.WorkshopId == workshopId).FirstOrDefaultAsync();

    if (invoice == null)
        return NotFound(new { message = "Invoice not found." });

    var result = await _invoices.DeleteOneAsync(i => i.Id == id);

    if (result.DeletedCount == 0)
        return NotFound(new { message = "Invoice not found." });

    return NoContent();
}
```

2.3.6. Backend – Api de mensajería

Microservicio API RESTful que permite la mensajería en tiempo real gracias a la notificación de websockets mediante SignalR. El cliente se conectará al websocket del servidor y será notificado de mensajes nuevos, mensajes editados, mensajes eliminados, chats nuevos o eliminados, etc. Al igual que la mayoría de los microservicios hace uso de una base de datos PostgreSQL y utiliza el framework ASP.NET

SignalR

A la hora de implementar SignalR nos encontramos con dos opciones:

- Enviar los mensajes nuevos como notificación al SignalR y que este se encargue de crearlos en la base de datos.
- Crear los mensajes con métodos post http y que dentro de estos métodos se envíe la notificación una vez se ha añadido el mensaje a la base de datos.

Optamos por la segunda opción, de tal forma que cada endpoint tiene una llamada a SignalR notificando a las conexiones correspondientes.

Para implementar SignalR primero hay que descargar las dependencias y luego hay que añadirlo a la aplicación en el main:

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddSignalR();
var app = builder.Build();
app.MapHub<MessageHub>("/hubs/messages");
```

El método MapHub nos permite indicar en qué url se conectarán los clientes al websocket, pese a su apariencia de endpoint realmente es un websocket. En caso de que el contenedor se levante en localhost el cliente debería conectarse a localhost:{puerto}/hubs/messages

4. Chats

Es necesario configurar las notificaciones que luego se llamarán al crear, eliminar o modificar chats:

```
0 referencias
public async Task NotifyChatUpdated(ChatReadDTO chat)
{
    var chatId = chat.Id.ToString();
    if (!string.IsNullOrEmpty(chatId))
    {
        await Clients.Group(chatId).SendAsync("ChatUpdated", chat);
    }
}

0 referencias
public async Task NotifyChatDeleted(int chatId)
{
    await Clients.Group(chatId.ToString()).SendAsync("ChatDeleted", chatId);
}
```

En caso de notificar al borrar o modificar un chat vemos que se notifica a un “grupo”, este grupo está compuesto por todos los integrantes de este chat.

```
0 referencias
public async Task JoinChat(string chatId)
{
    await Groups.AddToGroupAsync(Context.ConnectionId, chatId);
    await Clients.Group(chatId).SendAsync("UserJoined", $"{Context.ConnectionId} joined the chat {chatId}");
}

0 referencias
public async Task LeaveChat(string chatId)
{
    await Groups.RemoveFromGroupAsync(Context.ConnectionId, chatId);
    await Clients.Group(chatId).SendAsync("UserLeft", $"{Context.ConnectionId} left chat {chatId}");
}
```

Al unirse una persona nueva a un chat o al abandonarlo se unirá la conexión de websocket al grupo del chat y luego se notificará al resto de integrantes.

5. Mensajes

Configuración al crear, modificar eliminar mensajes:

```
0 referencias
public async Task NotifyMessageCreated(MessageReadDTO message)
{
    var chatId = message.ChatId.ToString();
    if (!string.IsNullOrEmpty(chatId))
    {
        await Clients.Group(chatId).SendAsync("ReceiveMessage", message);
    }
}

0 referencias
public async Task NotifyMessageUpdated(MessageReadDTO message)
{
    var chatId = message.ChatId.ToString();
    if (!string.IsNullOrEmpty(chatId))
    {
        await Clients.Group(chatId).SendAsync("MessageUpdated", message);
    }
}

0 referencias
public async Task NotifyMessageDeleted(int chatId, int messageId)
{
    await Clients.Group(chatId.ToString()).SendAsync("MessageDeleted", messageId);
}
```

De nuevo, se notifica sobre estos mensajes únicamente a los que pertenecen al chat

Controlador Chats

1. Crear chats

En este endpoint podemos crear un chat, lo que también creará una relación del usuario creador con el chat y notificará de la creación del chat.

```
[Authorize]
[HttpPost]
0 referencias
public async Task<ActionResult<ChatReadDTO>> CreateChat(ChatCreateDTO dto)
{
    var chat = new Chat
    {
        CreatorId = int.TryParse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value, out var creatorId) ? creatorId : 0,
        Name = dto.Name,
        CreatedAt = DateTime.UtcNow
    };

    _context.Chats.Add(chat);
    _context.SaveChanges();

    var chatReadDTO = new ChatReadDTO
    {
        Id = chat.Id,
        CreatorId = chat.CreatorId,
        Name = chat.Name,
        CreatedAt = chat.CreatedAt
    };

    await _hubContext.Clients.Group(chat.Id.ToString()).SendAsync("ChatCreated", chatReadDTO);

    var userChat = new User_Chat
    {
        UserId = chat.CreatorId,
        ChatId = chat.Id
    };

    _context.UserChats.Add(userChat);
    _context.SaveChanges();

    await _hubContext.Clients.Group(chat.Id.ToString())
        .SendAsync("UserJoined", new { UserId = chat.CreatorId, ChatId = chat.Id });

    return CreatedAtAction(nameof(GetChat), new { id = chat.Id }, chatReadDTO);
}
```

2. Ver chats

A la hora de ver los chats podemos acceder a todos ellos a los que pertenece el usuario que realiza la petición, con filtro de creador, nombre y fecha, o bien podemos acceder a un chat específico (si el usuario pertenece a ese chat)

```
[Authorize]
[HttpGet]
0 referencias
public ActionResult<IEnumerable<ChatReadDTO>> GetChats(
    [FromQuery] int? creatorId,
    [FromQuery] string? name,
    [FromQuery] DateTime? startDate,
    [FromQuery] DateTime? endDate)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    // Obtener los ChatId a los que pertenece el usuario
    var userChatIds = _context.UserChats
        .Where(uc => uc.UserId == userId)
        .Select(uc => uc.ChatId)
        .ToList();

    var query = _context.Chats
        .Where(c => userChatIds.Contains(c.Id))
        .AsQueryable();

    if (creatorId.HasValue)
        query = query.Where(c => c.CreatorId == creatorId.Value);

    if (!string.IsNullOrEmpty(name))
        query = query.Where(c => c.Name.Contains(name));

    if (startDate.HasValue)
        query = query.Where(c => c.CreatedAt >= startDate.Value);

    if (endDate.HasValue)
        query = query.Where(c => c.CreatedAt <= endDate.Value);

    var chats = query
        .Select(c => new ChatReadDTO
    {
        Id = c.Id,
        CreatorId = c.CreatorId,
        Name = c.Name,
        CreatedAt = c.CreatedAt
    })
        .ToList();

    return Ok(chats);
}
```

Para comprobar todos los chats a los que pertenece el usuario debemos realizar una consulta en el controlador UserChats, donde están las relaciones entre usuarios y chats, una vez recogidos todos los chats a los que pertenece el usuario se mostrará una lista de ChatReadDTO.

```
[Authorize]
[HttpGet("{id}")]
1 referencia
public ActionResult<ChatReadDTO> GetChat(int id)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    // Verifica si el usuario pertenece al chat
    var isMember = _context.UserChats.Any(uc => uc.UserId == userId && uc.ChatId == id);
    if (!isMember)
        return Forbid();

    var chat = _context.Chats
        .Where(c => c.Id == id)
        .Select(c => new ChatReadDTO
    {
        Id = c.Id,
        CreatorId = c.CreatorId,
        Name = c.Name,
        CreatedAt = c.CreatedAt
    })
        .FirstOrDefault();

    if (chat == null)
        return NotFound();

    return Ok(chat);
}
```

Para acceder a un chat específico se comprueba si ese usuario pertenece al chat y devuelve forbid en caso de que no pertenezca.

3. Modificar chats

Te permite modificar un chat únicamente si eres el creador de este y notifica de su modificación

```
[Authorize]
[HttpPut("{id}")]
0 referencias
public async Task<IActionResult> UpdateChat(int id, ChatUpdateDTO dto)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    var chat = _context.Chats.Find(id);
    if (chat == null)
        return NotFound();

    if (chat.CreatorId != userId)
        return Forbid();

    chat.Name = dto.Name;
    _context.SaveChanges();

    var chatReadDTO = new ChatReadDTO
    {
        Id = chat.Id,
        CreatorId = chat.CreatorId,
        Name = chat.Name,
        CreatedAt = chat.CreatedAt
    };

    await _hubContext.Clients.Group(chat.Id.ToString()).SendAsync("ChatUpdated", chatReadDTO);

    return NoContent();
}
```

4. Eliminar chats

Te permite eliminar un chat del que seas creador, notificándolo al resto de integrantes del chat.

```
[Authorize]
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> DeleteChat(int id)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    var chat = _context.Chats.Find(id);
    if (chat == null)
        return NotFound();

    if (chat.CreatorId != userId)
        return Forbid();

    _context.Chats.Remove(chat);
    _context.SaveChanges();

    await _hubContext.Clients.Group(id.ToString()).SendAsync("ChatDeleted", id);

    return NoContent();
}
```

Controlador Messages

Este controlador permite la mensajería en tiempo real dentro de un chat

1. Crear mensajes

Añade un mensaje al chat indicado si pertenece a el y notifica al grupo

```
[Authorize]
[HttpPost]
0 referencias
public async Task<ActionResult<MessageReadDTO>> CreateMessage(MessageCreateDTO dto)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    var isMember = _context.UserChats.Any(uc => uc.UserId == userId && uc.ChatId == dto.ChatId);
    if (!isMember)
        return Forbid();

    var message = new Message
    {
        ChatId = dto.ChatId,
        SenderId = userId,
        Content = dto.Content,
        SentAt = DateTime.UtcNow
    };

    _context.Messages.Add(message);
    _context.SaveChanges();

    var messageReadDTO = new MessageReadDTO
    {
        Id = message.Id,
        ChatId = message.ChatId,
        SenderId = message.SenderId,
        Content = message.Content,
        SentAt = message.SentAt
    };

    await _hubContext.Clients.Group(message.ChatId.ToString()).SendAsync("ReceiveMessage", messageReadDTO);

    return CreatedAtAction(nameof(GetMessage), new { id = message.Id }, messageReadDTO);
}
```

2. Ver mensajes

Te permite ver los mensajes enviados a los chats a los que pertenece el usuario, el getall tiene filtro por chat, por sender y por fecha.

```
[Authorize]
[HttpGet]
0 referencias
public ActionResult<IEnumerable<MessageReadDTO>> GetMessages(
    [FromQuery] int? chatId,
    [FromQuery] int? senderId,
    [FromQuery] DateTime? startDate,
    [FromQuery] DateTime? endDate)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    var userChatIds = _context.UserChats
        .Where(uc => uc.UserId == userId)
        .Select(uc => uc.ChatId)
        .ToList();

    var query = _context.Messages
        .Where(m => userChatIds.Contains(m.ChatId))
        .AsQueryable();

    if (chatId.HasValue)
        query = query.Where(m => m.ChatId == chatId.Value);

    if (senderId.HasValue)
        query = query.Where(m => m.SenderId == senderId.Value);

    if (startDate.HasValue)
        query = query.Where(m => m.SentAt >= startDate.Value);

    if (endDate.HasValue)
        query = query.Where(m => m.SentAt <= endDate.Value);

    var messages = query
        .Select(m => new MessageReadDTO
    {
        Id = m.Id,
        ChatId = m.ChatId,
        SenderId = m.SenderId,
        Content = m.Content,
        SentAt = m.SentAt
    })
        .ToList();
    return Ok(messages);
}

[Authorize]
[HttpGet("{id}")]
1 referencia
public ActionResult<MessageReadDTO> GetMessage(int id)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    var userChatIds = _context.UserChats
        .Where(uc => uc.UserId == userId)
        .Select(uc => uc.ChatId)
        .ToList();

    var message = _context.Messages
        .Where(m => m.Id == id && userChatIds.Contains(m.ChatId))
        .Select(m => new MessageReadDTO
    {
        Id = m.Id,
        ChatId = m.ChatId,
        SenderId = m.SenderId,
        Content = m.Content,
        SentAt = m.SentAt
    })
        .FirstOrDefault();
    if (message == null)
        return NotFound();
    return Ok(message);
}
```

3. Modificar mensajes

Te permite modificar un mensaje si el usuario que realiza la petición es el que lo ha enviado

```
[Authorize]
[HttpGet("{id}")]
1 referencia
public ActionResult<MessageReadDTO> GetMessage(int id)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    var userChatIds = _context.UserChats
        .Where(uc => uc.UserId == userId)
        .Select(uc => uc.ChatId)
        .ToList();

    var message = _context.Messages
        .Where(m => m.Id == id && userChatIds.Contains(m.ChatId))
        .Select(m => new MessageReadDTO
    {
        Id = m.Id,
        ChatId = m.ChatId,
        SenderId = m.SenderId,
        Content = m.Content,
        SentAt = m.SentAt
    })
        .FirstOrDefault();
    if (message == null)
        return NotFound();
    return Ok(message);
}
```

4. Borrar mensajes

Permite al usuario borrar un mensaje si es el que lo ha enviado

```
[Authorize]
[HttpDelete("{id}")]
0 referencias
public async Task<IActionResult> DeleteMessage(int id)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    var message = _context.Messages.Find(id);
    if (message == null)
        return NotFound();

    if (message.SenderId != userId)
        return Forbid();

    int chatId = message.ChatId;

    _context.Messages.Remove(message);
    _context.SaveChanges();

    await _hubContext.Clients.Group(chatId.ToString()).SendAsync("MessageDeleted", id);

    return NoContent();
}
```

Controlador UserChat

Este controlador te permite administrar las relaciones entre usuarios y chats, añadiendo usuarios a chats o eliminándolos

1. Ver usuarios en chats

Te permite ver el chat y los usuarios que pertenecen al mismo, puede filtrar por chatId o chatName y solo saldrán los chats a los que pertenezca el usuario.

```
[Authorize]
[HttpGet]
0 referencias
public ActionResult<IEnumerable<UserChatReadDTO>> GetUserChats(
    [FromQuery] int? chatId,
    [FromQuery] string? chatName)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();

    var query = _context.UserChats
        .Include(uc => uc.Chat)
        .Where(uc => uc.UserId == userId);

    if (chatId.HasValue)
        query = query.Where(uc => uc.ChatId == chatId.Value);

    if (!string.IsNullOrEmpty(chatName))
        query = query.Where(uc => uc.Chat.Name.Contains(chatName));

    var userChats = query
        .Select(uc => new UserChatReadDTO
        {
            UserId = uc.UserId,
            ChatId = uc.ChatId,
            Chat = new ChatReadDTO
            {
                Id = uc.Chat.Id,
                CreatorId = uc.Chat.CreatorId,
                Name = uc.Chat.Name,
                CreatedAt = uc.Chat.CreatedAt
            },
            Users = _context.UserChats
                .Where(ucc => ucc.ChatId == uc.ChatId)
                .Select(ucc => new UserInChatDTO { UserId = ucc.UserId })
                .ToList()
        })
        .ToList();

    return Ok(userChats);
}
```

2. Añadir usuario a chat

Te permite crear una nueva relación entre usuarios y chats.

```
[Authorize]
[HttpPost]
0 referencias
public async Task<IActionResult> CreateUserChat(UserChatCreateDTO dto)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var userId) || userId == 0)
        return Unauthorized();
    var userChat = new User_Chat
    {
        UserId = dto.UserId,
        ChatId = dto.ChatId
    };

    _context.UserChats.Add(userChat);
    await _context.SaveChangesAsync();

    await _hubContext.Clients.Group(dto.ChatId.ToString())
        .SendAsync("UserJoined", new { UserId = dto.UserId, ChatId = dto.ChatId });

    return NoContent();
}
```

3. Eliminar usuario de chat

Te permite eliminar a un usuario de un chat si eres el creador del chat.

```
[Authorize]
[HttpDelete]
0 referencias
public async Task<IActionResult> DeleteUserChat(int userId, int chatId)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!int.TryParse(userIdClaim, out var currentUserId) || currentUserId == 0)
        return Unauthorized();

    var userChat = _context.UserChats
        .Include(uc => uc.Chat)
        .FirstOrDefault(uc => uc.UserId == userId && uc.ChatId == chatId);

    if (userChat == null)
        return NotFound();

    if (currentUserId != userChat.UserId && currentUserId != userChat.Chat.CreatorId)
        return Forbid();

    await _hubContext.Clients.Group(chatId.ToString())
        .SendAsync("UserLeft", new { UserId = userId, ChatId = chatId });

    _context.UserChats.Remove(userChat);
    _context.SaveChanges();

    return NoContent();
}
```

2.3.7. Backend – Api Gateway

Este microservicio se encarga de unir al resto de microservicios para que puedan funcionar como uno solo. Todas las peticiones de los clientes pasarán a través de este y permitirá controlar el volumen y origen de las peticiones.

En cuanto a la configuración inicial es necesario instalar la dependencia de Ocelot y configurarlo en el Program.cs

```
var builder = WebApplication.CreateBuilder(args);

builder.Configuration
    .SetBasePath(builder.Environment.ContentRootPath)
    .AddJsonFile("ocelot.json", optional: false, reloadOnChange: true)
    .AddEnvironmentVariables();

builder.Services.AddSwaggerForOcelot(builder.Configuration);
builder.Services.AddOcelot();
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowLocalhost", policy =>
    {
        policy.WithOrigins("http://vms.iesluisvives.org:25002")
            .AllowAnyMethod()
            .AllowAnyHeader()
            .AllowCredentials();
    });
});

builder.Services.AddMvcCore().AddApiExplorer();
var app = builder.Build();
app.UseCors("AllowLocalhost");
app.UseWebSockets();
app.UseSwaggerForOcelotUI(options =>
{
    options.PathToSwaggerGenerator = "/swagger/docs";
});

await app.UseOcelot();
```

Para la correcta integración de Ocelot con un cliente web es necesario añadir y configurar CORS, indicando la url del cliente web. También para la notificación de websockets mediante SignalR es necesario indicar la utilización de Websockets.

Toda la configuración de los endpoints se hace a través de un archivo json llamado Ocelot.json, en este se añadirá el endpoint original del microservicio, el endpoint que sale en el Gateway, el método http asociado al endpoint y la dirección con puerto del endpoint original.

```
{  
    "UpstreamHttpMethod": [ "Get", "Post" ],  
    "UpstreamPathTemplate": "/api/messages",  
    "DownstreamPathTemplate": "/api/messages",  
    "DownstreamScheme": "http",  
    "DownstreamHostAndPorts": [  
        {  
            "Host": "webapimessages",  
            "Port": 8080  
        }  
    ],  
    "SwaggerKey": "messages"  
},
```

También es necesario configurar la url desde la que accederemos al Gateway, en este caso:

```
"GlobalConfiguration": {  
    "BaseUrl": "http://vms.iesluisvives.org:25003"  
}
```

Para que el Gateway tenga documentación swagger es necesario instalar la extensión MMLib.SwaggerForOcelot, que se encargará de coger los swagger de cada microservicio y unirlos en uno mismo.

```
{  
    "Key": "workshop",  
    "TransformByOcelotConfig": true,  
    "Config": [  
        {  
            "Name": "workshop",  
            "Version": "v1",  
            "Url": "http://workshop_api:8080/swagger/v1/swagger.json"  
        }  
    ]  
}
```

2.3.8. Frontend Web:

El Frontend se programó usando React, JavaScript y TailwindCSS. El Frontend contiene gran cantidad de archivos, métodos y componentes, sin embargo, lo más relevante es:

Main.jsx

```
async function initializeApp() {
  try {
    await loadConfig();
    ReactDOM.createRoot(document.getElementById('root')).render(
      <StrictMode>
        <AuthProvider>
          <SignalRProvider>
            <BrowserRouter>
              <AppRoutes />
            </BrowserRouter>
          </SignalRProvider>
        </AuthProvider>
      </StrictMode>
    );
  } catch (error) {
    console.error("Failed to load configuration:", error);
  }
}
```

Debido a que la navegación web permite acceso a todas las páginas que componen la aplicación fue necesaria proveerla de un contexto de autorización. Además, para controlar la escucha de eventos se usó otro contexto, el de SignalR.

AuthProvider

El AuthProvider contiene 2 métodos importantes que manejan la sesión. Uno guarda la información principal al hacer el login, y se recupera en las distintas peticiones al backend mientras que el otro (logout) elimina la información de sesión.

```
const login = async (token) => {
  const payload = parseJwt(token);
  const userId = payload?.nameid ? parseInt(payload.nameid, 10) : null;
  const userType = payload?.UserType ? parseInt(payload.UserType, 10) : null;

  console.log('[login] Payload:', payload);
  console.log('[login] userId:', userId);
  console.log('[login] userType:', userType);

  if (userId) {
    sessionStorage.setItem('token', token);
    sessionStorage.setItem('userId', userId.toString());
    if (userType !== null) {
      sessionStorage.setItem('userType', userType.toString());
      console.log('[login] Guardado userType en sessionStorage:', userType);
    }
  }

  setAuthState({ status: 'authenticated', userId, token, hasProfile: null, userType });

  const exists = await checkUserProfile(userId, token);
  setAuthState((prev) => ({ ...prev, hasProfile: exists }));
} else {
  console.warn('[login] No se pudo obtener el userId desde el token');
  setAuthState({ status: 'unauthenticated', userId: null, token: null, hasProfile: null, userType: null });
}
};

const logout = () => {
  console.log('[logout] Cerrando sesión');
  sessionStorage.removeItem('token');
  sessionStorage.removeItem('userId');
  sessionStorage.removeItem('userType');
  setAuthState({ status: 'unauthenticated', userId: null, token: null, hasProfile: null, userType: null });
};
```

PrivateRoute

Este componente se encarga de proteger las rutas privadas y obliga al usuario a generar un perfil antes de poder acceder a ella. Está presente en todas las rutas privadas.

```
import { useContext, useState, useEffect } from 'react';
import { AuthContext } from '../context/AuthContext';
import { Navigate, useLocation } from 'react-router-dom';
import CreateProfileModalContainer from '../Container/ProfileModal';

export function PrivateRoute({ children }) {
  const { status, hasProfile } = useContext(AuthContext);
  const location = useLocation();
  const [modalOpen, setModalOpen] = useState(false);

  useEffect(() => {
    if (status === 'authenticated' && hasProfile === false) {
      setModalOpen(true);
    } else {
      setModalOpen(false);
    }
  }, [status, hasProfile]);

  if (status === 'loading' || hasProfile === undefined) {
    return <div>Cargando...</div>;
  }

  if (status !== 'authenticated') {
    return <Navigate to="/login" replace state={{ from: location }} />;
  }

  return (
    <>
      {modalOpen && (
        <CreateProfileModalContainer onClose={() => setModalOpen(false)} />
      )}
      {!modalOpen && children}
    </>
  );
}
```

Este componente protege la página entera y la bloquea con un modal de creación de perfil hasta que te crees uno.

SignalRProvider

Este context se encarga de mantener la conexión a los eventos de SignalR. Contiene 3 métodos useEffect, que ajustarán el contexto al usuario y les subscribirá a todos los eventos para que el servicio de chats ejecute los cambios en vivo.

1. useEffect de conexión.

Este useEffect se encarga de inicializar la conexión con SignalR saltando la negociación y usando websockets

```
useEffect(() => {
  if (!token) return;

  const connect = async () => {
    const hubConnection = new signalR.HubConnectionBuilder()
      .withUrl(getSignalRHubUrl(), {
        skipNegotiation: true,
        transport: signalR.HttpTransportType.WebSockets,
        accessTokenFactory: () => token,
      })
      .withAutomaticReconnect()
      .configureLogging(signalR.LogLevel.Information)
      .build();

    hubConnection.onreconnecting((error) => {
      console.warn(" Reconnectando a SignalR...", error);
    });

    hubConnection.onreconnected(() => {
      console.log(" Reconectado a SignalR");
      joinedChatsRef.current.forEach(chatId => {
        hubConnection.invoke("JoinChat", chatId.toString()).catch(console.error);
      });
    });

    try {
      await hubConnection.start();
      console.log(" SignalR conectado");
      setConnection(hubConnection);
    } catch (err) {
      console.error(" Error al conectar SignalR:", err);
    }
  };

  connect();
}

return () => {
  if (connection) connection.stop();
}, [token]);
```

2. useEffect de carga de chats.

Cuando se produce un cambio de token o de conexión, se produce una nueva carga de chats. Esto es necesario para que cada vez que inicie sesión un usuario o se reinicie la conexión se actualicen los chats en caso de que haya nuevos.

```
useEffect(() => {
  if (!token || !connection) return;
  fetchChats();
}, [token, connection]);
```

3. useEffect de configuración de eventos.

Este endpoint se encarga de suscribir y eliminar suscripciones la de la conexión actual a los eventos de SignalR programados. Y añade las acciones que se deben realizar por cada evento.

```
connection.on("ReceiveMessage", handleReceiveMessage);
connection.on("MessageUpdated", handleMessageUpdated);
connection.on("MessageDeleted", handleMessageDeleted);
connection.on("ChatUpdated", handleChatUpdated);
connection.on("ChatDeleted", handleChatDeleted);
connection.on("UserJoined", handleUserJoinedChat);
connection.on("UserLeft", handleUserLeft);

return () => {
  connection.off("ReceiveMessage", handleReceiveMessage);
  connection.off("MessageUpdated", handleMessageUpdated);
  connection.off("MessageDeleted", handleMessageDeleted);
  connection.off("ChatUpdated", handleChatUpdated);
  connection.off("ChatDeleted", handleChatDeleted);
  connection.off("UserJoined", handleUserJoinedChat);
  connection.off("UserLeft", handleUserLeft);
};

}, [connection, userId, token, onMessageUpdated, onMessageDeleted]);
```

Routing

Las rutas de la aplicación se dividieron en rutas privadas y públicas:

```
export default function PublicRoutes() {
  return (
    <Routes>
      <Route path="/login" element={<LoginPage />} />
      <Route path="/register" element={<RegisterPage />} />
      <Route path="/confirm-email/:token" element={<ConfirmEmailPage />} />
      <Route path="/reset-password" element={<ForgotPasswordPage />} />
      <Route path="/pass-change/:token" element={<ResetPasswordPage />} />
      <Route path="/" element={<Navigate to="/login" replace />} />
    </Routes>
  );
}

export default function PrivateRoutes() {
  return (
    <Routes>
      <Route
        path="/dashboard"
        element={
          <PrivateRoute>
            <DashboardPage />
          </PrivateRoute>
        }
      />
      <Route
        path="/profile/:id"
        element={
          <PrivateRoute>
            <ProfilePage />
          </PrivateRoute>
        }
      />
    </Routes>
  );
}
```

Cuando se intentaba entrar a las rutas privadas, actuaba un componente de protección que si detectaba un acceso no autorizado te mandaba de vuelta a la página de Login. Así se evita un uso indebido de la aplicación

2.3.9. Frontend Móvil:

Nuestra aplicación móvil, construida con **Android Studio**, **Jetpack Compose** y **Kotlin**, implementa el patrón **Modelo-Vista-VistaModelo (MVVM)**. Al integrar **Hilt**, una solución de inyección de dependencias basada en Dagger, optimizamos la arquitectura MVVM. Este enfoque no solo minimizó el código repetitivo, sino que también permitió compartir sin problemas repositorios, Viewmodels y otros componentes cruciales, dando como resultado una aplicación más robusta y un código significativamente más limpio.

Seguridad

Para acceder a los recursos de la aplicación hay que estar autenticado y autorizado con el servidor mediante un token JWT válido. Para la autenticación es necesario logarse en la aplicación, si el usuario es válido el servidor devolverá el JWT correspondiente, este token lo maneja el SessionManager, que permite mantener la sesión iniciada siempre y cuando el token siga siendo válido.

```
class SessionManager @Inject constructor(
    @ApplicationContext private val context: Context
) {
    companion object {
        private val JWT_TOKEN = stringPreferencesKey("jwt_token")
    }

    suspend fun saveToken(token: String) {
        context.dataStore.edit { prefs ->
            prefs[JWT_TOKEN] = token
        }
    }

    val token: Flow<String?> = context.dataStore.data
        .map { prefs -> prefs[JWT_TOKEN] }

    suspend fun clearToken() {
        context.dataStore.edit { prefs ->
            prefs.remove(JWT_TOKEN)
        }
    }

    private fun decodeJwt(token: String): Triple<String, String, Int>? {
        return try {
            val parts = token.split(".")
            if (parts.size < 2) return null
            val payloadJson = String(
                Base64.decode(parts[1], Base64.URL_SAFE or Base64.NO_WRAP
or Base64.NO_PADDING)
            )
            val payload = JSONObject(payloadJson)

            val userId = payload.getString("nameid")
            val email = payload.getString("email")
            val userType = payload.getInt("UserType")

            Triple(userId, email, userType)
        } catch (e: Exception) {
            null
        }
    }
}
```

Autenticación

Para hacer las peticiones al servidor es necesario incluir el token en las mismas, de esto se encarga AuthInterceptor.

```
class AuthInterceptor @Inject constructor(private val sessionManager: SessionManager) : Interceptor {
    override fun intercept(chain: Interceptor.Chain): Response {
        val original = chain.request()
        val token = runBlocking { sessionManager.token.firstOrNull() }
        val request = if (token != null) {
            original.newBuilder()
                .addHeader("Authorization", "Bearer $token")
                .build()
        } else original
        return chain.proceed(request)
    }
}
```

Api RESTful

Y de las peticiones se encarga Retrofit

```
@Module
@InstallIn(SingletonComponent::class)
object RetrofitInstance {

    private const val BASE_URL = "http://vms.iesluisvives.org:25003/"

    @Provides
    @Singleton
    fun provideLoggingInterceptor(): HttpLoggingInterceptor {
        return HttpLoggingInterceptor().apply {
            level = HttpLoggingInterceptor.Level.BODY
        }
    }

    @Provides
    @Singleton
    fun provideAuthInterceptor(sessionManager: SessionManager): AuthInterceptor {
        return AuthInterceptor(sessionManager)
    }

    @Provides
    @Singleton
    fun provideOkHttpClient(
        logging: HttpLoggingInterceptor,
        authInterceptor: AuthInterceptor
    ): OkHttpClient {
        return OkHttpClient.Builder()
            .addInterceptor(logging)
            .addInterceptor(authInterceptor)
            .connectTimeout(15, TimeUnit.SECONDS)
            .readTimeout(20, TimeUnit.SECONDS)
            .build()
    }
}
```

```
@Provides
@Singleton
fun provideRetrofit(client: OkHttpClient): Retrofit {
    return Retrofit.Builder()
        .baseUrl(BASE_URL)
        .client(client)
        .addConverterFactory(GsonConverterFactory.create()).build()
}
```

Websockets

Para la notificación mediante SignalR se ha utilizado la dependencia de Microsoft SignalR, creando la conexión con el hub correspondiente del back:

```
private lateinit var hubConnection: HubConnection

fun connect(
    chatId: String,
    onReceive: (Message) -> Unit,
    onUpdate: (Message) -> Unit,
    onDelete: (String) -> Unit
) {
    hubConnection =
        HubConnectionBuilder.create("http://vms.iesluisvives.org:25003/hubs/messages")
            .build()

    hubConnection.on("ReceiveMessage", { message -> onReceive(message) },
        Message::class.java)
    hubConnection.on("MessageUpdated", { message -> onUpdate(message) },
        Message::class.java)
    hubConnection.on(
        "MessageDeleted",
        { id: String -> onDelete(id) },
        String::class.java
    )
    hubConnection.start().blockingAwait()
    hubConnection.send("JoinChat", chatId)
}

fun disconnect() {
    if (::hubConnection.isInitialized) {
        hubConnection.stop()
    }
}
```

Autorización

De la autorización se encarga el servidor, por lo tanto, al iniciar la aplicación se hace una comprobación:

```
@Composable
fun StartScreen(navController: NavHostController, viewModel: AuthViewModel) {
    val token by viewModel.token.collectAsState(initial = null)

    LaunchedEffect(token) {
        if (token != null) {
            when (viewModel.checkUserStatus()) {
                "OK" -> navController.navigate(Routes.MAIN)

                "NOT_FOUND" -> navController.navigate(Routes.CREATE_USER)

                "UNAUTHORIZED", "UNKNOWN" -> {
                    viewModel.clearToken()
                    navController.navigate(Routes.LOGIN)
                }
            }
        } else {
            navController.navigate(Routes.LOGIN)
        }
    }

    Box(modifier = Modifier.fillMaxSize(), contentAlignment =
Alignment.Center) {
        CircularProgressIndicator()
    }
}
```

Este código comprueba si hay un token almacenado en la aplicación. Si lo hay entonces realiza una consulta de prueba al backend y dependiendo de la respuesta de este inicia la aplicación, lleva a la pantalla de creación de usuario o si el token no es válido/está caducado entonces se borra y se pide el login de nuevo.

Por lo tanto, comprueba:

- Validez del token
- Existencia de un perfil válido asociado a ese token
- Correcto funcionamiento del servidor

Navegación

Para la navegación se utiliza la navegación propia de androidx adaptada a Jetpack Compose. Esta navegación hace uso de rutas como las que se pueden ver aquí:

```
object Routes {
    const val START = "start"
    const val LOGIN = "login"
    const val REGISTER = "register"
    const val RECOVER = "recover"
    const val CREATE_USER = "create_user"
    const val MAIN = "main"

    const val HOME = "home"

    const val COMMUNITIES = "communities"
    const val COMMUNITY_DETAIL = "communityDetail/{communityId}"
    const val COMMUNITY_DETAIL_BASE = "communityDetail"

    const val POST = "post"
    const val POST_DETAIL = "post_detail"
}
```

El funcionamiento de estas rutas se configura en el NavHostController, permitiendo también el envío de información básica entre las pantallas:

```
NavHost(
    navController = navController, startDestination = Routes.HOME,
    modifier = modifier
) {
    composable(Routes.COMMUNITIES) {
        PlaceholderScreen("Comunidades")
    }
    composable(Routes.HOME) {
        HomeScreen()
    }
    composable(
        route = Routes.COMMUNITY_DETAIL,
        arguments = listOf(navArgument("communityId") { type =
        NavType.StringType })
    ) { backStackEntry ->
        val communityId =
            backStackEntry.arguments?.getString("communityId") ?:
        return@composable
        CommunityDetailScreen(
            communityId = communityId, viewModel = communityViewModel,
            navController = navController
        )
    }
    composable(Routes.USERS) {
        UserListScreen(
            viewModel = profileViewModel,
            onUserClick = { user ->
                navController.navigate("${Routes.USER_DETAIL_BASE}/${user.userId}")
            }
        )
    }
}
```

Vistas

Al hacer uso del framework Jetpack Compose, todas las vistas se realizan en Kotlin, sin XML. Por ejemplo, la vista de login:

```
@Composable
fun LoginScreen(
    viewModel: AuthViewModel,
    navController: NavHostController
) {
    var email by rememberSaveable { mutableStateOf("") }
    var password by rememberSaveable { mutableStateOf("") }
    val scrollState = rememberScrollState()

    val isLoading = viewModel.isLoading
    val result = viewModel.loginResult

    LaunchedEffect(result) {
        if (result?.isSuccess == true) {
            navController.navigate(Routes.START)
        }
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .verticalScroll(scrollState)
            .padding(24.dp),
        verticalArrangement = Arrangement.Center
    ) {
        Text("Iniciar sesión", style =
MaterialTheme.typography.headlineMedium)

        Spacer(modifier = Modifier.height(24.dp))

        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Correo electrónico") },
            modifier = Modifier.fillMaxWidth()
        )

        Spacer(modifier = Modifier.height(16.dp))

        OutlinedTextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Contraseña") },
            modifier = Modifier.fillMaxWidth(),
            visualTransformation = PasswordVisualTransformation()
        )

        Spacer(modifier = Modifier.height(24.dp))

        Button(
            onClick = { viewModel.login(email, password) },
            modifier = Modifier.fillMaxWidth(),
            enabled = !isLoading
        ) {
            Text("Entrar")
        }
    }
}
```

En las múltiples vistas se repiten ciertos fragmentos de código que permiten la persistencia de datos a lo largo del ciclo de vida de la aplicación, incluyendo al rotar la pantalla:

- `rememberSaveable { mutableStateOf("") }`

Usado en caso de tipos básicos como String o Int, en caso de objetos complejos se utiliza el Viewmodel

- `LaunchedEffect(Unit) {}`

Hace que al iniciarse la vista se ejecute el código entre corchetes, útil para cargar datos en el viewmodel y mostrarlos en las vistas

Viewmodels

Intermediario entre los datos recogidos de la api y los datos que se muestran en las vistas, facilita la persistencia de datos complejos y la visualización de estos, hace las llamadas necesarias para recoger o guardar datos. Como las llamadas a la api no son instantáneas y tienen un tiempo muy variable se ha incluido un booleano en la mayoría de viewmodels que permite darle feedback al usuario y que sepa cuando están cargando sus datos para esperar y mejorar la experiencia de usuario:

```
var loginResult by mutableStateOf<Result<LoginResponse>?>(null)
when {
    isLoading -> {
        CircularProgressIndicator()
    }
}
```

En cuanto a las listas de objetos normalmente se ha usado `mutableListOf`:

```
val usersSelected = mutableListOf<UserReduction>()
```

Aunque en algunos casos ha sido necesaria una lista más reactiva que muestre inmediatamente cualquier cambio en las vistas, haciendo una aplicación completamente reactiva, entonces se ha usado `StateFlow`:

```
val users: StateFlow<List<UserReduction>> = _users
```

Personalización

Además de la personalización individual de los componentes que la necesitaban se ha utilizado un esquema de estilos global:

```
private val DarkColorScheme = darkColorScheme(
    primary = Color(0xFFBF3434),
    onPrimary = Color.White,
    primaryContainer = Color(0xFF3A1A1A),
    onPrimaryContainer = Color(0xFFFFDAD6),

    secondary = Color(0xFFE63E3E),
    onSecondary = Color.White,
    secondaryContainer = Color(0xFF4B2323),
    onSecondaryContainer = Color(0xFFFFDAD6),

    tertiary = Color(0xFFD93B3B),
    onTertiary = Color.White,
    tertiaryContainer = Color(0xFF3A2323),
    onTertiaryContainer = Color(0xFFFFDAD6)
)
```

2.4. IMPLANTACIÓN

La implantación del sistema se lleva a cabo mediante contenedores Docker, gestionados con la herramienta Docker-Compose. Este método facilita la distribución, el despliegue y la ejecución de múltiples servicios interconectados de forma sencilla y consistente en cualquier entorno compatible con Docker.

2.4.1. Plataformas utilizadas

- [Docker Engine](#): Motor principal de ejecución de contenedores.
- [Docker Compose](#): Herramienta para definir y orquestar el despliegue de múltiples servicios.
- [Docker Hub](#): Todas las imágenes necesarias para los servicios han sido previamente construidas y subidas a Docker Hub, por lo que no se requiere clonar ningún repositorio ni compilar localmente.
- [Compatibilidad](#): El sistema puede ejecutarse en cualquier sistema operativo que soporte Docker (Ubuntu, Windows, macOS).

2.4.2. Servicios desplegados

El sistema está formado por varios microservicios que se comunican entre sí mediante redes definidas en Docker:

Api Gateway

Se encarga de redireccionar las peticiones HTTP al servicio correspondiente

Microservicios

- [auth_api](#): Autenticación y gestión de usuarios, incluyendo envío de correos.
- [com_api](#): Gestión de comunidades y sus contenidos.
- [web-api-messages](#): Servicio de mensajería entre usuarios.
- [prof_api](#): Gestión de perfiles de usuario.
- [rev_api](#): Sistema de valoraciones y reseñas.
- [webapitaller](#): Servicio de taller mecánico, incluyendo vehículos y componentes.

Bases de datos

- [PostgreSQL](#): Utilizada por la mayoría de los servicios (uno por microservicio).
- [MongoDB](#): Utilizada únicamente por el microservicio del taller.

Frontend Web

Interfaz web de usuario.

Frontend Móvil

Interfaz móvil de usuario, creado el APK firmado y subido a Google Drive (Anexo V)

2.4.3. Variables de entorno

Variable	Explicación	Microservicios
POSTGRES_USER	Usuario para autenticar la conexión a la base de datos PostgreSQL	Todos los servicios con PostgreSQL
POSTGRES_PASSWORD	Contraseña para autenticar la conexión a la base de datos PostgreSQL	Todos los servicios con PostgreSQL
POSTGRES_DB	Nombre de la base de datos	Todos los servicios con PostgreSQL
DbSettings__Host	Nombre del contenedor donde está alojada la base de datos	Todos los microservicios que usan base de datos
DbSettings__Port	Puerto del host de la base de datos	Igual que DbSettings__Host
DbSettings__Username	Usuario para conectarse a la base de datos desde el microservicio	Igual que DbSettings__Host
DbSettings__Password	Contraseña para conectarse a la base de datos desde el microservicio	Igual que DbSettings__Host
DbSettings__Database	Nombre de la base de datos a la que se conecta el microservicio	Igual que DbSettings__Host
JwtSettings__SecretKey	Clave secreta usada para firmar y validar tokens JWT para autenticación	Todos los microservicios con
EmailSettings__UserEmail	Dirección de email utilizada para enviar correos	auth_api
EmailSettings__UserName	Nombre de usuario que aparece en el correo	auth_api
EmailSettings__UserApiKey	Clave API para autenticar el envío de correos	auth_api
EmailSettings__Host	Ruta del front	auth_api
MONGO_INITDB_ROOT_USERNAME	Usuario administrador para MongoDB	mongodb
MONGO_INITDB_ROOT_PASSWORD	Contraseña para MongoDB	mongodb
MONGO_INITDB_DATABASE	Base de datos inicial que se crea al iniciar MongoDB	mongodb

Tabla 2-1: Variables de entorno

2.4.4. Persistencia de datos

El sistema utiliza volúmenes para mantener la persistencia de los datos, incluso si los contenedores son reiniciados o eliminados.

Volumen	Servicio	Uso
postgres_messages	postgres_messages	Datos persistentes de la base de datos de mensajes
postgres_auth_data	auth_database	Datos persistentes de la base de datos de autenticación
postgres_com_data	com_database	Datos persistentes de la base de datos de comunidades
postgres_prof_data	prof_database	Datos persistentes de la base de datos de perfiles
postgres_rev_data	rev_database	Datos persistentes de la base de datos de reseñas
mongo_workshop_data	mongodb	Datos persistentes de la base de datos del taller (MongoDB)
profile_images	prof_api	Imágenes de perfil de usuario almacenadas en el microservicio de perfiles
community_images	com_api	Imágenes asociadas a comunidades almacenadas en el microservicio de comunidades

2.4.5. Redes

El sistema usa redes para comunicar contenedores entre sí.

Red	Servicios
backend	webapigateway, web-api-messages, auth_api, com_api, prof_api, rev_api, webapitaller, web_frontend
messages	postgres_messages, web-api-messages
authentication	auth_database, auth_api
communities	com_database, com_api
prof	prof_database, prof_api
rev	rev_database, rev_api
workshop	mongodb, webapitaller

2.4.6. Procedimiento de despliegue

Instalación de Docker y Docker Compose

Descargar e instalar Docker.

Colocar el fichero docker-compose.yml

En una carpeta local del servidor o máquina donde se desea ejecutar el sistema.

Ejecutar el despliegue

Abrir una terminal en la carpeta del fichero y ejecutar: docker-compose up -d

Acceso a la aplicación

Servicio	Contenedor	Puerto externo	Descripción
Frontend web	web_frontend	25002	Interfaz de usuario
API Gateway	webaPIgateway	25003	Peticiones HTTP

Suponiendo que se levante en local, la ruta sería **http://localhost: {Puerto externo}**

El archivo docker-compose.yml completo se incluye en los anexos para su consulta.

2.5. DOCUMENTACIÓN

Durante el desarrollo del proyecto se ha generado la siguiente documentación, que se adjunta como Anexos:

Manual del Usuario Web

Guía dirigida a los usuarios finales sobre el uso de la aplicación web, incluyendo navegación, funcionalidades principales y resolución de problemas comunes.

Manual del Usuario Móvil

Guía que facilita la familiarización de los usuarios con la aplicación móvil, incluyendo las funcionalidades principales.

Documentación Técnica

Disponible en los respectivos archivos README.md de los repositorios de código fuente (Frontend, backend y móvil). Incluye instrucciones de instalación, configuración y uso de contenedores Docker.

Para más detalles y enlaces de acceso, consultar los Anexos correspondientes:

- Manual del Usuario Web.
- Manual del Usuario Móvil
- Repositorios.

3. CONCLUSIONES

La realización de este proyecto ha supuesto uno de los mayores retos a los que nos hemos enfrentado durante el ciclo. El objetivo era desarrollar una **aplicación basada en microservicios**, una arquitectura completamente nueva para nosotros, que nos ha obligado a enfrentarnos a numerosos desafíos técnicos y organizativos. Desde el inicio, adoptamos una dinámica de trabajo muy cercana a la de un entorno profesional real: **cada integrante del equipo tenía sus responsabilidades claramente delimitadas y no podíamos modificar el código del resto**, lo que implicó una coordinación constante, una planificación rigurosa y una comunicación fluida.

Durante el desarrollo, trabajamos con una gran variedad de tecnologías. En el **Frontend web**, utilizamos **JavaScript con React**, mientras que para el **Frontend móvil** nos adentramos en **Kotlin y Jetpack Compose**, herramientas que no habíamos utilizado previamente. Además, integramos el uso de **Docker** para crear contenedores con los distintos servicios y aprendimos a implementar un **API Gateway**, un componente clave en arquitecturas de microservicios que desconocíamos por completo al empezar. Este aspecto fue especialmente complejo, ya que no solo se trataba de configurarlo correctamente, sino de comprender su papel en la gestión del tráfico entre servicios y cómo mejorar la escalabilidad y mantenibilidad del sistema.

Todo este ecosistema de tecnologías supuso una **curva de aprendizaje muy pronunciada**. A menudo nos vimos sobrepasados por la cantidad de elementos nuevos a dominar, y el ritmo del proyecto hizo que llegáramos al límite de las fechas de entrega en más de una ocasión. También tuvimos momentos de **descoordinación y dificultades de comunicación** dentro del equipo, lo cual nos obligó a mejorar nuestra manera de organizarnos, tomar decisiones de forma más estructurada y ser mucho más estrictos con la gestión del tiempo y los plazos.

Sin embargo, todos estos obstáculos también fueron oportunidades de crecimiento. A nivel técnico, hemos aprendido muchísimo: no solo sobre herramientas y lenguajes concretos, sino también sobre **principios de arquitectura, despliegue con contenedores, autenticación, comunicación entre servicios**.

Desde una perspectiva personal y académica, este proyecto nos ha servido para comprobar que realmente tenemos lo necesario para trabajar como programadores. Ha sido una experiencia exigente, en la que no solo hemos demostrado nuestra capacidad técnica, sino también nuestra **resiliencia**, nuestra **motivación por aprender**, y, sobre todo, nuestras **ganas de seguir adelante incluso cuando las cosas no salían como esperábamos**.

En conclusión, este proyecto ha sido la culminación perfecta del ciclo, ya que nos ha enfrentado a una situación realista, con problemas reales, decisiones técnicas importantes y responsabilidades claras. Nos ha permitido aplicar lo aprendido y descubrir todo lo que todavía nos queda por mejorar. Pero, sobre todo, nos ha dejado claro que estamos preparados para dar el siguiente paso y afrontar nuevos retos en el mundo del desarrollo profesional.

3.1. RESULTADOS Y DISCUSIÓN

Durante el proyecto se experimentaron diferencias significativas entre la temporalización inicialmente planificada y la ejecución real. En varias fases, especialmente durante la integración de tecnologías novedosas y la configuración de la arquitectura basada en microservicios, surgieron dificultades técnicas que ralentizaron el progreso. La curva de aprendizaje resultó considerable debido al uso de tecnologías no conocidas previamente, como React para Frontend web y Jetpack Compose para Frontend móvil, así como la implementación de un API Gateway y la gestión mediante Docker.

Además, la dinámica de trabajo basada en la asignación de responsabilidades individuales y la imposibilidad de modificar el código desarrollado por otros generó la necesidad de mantener una comunicación constante y una coordinación estricta para evitar bloqueos y malentendidos. En ocasiones, la falta de comunicación efectiva y la presión del tiempo contribuyeron a retrasos y cuellos de botella, especialmente en las fases finales cerca de las fechas límite.

Entre las dificultades más importantes también se destaca la falta de experiencia previa en el manejo de arquitecturas distribuidas, lo que requirió múltiples ajustes y revisiones para asegurar la correcta comunicación entre servicios y la coherencia global del sistema.

A pesar de estos retos, el proyecto concluyó con la obtención de un producto funcional y estable, que cumple con los objetivos principales planteados y representa un avance significativo en el aprendizaje y la aplicación práctica de nuevas tecnologías y metodologías.

3.2. TRABAJO FUTURO

Aunque el proyecto ha alcanzado un estado funcional y estable, han quedado algunos aspectos pendientes de integrar por completo, principalmente debido a limitaciones de tiempo y carga de trabajo.

Uno de los elementos más relevantes es el sistema de roles por comunidad. Este sistema ya está implementado de forma completa en el backend y permite que un mismo usuario tenga diferentes roles según la comunidad en la que participe (por ejemplo, administrador en una comunidad y miembro en otra). Sin embargo, por falta de tiempo, no se ha reflejado aún en el Frontend, por lo que el usuario no puede visualizar ni gestionar sus roles desde la interfaz. Se trataría de una mejora prioritaria para futuras versiones, ya que potenciaría significativamente la usabilidad y el control por parte del usuario.

Otra funcionalidad pendiente de desarrollo es la integración de elementos de inteligencia artificial. Desde el planteamiento inicial, se propuso la posibilidad de incorporar un sistema de recomendaciones personalizadas (por ejemplo, comunidades sugeridas, contenido destacado según intereses o actividad previa), así como un módulo de análisis de datos que permitiera extraer métricas útiles del comportamiento de los usuarios y los mantenimientos de los vehículos, pudiendo ofrecer información relevante tanto a los administradores como a los propios usuarios. Aunque parte de esta lógica fue esbozada, la integración técnica no se llegó a realizar, ni en el backend ni en el Frontend, principalmente por falta de tiempo y recursos.

Funcionalidad	Estado actual	Prioridad	Descripción
Visualización y gestión de roles en el Frontend	No implementado (backend funcional)	Alta	Permitir a los usuarios ver y administrar sus roles por comunidad desde la interfaz.
Sistema de recomendaciones personalizadas (IA)	No implementado	Media	Sugerencias automáticas basadas en actividad e intereses del usuario.
Análisis de datos y métricas de uso (IA)	No implementado	Media	Herramientas para visualizar estadísticas de participación y uso.
Gestión avanzada de permisos por rol	Parcialmente implementado	Media	Ampliar las restricciones y capacidades asociadas a cada rol dentro de la comunidad.
Panel de administración para moderadores	No implementado	Baja	Crear una interfaz específica para la gestión de contenido y usuarios por parte de moderadores.

Tabla 3-1: Roadmap futuro.

4. BIBLIOGRAFIA

4.1. ASP.NET Core

- Microsoft. *Build a web API with ASP.NET Core*. Available at: <https://learn.microsoft.com/en-us/training/modules/build-web-api-aspnet-core/> (Accessed: 24 March 2025)
- Microsoft. *Use Swagger to document and test ASP.NET Core web APIs*. Available at: <https://learn.microsoft.com/es-es/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-8.0&viewFallbackFrom=aspnetcore-9.0> (Accessed: 27 March 2025)
- Microsoft. *Configure JWT bearer authentication in ASP.NET Core*. Available at: <https://learn.microsoft.com/es-es/aspnet/core/security/authentication/configure-jwt-bearer-authentication?view=aspnetcore-9.0> (Accessed: 30 March 2025)
- Microsoft. *Action return types in ASP.NET Core Web API*. Available at: <https://learn.microsoft.com/en-us/aspnet/core/web-api/action-return-types?view=aspnetcore-9.0#actionresultt-type> (Accessed: 2 April 2025)
- Microsoft. *Authorization with roles in ASP.NET Core*. Available at: <https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-9.0> (Accessed: 5 April 2025)
- Pietzko, P. *Uploading and storing files with ASP.NET and wwwroot*. Available at: <https://medium.com/@paul.pietzko/uploading-and-storing-files-with-asp-net-and-wwwroot-1efc23d34ee5> (Accessed: 7 April 2025)
- Microsoft. *Static files in ASP.NET Core*. Available at: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/static-files?view=aspnetcore-9.0> (Accessed: 10 April 2025)
- Microsoft. *ASP.NET Core best practices*. Available at: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/best-practices?view=aspnetcore-9.0> (Accessed: 13 April 2025)
- DrSimpleGraffiti. *Pagination in .NET API*. Available at: <https://dev.to/drsimplegraffiti/pagination-in-net-api-4opp> (Accessed: 15 April 2025)
- ByteHide. *Pagination in C#: Complete guide with easy code examples*. Available at: <https://www.bytehide.com/blog/pagination-in-c-complete-guide-with-easy-code-examples> (Accessed: 18 April 2025)

- Microsoft. *Real time ASP.NET Introduction*. Available at:
https://learn.microsoft.com/es-es/aspnet/core/tutorials/signalr?view=aspnetcore-9.0&WT.mc_id=dotnet-35129-website&tabs=visual-studio (Accessed: 19 May 2025)
- Ocelot. *Ocelot Gateway Configuration*. Available at:
<https://ocelot.readthedocs.io/en/latest/features/configuration.html> (Accessed: 10 May 2025)
- Ocelot. *Ocelot Gateway Websockets and SignalR Implementation*. Available at:
<https://ocelot.readthedocs.io/en/latest/features/websockets.html> (Accessed: 21 May 2025)

4.2.Docker

- Docker. *Compose file reference*. Available at:
<https://docs.docker.com/reference/compose-file/> (Accessed: 21 April 2025)

4.3.Tailwind, React y JavaScript

- Tailwind CSS. *Styling with utility classes*. Available at:
<https://tailwindcss.com/docs/styling-with-utility-classes> (Accessed: 23 April 2025)
- React. *Thinking in React*. Available at: <https://es.react.dev/learn/thinking-in-react> (Accessed: 26 April 2025)
- React. *Adding interactivity*. Available at: <https://es.react.dev/learn/adding-interactivity> (Accessed: 29 April 2025)
- React. *Context*. Available at: <https://legacy.reactjs.org/docs/context.html> (Accessed: 1 May 2025)
- LenguajeJS. *JavaScript: Fundamentos y conceptos clave*. Available at:
<https://lenguajejs.com/javascript/#tema-2> (Accessed: 4 May 2025)
- InfiniteScrollComponent. *react-infinite-scroll-component*. Available at:
<https://www.npmjs.com/package/react-infinite-scroll-component> (Accessed: 7 May 2025)

4.4. Jetpack Compose and Hilt

- Android Developer. *Introduction to Jetpack Compose*. Available at: <https://developer.android.com/develop/ui/compose/tutorial?hl=es-419> (Accessed: 2 June 2025)
- Medium. *Jetpack Compose and MVVM*. Available at: <https://medium.com/@sks727633/mvvm-with-jetpack-compose-structuring-your-app-for-clean-architecture-42f4bad4c99e> (Accessed: 4 June 2025)
- Android Developer. *Hilt Dependency Injector*. Available at: <https://developer.android.com/training/dependency-injection/hilt-android?hl=es-419> (Accessed: 4 June 2025)
- Android Developer. *Thinking in compose*. Available at: <https://developer.android.com/develop/ui/compose/mental-model?hl=es-419> (Accessed: 6 June 2025)
- Medium. *SignalR for Real-Time Communication in Android*. Available at: <https://medium.com/@khorassani64/implementing-signalr-for-real-time-communication-in-android-bb44ef8d0c2b> (Accessed: 8 June 2025)

ANEXOS

En esta sección se incluye documentación adicional relevante para la comprensión, despliegue y mantenimiento del sistema.

Anexo.I. Manual de usuario Web

Aplicación web

Creado por:
José María García
Rodrigo Tapiador Cano



índice

1.	REGISTRO Y LOGIN.....	98
2.	CAMBIO DE CONTRASEÑA	100
3.	CREACIÓN DE PERFIL.....	101
4.	NAVEGACIÓN BÁSICA	102
5.	PERFIL	104
6.	REVIEWS	107
7.	COMUNIDADES	108
8.	THREADS.....	109
9.	CHATS	112
10.	TALLER	116

1. Registro y login

Registro

The image shows a 'Login' screen with the following elements:

- A title 'Login' at the top.
- Two input fields: 'Correo' (Email) and 'Contraseña' (Password).
- A blue 'Iniciar sesión' (Sign In) button.
- Text links at the bottom: '¿No tienes una cuenta? [Regístrate](#)' and '¿Olvidaste tu contraseña?'.

Desde la página de Login haremos click en Regístrate. Esto nos llevará a una página con un pequeño formulario.

Imagen 1-1:Pantalla login

Una vez rellenemos el formulario, haremos click en registrarse. Si todo ha salido bien, nos llegará un email al correo. Y al hacer click en el enlace nos llevará de nuevo a la pantalla de Login.

The image shows a 'Crear cuenta' (Create account) screen with the following elements:

- A title 'Crear cuenta' at the top.
- Input fields: 'Nombre' (Name), 'Correo' (Email), 'Contraseña' (Password), and 'Repetir contraseña' (Repeat password).
- A checkbox labeled '¿Eres un taller?' (Are you a workshop?).
- A large blue 'Registrarse' (Register) button.
- Links at the bottom: 'Email confirmado' (Email confirmed) and 'Ir a Login' (Go to Login).
- Text link at the bottom: '¿Ya tienes una cuenta? [Inicia sesión](#)' (Do you have an account? [Start session](#)).

Imagen 1-2:Pantalla registro

Imagen 1-3:Pantalla email confirmado

Login

Una vez nos hayamos registrado, introduciremos nuestras credenciales en el formulario. Y entraremos a la página principal de la aplicación

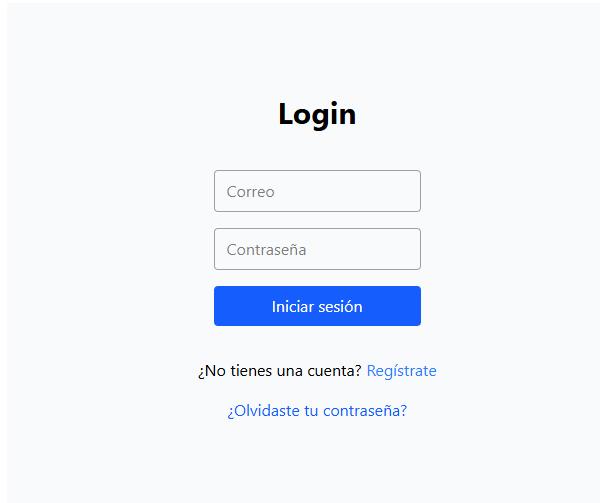


Imagen 1-4: Pantalla login

2. Cambio de contraseña

En caso de pérdida de contraseña existe la opción de cambiarla desde la página de login.

Login

Correo

Contraseña

Iniciar sesión

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

[¿Olvidaste tu contraseña?](#)

Recuperar contraseña

Introduce tu email:

Enviar solicitud

Recuperar contraseña

Solicitud enviada.
Si ese correo está registrado, recibirás un email con instrucciones para cambiar tu contraseña.

Imagen 2-1:Pantalla recuperación de contraseña

Una vez introducido tu email te llegará un enlace al correo para renovar tu

Cambiar contraseña

Nueva contraseña

Confirmar contraseña

Cambiar contraseña

Imagen 2-2:Formulario cambio de contraseña

contraseña.

Rellenarías el formulario y si todo va bien volverías a la página del login automáticamente.

3. Creación de perfil

La primera vez que un usuario ingrese al Dashboard, se abrirá un modal de creación de perfil (Se recomienda pulsar f5 en caso de que no aparezca el modal pasados unos segundos)

The screenshot shows a white modal window titled 'Crear Perfil'. It contains five input fields: 'Nombre *' (Name), 'UserName *' (User Name), 'Descripción' (Description) with placeholder text 'Cuéntanos algo sobre ti...', 'Dirección' (Address) with placeholder text 'Ciudad, país...', and an 'Imagen de perfil' (Profile Picture) input field with the placeholder 'Seleccionar archivo Ningún archivo seleccionado' (Select file). A blue 'Crear perfil' (Create profile) button is at the bottom.

Una vez se rellene el perfil al completo (es necesario subir imagen) se podrá acceder a todas las partes de la aplicación.

Imagen 3-1:Formulario de perfil

4. Navegación básica

El dashboard está dividido en tres partes esenciales.



Imagen 4-1: Dashboard



Header

HEADER

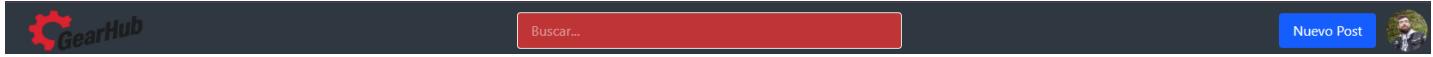


Imagen 4-2: Header

Búsqueda de elementos, creación de posts y navegación a perfil del usuario.

SIDEBAR

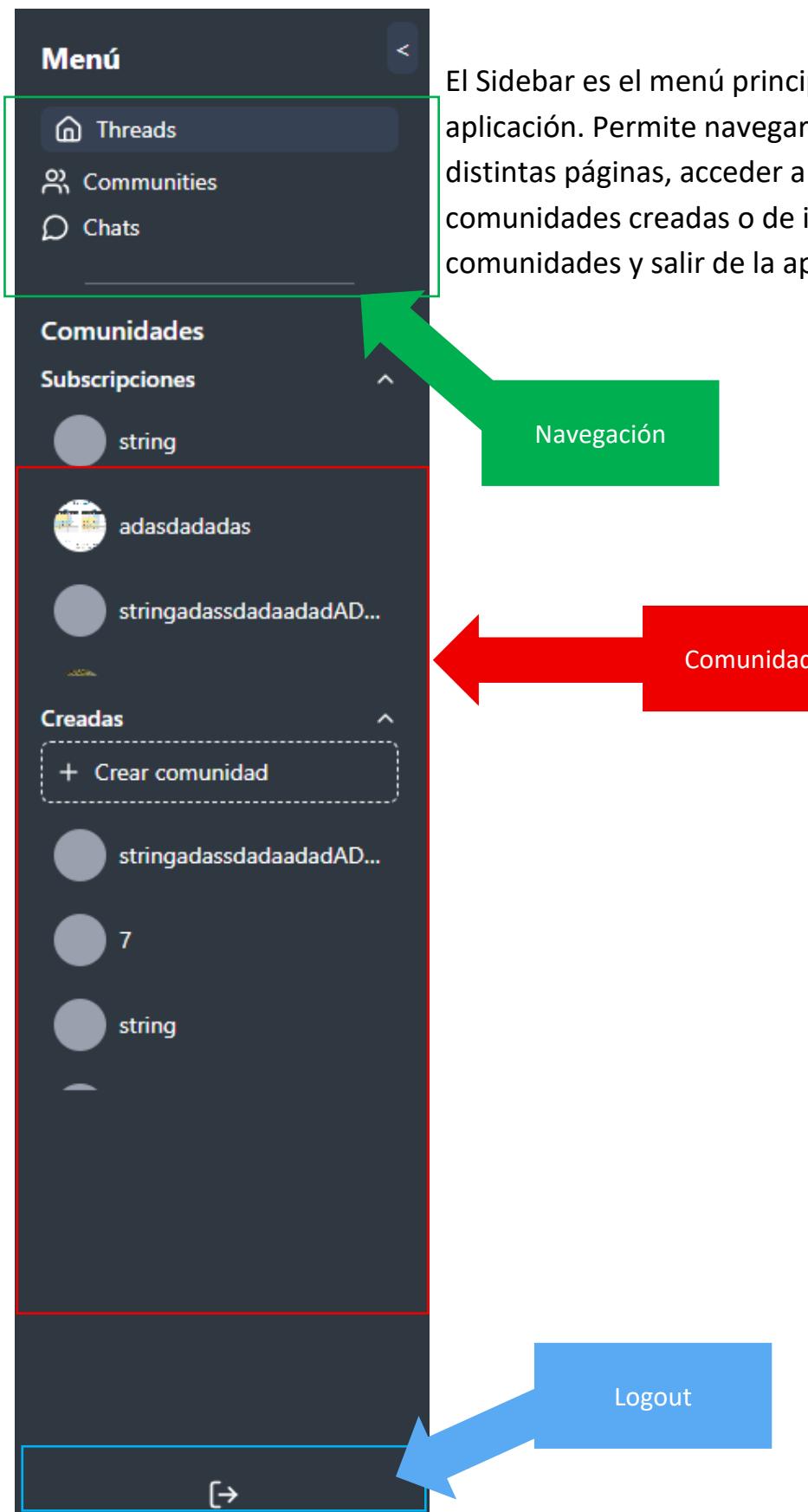


Imagen 4-3:Sidebar

5. Perfil

En el perfil se pueden tener 2 tipos de vista. Una es para perfiles de usuarios normales y otra para talleres. La principal diferencia es la aparición de la pestaña de reviews.

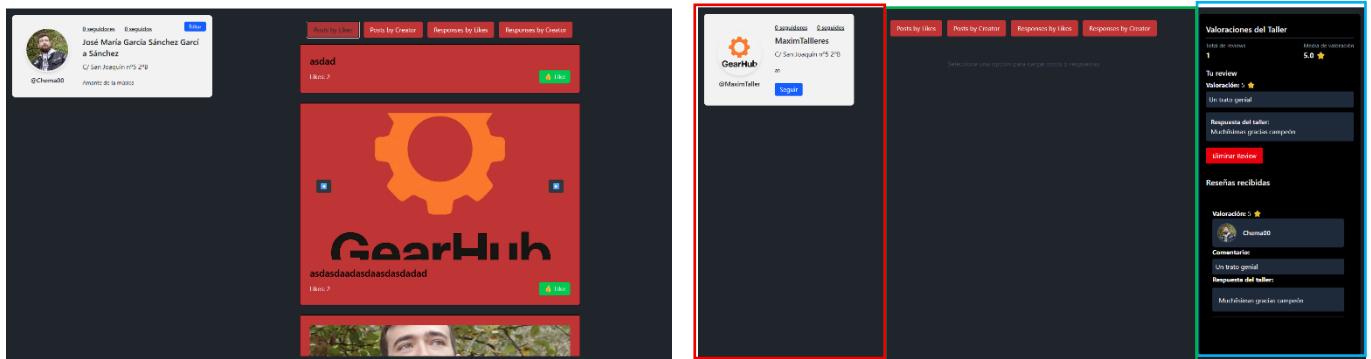


Imagen 5-1:Perfiles

Podemos distinguir 3 partes:

- La tarjeta de perfil: Aquí muestra la información del usuario. Se puede acceder a sus seguidores y seguidos. Y puedes seguirle, dejar de seguir o quitar de seguidores. Además, si es tu tarjeta, podrás editar tu información.
- El escaparate de threads y respuestas: Se muestran los hilos y respuestas creadas o que le gusten al usuario del perfil en el que te encuentres
- La tarjeta de reviews: Solo aparece cuando estás en el perfil de un taller. Permite ver las críticas de un taller, la cantidad y la valoración media.

TARJETA DE PERFIL

En esta tarjeta se muestra la info principal del usuario al que pertenezca el perfil que visites. Si es tu tarjeta podrás editar tu info de contacto.



Imagen 5-2: Tarjeta de perfil

Si haces click en seguidores o seguidos, te saldrá un modal con tus seguidores.

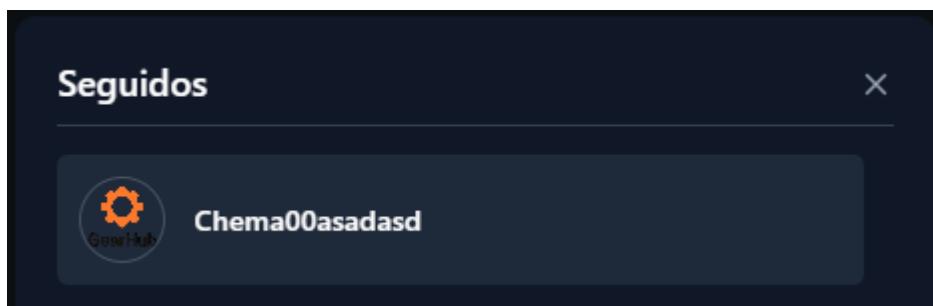


Imagen 5-3: Modal de seguidos

Y si se hace click en la tarjeta del modal, te lleva al perfil del otro usuario

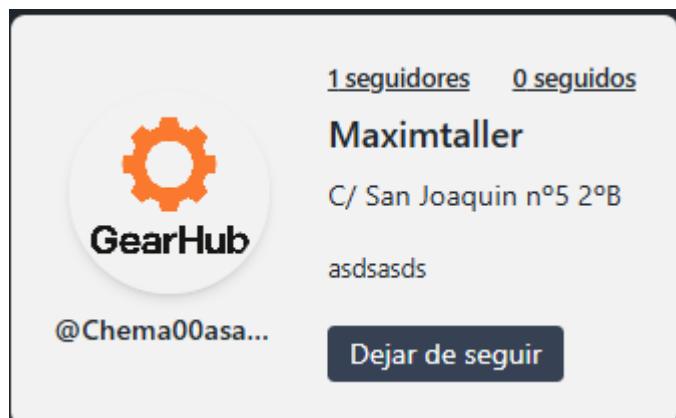


Imagen 5-4: Tarjeta de perfil

ESCAPARATE

Permite conocer los threads y las respuestas que ha creado o dado like.

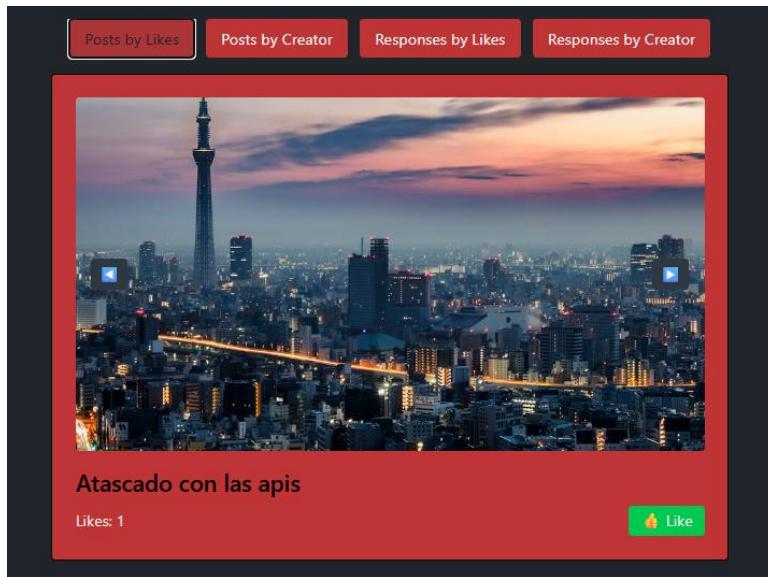


Imagen 5-6: Escaparate con threads

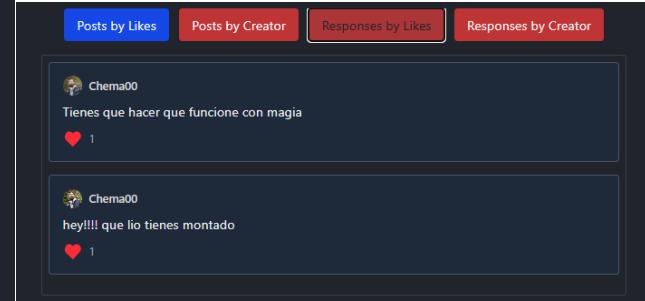


Imagen 5-5: Escaparate con respuestas

6. Reviews

En la tarjeta de reviews podremos poner reviews a otros talleres si somos usuarios. En caso de ser un taller, solo podremos responder a las reviews y ver las del resto, pero no podremos responder ni poner reviews en el resto de perfiles

Valoraciones del Taller

Total de reviews	Media de valoración
0	0.0 ★

Deja tu review

Valoración (0-5):

Comentario:

Enviar Review

Imagen 6-1:Formulario review

Valoraciones del Taller

Total de reviews	Media de valoración
1	5.0 ★

Tu review

Valoración: 5 ★

Un trato genial

Respuesta del taller:
Muchísimas gracias campeón

Eliminar Review

Reseñas recibidas

Valoración: 5 ★

 Chema00

Comentario:
Un trato genial

Respuesta del taller:
Muchísimas gracias campeón

Imagen 6-1:Vista tarjeta review

7. Comunidades

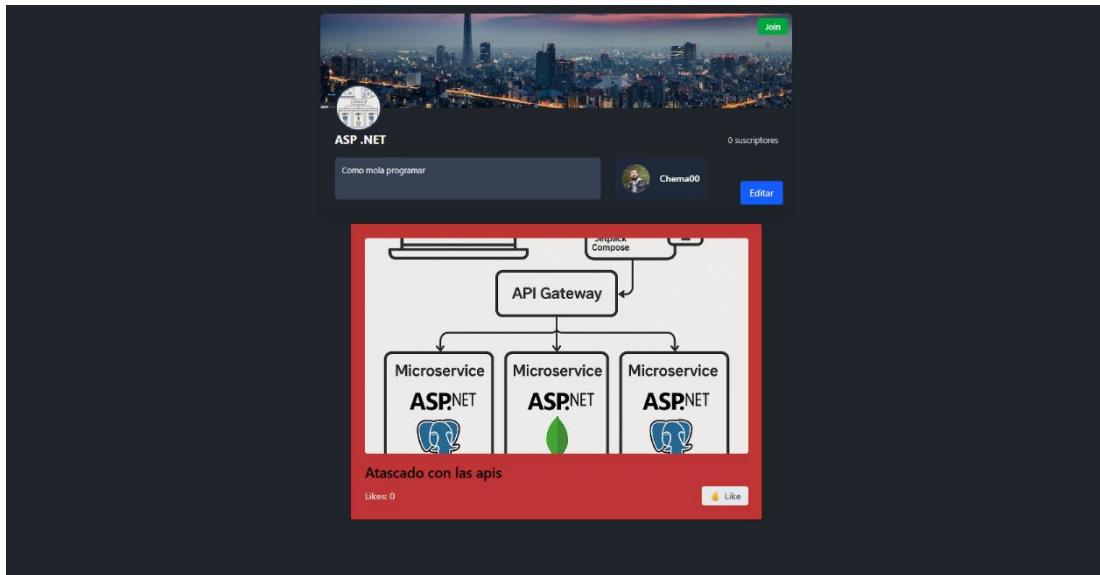


Imagen 7-1:Vista Comunidad

Cuando el usuario hace click en una comunidad, este llega a la vista detalle de la misma. En esta se ve el panel de la comunidad, junto con los threads de la misma.

En la tarjeta de comunidad puedes unirte o dejar la comunidad. Y si eres el dueño puedes editar.

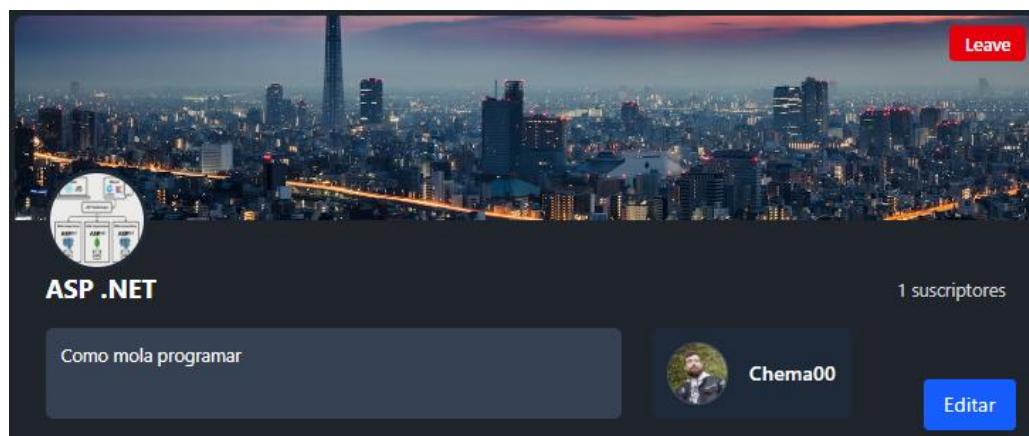


Imagen 7-2:Tarjeta comunidad

8. Threads

Al hacer click encima de un thread, el usuario accederá a su vista detalle en la que aparecerán el thread en sí y las respuestas de este.

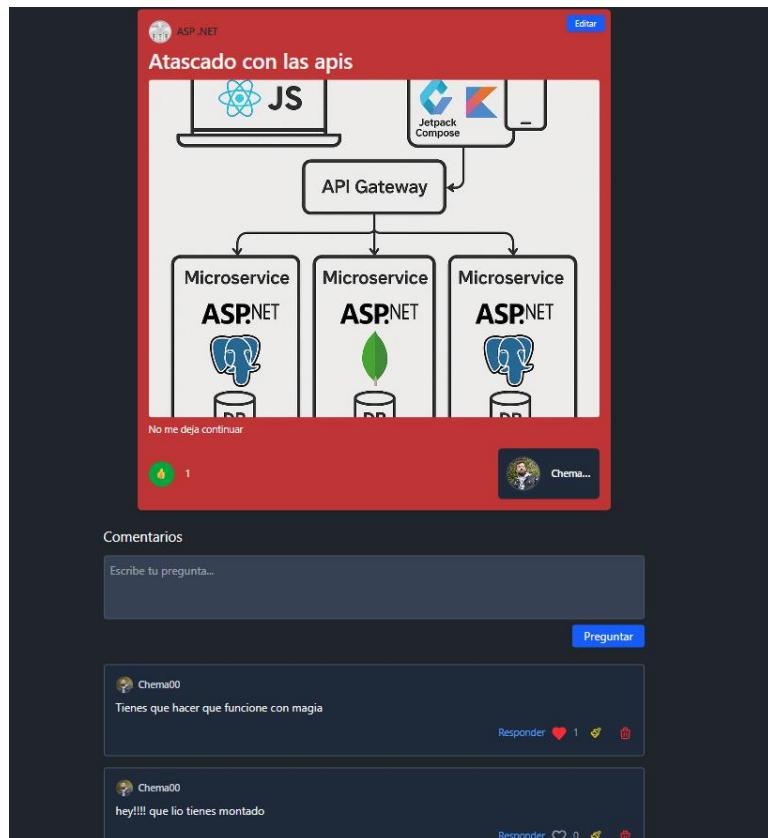


Imagen 8-1:Vista thread

En el detail se puede dar like al thread, ir al perfil del creador o a la comunidad. Además, puedes ir pasando de foto en foto, y si pulsas encima de ella puedes entrar al carrusel de imágenes.

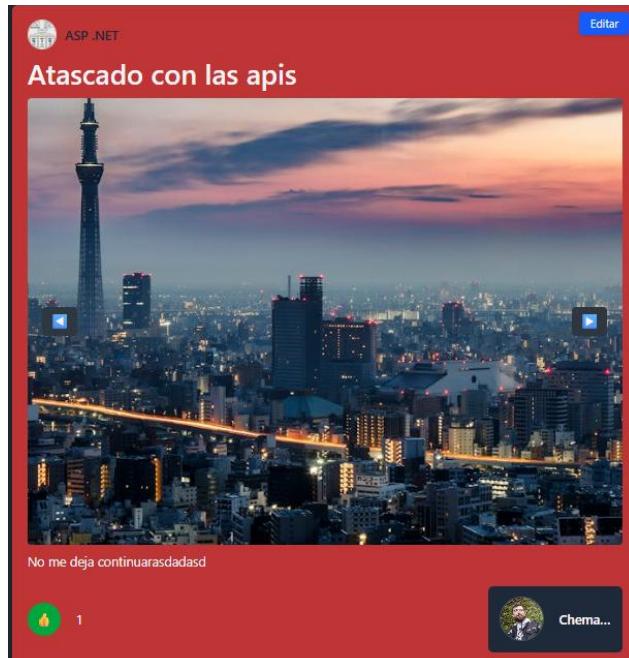


Imagen 8-2: Tarjeta thread

Así se vería el carrusel de imágenes.

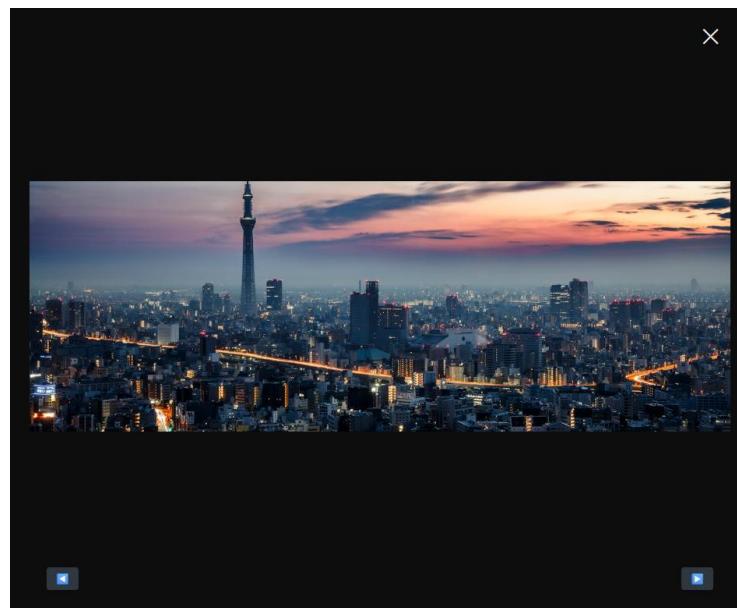


Imagen 8-3: Carrusel de imágenes

En la mitad inferior se encuentra el apartado de comentarios y preguntas. Las respuestas se muestran anidadas. Puedes eliminar o editar tus respuestas. Y puedes responder a otros usuarios

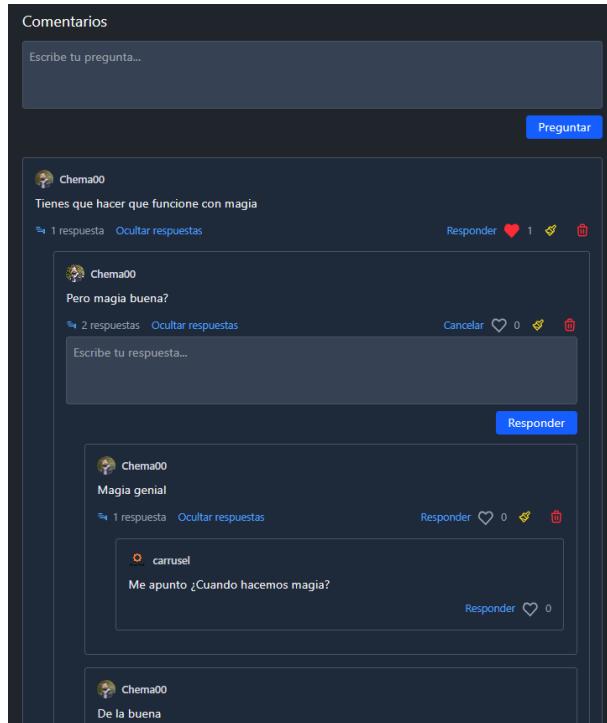


Imagen 8-4:Respuestas

9. Chats

En la página de chats el usuario tendrá la capacidad de ver sus chats, de crear nuevos y de dejar de ser parte de estos. Esta página se divide en dos partes principales.

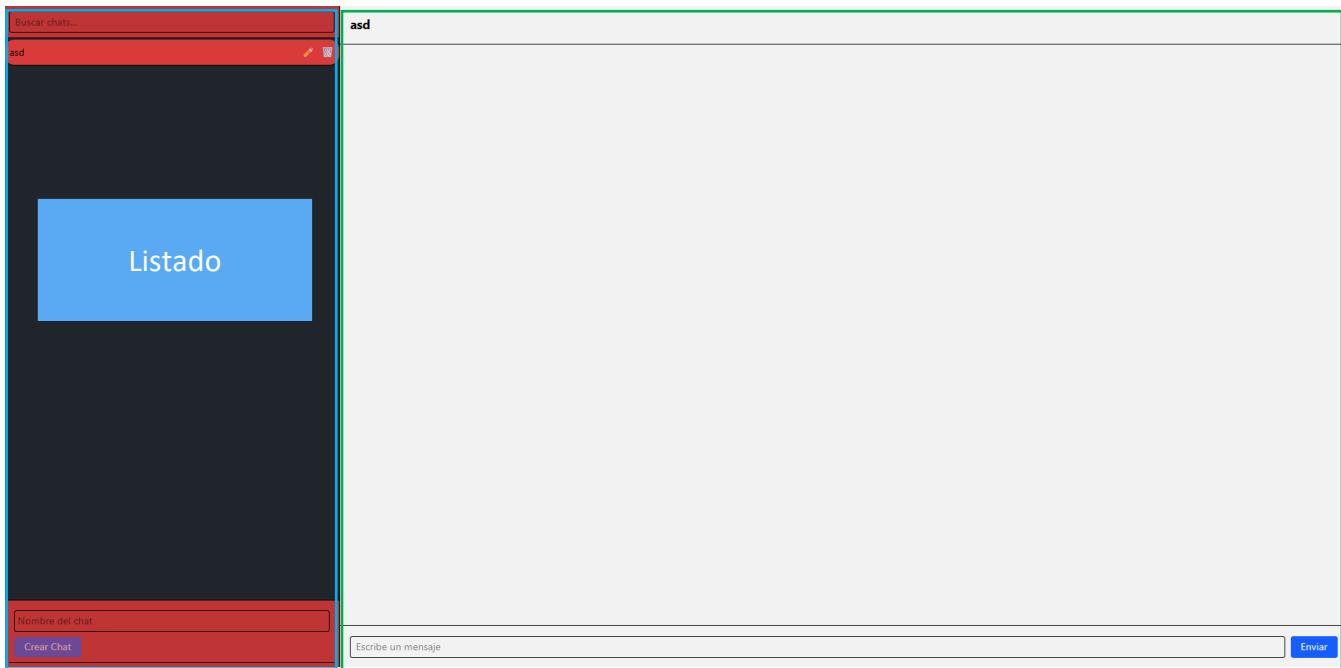


Imagen 9-1:Vista principal

- El listado se usa para controlar principalmente los chats, editarlos, borrarlos o generar nuevos.
- La parte de detalle muestra la info de los chats, los mensajes y sus miembros. Además permite enviar mensajes.

LISTADO

La parte superior permite filtrar los chats por nombre. Justo debajo está la lista con los chats de los que somos miembros. Si somos los propietarios del chat podremos editarlos y borrarlos.

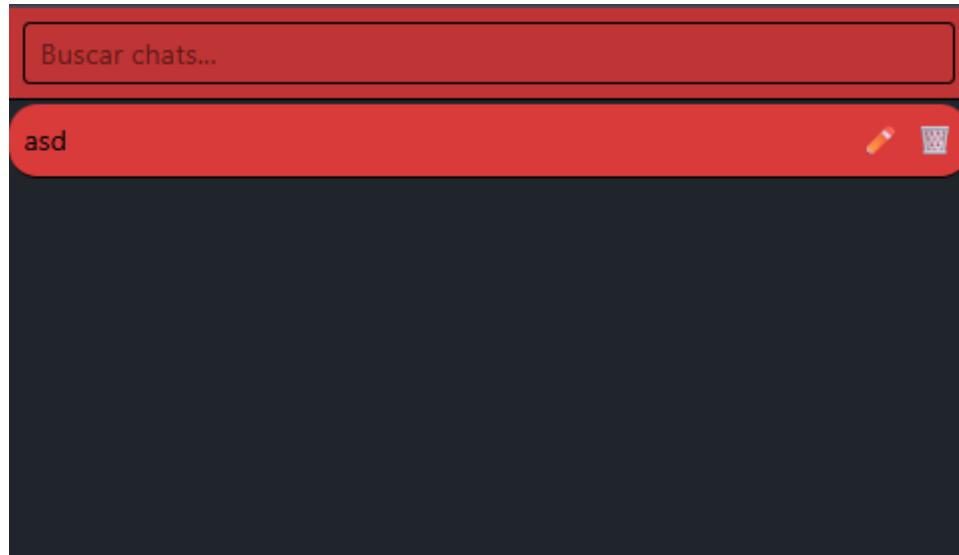


Imagen 9-2:Lista de chats

En la parte inferior encontramos el formulario para generar chats. El botón está inactivo hasta que escribes un nombre del chat.



Imagen 9-3:Creación de chat

DETALLE

En la vista detalle podremos mandar mensajes e interactuar con los detalles de este chat.



Imagen 9-4:Chat en vivo

En la parte baja se escriben y se envían los mensajes

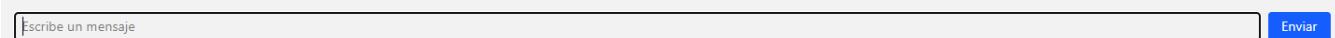


Imagen 9-5:Formulario mensaje

En la parte intermedia se muestran los mensajes enviados al grupo

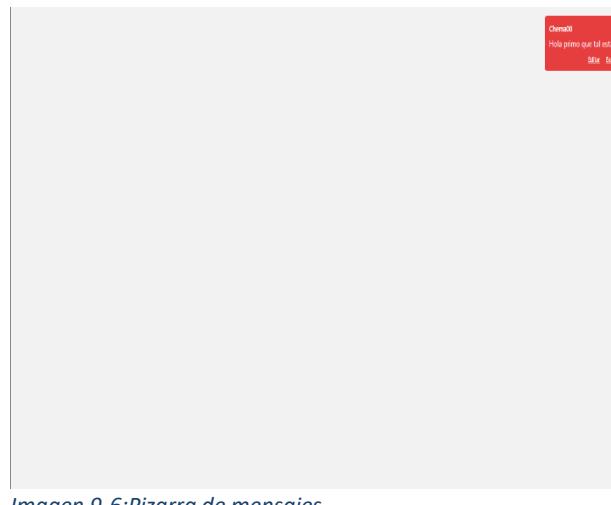


Imagen 9-6:Pizarra de mensajes

Al hacer click en el nombre del grupo se nos cambiará la vista a la de miembros del grupo. Si se hace click en volver, se volverá a la vista anterior.

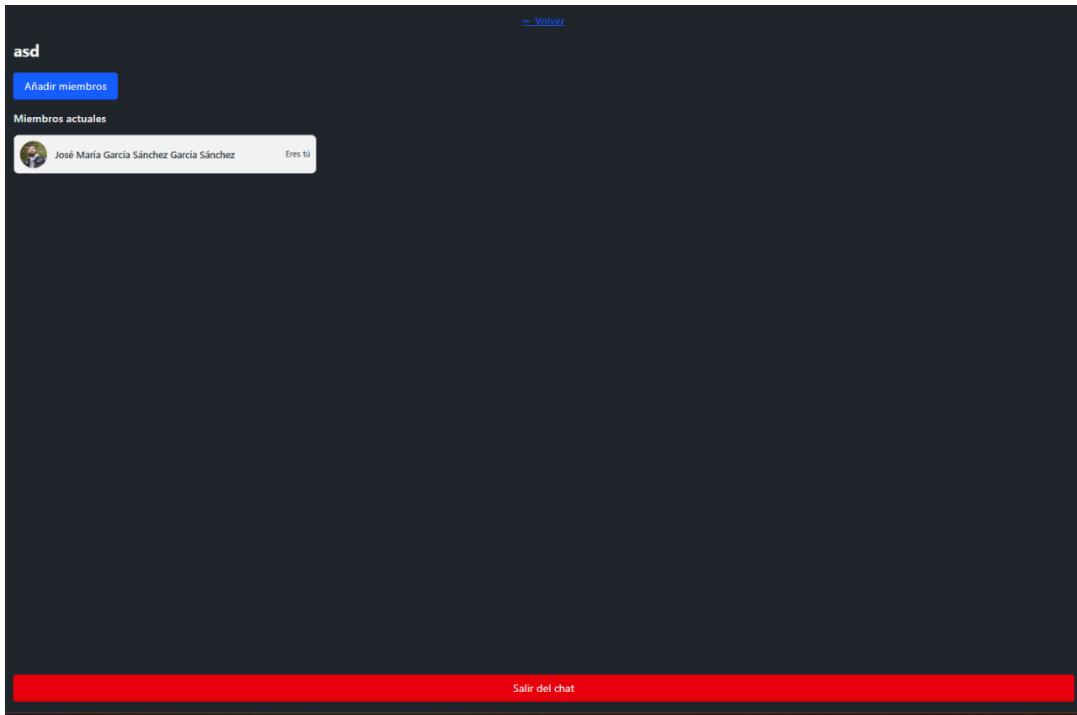


Imagen 9-7:Detalle de chat

Si eres administrador del grupo puedes añadir miembros. Y si quieres salir del mismo le das al botón de salir del grupo.

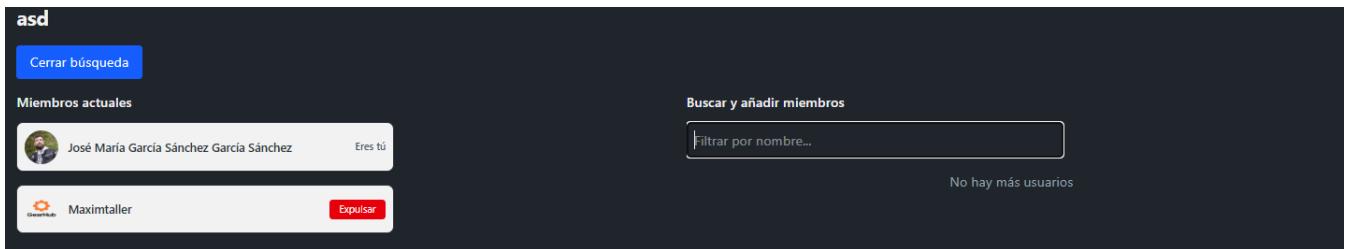


Imagen 9-8:Añadir o eliminar miembros

Si añades a un miembro y quieras eliminarlo del grupo le das al botón “Expulsar” de su tarjeta

10. Taller

La aplicación web está enfocada a los talleres por lo que estos tienen habilitada una página llamada taller en el que pueden organizar sus ordenes de mantenimiento y sus facturas.

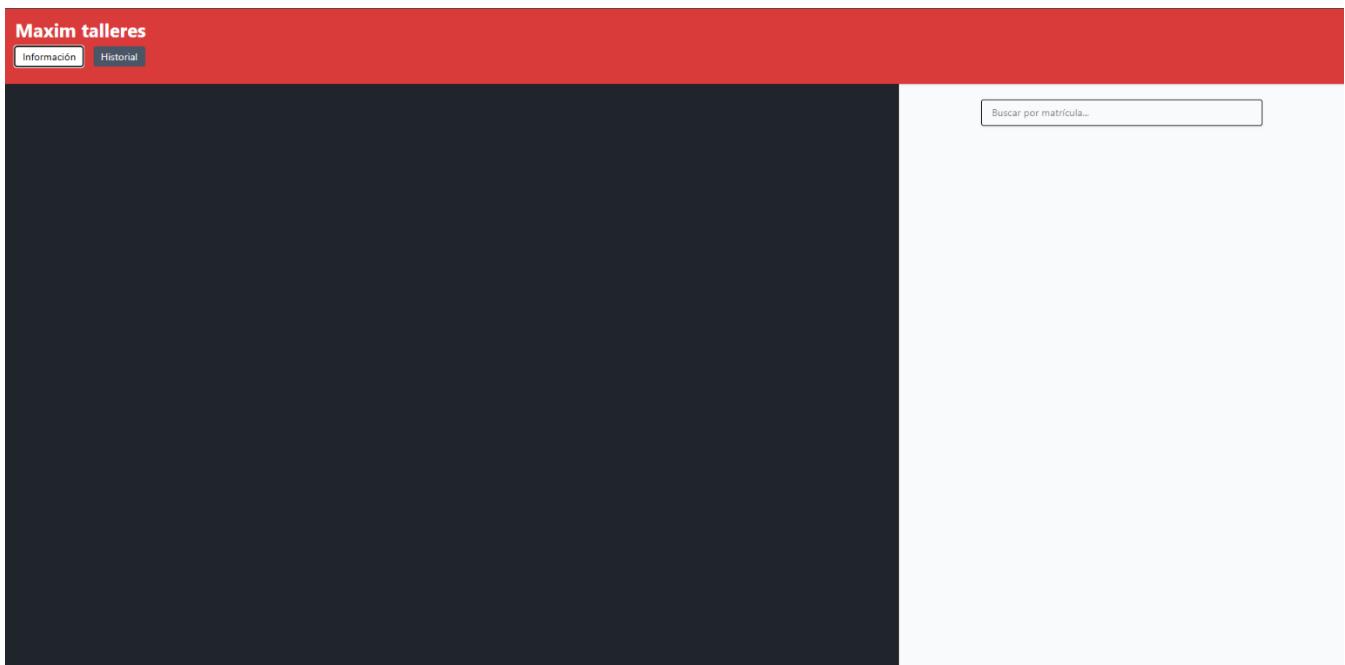


Imagen 10-1:Vista principal

INFORMACIÓN

En esta pestaña se encuentra el buscador por matrícula. Este permitirá ver las órdenes de mantenimiento que hemos realizado a un coche, el borrado de las mismas y la creación de órdenes nuevas.

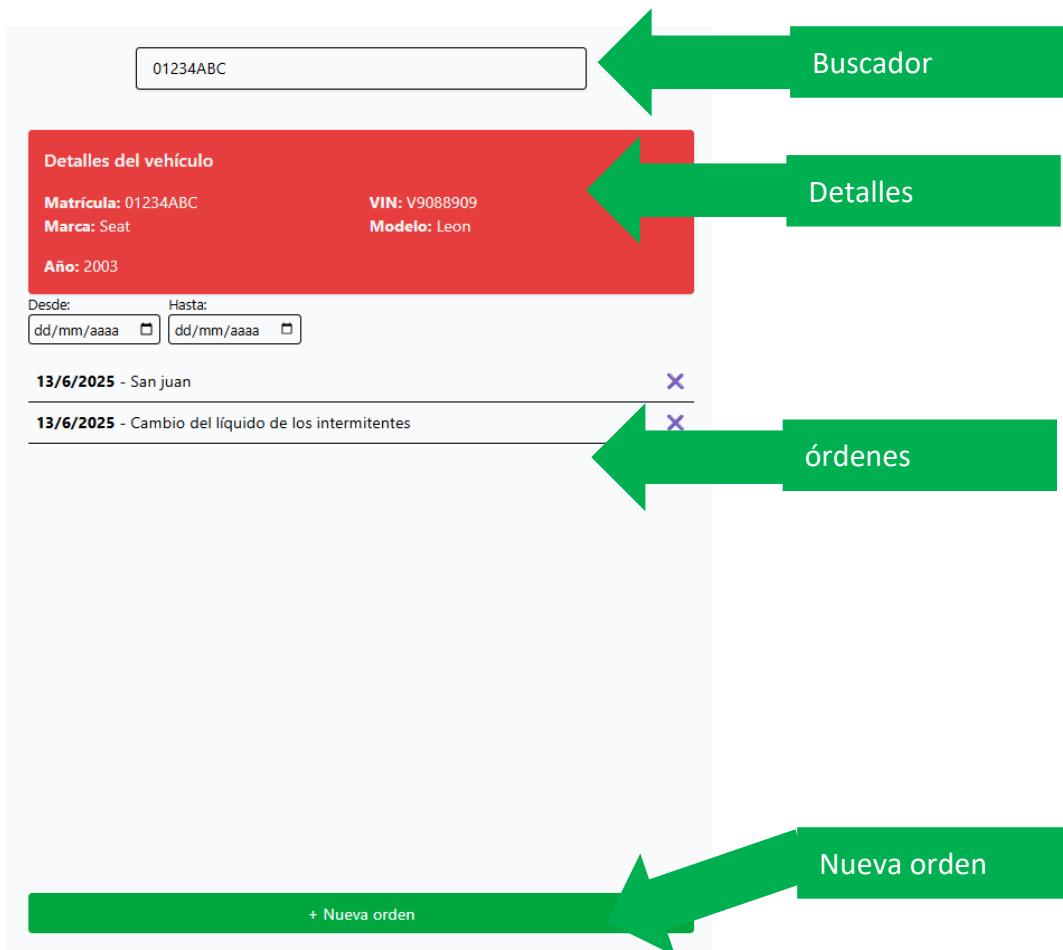


Imagen 10-2:Detalle vehículo

Al darle a nueva orden se nos abrirá la vista detalle en la que podremos hacer cambios sobre el vehículo y sus componentes

Descripción

Describe la orden de mantenimiento

Componentes principales + Añadir componente padre

Motor 1.9 TDI Eliminar

Intemitente derecho Eliminar

Componentes hijos + Añadir componente hijo

Válvula de inyección Eliminar

Cancelar Guardar orden

Imagen 10-3:Formulario de orden

Una vez guardemos la orden se nos abrirá la vista detalle con los componentes añadidos, la descripción y la posibilidad de crear una factura

Detalles de la orden

ID de la orden: 684c03868b18dfefa608f38b

Matrícula: 01234ABC

Fecha: 13/6/2025, 12:55:03

Descripción:

Componentes:
No hay componentes registrados

Factura asociada

Total (€):

Generar factura

Imagen 10-4:Detalle de factura

HISTORIAL

En historial podremos ver todas las órdenes que hemos ido generando con el tiempo. Podremos ver, como en información, la vista detalle, la factura y podremos eliminar tanto la factura como la orden

Maxim talleres

Información Historial

Historial de órdenes

Desde dd/mm/aaaa Hasta dd/mm/aaaa

13/6/2025 - San juan X

13/6/2025 - Cambio del líquido de los intermitentes X

13/6/2025 - X

Detalles de la orden

ID de la orden: 684bfaf88b18dfe608f386

Matrícula: 01234ABC

Fecha: 13/6/2025, 12:18:16

Descripción: San juan

Componentes:

- Motor 1.9 TDI: Motor del coche
- Válvula de inyección:

Factura asociada

ID: 684bfaf18b18dfe608f387

Total: 80 €

Fecha: 13/6/2025

Eliminar factura

Imagen 10-5:Historial de órdenes general

Anexo.II. Manual de usuario móvil

Aplicación Móvil

Creado por:

José María García Sánchez

Rodrigo Tapiador Cano



índice

<u>1.</u>	<u>REGISTRO Y LOGIN.....</u>	<u>123</u>
<u>2.</u>	<u>NAVEGACIÓN BÁSICA</u>	<u>125</u>
<u>3.</u>	<u>PERFIL</u>	<u>126</u>
<u>4.</u>	<u>VEHÍCULOS.....</u>	<u>127</u>
<u>5.</u>	<u>REVIEWS</u>	<u>128</u>
<u>6.</u>	<u>COMUNIDADES</u>	<u>130</u>
<u>7.</u>	<u>THREADS.....</u>	<u>131</u>
<u>8.</u>	<u>CHATS</u>	<u>133</u>

Registro y login

Inicio

Al iniciar la aplicación dependiendo de si es la primera vez que el usuario se loguea o si ya lo ha hecho con anterioridad pueden ocurrir diversos escenarios:

1. Primer inicio de la aplicación, pantalla de login:



Al iniciar la aplicación podemos ver la pantalla de inicio de sesión, aquí pondremos los datos si ya tenemos una cuenta y si no la tenemos le daremos a ¿No estás registrado?

2. Pantalla de registro

Si queremos registrarnos rellenaremos los datos de registro y al darle a registrarse nos mandará de vuelta a la pantalla de login, si intentamos logarnos sin haber confirmado el correo electrónico nos saltará un error pidiendo la confirmación de correo

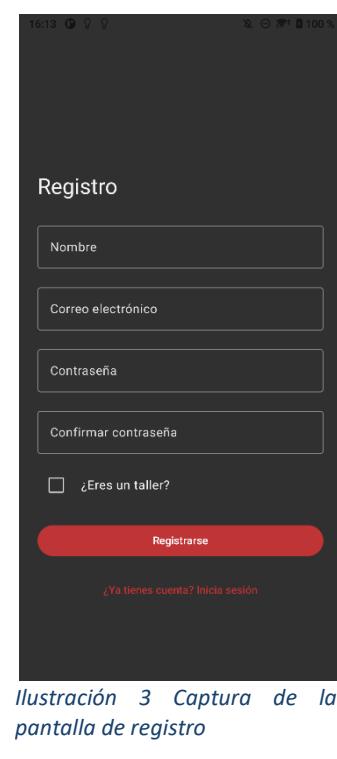




Ilustración 4 Captura de la pantalla de recuperar contraseña

3. Pantalla de recuperación de la contraseña

En esta pantalla debemos poner nuestro correo electrónico y nos llegará un email en el que podremos cambiar la contraseña, una vez cambiada ya podremos entrar a la aplicación con normalidad



Ilustración 5 Captura de la pantalla de creación de usuario

4. Pantalla de creación de usuario

Si nos logamos, pero no tenemos un perfil creado, nos saltará a esta pantalla donde podremos crear nuestro perfil:

Aquí podemos seleccionar una imagen de perfil desde el almacenamiento de nuestro dispositivo.

Navegación básica

Al iniciar sesión llegamos a la navegación principal de la aplicación, tenemos 4 pestañas en la parte inferior, pudiendo navegar entre home, usuarios, publicar y chats.

En la parte superior tenemos el menú lateral y el apartado de perfil:

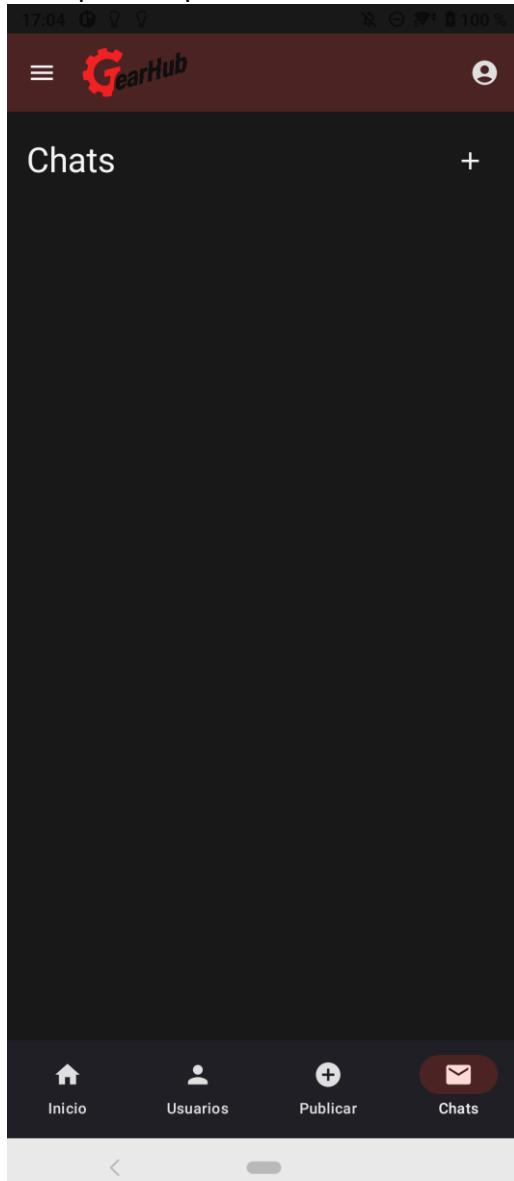


Ilustración 7 Captura para mostrar la navegacion de la alicacion

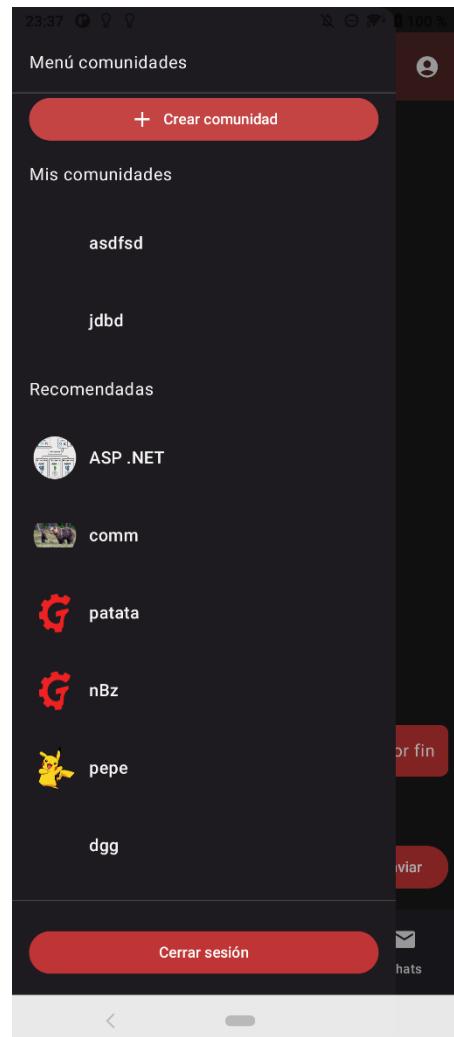
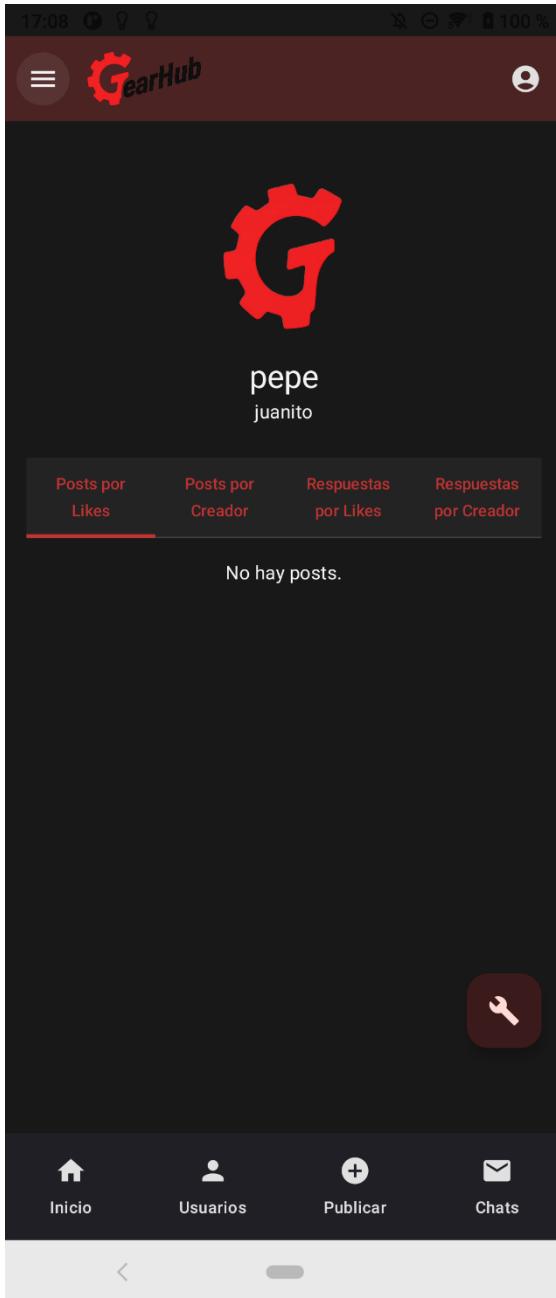


Ilustración 6 Captura para mostrar la navegacion lateral

Perfil

Si pulsamos en la parte superior derecha podemos acceder a nuestro perfil, ahí podemos encontrar varios elementos:



Tenemos los detalles del usuario, en este caso nosotros mismos. Podemos navegar entre los posts a los que le hemos dado like, los posts creados, las respuestas a las que le dimos like y las respuestas que hemos creado.

Abajo a la derecha se puede ver un botón flotante con el símbolo de una llave inglesa, ahí es donde podremos acceder a nuestros vehículos, en caso de que seamos un taller ahí accederemos a nuestras reviews.

Ilustración 8 Captura para mostrar la pantalla de perfil

Vehículos

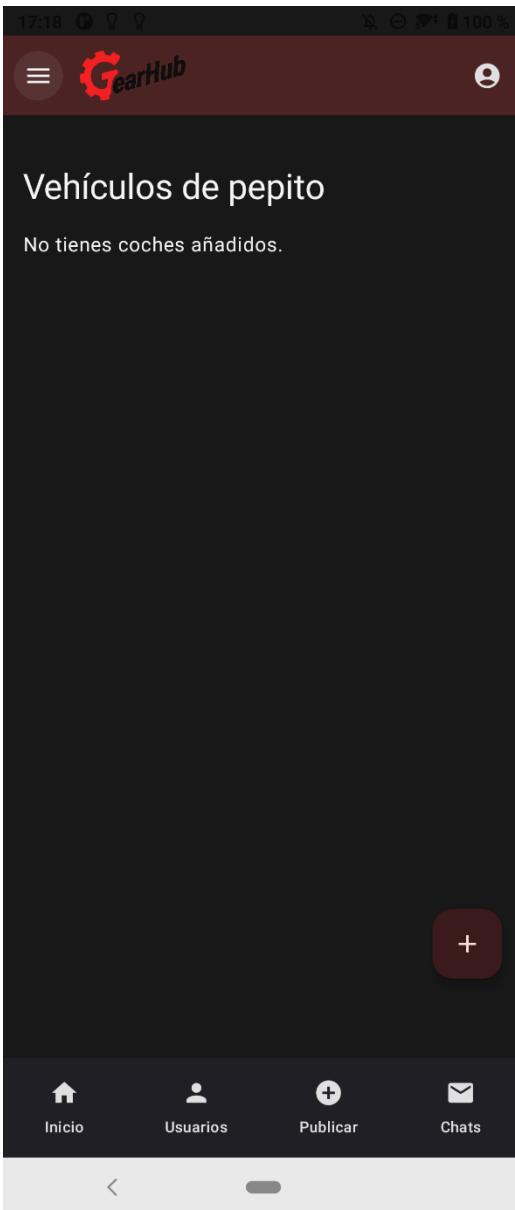


Ilustración 9 Captura para mostrar la pantalla de vehículos

En el apartado de vehículos podemos ver nuestros vehículos o los de el usuario que estemos mirando, si mantenemos pulsado en el vehículo y es nuestro perfil podremos editarlo o eliminarlo, en el botón flotante de abajo a la derecha podemos añadir un vehículo si es nuestro perfil:

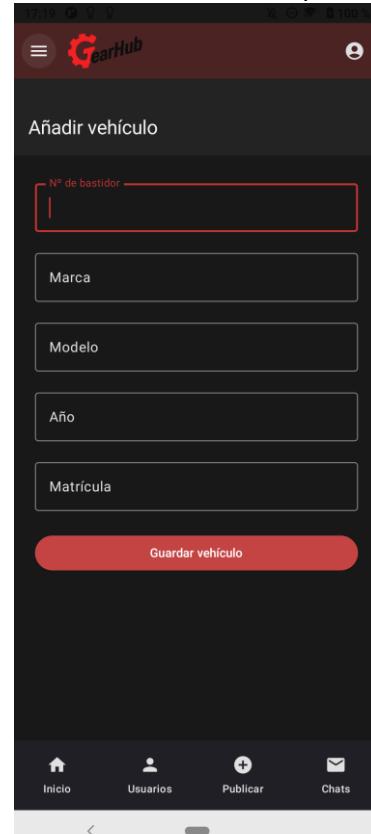


Ilustración 10 Captura para mostrar como añadir vehiculos

Reviews

Si accedemos al perfil de un taller (principalmente desde la pestaña Users) veremos abajo a la derecha el botón flotante con un corazón mediante el cual accederemos a las reviews.

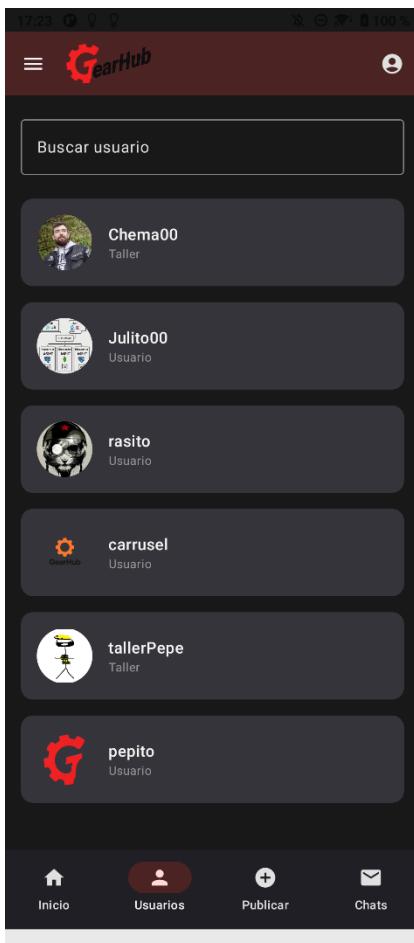


Ilustración 13 Captura para mostrar los usuarios y tipo de usuario

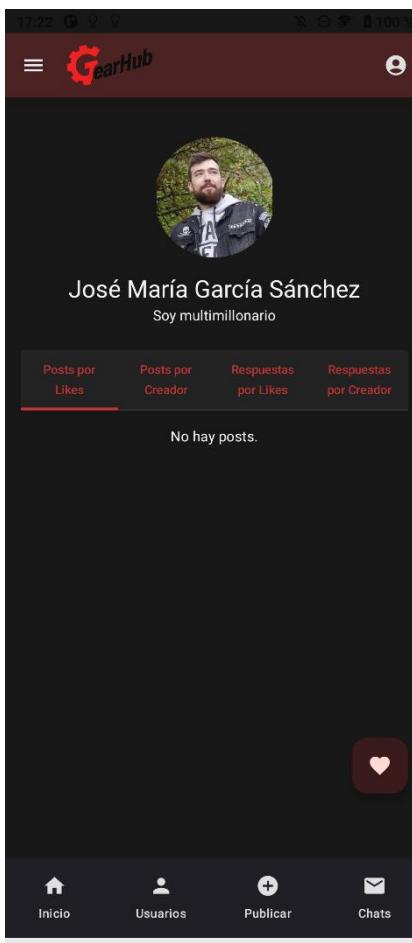


Ilustración 11 Captura para mostrar el detalle de un taller

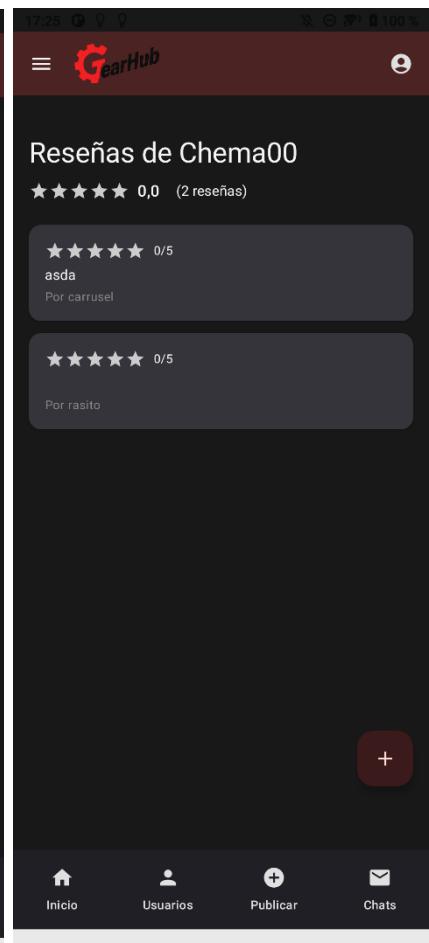


Ilustración 12 Captura para mostrar las reviews de un taller

Si somos talleres el botón flotante de añadir reseña no aparecerá, en caso de que sean reseñas de nuestro propio taller tendremos la opción de responder a la reseña.

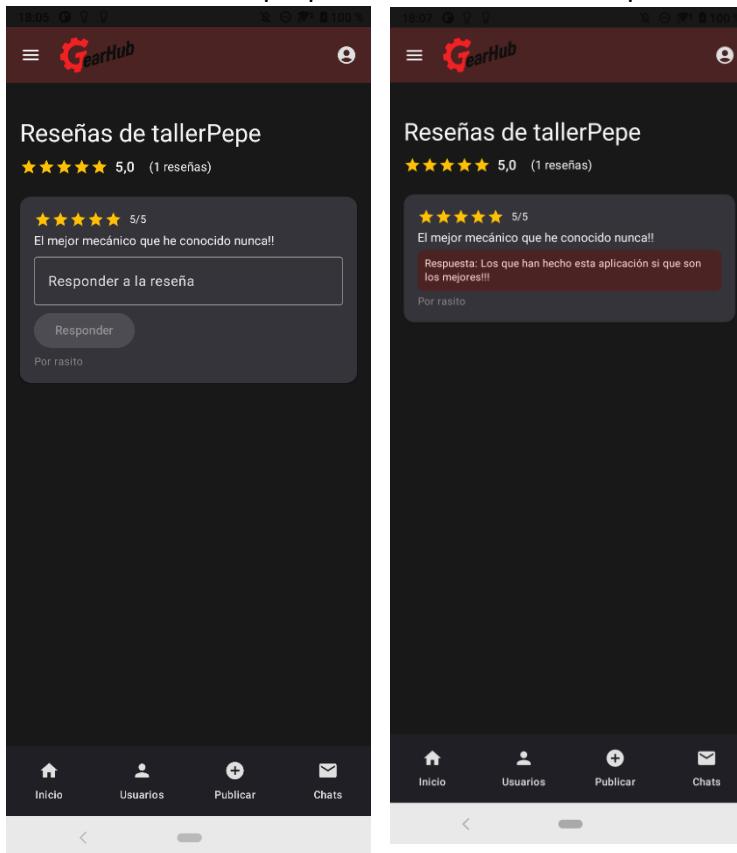


Ilustración 14 Captura para mostrar como responder a una reseña

Ilustración 15 Captura para mostrar la respuesta a una reseña

Comunidades

Al pulsar en el icono de menú de arriba a la izquierda se abre la lista de comunidades, divididas entre las comunidades de las que somos miembros o creadores y las comunidades recomendadas:

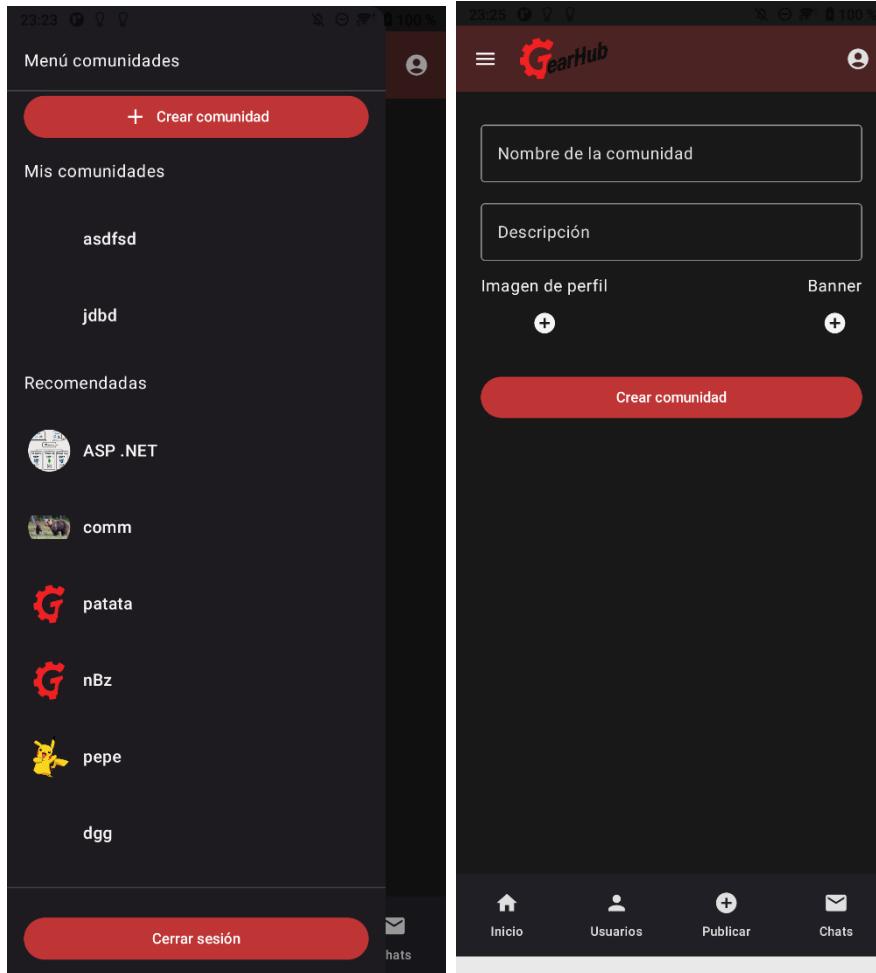


Ilustración 17 Captura para mostrar la lista de comunidades

Ilustración 16 Captura para mostrar la creacion de una comunidad

Podemos ver el botón de crear comunidad, si pulsamos en el se abre la ventana para crear comunidad, que nos permite seleccionar la imagen de banner y la foto de perfil:

Al pulsar en una de las comunidades podemos navegar a ella, entonces veremos el banner, la foto de perfil, descripción y todos los hilos que hay en la comunidad:

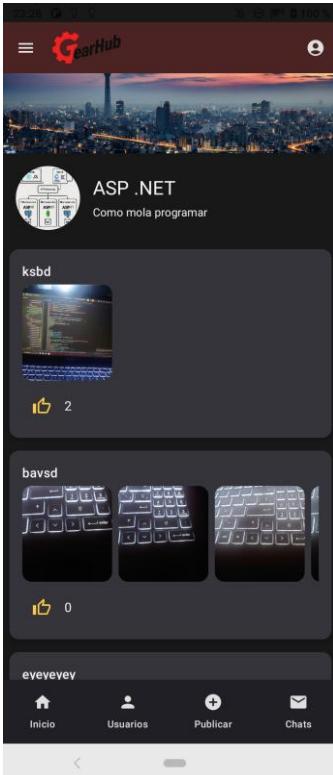


Ilustración 18 Captura para mostrar el detalle de una comunidad

Threads

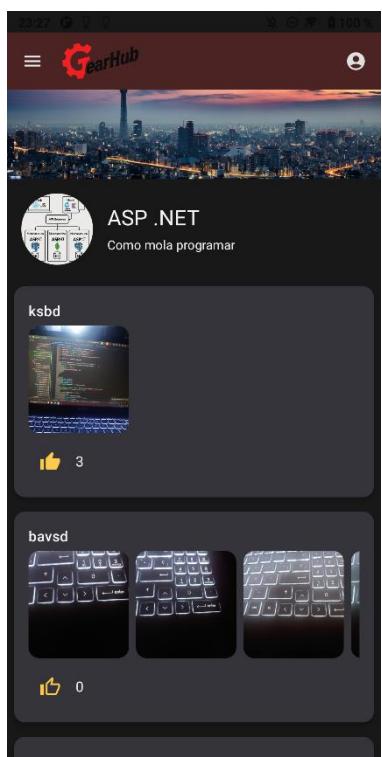


Ilustración 19 Captura para mostrar el detalle de una comunidad y los likes a los threads

Se puede dar like a un Thread: y podemos ver cuantos likes tiene, si somos los creadores del thread podremos ver un botón que nos permita modificar el thread.

Al pulsar en uno de los threads podemos acceder, además de a la publicación original, a todas sus respuestas, divididas en “hilos” que cuelgan unos de otros, dejando una estructura de respuestas ordenada.

Si somos los creadores del thread podemos modificarlo y si somos el creador de la respuesta podemos mantener pulsado en ella para editarla o borrarla

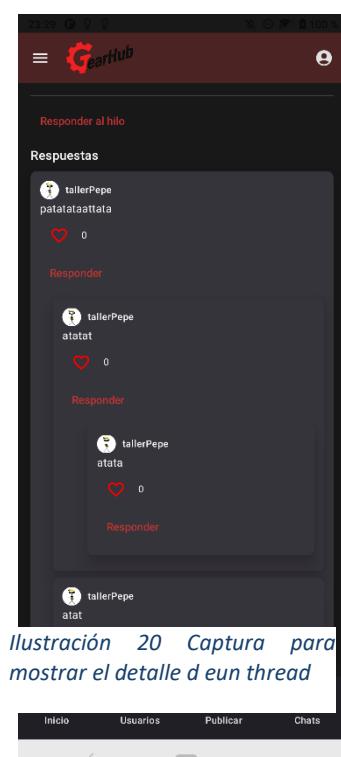


Ilustración 20 Captura para mostrar el detalle de un thread

Para publicar un thread tenemos la pantalla de publicaciones, desde esta pantalla podemos publicar un thread en cualquier comunidad, y se accede desde la barra de navegación inferior.



Ilustración 21 Captura para mostrar la publicación de un thread

Chats

Accederemos a los chats desde la barra de navegación inferior, donde podremos ver la lista de chats y un botón para crear un chat nuevo:

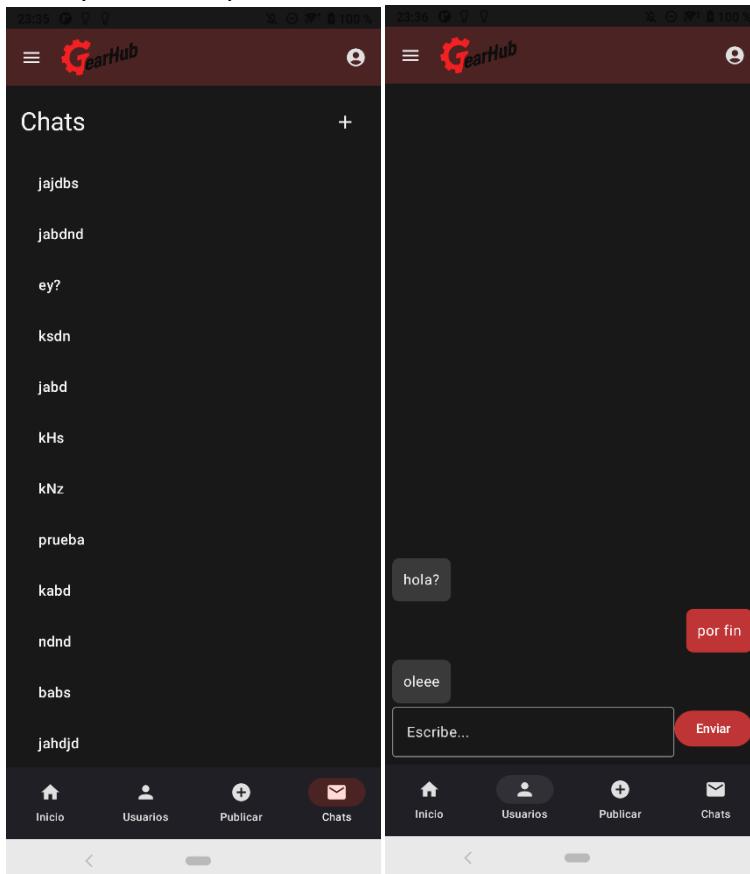


Ilustración 22 Captura para mostrar la lista de chats

Al pulsar en uno de los chats podemos acceder a los mensajes de este chat, aquí podremos conversar en tiempo real con otro usuario:

Ilustración 23 Captura para mostrar la ventana de chats

Anexo.III. Repositorios

Enlace a los repositorios de Git.

- [Api de autorización](#)
- [Api de comunidades](#)
- [Api de perfiles](#)
- [Api de reviews](#)
- [Api de mensajería](#)
- [Api de talleres](#)
- [Api Gateway](#)
- [Frontend web](#)
- [Frontend móvil](#)

Enlace a los repositorios de Docker.

- [Api de autorización](#)
- [Api de comunidades](#)
- [Api de perfiles](#)
- [Api de reviews](#)
- [Api de mensajería](#)
- [Api de talleres](#)
- [Api Gateway](#)
- [Frontend web](#)

Anexo.IV. Enlaces a las aplicaciones

[Aplicación web](#)

[Aplicación móvil](#)

Anexo.V. Docker-compose

```
1 services:
2   webapigateway:
3     image: rasito/webapigatewayserver:latest
4     container_name: webapigateway
5     ports:
6       - 25003:8080
7     networks:
8       - backend
9   postgres_messages:
10    image: postgres:latest
11    container_name: postgres_messages
12    environment:
13      POSTGRES_USER: postgres
14      POSTGRES_PASSWORD: postgres
15      POSTGRES_DB: postgres
16    volumes:
17      - postgres_messages:/var/lib/postgresql/data
18    networks:
19      - messages
20   web-api-messages:
21    image: rasito/webapimessages:latest
22    container_name: webapimessages
23    environment:
24      DbSettings_Host: postgres messages
25      DbSettings_Port: 5432
26      DbSettings_Username: postgres
27      DbSettings_Password: postgres
28      DbSettings_Database: postgres
29      JwtSettings_SecretKey: qwertyuiopasdfghjklzxcvbnmqwertyuio
30      depends_on:
31        - postgres_messages
32    networks:
33      - messages
34      - backend
35   auth_database:
36    image: postgres:latest
37    container_name: database_authentication
38    environment:
39      POSTGRES_USER: common
40      POSTGRES_PASSWORD: auth
41      POSTGRES_DB: Authapi
42    volumes:
43      - postgres auth data:/var/lib/postgresql/data
44    networks:
45      - authentication
46   auth_api:
47    image: chema00/authapi:latest
48    container_name: authentication api
49    environment:
50      DbSettings_Host: auth database
51      DbSettings_Port: 5432
52      DbSettings_Username: common
53      DbSettings_Password: auth
54      DbSettings_Database: Authapi
55      JwtSettings_SecretKey: qwertyuiopasdfghjklzxcvbnmqwertyuio
56      EmailSettings_UserEmail: noresponseapp@gmail.com
57      EmailSettings_UserName: onboarding@resend.dev
58      EmailSettings_UserApiKey:
59      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
60      EmailSettings_Host: vms.iesluisvives.org:25002
61    depends_on:
62      - auth_database
63    networks:
64      - authentication
      - backend
```

```
65 com database:
66   image: postgres:latest
67   container_name: database_community
68   environment:
69     POSTGRES_USER: common
70     POSTGRES_PASSWORD: com
71     POSTGRES_DB: Comapi
72   volumes:
73     - postgres_com_data:/var/lib/postgresql/data
74   networks:
75     - communities
76 com api:
77   image: chema00/comunapi:latest
78   container_name: community_api
79   environment:
80     DbSettings__Host: com_database
81     DbSettings__Port: 5432
82     DbSettings__Username: common
83     DbSettings__Password: com
84     DbSettings__Database: Comapi
85     JwtSettings__SecretKey: qwertyuiopasdfghjklzxcvbnmqwertyuio
86   depends_on:
87     - com_database
88   volumes:
89     - community_images:/app/community_images
90   networks:
91     - communities
92     - backend
93 prof database:
94   image: postgres:latest
95   container_name: database_prof
96   environment:
97     POSTGRES_USER: prof
98     POSTGRES_PASSWORD: prof
99     POSTGRES_DB: Prof
100  volumes:
101    - postgres_prof_data:/var/lib/postgresql/data
102  networks:
103    - prof
104 prof_api:
105   image: chema00/profapi:latest
106   container_name: prof_api
107   environment:
108     DbSettings__Host: prof_database
109     DbSettings__Port: 5432
110     DbSettings__Username: prof
111     DbSettings__Password: prof
112     DbSettings__Database: Prof
113     JwtSettings__SecretKey: qwertyuiopasdfghjklzxcvbnmqwertyuio
114   depends_on:
115     - prof_database
116   volumes:
117     - profile_images:/app/profile_images
118   networks:
119     - prof
120     - backend
121 rev database:
122   image: postgres:latest
123   container_name: database_rev
124   environment:
125     POSTGRES_USER: rev
126     POSTGRES_PASSWORD: rev
127     POSTGRES_DB: Rev
128   volumes:
129     - postgres_rev_data:/var/lib/postgresql/data
130   networks:
131     - rev
132 rev_api:
133   image: chema00/revapi:latest
134   container_name: rev_api
135   environment:
136     DbSettings__Host: rev_database
137     DbSettings__Port: 5432
138     DbSettings__Username: rev
139     DbSettings__Password: rev
140     DbSettings__Database: Rev
141     JwtSettings__SecretKey: qwertyuiopasdfghjklzxcvbnmqwertyuio
142   depends_on:
143     - rev_database
```

```
144     networks:
145         - rev
146         - backend
147
mongodb:
148     image: mongo:4.2.1B
149     container_name: mongo
150     environment:
151         MONGO_INITDB_ROOT_USERNAME: root
152         MONGO_INITDB_ROOT_PASSWORD: example
153         MONGO_INITDB_DATABASE: mydatabase
154     volumes:
155         - mongo_workshop_data:/data/db
156
networks:
157     - workshop
158
webapitaller:
159     image: rasito/webapiworkshop:latest
160     container_name: workshop_api
161     depends_on:
162         - mongodb
163     environment:
164         DbSettings Host: mongo
165         DbSettings Port: 27017
166         DbSettings Username: root
167         DbSettings Password: example
168         DbSettings Database: mydatabase
169         JwtSettings SecretKey: qwertyuiopasdfghjklzxcvbnmqwertyuioasdfghjklzxcvbnm
170     networks:
171         - workshop
172         - backend
173
web frontend:
174     image: chema00/gearhub:latest
175     container_name: webfrontend
176     ports:
177         - 25002:80
178     networks:
179         - backend
180
networks:
181     backend:
182     messages:
183     authentication:
184     communities:
185     prof:
186     rev:
187     workshop:
188
volumes:
189     postgres messages:
190     postgres auth data:
191     postgres com data:
192     postgres prof data:
193     postgres rev data:
194     mongo workshop data:
195     profile images:
196     community images:
```