

PROYECTO FIN DE CICLO FORMATIVO



IES GINÉS PÉREZ CHIRINOS
DEPARTAMENTO DE INFORMÁTICA

TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

GuilNet: Red Social de Comunidades Temáticas

AUTOR(ES)

D. Álvaro Gómez Alfaro

TUTOR

D. María Teresa Rubio Moreno

CARAVACA DE LA CRUZ, 10 DE ABRIL DE 2025



Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License

ÍNDICE GENERAL

1.	INTRODUCCIÓN	1
2.	ANÁLISIS	3
2.1.	OBJETIVOS	3
2.2.	REQUISITOS FUNCIONALES.....	3
2.3.	CASOS DE USO	4
2.4.	PLANIFICACIÓN.....	9
3.	DISEÑO DEL PROYECTO.....	11
3.1.	MARCO DE TRABAJO Y HERRAMIENTAS.....	11
3.2.	ARQUITECTURA.....	12
3.3.	INTERFACES GRÁFICAS	15
4.	DESARROLLO	21
4.1.	PROGRAMACIÓN	21
4.2.	PRUEBAS	24
4.3.	CONTROL DE VERSIONES Y REPOSITORIO.....	33
5.	MANUAL DE PUESTA EN MARCHA.....	35
5.1.	REQUISITOS PREVIOS	35
5.2.	CLONAR EL PROYECTO.....	35
5.3.	CONFIGURAR Y EJECUTAR EL BACKEND (SPRING BOOT)	36
5.4.	CONFIGURAR Y EJECUTAR EL FRONTEND (ANGULAR).....	36
5.5.	USUARIO DE PRUEBA	36
5.6.	CAPTURAS DE PANTALLA REPRESENTATIVAS	37
6.	CONCLUSIONES Y VÍAS FUTURAS.....	43
6.1.	CONCLUSIONES.....	43
6.2.	VÍAS FUTURAS Y MEJORAS	43
6.3.	OBJETIVOS NO DESARROLLADOS	44
7.	REFERENCIAS Y BIBLIOGRAFÍA.....	45
7.1.	RECURSOS TÉCNICOS Y DOCUMENTACIÓN UTILIZADA.....	45
7.2.	APUNTES Y MATERIAL DEL CICLO	46

1. INTRODUCCIÓN

Este proyecto consiste en el desarrollo de una **red social temática** centrada en la creación y gestión de comunidades. El objetivo principal es permitir que los usuarios interactúen de forma organizada según sus intereses, creando espacios personalizados en los que puedan expresarse, compartir contenido y comunicarse con otros usuarios.

La aplicación se basa en un enfoque **modular, dinámico y personalizable**, permitiendo a los usuarios tener un perfil general básico y perfiles independientes dentro de cada comunidad a la que pertenezcan. El contenido publicado en las comunidades tendrá un formato enriquecido tipo blog, con soporte para texto, imágenes, lo que permite una experiencia más completa y creativa.

2. ANÁLISIS

2.1. OBJETIVOS

- O1: Permitir a los usuarios registrarse, iniciar sesión y gestionar su perfil general básico.
- O2: Permitir crear comunidades temáticas y unirse a ellas mediante un sistema de perfiles personalizados.
- O3: Facilitar la publicación de contenido enriquecido (tipo blog) dentro de cada comunidad.
- O4: Permitir comentar publicaciones y reaccionar con likes.
- O5: Gestionar roles dentro de las comunidades, permitiendo definir permisos personalizados.
- O6: Ofrecer un sistema de notificaciones para interacciones y eventos importantes.
- O7: Ofrecer un sistema de chat grupal dentro de cada comunidad.
- O8: Proporcionar un entorno visual atractivo, funcional y responsive mediante Angular y Quill.js.

2.2. REQUISITOS FUNCIONALES

- RF1: El sistema debe permitir a los usuarios registrarse con nombre de usuario, correo electrónico y contraseña.
- RF2: El sistema debe permitir a los usuarios iniciar sesión con sus credenciales.
- RF3: El sistema debe permitir editar los datos básicos del perfil global (nombre, correo, imagen).
- RF4: El sistema debe permitir la creación de comunidades con nombre, imagen, banner, descripción y reglas.
- RF5: El sistema debe permitir buscar comunidades y unirse a ellas.
- RF6: El sistema debe crear automáticamente un perfil personalizado para el usuario al unirse a una comunidad.
- RF7: El sistema debe permitir editar el perfil dentro de una comunidad con nombre, imagen y descripción enriquecida.
- RF8: Los líderes o moderadores deben poder asignar roles y títulos a los miembros de su comunidad.

- RF9: Los usuarios deben poder crear publicaciones dentro de comunidades con formato enriquecido.
- RF10: El sistema debe permitir editar o eliminar las publicaciones propias.
- RF11: Los usuarios deben poder comentar publicaciones dentro de las comunidades.
- RF12: Los usuarios deben poder dar «me gusta» a publicaciones dentro de comunidades.
- RF13: El sistema debe generar notificaciones ante likes, comentarios, asignación de títulos.
- RF14: El sistema debe permitir comunicarse en tiempo real en el chat general de cada comunidad mediante WebSockets.
- RF15: El sistema debe permitir eliminar o editar los comentarios en el perfil de comunidad del usuario.
- RF16: El sistema debe restringir las acciones según el rol y permisos del usuario dentro de cada comunidad.

2.3. CASOS DE USO

CU1 – Registrarse en la plataforma

Descripción: El usuario accede al formulario de registro, introduce sus datos personales y crea una cuenta en la plataforma. **Actor principal:** Usuario no autenticado **Flujo principal:**

1. El usuario abre el formulario de registro.
2. Introduce nombre de usuario, correo electrónico, contraseña y datos mínimos de perfil.
3. El sistema valida que el correo no esté en uso.
4. Si es válido, se crea la cuenta y se inicia sesión automáticamente o se envía correo de verificación (opcional). **Resultado esperado:** Se crea una cuenta nueva y el usuario accede a la plataforma.

CU2 – Iniciar sesión

Descripción: Un usuario registrado accede a la plataforma introduciendo su correo electrónico y contraseña. **Actor principal:** Usuario registrado **Flujo principal:**

1. El usuario accede al formulario de inicio de sesión.
2. Introduce su correo y contraseña.
3. El sistema valida las credenciales.
4. Si son correctas, inicia sesión y accede al panel principal. **Resultado esperado:** El usuario entra en la plataforma con su cuenta

CU3 – Editar el perfil global del usuario

Descripción: El usuario modifica los datos básicos de su perfil global: nombre de usuario, correo e imagen de perfil. **Actor principal:** Usuario autenticado **Flujo principal:**

1. Accede a su perfil.
2. Pulsa en editar.
3. Cambia los datos básicos.
4. Guarda los cambios. **Resultado esperado:** Los cambios se reflejan como base para nuevos perfiles en comunidades.

CU4 – Crear una comunidad

Descripción: El usuario crea una nueva comunidad temática y se convierte automáticamente en su líder. **Actor principal:** Usuario autenticado **Flujo principal:**

1. Accede a la sección de comunidades.
2. Pulsa en "Crear comunidad".
3. Rellena los datos: nombre, descripción, imagen, banner, reglas.
4. Envía el formulario.
5. Se crea también automáticamente el chat asociado. **Resultado esperado:** La comunidad aparece en el sistema y el usuario pasa a ser su líder.

CU5 – Buscar y unirse a una comunidad

Descripción: Permite al usuario encontrar comunidades existentes y unirse a ellas. **Actor principal:** Usuario autenticado **Flujo principal:**

1. Accede a la sección de búsqueda de comunidades.
2. Usa filtros o buscador.
3. Entra en la comunidad deseada.
4. Pulsa "Unirse".
5. El sistema crea un perfil del usuario dentro de esa comunidad. **Resultado esperado:** El usuario forma parte de la comunidad y puede interactuar.

CU6 – Gestionar el perfil dentro de una comunidad

Descripción: El usuario puede modificar su perfil personalizado dentro de una comunidad. **Actor principal:** Usuario autenticado y miembro de una comunidad **Flujo principal:**

1. Entra a la comunidad.
2. Accede a su perfil.
3. Pulsa en editar.
4. Cambia datos como nombre, imagen, descripción, etc.
5. Guarda los cambios. **Resultado esperado:** El perfil se actualiza dentro de esa comunidad.

CU7 – Asignar roles y títulos dentro de una comunidad

Descripción: Permite a líderes o moderadores asignar roles y títulos a los miembros. **Actor principal:** Usuario con rol de líder o moderador **Flujo principal:**

1. Accede a la configuración de la comunidad.
2. Selecciona un miembro.
3. Asigna o modifica su rol.
4. Añade uno o varios títulos. **Resultado esperado:** El usuario tiene nuevos permisos y títulos visibles en su perfil y publicaciones.

CU8 – Crear una publicación en una comunidad

Descripción: El usuario crea una publicación tipo blog dentro de una comunidad. **Actor**

principal: Usuario autenticado y miembro de la comunidad **Flujo principal:**

1. Entra a la comunidad.
2. Pulsa en "Nueva publicación".
3. Escribe contenido con el editor.
4. Publica. **Resultado esperado:** La publicación aparece en el feed de la comunidad.

CU9 – Editar o eliminar una publicación propia

Descripción: El autor de una publicación puede editarla o eliminarla. **Actor**

principal: Usuario autenticado y autor de la publicación **Flujo principal:**

1. Localiza su publicación.
2. Pulsa en editar o eliminar.
3. Realiza los cambios o confirma la eliminación. **Resultado esperado:** La publicación se actualiza o se elimina.

CU10 – Comentar en una publicación

Descripción: Los usuarios pueden comentar publicaciones dentro de una comunidad. **Actor principal:** Usuario autenticado y miembro de la comunidad **Flujo**

principal:

1. Visualiza una publicación.
2. Escribe un comentario.
3. Envía el comentario. **Resultado esperado:** El comentario queda visible para otros usuarios.

CU11 – Dar «me gusta» a una publicación

Descripción: El usuario puede reaccionar positivamente a una publicación. **Actor**

principal: Usuario autenticado y miembro de la comunidad **Flujo principal:**

1. Visualiza una publicación.
2. Pulsa el botón de like.

3. El sistema registra o elimina el like. **Resultado esperado:** La publicación muestra el número actualizado de likes.

CU12 – Recibir y consultar notificaciones

Descripción: El usuario recibe alertas por interacciones importantes. **Actor principal:** Usuario autenticado **Flujo principal:**

1. Accede al sistema o permanece conectado.
2. Recibe notificaciones.
3. Visualiza o marca como leídas. **Resultado esperado:** El usuario está informado sobre eventos importantes.

CU13 – Usar el chat en tiempo real de una comunidad

Descripción: Permite a los miembros comunicarse mediante un chat general en tiempo real. **Actor principal:** Usuario autenticado y miembro de la comunidad **Flujo principal:**

1. Accede a la comunidad.
2. Entra a la sección de chat.
3. Escribe y envía mensajes. **Resultado esperado:** Los mensajes se envían y reciben en tiempo.

CU14 – Eliminar o editar comentarios del perfil de comunidad

Descripción: Permite a los usuarios eliminar comentarios recibidos o editar los propios dentro de su perfil de comunidad. **Actor principal:** Usuario autenticado **Flujo principal:**

1. Accede a su perfil en una comunidad.
2. Localiza el comentario.
3. Pulsa "Eliminar" o "Editar". **Resultado esperado:** El comentario desaparece o se actualiza.

2.4. PLANIFICACIÓN

Semana	Actividad
1	Análisis, diseño y configuración inicial
3-4	Desarrollo de entidades y funcionalidades en el back-end con Spring Boot
5-6	Desarrollo del front-end con Angular
3-4	Pruebas unitarias e integración
3	Documentación y entrega final

3. DISEÑO DEL PROYECTO

3.1. MARCO DE TRABAJO Y HERRAMIENTAS

Para el desarrollo de este proyecto he utilizado un conjunto de herramientas y tecnologías modernas, organizadas bajo una arquitectura cliente-servidor con separación clara entre frontend y backend. A continuación, se detallan los componentes más relevantes:

Lenguajes de programación

- **Java:** lenguaje principal para el desarrollo del backend.
- **TypeScript:** lenguaje principal del frontend mediante el uso del framework Angular.
- **HTML y CSS:** para la estructura y estilos de la interfaz gráfica.

Entornos de desarrollo

- **IntelliJ IDEA:** IDE utilizado para el desarrollo del backend con Spring Boot.
- **Visual Studio Code:** editor utilizado para el desarrollo del frontend con Angular.

Frameworks y librerías

- **Spring Boot:** Framework base del backend.
- **Spring Security + JWT:** para la implementación de autenticación y autorización segura basada en tokens.
- **Spring WebSocket + STOMP:** para la funcionalidad de chat en tiempo real.
- **Spring Data JPA + Hibernate:** para la persistencia de datos y el mapeo objeto-relacional.
- **Lombok:** para reducir el código en las entidades y clases DTO.
- **Angular:** framework frontend para construir una SPA (Single Page Application).
- **SockJS y @stomp/ng2-stompjs:** librerías de Angular utilizadas para conectar con WebSocket de forma compatible.
- **Bootstrap:** para los estilos de la interfaz gráfica

Base de datos

- **MySQL:** sistema gestor de bases de datos relacional utilizado para almacenar todos los datos de la aplicación.

Otros componentes

- **Maven:** herramienta de gestión de dependencias y construcción del proyecto backend.
- **Git + GitHub:** para el control de versiones y colaboración.
- **Postman:** herramienta utilizada para probar y documentar los endpoints de la API.
- **Draw.io:** empleada para la elaboración de diagramas.

Sistema operativo

- **Windows 11:** entorno principal de desarrollo.

3.2. ARQUITECTURA

El proyecto se ha diseñado siguiendo una arquitectura en capas basada en el patrón cliente-servidor, con una clara separación entre el frontend (cliente), el backend (servidor) y la base de datos. Este enfoque permite una mayor modularidad, escalabilidad y facilidad de mantenimiento.

Estructura general

La aplicación está dividida en tres capas principales:

- **Frontend (cliente web):** desarrollado con Angular y TypeScript. Es responsable de la interfaz de usuario, las interacciones con el usuario y el consumo de la API REST expuesta por el backend.
- **Backend (servidor):** implementado con Java y Spring Boot. Expone una API RESTful y gestiona la lógica de negocio, la persistencia de datos y la seguridad. También incorpora comunicación en tiempo real mediante WebSockets.
- **Base de datos:** se utiliza MySQL como sistema de gestión de bases de datos relacional para almacenar la información del sistema: usuarios, comunidades, publicaciones, comentarios, perfiles, mensajes, etc.

Comunicación entre capas

- La comunicación entre el frontend y el backend se realiza mediante llamadas HTTP a los endpoints REST, y mediante WebSockets para el sistema de chat en tiempo real.

- El backend accede a la base de datos a través de JPA (Java Persistence API).

3.2.1. Diagrama entidad-relación

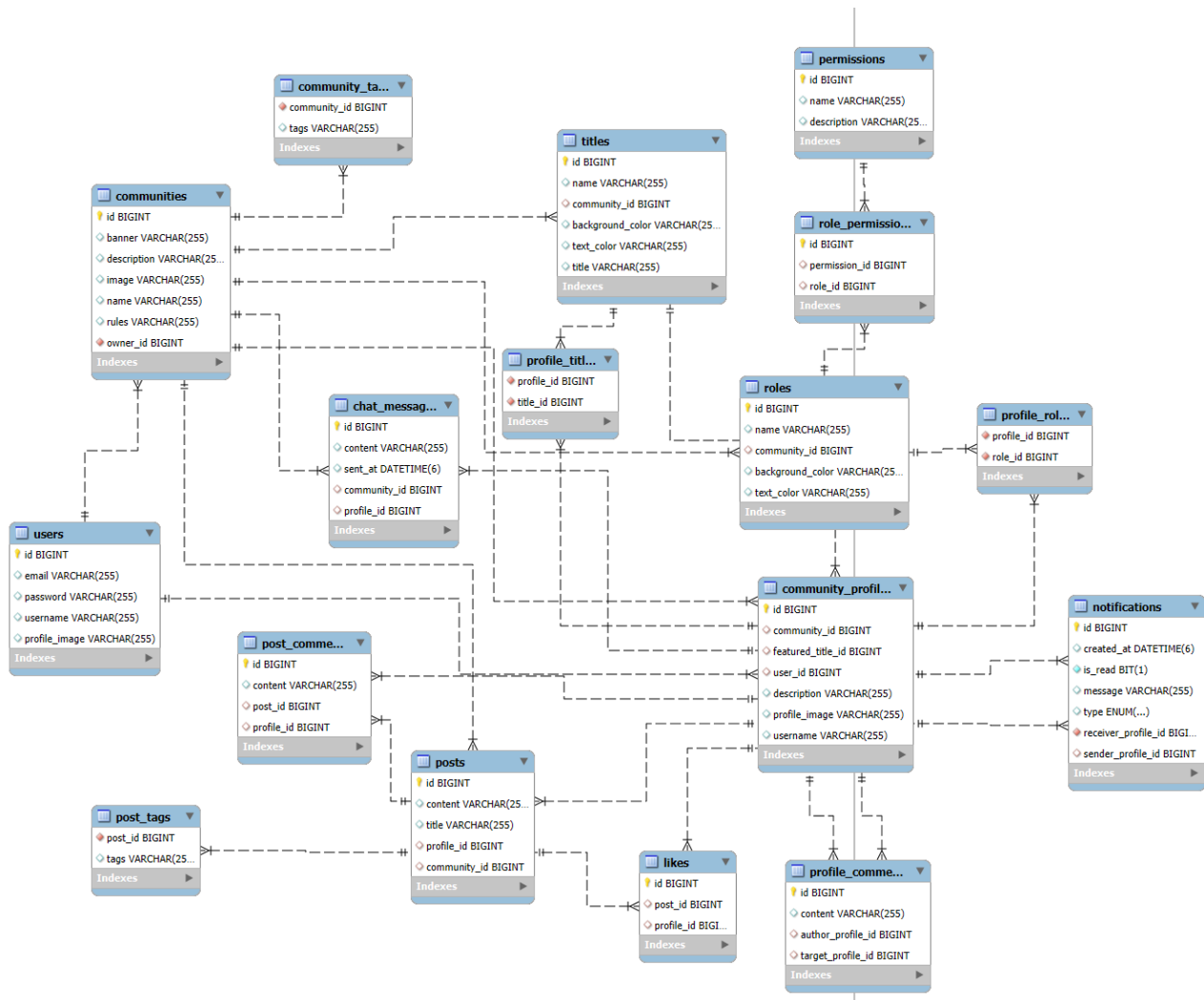
El modelo relacional es utilizado para generar la base de datos, y se fundamenta en la representación de los datos a través de tablas, donde cada tabla contiene diferentes atributos dispuestos en columnas. Los vínculos entre las tablas se establecen mediante el uso de claves primarias y claves foráneas o ajenas. En el siguiente diagrama observaremos las entidades relevantes que componen la totalidad del sistema



- **Chat messages:** Almacena los mensajes enviados en los chats de cada comunidad por los perfiles de comunidad.
- **Communities:** Almacena los datos principales de cada comunidad creada en la red social, incluyendo nombre, descripción y su propietario.
- **Community profiles:** Representa los perfiles que los usuarios tienen dentro de cada comunidad, vinculándolos a usuarios y comunidades.
- **Community tags:** Permite asignar etiquetas o categorías a las comunidades.
- **Likes:** Registra los 'me gusta' que los perfiles dan a las publicaciones dentro de las comunidades.
- **Notifications:** Guarda las notificaciones entre perfiles, indicando quién la envía, quien la recibe y su tipo.
- **Permissions:** Agrupa los permisos específicos que pueden estar asociados a los distintos roles del sistema.
- **Post comments:** Guarda los comentarios realizados en publicaciones por parte de los perfiles de comunidad.
- **Post tags:** Permite etiquetar las publicaciones con una o varias etiquetas de texto libre.
- **Posts:** Contiene las publicaciones realizadas por los perfiles dentro de las comunidades.
- **Profile comments:** Guarda comentarios que un perfil puede dejar en el perfil de otro usuario dentro de una comunidad.
- **Profile roles:** Relaciona los perfiles de comunidad con los roles que tienen dentro de una comunidad concreta.
- **Profile titles:** Relaciona los perfiles de comunidad con los títulos que pueden tener, permitiendo una relación N:M.
- **Role permissions:** Relaciona los roles con los permisos que tienen asignados, estableciendo una relación N:M entre ambos.
- **Roles:** Define los distintos roles disponibles para los usuarios o perfiles de comunidad.
- **Titles:** Contiene los títulos personalizables que pueden asignarse a los perfiles de comunidad como distintivo.
- **Users:** Contiene los datos de los usuarios registrados en la aplicación, incluyendo información personal y de acceso.

3.2.2. Modelo Lógico Relacional.

Este es el modelo relacional resultado de transformar el modelo entidad relación a modelo relacional.



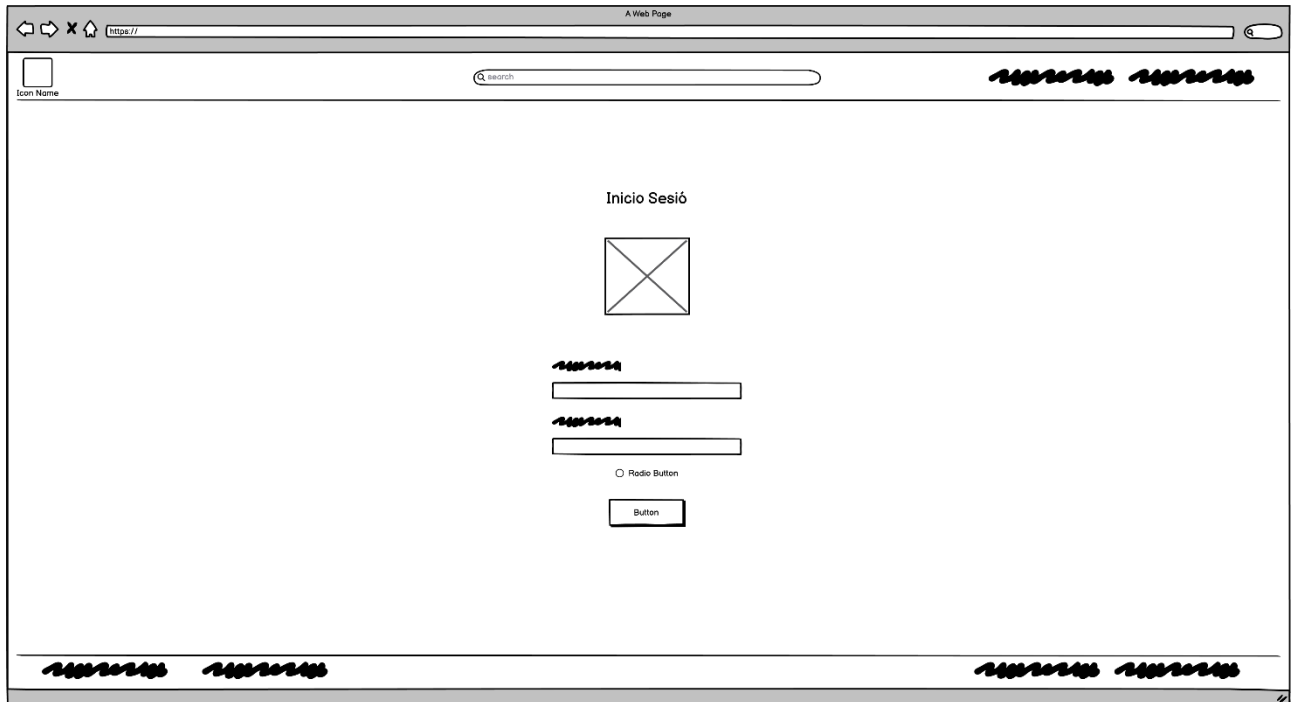
3.3. INTERFACES GRÁFICAS

Para el diseño de la aplicación web he optado por un diseño muy minimalista y fácil de entender para el usuario final. Todos los textos tienen un tamaño legible y la información en pantalla es la justa para que el usuario tenga suficiente información y a la vez no sea demasiada para no saturar al usuario, de esta forma la experiencia de usuario será agradable y fluida.

En cuanto al diseño de las interfaces, estas se componen principalmente por un header en la parte superior con las opciones de navegación necesarias para usar de forma sencilla la aplicación, el cuerpo; situado en la parte central de la aplicación; es donde se muestra la mayor parte de la aplicación.

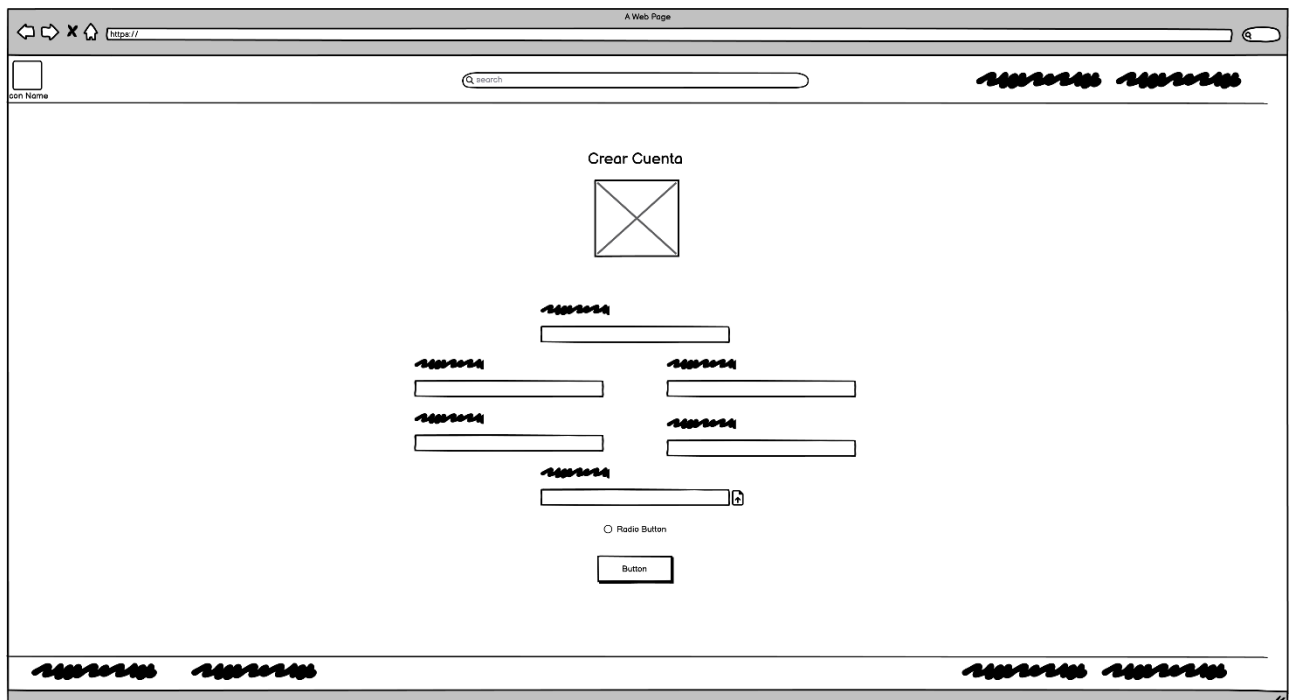
La última parte de la interfaz es el pie de página encontrándose en la parte más baja de la aplicación se encuentra un pequeño footer con un poco de información sobre la

aplicación. Para finalizar este apartado mostraremos unos diseños preliminares hechos en Balsamiq Wireframe:



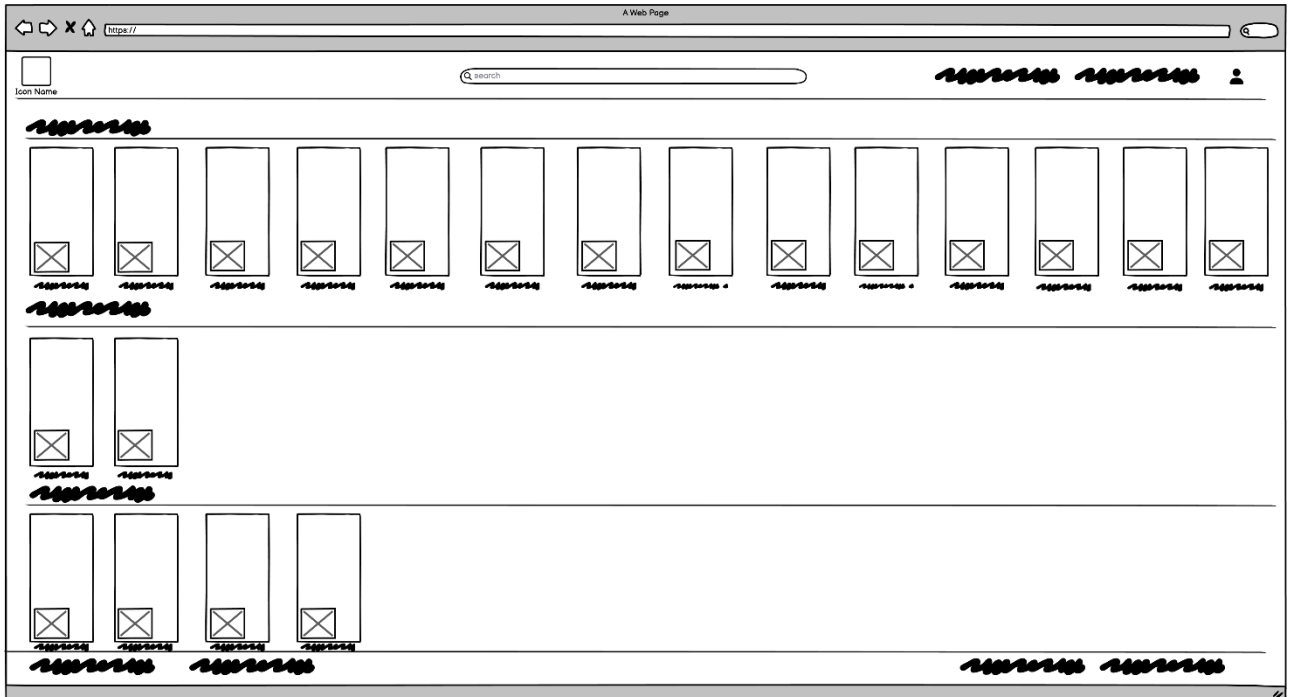
Esta pantalla se compone del título de la pagina y los campos que rellenar para iniciar sesión y un botón para recordar la cuenta.

Crear una cuenta:



Es similar a la pantalla anterior, pero con los campos necesarios para registrarse y los de confirmación de la contraseña y correo electrónico, a parte del campo para subir la imagen.

Inicio de la aplicación:



Aquí principalmente se muestran las comunidades existentes en la aplicación. Saldrán listas de comunidades recomendadas por tus gustos, a las que estas suscrito y comunidades populares. En las tarjetas de las comunidades se verán la imagen de fondo de la comunidad y el icono la imagen principal de la comunidad junto a su nombre.

Crear una comunidad:

La pagina de crear una comunidad será similar a la de registro e inicio de sesión. Un campo para el nombre para subir las dos imágenes necesarias para la comunidad y os campos más, uno para las reglas de la comunidad y su descripción.

Inicio de una comunidad:

En el centro de la pantalla se verán los posts de la comunidad, una pequeña previsualización y los «me gusta» y comentarios que tiene. El lateral izquierdo tendrás

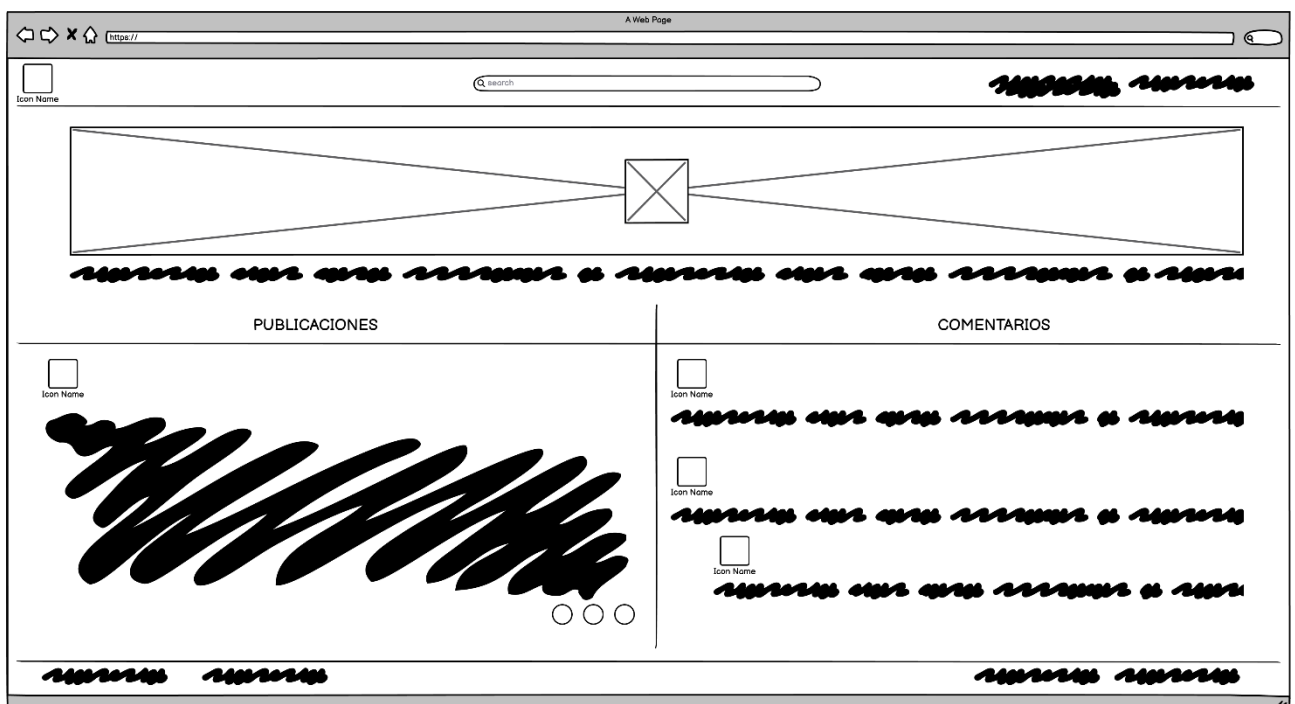
tu perfil de esa comunidad y varias opciones de la gestión y uso de la comunidad. En el lateral derecho post populares.

Contenido de un post:



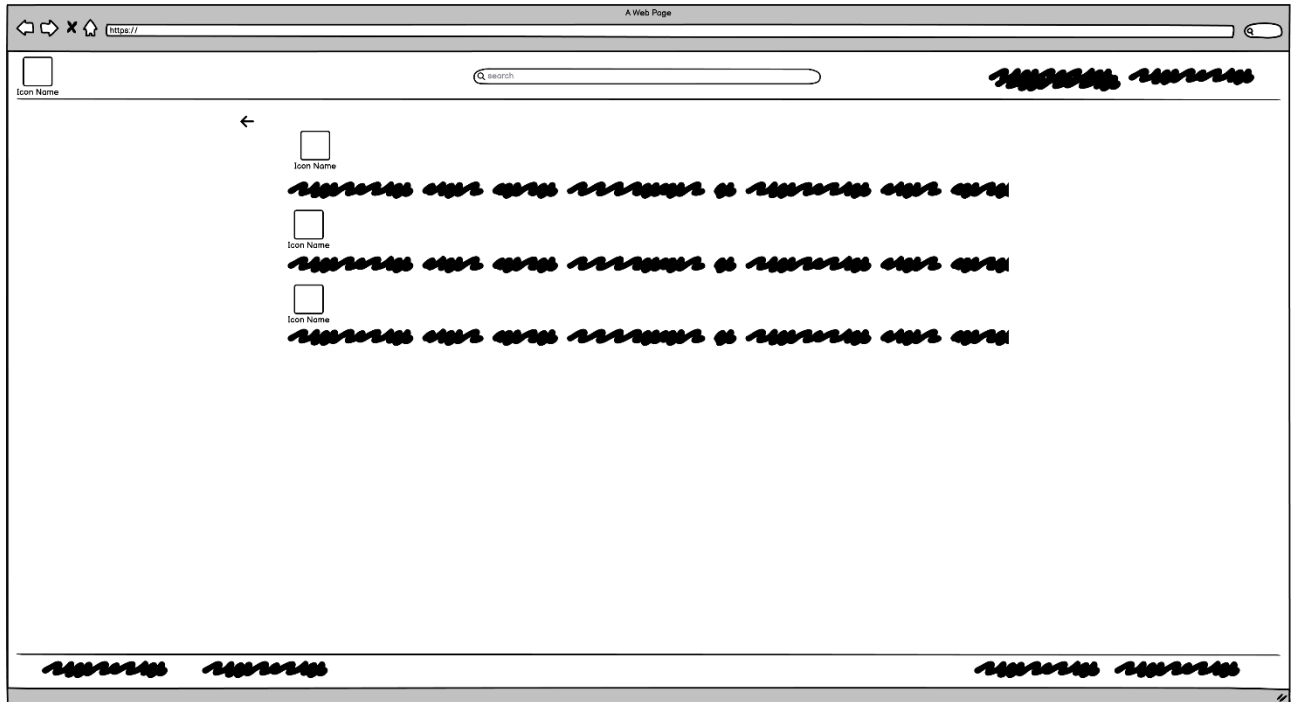
Tendrá el contenido del post y en la parte inferior la opción de dar «me gusta» o comentar en la publicación.

Perfil de una comunidad:



Tendrás tu imagen de perfil en el centro de la parte superior con una imagen de fondo, en la parte inferior tus títulos o roles de la comunidad. En la parte inferior podrás ver tus posts y también los comentarios que te han dejado en tu perfil.

Notificaciones:



Se mostrar la imagen de la persona que ha creado la notificación con un mensaje de lo que se trata la notificación.

4. DESARROLLO

4.1. PROGRAMACIÓN

El desarrollo de la aplicación **Guildnet** se ha abordado de forma modular, siguiendo una arquitectura basada en servicios independientes y componentes reutilizables. Esta aplicación representa una red social temática centrada en comunidades, y se ha implementado utilizando **Spring Boot** para el backend y **Angular** para el frontend, con una base de datos relacional **MySQL**.

El objetivo principal ha sido construir una plataforma funcional, escalable y segura, que permita a los usuarios interactuar dentro de comunidades, compartir contenido, comunicarse en tiempo real y personalizar su experiencia mediante roles y títulos.

Arquitectura general y organización del proyecto

Desde el inicio del desarrollo, se optó por una separación clara entre el **backend (servidor de lógica de negocio)** y el **frontend (interfaz de usuario)**. Ambos se comunican exclusivamente mediante una **API RESTful** protegida mediante **autenticación JWT**, a excepción del sistema de chat.

En el backend, los distintos dominios del sistema (usuarios, comunidades, publicaciones, perfiles, etc.) se organizaron en **módulos por funcionalidades** (features), siguiendo principios de bajo acoplamiento y alta cohesión. Esta decisión permite escalar fácilmente el proyecto y mantener cada parte del código bien delimitada. En el frontend, se siguió un enfoque similar: los componentes, servicios y modelos se organizaron también por funcionalidades, respetando las buenas prácticas de Angular.

4.1.1. Backend - Spring Boot

El backend se desarrolló utilizando **Spring Boot**, aprovechando sus capacidades para la creación rápida de aplicaciones RESTful, la configuración automática, y el soporte a tecnologías como **JPA**, **WebSocket**, **Spring Security**.

4.1.2. Entidades y relaciones

La base de datos se diseñó con múltiples entidades interrelacionadas:

- **Usuario:** almacena información básica, imagen de perfil, correo y nombre de usuario.

- **Comunidad:** contiene nombre, descripción, imagen de portada, etiquetas, etc.
- **Perfil de comunidad:** representa a un usuario dentro de una comunidad. Permite asociar roles, títulos, publicaciones, etc.
- **Publicaciones (Post):** creadas por perfiles, pueden tener contenido, etiquetas, comentarios y 'me gusta'.
- **Comentarios:** asociados a publicaciones y perfiles.
- **Roles y títulos:** asignables a perfiles dentro de una comunidad para definir jerarquías o distinciones.
- **Mensajes de chat:** enviados por perfiles en una comunidad, con fecha y contenido.

Se aplicaron **relaciones bidireccionales** cuando fue necesario (por ejemplo, entre publicaciones y comentarios) y se definieron correctamente las **cascadas y restricciones** de integridad para garantizar la consistencia.

4.1.3. Servicios y lógica de negocio

Cada entidad tiene su correspondiente **servicio (Service)** e **interfaz (Repository)**. Se utilizó el patrón **DTO (Data Transfer Object)** para encapsular los datos intercambiados entre capas y evitar exponer directamente las entidades.

La conversión entre entidades y DTOs se realiza manualmente.

La lógica de negocio más compleja se concentra en los servicios. Algunos ejemplos relevantes:

- **Creación automática de perfiles de comunidad al unirse a una comunidad.**
- **Asignación de roles y títulos a perfiles, con verificación de permisos.**
- **Validaciones personalizadas**, como evitar duplicidad de nombres de usuario o comunidades.
- **Gestión de imágenes:** guardado en disco, asignación de nombre único y exposición mediante ruta pública.

4.1.4. Seguridad y autenticación

Se implementó un sistema de autenticación **JWT (JSON Web Token)** utilizando **Spring Security**. Los usuarios deben iniciar sesión para obtener un token, el cual deben enviar en cada petición protegida.

Se añadieron filtros personalizados para verificar el token, obtener el usuario autenticado, y cargar sus detalles. Los endpoints se clasificaron como **públicos o protegidos** según su función.

Además, se creó un endpoint especial /me para obtener la información del usuario autenticado y simplificar la lógica del frontend.

4.1.5. Frontend – Angular

El frontend se desarrolló con Angular, utilizando **componentes reutilizables, servicios para llamadas HTTP, formularios reactivos y navegación protegida con guards**.

4.1.6. Estructura y navegación

La estructura se organizó en rutas principales:

- /login y /register para autenticación.
- /communities como página principal, con comunidades populares y suscritas.
- /community/:id para el detalle de una comunidad, incluyendo publicaciones, miembros y chat.
- /profile para gestionar el perfil personal.
- /post/:id para ver una publicación completa.

La navegación se gestiona con RouterModule, y se protegieron rutas privadas mediante un **AuthGuard** que comprueba la existencia del token JWT.

4.1.7. Formularios y subida de archivos

Para el registro y creación de comunidades, se utilizaron **formularios reactivos** (FormGroup, FormControl). La subida de imágenes (perfil, banner, etc.) se integró mediante FormData, combinando JSON y archivo en una sola petición.

Una de las dificultades superadas fue la correcta construcción de estas peticiones multipart y su recepción por parte del backend, que requería una configuración específica para mapear los datos correctamente.

4.1.8. Comunicación con el backend

Cada módulo cuenta con un **servicio Angular** (CommunityService, UserService, PostService, etc.) que centraliza las llamadas HTTP. Se utiliza HttpClient.

Los datos se intercambian siempre mediante DTOs, lo que simplifica el tratamiento y evita exponer estructuras complejas en el cliente.

4.1.9. Interfaz de usuario y experiencia

Se ha puesto énfasis en una interfaz clara y moderna, con botones intuitivos, imágenes de perfil visibles, formularios con validación, estados colapsables para contenido largo ("Ver más / Ver menos").

4.1.10. Decisiones y problemas resueltos

Durante el desarrollo se tomaron decisiones clave y se resolvieron problemas técnicos relevantes:

- **Modularización desde el principio**, que facilitó la escalabilidad.
- **Separación de responsabilidades**, evitando mezclas de lógica de negocio con lógica de presentación.
- **Validación de archivos multipart** entre Angular y Spring Boot, resuelta mediante pruebas y ajustes en la configuración del controlador.
- **Problemas de CORS** solucionados configurando correctamente los filtros de Spring.
- **Gestión de relaciones complejas** (por ejemplo, una publicación que pertenece a un perfil, que a su vez pertenece a una comunidad), con estructuras de DTO simplificadas.
- **Eliminación en cascada controlada** para evitar errores de borrado en entidades relacionadas.

4.2. PRUEBAS

La fase de pruebas del proyecto se centró en la **verificación funcional** de los distintos endpoints desarrollados en el backend. Para ello se utilizó la herramienta **Postman**.

El objetivo de estas pruebas fue garantizar que cada funcionalidad implementada respondiera conforme a lo definido en los requisitos, tanto en situaciones normales como en posibles casos de error.

Metodología de prueba

Se diseñó una batería de pruebas manuales agrupadas por funcionalidades principales:

- **Autenticación:** registro, login y obtención de datos del usuario autenticado.
- **Gestión de comunidades:** creación, búsqueda, suscripción y obtención de detalles.
- **Perfiles de comunidad:** creación automática al unirse, obtención por comunidad o ID.
- **Publicaciones:** creación de posts, visualización individual, comentarios y likes.
- **Roles y títulos:** asignación y eliminación por parte de usuarios con permisos.
- **Notificaciones:** generación automática en respuesta a eventos clave.

Cada conjunto de pruebas incluía:

- La **URL del endpoint** correspondiente.
- El **método HTTP** utilizado (GET, POST, PUT, DELETE).
- El **cuerpo de la petición** cuando aplicaba (JSON o multipart/form-data).
- Los **tokens de autenticación** JWT para los endpoints protegidos.
- La **respuesta esperada** y el análisis del código de estado (200, 201, 400, 403, 404...).

Resultados obtenidos

Las pruebas realizadas con Postman confirmaron que el backend responde correctamente en la mayoría de las situaciones esperadas. Algunos de los resultados más destacados fueron:

- Las operaciones protegidas devuelven error si el token es inválido o no se proporciona.
- Las validaciones (por ejemplo, no permitir dos usuarios con el mismo nombre) funcionan adecuadamente.
- Los datos relacionados (por ejemplo, publicaciones dentro de una comunidad, comentarios en un post, perfiles con sus roles y títulos) se devuelven correctamente estructurados.
- Las imágenes subidas en formularios multipart se guardan y se exponen mediante URL accesibles.

4.2.1. Pruebas de API con Postman

Durante el desarrollo del proyecto, se realizaron múltiples pruebas manuales a través de Postman para verificar el correcto funcionamiento de los distintos endpoints de la API. A continuación, se describen algunas de las pruebas más importantes realizadas:

Autenticación de usuarios

- **Registro:**

POST: /api/v1/auth/register

POST http://localhost:8080/api/v1/auth/register

Params Authorization Headers (8) Body Scripts Settings Cookies

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	Key	Value	Content-Type	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	user	Text	{ "username": "alvaro", "email": "alvaro4@ex..." }	application/json		
<input checked="" type="checkbox"/>	image	File	2d7a6d4859c1b17f3f468091443...	Auto		
	Key	Text	Value	Auto	Description	

Body Cookies Headers (14) Test Results 201 Created • 501 ms • 589 B • Save Response

```

1 {
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhbHZhcm80QGV4YW1wbGUuY29tIiwiaWF0IjoxNzQ5NjY4NjYyLCJleHAiOiJlE3NDk3NTUwNjI3J9.YcKlZ2yqWt6okzyYV3Htz630cjxD5wRUTetLiYxHvE"
3 }
  
```

- **Login**

POST /api/v1/auth/login

POST http://localhost:8080/api/v1/auth/login

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```

1 {
2   "email": "alvaro3@example.com",
3   "password": "1234"
4 }
5
  
```

Body Cookies Headers (14) Test Results 200 OK • 89 ms • 584 B • Save Response

```

1 {
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhbHZhcm80QGV4YW1wbGUuY29tIiwiaWF0IjoxNzQ5NjY4NjYyLCJleHAiOiJlE3NDk3NTUwOTB9.412J8cxysBPgrUa3XF6r4fy0dL0bevSV6sLJg0dRC0c"
3 }
  
```

Body Cookies Headers (14) Test Results 401 Unauthorized • 173 ms • 456 B • Save Response

Raw

```

1 Credenciales inválidas
  
```

- **Obtener usuario autenticado**

GET /api/v1/user/me

The screenshot shows a Postman interface for a GET request to `http://localhost:8080/api/v1/user/me`. The 'Headers' tab is active, showing standard headers like Authorization, Postman-Token, Host, User-Agent, Accept, Accept-Encoding, and Connection. The 'Body' tab is also visible, showing a JSON response with the following structure:

```

{
  "id": 3,
  "username": "alvaro",
  "email": "alvaro4@example.com",
  "profileImage": "http://localhost:8080/uploads/users/acf9f6b2-1215-4d48-a7c0-56f016144e6d_20250611_210422.png",
  "tags": []
}

```

Gestión de comunidades

- **Crear comunidad**

POST /api/v1/communities/create

The screenshot shows a Postman interface for a POST request to `http://localhost:8080/api/v1/communities/create`. The 'Body' tab is active, showing form-data with three fields: community, image, and banner. The 'community' field has a JSON value, while 'image' and 'banner' are file uploads. The 'Test Results' tab shows a 201 Created status. The 'Body' tab shows a JSON response with the following structure:

```

{
  "id": 2,
  "name": "Retro Gamers",
  "description": "Comunidad de juegos clásicos",
  "rules": "No spam",
  "image": "http://localhost:8080/uploads/communities/7f3c77d8-3bb2-46a8-b66c-00568214223c_20250611_210953.png",
  "banner": "http://localhost:8080/uploads/communities/0d0099bb-0d29-41ae-83e0-c84852792d71_20250611_210953.png",
  "tags": [
    "retro",
    "gaming"
  ],
  "ownerId": 3
}

```


- Buscar comunidades

/api/v1/communities/search

Por etiquetas:

GET ⌵ http://localhost:8080/api/v1/communities/search?tag=gaming&tag=retro Send ⌵

Por nombre:

GET ⌵ http://localhost:8080/api/v1/communities/search?name=retro Send ⌵

Por nombre y etiquetas:

GET ⌵ http://localhost:8080/api/v1/communities/search?name=retr&tag=gaming&tag=techweqw Send ⌵

Body Cookies Headers (14) Test Results 200 OK 46 ms 764 B Save Response ⋮

{} JSON ⌵ ▶ Preview 🔗 Visualize ⌵ ↻ 🔍 📄 🔗

```

1  [
2    {
3      "id": 2,
4      "name": "Retro Gamers",
5      "description": "Comunidad de juegos clásicos",
6      "rules": "No spam",
7      "image": "http://localhost:8080/uploads/communities/7f3c77d8-3bb2-46a8-b66c-00568214223c_20250611_210953.png",
8      "banner": "http://localhost:8080/uploads/communities/0d0099bb-0d29-41ae-83e0-c84852792d71_20250611_210953.png",
9      "tags": [
10       "retro",
11       "gaming"
12     ]
13   }
14 ]

```

- Actualizar comunidad

PUT /api/v1/communities/{id}

PUT ⌵ http://localhost:8080/api/v1/communities/1 Send ⌵

Params Authorization Headers (10) **Body** • Scripts Settings Cookies

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	Key	Value	Content-Type	Description	⋮ Bulk Edit
<input checked="" type="checkbox"/>	data	Text ⌵	{ "name": "Actualizado", "description": "...	application/json	
<input checked="" type="checkbox"/>	image	File ⌵	Select files	Auto	
<input checked="" type="checkbox"/>	banner	File ⌵	Select files	Auto	
	Key	Text ⌵	Value	Auto	Description

Body Cookies Headers (14) Test Results 200 OK 148 ms 458 B Save Response ⋮

📄 Raw ⌵ ▶ Preview 🔗 Visualize ⌵ ↻ 🔍 📄 🔗

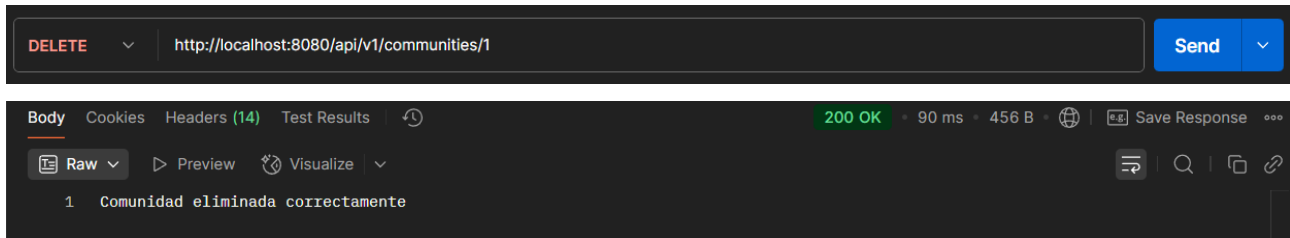
```

1  Comunidad actualizada correctamente

```

- **Eliminar comunidad**

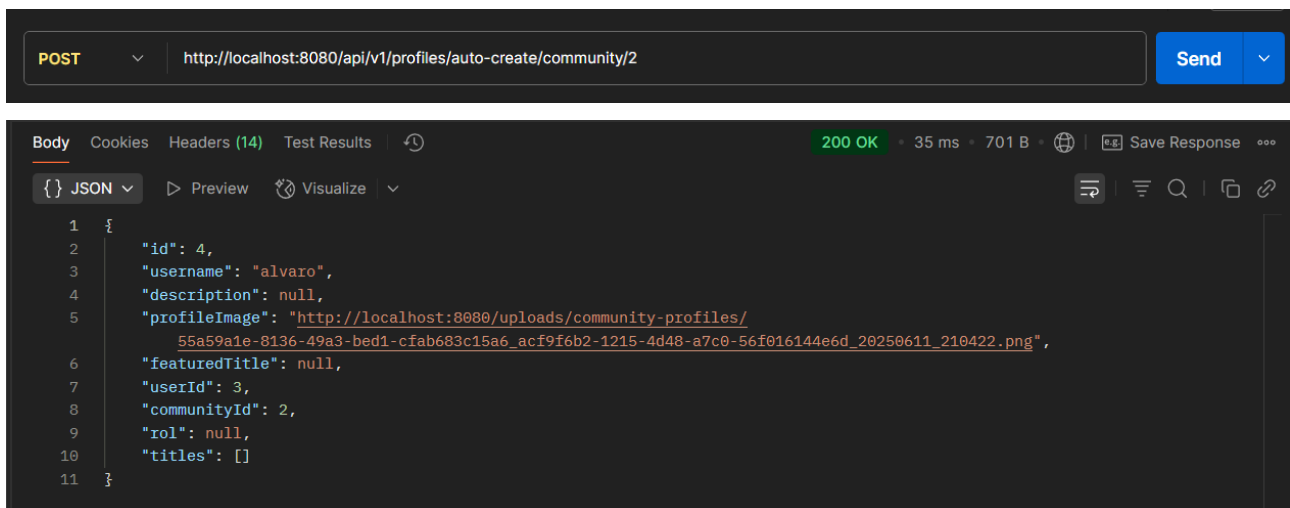
DELETE /api/v1/communities/{id}



Perfiles de comunidad

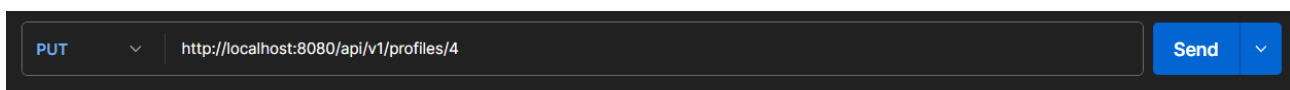
- **Creación automática**

POST /api/v1/profiles/auto-create/community/{id}



- **Edición de perfil**

PUT /api/v1/profiles/{id}



Por algún motivo la actualización del perfil no me está dejando realizarla desde postman. Y me esta dando error, pero si esta funcionando correctamente desde el front de la aplicación.



aa

Lider

Editar perfil

Ver descripción

AAAAA

Editar perfil



Nombre de usuario

Actualizado

Descripción

Rich text editor toolbar with icons for Bold (B), Italic (I), Underline (U), Strikethrough (ABC), Quote (”), Code (), Header 1 (H1), Header 2 (H2), Bulleted List (•), Numbered List (1), Subscript (x₂), Superscript (x²), Indent (→), Outdent (←), Text Color (A), Background Color (■), Link (🔗), Image (🖼️), Table (📊), and Font Family (Normal). Below the toolbar is a text area containing the word "Actualizado".

Cambiar imagen

Seleccionar archivo 20210805_183516.jpg

Perfil actualizado correctamente.

Cerrar

Guardar cambios

No hay publicaciones de este usuario aún.



Actualizado

Lider

[Editar perfil](#)

[Ver descripción](#)

Actualizado

Sistema de publicaciones

- Crear publicación

POST /api/v1/posts

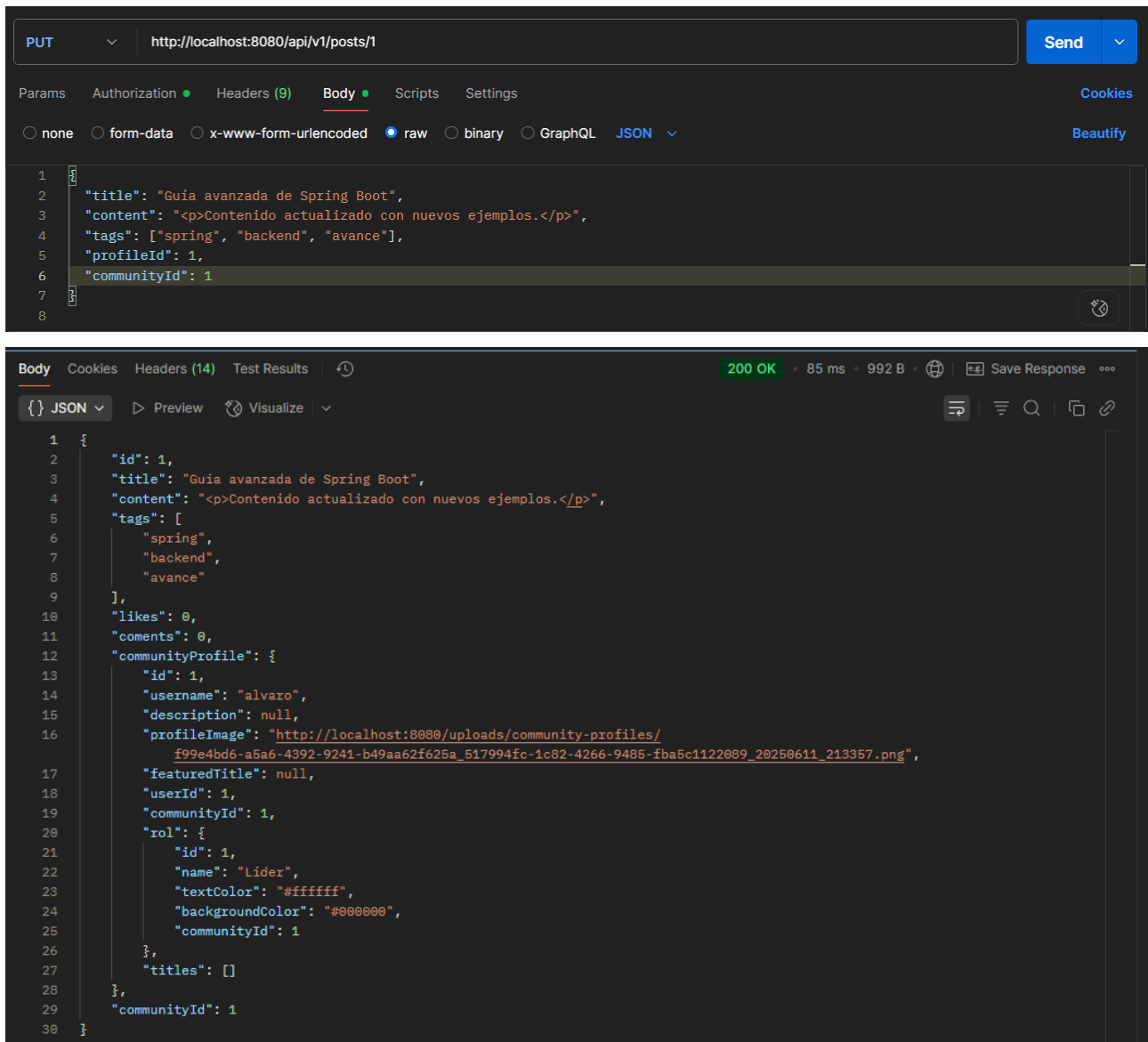
```
POST http://localhost:8080/api/v1/posts

{
  "title": "Guía de iniciación a Spring Boot",
  "content": "<p>Este post contiene una introducción a <strong>Spring Boot</strong>.</p>",
  "tags": ["spring", "java", "backend"],
  "profileId": 1,
  "communityId": 1
}
```

```
201 Created • 41 ms • 1 KB

{
  "id": 1,
  "title": "Guía de iniciación a Spring Boot",
  "content": "<p>Este post contiene una introducción a <strong>Spring Boot</strong>.</p>",
  "tags": [
    "spring",
    "java",
    "backend"
  ],
  "likes": 0,
  "comments": 0,
  "communityProfile": {
    "id": 1,
    "username": "alvaro",
    "description": null,
    "profileImage": "http://localhost:8080/uploads/community-profiles/f99e4bd6-a5a6-4392-9241-b49aa62f625a_517994fc-1c82-4266-9485-fba5c1122089_20250611_213357.png",
    "featuredTitle": null,
    "userId": 1,
    "communityId": 1,
    "rol": {
      "id": 1,
      "name": "Lider",
      "textColor": "#ffffff",
      "backgroundColor": "#000000",
      "communityId": 1
    },
    "titles": []
  },
  "communityId": 1
}
```

- Actualizar publicaciones
PUT /api/v1/posts/{id}



The screenshot displays a REST client interface with a PUT request to `http://localhost:8080/api/v1/posts/1`. The request body is a JSON object with the following structure:

```

1 {
2   "title": "Guía avanzada de Spring Boot",
3   "content": "<p>Contenido actualizado con nuevos ejemplos.</p>",
4   "tags": ["spring", "backend", "avance"],
5   "profileId": 1,
6   "communityId": 1
7 }

```

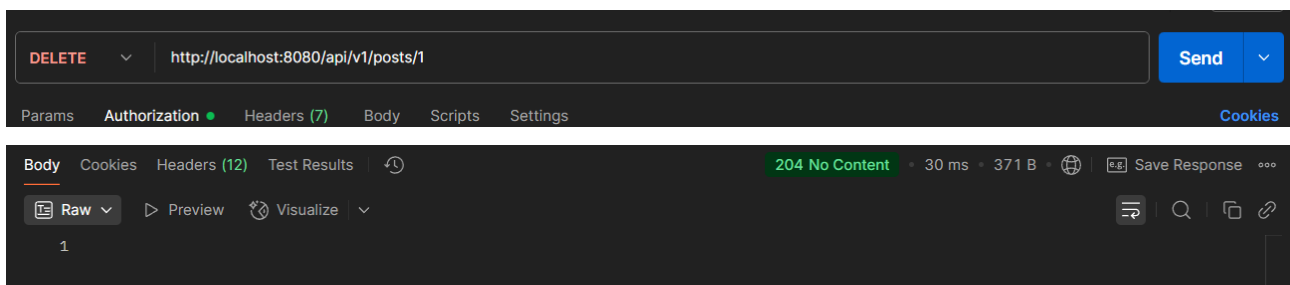
The response is a 200 OK status with a JSON body containing detailed information about the post and its associated community profile:

```

1 {
2   "id": 1,
3   "title": "Guía avanzada de Spring Boot",
4   "content": "<p>Contenido actualizado con nuevos ejemplos.</p>",
5   "tags": [
6     "spring",
7     "backend",
8     "avance"
9   ],
10  "likes": 0,
11  "coments": 0,
12  "communityProfile": {
13    "id": 1,
14    "username": "alvaro",
15    "description": null,
16    "profileImage": "http://localhost:8080/uploads/community-profiles/f99e4bd6-a5a6-4392-9241-b49aa62f625a_517994fc-1c82-4266-9485-fba5c1122089_20250611_213357.png",
17    "featuredTitle": null,
18    "userId": 1,
19    "communityId": 1,
20    "rol": {
21      "id": 1,
22      "name": "Lider",
23      "textColor": "#ffffff",
24      "backgroundColor": "#000000",
25      "communityId": 1
26    },
27    "titles": []
28  },
29  "communityId": 1
30 }

```

- Eliminar publicaciones
DELETE /api/v1/posts/{id}



The screenshot displays a REST client interface with a DELETE request to `http://localhost:8080/api/v1/posts/1`. The response is a 204 No Content status, indicating that the resource has been successfully deleted.

4.3. Control de versiones y repositorio

Durante todo el desarrollo del proyecto se ha utilizado **Git** como sistema de control de versiones, y **GitHub** como plataforma para alojar el código fuente, gestionar cambios, realizar seguimientos y mantener el historial de versiones.

Esto ha permitido trabajar de forma organizada, realizar pruebas sin afectar al código principal y tener una copia de seguridad continua del proyecto.

El repositorio se encuentra disponible en:

Repositorio GitHub: <https://github.com/AlvaroGomezNerpio/Guildnet-Proyecto-Red-social-de-comunidades>

- **/backend:** Contiene el proyecto Spring Boot (servidor, lógica, API REST, etc.)
- **/frontend:** Contiene el proyecto Angular (interfaz gráfica de usuario)
- **README.md:** Documento principal

5. MANUAL DE PUESTA EN MARCHA

Este manual describe paso a paso cómo instalar, configurar y ejecutar el proyecto **Guildnet - Red social de comunidades**, el cual incluye tanto el backend (Spring Boot) como el frontend (Angular), organizados en un único repositorio.

5.1. Requisitos previos

Herramientas necesarias

- Java 17 o superior
- Node.js 18+ y npm
- Angular CLI
- MySQL 8 o superior
- Maven (si no usas un IDE)
- Postman (opcional para pruebas)
- IDE recomendado: IntelliJ IDEA (backend) y Visual Studio Code (frontend)

Base de datos

- Crear una base de datos MySQL llamada guildnet
- Usuario por defecto: root
- Contraseña por defecto: password

Estos datos se pueden modificar en la configuración del backend (application.properties)

5.2. Clonar el proyecto

En GitHub está subido el proyecto entero tanto con el front y el back en el mismo repositorio, para clonarlo son los siguientes pasos:

```
git clone https://github.com/ tuusuario/Guildnet-Proyecto-Red-social-de-comunidades.git
```

```
cd Guildnet-Proyecto-Red-social-de-comunidades
```

El repositorio contiene dos carpetas principales:

- backend/: contiene la API desarrollada con Spring Boot.
- frontend/: contiene la aplicación web desarrollada con Angular.

5.3. Configurar y ejecutar el backend (Spring Boot)

1. Acceder al directorio del backend:

```
cd backend
```

2. Configurar el archivo src/main/resources/application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/guildnet  
spring.datasource.username=root  
spring.datasource.password=password
```

3. Ejecutar la aplicación:

Desde el IDE ejecutar la clase GuildnetApplication

4. El backend quedará disponible en:

<http://localhost:8080>

5.4. Configurar y ejecutar el frontend (Angular)

1. Abrir una nueva terminal y acceder al directorio frontend:

```
cd ../frontend
```

2. Instalar las dependencias:

```
npm install
```

3. Configurar la URL del backend

Por el momento el front esta configurado para funcionar con la <http://localhost:8080>

4. Ejecutar la aplicación Angular:

```
ng serve
```

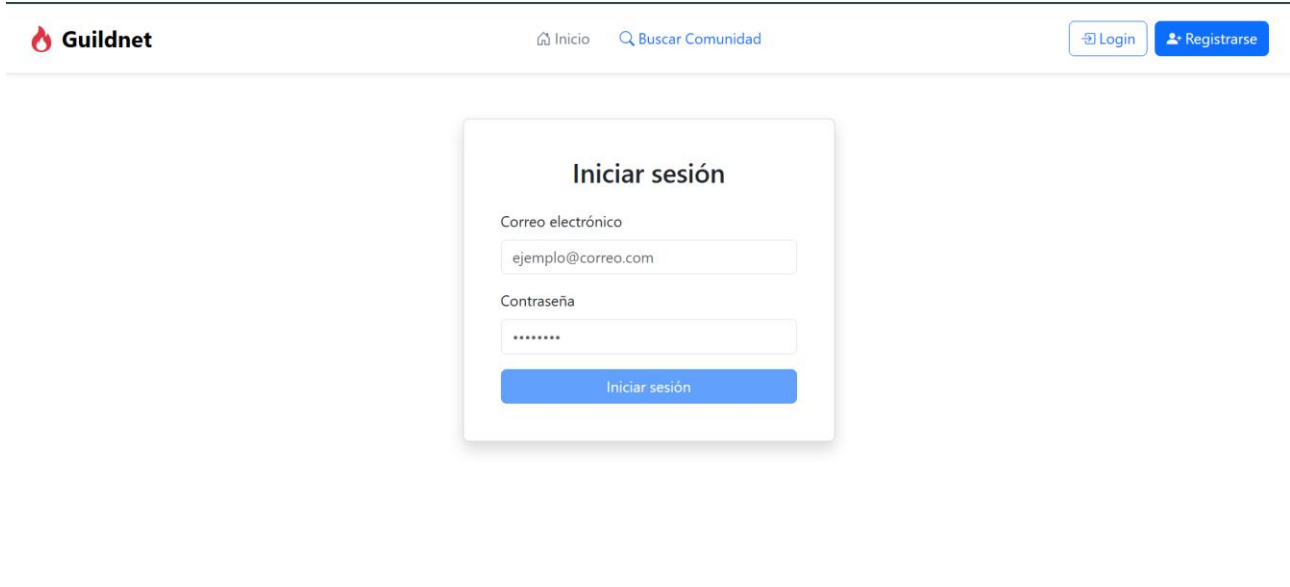
5.5. Usuario de prueba

Se puede registrar desde el formulario de la aplicación o mediante Postman (</api/v1/auth/register>).

5.6. Capturas de pantalla representativas

A continuación, se incluyen algunas vistas principales de la aplicación:

1. Pantalla de inicio de sesión



Guilnet Inicio Buscar Comunidad Login Registrarse

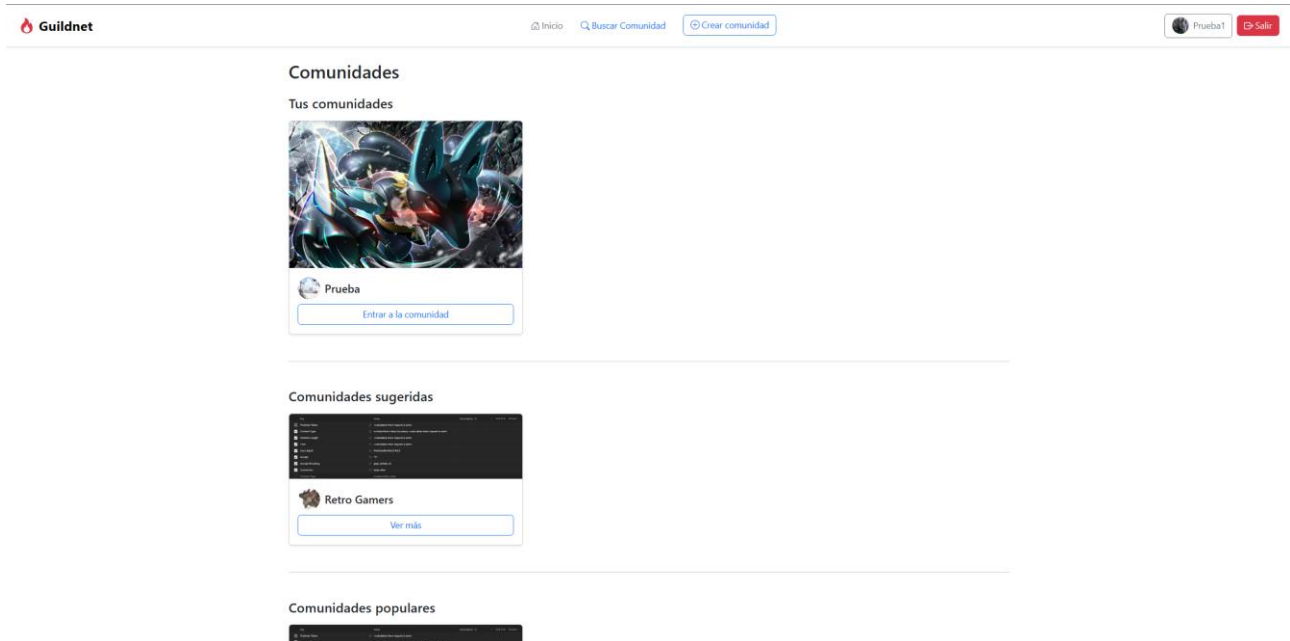
Iniciar sesión

Correo electrónico
ejemplo@correo.com

Contraseña

Iniciar sesión

2. Listado de comunidades



Guilnet Inicio Buscar Comunidad Crear comunidad Prueba1 Salir

Comunidades

Tus comunidades

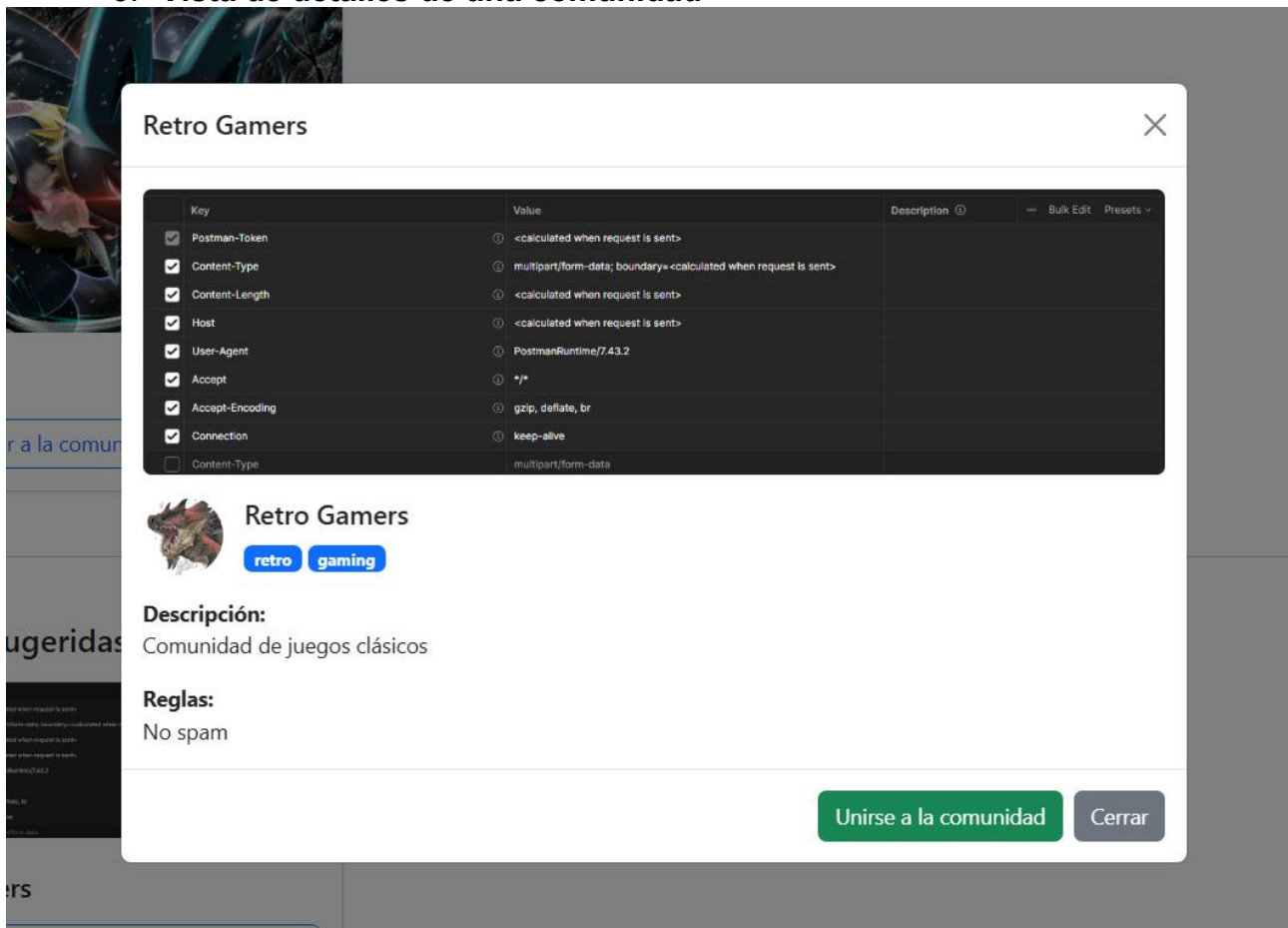
Prueba
Entrar a la comunidad

Comunidades sugeridas

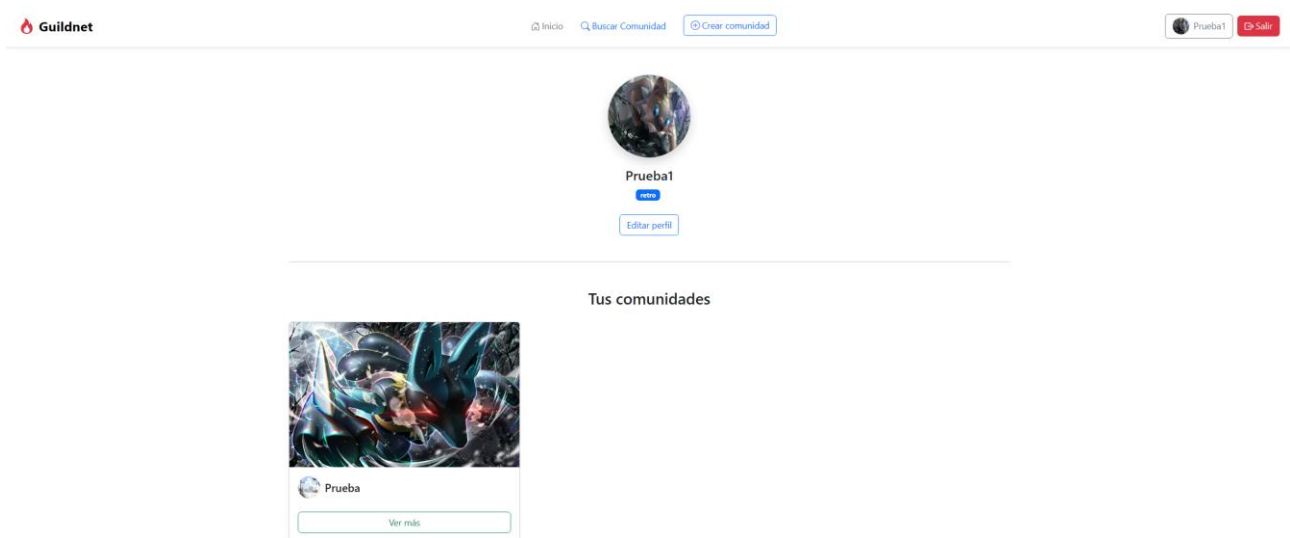
Retro Gamers
Ver más

Comunidades populares

3. Vista de detalles de una comunidad



4. Perfil del usuario



5. Inicio de una comunidad

[Inicio](#)
[Buscar Comunidad](#)
[Crear comunidad](#)

Prueba1

Salir

Prueba1
#Prueba1

Ver reglas
Ver descripción

Prueba1
Ver perfil

Opciones

Crear publicación

Ver miembros

Buscar publicaciones

Ver notificaciones

Crear rol

Unirse a comunidad

Publicaciones recientes

Prueba1

Prueba1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris at neque a sapien condimentum lacinia. Sed sit amet metus sit a met odio hendrerit sagittis imperdiet ut odio. Donec lorem felis, suscipit vitae tincidunt in, blandit ut nibh. Aliquam sed turpis te mpor, auctor nisi eget, ultrices turpis. Sed vitae ante dolor. Pellentesque et nulla et orci facilisis tristique euismod nec diam. Donec fermentum metus velit, in congue risus pulvinar a. Duis lacinia, libero ut fringilla posuere, ipsum nisi accumsan turpis, sit amet ultricies nisi nunc ac odio. In hac habitasse platea dictumst. Sed ut blandit lorem.

Donec consequat vestibulum ipsum vitae scelerisque. Suspendisse tincidunt tellus tortor, in mattis neque semper nec. Quisque eleifend sit amet eros sed fermentum. Mauris fringilla neque vel eleifend euismod. Mauris mi enim, tincidunt semper mauris ac, porta consectetur neque. Sed quis ipsum turpis. Ut nisl eros, gravida eu euismod quis, pretium a lacus. Pellentesque ut mi eget, eros pellentesque fringilla id quis dui. Donec et malesuada lectus. Suspendisse condimentum justo aliquam elit lobortis interdum. Nunc rutrum ante ante, in efficitur una accumsan eget. Aliquam sit amet lorem eu nisl gravida mattis id quis nunc. Quisque vel posuere tortor, ac egestas eros.

Suspendisse scelerisque vulputate ex non hendrerit. Phasellus eget tellus blandit, egestas erat at, dignissim eros. Etiam ac condimentum turpis. Praesent suscipit mi ut quam volutpat accumsan sit amet eget mi. Maecenas sit amet viverra tortor, nec luctus eros. Fusce nec ligula gravida, mollis augue non, iaculis quam. Nulla placerat, purus vel tincidunt tincidunt, mauris lorem pellentesque tellus, bibendum rutrum neque ligula ut ante. Quisque consectetur lectus at elit ultricies, eget fermentum enim scelerisque. Donec at quam a mauris efficitur eleifend. Suspendisse fringilla neque a risus pulvinar rutrum. Integer sodales ante id velit finibus fringilla. In consono magna tellus, eu porta magna ultricies in. Etiam vel sit amet turpis tincidunt imperdiet. Phasellus suscipit ullamcorper purus eget dictum.

Etiam nec consequat enim. Morbi semper vehicula enim. Pellentesque vitae erat massa. Aenean sed purus nec lacus tristique eleifend. Donec rutrum, libero id blandit egestas, tellus lacus pretium nisi, non efficitur ex eros at est. Morbi arcu ipsum, fermentum vitae turpis non, aliquam fringilla tortor. Etiam vitae hendrerit neque. In elementum justo ac tristique ornare. Donec luctus co

6. Perfil de una comunidad

[Inicio](#) [Buscar Comunidad](#) [Crear comunidad](#)

Prueba1

Salir

Prueba1

Lider

Prueba1

Editar perfil

Gestionar títulos

Asignar nuevo rol

Eliminar de la comunidad

Ver descripción

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris at neque a sapien condimentum lacinia. Sed sit amet metus sit amet odio hendrerit sagittis imperdiet ut odio. Donec e lorem felis, suscipit vitae tincidunt in, blandit ut nibh. Aliquam sed turpis tempor, auctor nisi eget, ultrices turpis. Sed vitae ante dolor. Pellentesque et nulla et orci facilisis tristique euismod nec diam. Donec fermentum metus velit, in congue risus pulvinar a. Duis lacinia, libero ut fringilla posuere, ipsum nisl accumsan turpis, sit amet ultricies nisi nunc ac odio. In hac habitasse platea dictumst. Sed ut blandit lorem.

Donec consequat vestibulum ipsum vitae scelerisque. Suspendisse tincidunt tellus tortor. In mattis neque semper nec. Quisque eleifend sit amet eros sed fermentum. Mauris fringilla a neque vel eleifend euismod. Mauris mi enim, tincidunt semper mauris ac, porta consectetur neque. Sed quis ipsum turpis. Ut nisl eros, gravida eu euismod quis, pretium a lacus. Pellentesque ut mi eget eros pellentesque fringilla id quis dui. Donec et malesuada lectus. Suspendisse condimentum justo aliquam elit lobortis interdum. Nunc rutrum ante ante, in efficitur urna accumsan eget. Aliquam sit amet lorem eu nisl gravida mattis id quis nunc. Quisque vel posuere tortor, ac egestas eros.

Publicaciones

Comentarios

Prueba

Ver más

Etiquetas: prueba1, prueba2

0 0 0

7. Contenido de un post

Prueba1

Prueba

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris at neque a sapien condimentum lacinia. Sed sit amet metus sit amet odio hendrerit sagittis imperdiet ut odio. Donec lor em felis, suscipit vitae tincidunt in, blandit ut nibh. Aliquam sed turpis tempor, auctor nisi eget, ultrices turpis. Sed vitae ante dolor. Pellentesque et nulla et orci facilisis tristique euismod nec diam. Donec fermentum metus velit, in congue risus pulvinar a. Duis lacinia, libero ut fringilla posuere, ipsum nisl accumsan turpis, sit amet ultricies nisi nunc ac odio. In hac habitasse platea dictumst. Sed ut blandit lorem.

Donec consequat vestibulum ipsum vitae scelerisque. Suspendisse tincidunt tellus tortor. In mattis neque semper nec. Quisque eleifend sit amet eros sed fermentum. Mauris fringilla ne que vel eleifend euismod. Mauris mi enim, tincidunt semper mauris ac, porta consectetur neque. Sed quis ipsum turpis. Ut nisl eros, gravida eu euismod quis, pretium a lacus. Pellentesq ue ut mi eget eros pellentesque fringilla id quis dui. Donec et malesuada lectus. Suspendisse condimentum justo aliquam elit lobortis interdum. Nunc rutrum ante ante, in efficitur urna accumsan eget. Aliquam sit amet lorem eu nisl gravida mattis id quis nunc. Quisque vel posuere tortor, ac egestas eros.

Suspendisse scelerisque vulputate ex non hendrerit. Phasellus eget tellus blandit, egestas erat at, dignissim eros. Etiam ac condimentum turpis. Praesent suscipit mi ut quam volutpat ac cursum sit amet eget mi. Maecenas sit amet viverra tortor, nec luctus eros. Fusce nec ligula gravida, mollis augue non, iaculis quam. Nulla placerat, purus vel tincidunt tincidunt, mauris lorem pellentesque tellus, bibendum rutrum neque ligula ut ante. Quisque consectetur lectus at elit ultricies, eget fermentum enim scelerisque. Donec at quam a mauris efficitur eleifend. Suspendisse fringilla neque a risus pulvinar rutrum. Integer sodales ante id velit finibus fringilla. In commodo magna tellus, eu porta magna ultricies in. Etiam vel mi sit amet turpis tincidunt imperdiet. Phasellus suscipit ullamcorper purus eget dictum.

Etiam nec consequat enim. Morbi semper vehicula enim. Pellentesque vitae erat massa. Aenean sed purus nec lacus tristique eleifend. Donec rutrum, libero id blandit egestas, tellus lac



Suspendisse scelerisque vulputate ex non hendrerit. Phasellus eget tellus blandit, egestas erat at, dignissim eros. Etiam ac condimentum turpis. Praesent suscipit mi ut quam volutpat ac cumsan sit amet eget mi. Maecenas sit amet viverra tortor, nec luctus eros. Fusce nec ligula gravida, mollis augue non, iaculis quam. Nulla placerat, purus vel tincidunt tincidunt, mauris lorem pellentesque tellus, bibendum rutrum neque ligula ut ante. Quisque consectetur lectus at elit ultricies, eget fermentum enim scelerisque. Donec at quam a mauris efficitur eleifen d. Suspendisse fringilla neque a risus pulvinar rutrum. Integer sodales ante id velit finibus fringilla. In commodo magna tellus, eu porta magna ultricies in. Etiam vel mi sit amet turpis tincidunt imperdiet. Phasellus suscipit ullamcorper purus eget dictum.

Etiam nec consequat enim. Morbi semper vehicula enim. Pellentesque vitae erat massa. Aenean sed purus nec lacus tristique eleifend. Donec rutrum, libero id blandit egestas, tellus lacus pretium nisi, non efficitur ex eros at est. Morbi arcu ipsum, fermentum vitae turpis non, aliquam fringilla tortor. Etiam vitae hendrerit neque. In elementum justo ac tristique ornare. Donec luctus congue lacus at porttitor. Fusce sed massa vel ex ultrices tempus. Nunc quis mi imperdiet, viverra lacus eget, imperdiet tortor.

Nulla et nibh non nibh iaculis suscipit. Duis ac vestibulum dui. Nunc lobortis ante at lorem congue pharetra. Ut tincidunt sapien vel consequat dignissim. Pellentesque ut lectus felis. Sed maximus egestas urna non imperdiet. Donec accumsan, sem in tincidunt porta, lorem diam tristique mi, commodo aliquam felis massa sed metus. Nam non erat sed erat pulvinar pelle ntesque id eget neque. Sed ornare suscipit turpis, et congue odio blandit sed. Donec mollis scelerisque libero, non dictum odio scelerisque et. Nam at metus mauris. In id lacus arcu. Sed ut fermentum dui, id elementum ligula.

[Publicar](#) [Cancelar](#)

[Editar publicación](#) [Eliminar publicación](#)

1 me gusta

Comentarios

Escribe un comentario...

[Publicar comentario](#)



Prueba1

Comentario

[Editar](#) [Eliminar](#)

6. CONCLUSIONES Y VÍAS FUTURAS

6.1. Conclusiones

Desarrollar **Guildnet** ha sido un trabajo muy completo que me ha permitido aplicar muchos de los conocimientos aprendidos durante el ciclo. A lo largo del proyecto he trabajado con tecnologías como **Spring Boot**, **Angular** y **MySQL**, además de aprender a integrar todas ellas en un sistema funcional y bien organizado.

Entre los principales logros del proyecto destacan:

- La implementación completa de un sistema de autenticación con **JWT**.
- La creación de un sistema modular basado en entidades como comunidades, perfiles, publicaciones, comentarios, likes, roles y permisos.
- La gestión dinámica de roles y permisos para perfiles de comunidad.
- La integración de formularios con subida de imágenes (perfil, comunidad, banner).
- El desarrollo de una interfaz intuitiva y funcional utilizando Angular y servicios REST.
- La puesta en marcha de un sistema básico de búsqueda por nombre y etiqueta.

Durante el desarrollo surgieron diversos desafíos técnicos, como el tratamiento de formularios multipart en Angular, la gestión de seguridad y autorización con JWT, la persistencia de relaciones complejas en base de datos, y la organización modular del código para mantener la escalabilidad del proyecto.

Pese a las dificultades encontradas, se han cumplido satisfactoriamente los principales **objetivos iniciales** definidos para este proyecto. Se ha logrado una aplicación funcional, usable, y lista para evolucionar.

6.2. Vías futuras y mejoras

Aunque el proyecto está completo y funcional, hay varias mejoras que me gustaría añadir en el futuro:

- **Sistema de notificaciones en tiempo real**, tanto internas como externas (WebSocket / Push).
- **Mejora del sistema de permisos**, con más granularidad y jerarquías por comunidad.
- **Sistema de chats** grupales o individuales por comunidad.
- **Moderación de contenido**, con reportes, bloqueo de usuarios.
- **Sistema de reputación o puntos** para gamificar la participación en comunidades.
- **Panel de administración global** para monitorizar estadísticas y gestionar usuarios o comunidades.
- Implementar un **bot o sistema automático que detecte contenido no adecuado**, como insultos, spam o contenido sexual explícito, y lo bloquee o marque para revisión.
- Aplicación de móvil nativa.

6.3. **Objetivos no desarrollados**

Por falta de tiempo, hay algunas cosas que no he podido desarrollar del todo:

- El **chat en tiempo real** solo está implementado parcialmente en el backend.
- No llegué a añadir un sistema completo de **moderación de publicaciones y comentarios**.
- Tenía pensado implementar un sistema de **recomendaciones personalizadas** de post según los intereses del usuario, pero no me dio tiempo, al mismo estilo que el sistema de recomendación de comunidades.
- Mejoras en las interfaces del front, como que al lado de tu nombre se muestre tu rol si lo tienes.

Aun así, el proyecto está preparado para que estas funciones se puedan añadir más adelante sin necesidad de rehacer lo que ya está hecho.

7. REFERENCIAS Y BIBLIOGRAFÍA

7.1. Recursos técnicos y documentación utilizada

- Spring Boot Documentation.
Disponible en: <https://docs.spring.io/spring-boot/index.html>
- Spring Security – JWT Authentication Guide.
Disponible en: <https://spring.io/projects/spring-security>,
<https://www.toptal.com/spring/spring-security-tutorial>,
<https://www.youtube.com/watch?v=-Z4a0bKr2Pg>,
<https://medium.com/@tericcabrel/implement-jwt-authentication-in-a-spring-boot-3-application-5839e4fd8fac>
- Angular - Guía oficial.
Disponible en: <https://angular.dev/overview>, <https://angular.dev/tutorials/learn-angular>
- Angular - Formularios reactivos
Disponible en: <https://angular.dev/guide/forms/reactive-forms>,
- Angular – Servicios
Disponible en: <https://v17.angular.io/guide/architecture-services>,
<https://angular.dev/tutorials/first-app/09services>
- Angular - Formularios reactivos
Disponible en: <https://angular.dev/guide/components>
- Angular - Formularios reactivos
Disponible en: <https://v17.angular.io/tutorial/tour-of-heroes/toh-pt5>
- Angular – Quill js.
Disponible en: <https://quilljs.com/docs/installation>,
<https://quilljs.com/docs/configuration>
- Documentación oficial de MySQL.
Disponible en: <https://dev.mysql.com/doc/refman/8.0/en/>
- Implementación de WebSocket en Spring Boot.
Disponible en: <https://spring.io/guides/gs/messaging-stomp-websocket>,
<https://docs.spring.io/spring-framework/reference/web/websocket.html>
- Guía sobre relaciones JPA y entidades en Spring.

Disponible en: <https://spring.io/projects/spring-data-jpa>,
<https://certidevs.com/tutorial-spring-boot-entidades-jpa>

- ChatGPT

Disponible en: <https://chat.openai.com>

7.2. Apuntes y material del ciclo

- Apuntes de **Servicios y procesos** – 2º DAM
- Apuntes de **Acceso a datos** -2º DAM
- Apuntes de **Desarrollo web en entorno cliente** – 2º DAW
- Apuntes de **Desarrollo web en entorno servidor** - 2º DAW
- Apuntes de **Diseño de interfaces web** - 2º DAW
- Proyectos realizados por mi cuenta anterior mente