Documentación del Proyecto - Red Social con Microservicios

Juan Felipe Jaramillo Losada

20 de marzo de 2025

Resumen

Este documento describe la implementación de una red social con arquitectura de microservicios, desarrollada en Java con Spring Boot para el backend y React con Zustand para el frontend. Se detalla la instalación, configuración, uso y documentación de la API.

Índice

1. Introducción

En esta prueba técnica se desarrolla una red social utilizando una arquitectura de microservicios, con un backend desarrollado en Java Spring Boot y un frontend en React. La aplicación permite a los usuarios autenticarse mediante JWT, gestionar su perfil, crear publicaciones y dar "likes.ª las publicaciones de otros usuarios.

El objetivo principal de esta red social es proporcionar una plataforma sencilla y funcional en la que los usuarios puedan interactuar mediante la creación y reacción a publicaciones. Para ello, se han diseñado y desarrollado los siguientes servicios:

- **Servicio de Autenticación y Usuarios (Atenea)**: Responsable del registro, autenticación y gestión de perfiles de usuario.
- **Servicio de Publicaciones (Hermes)**: Gestiona la creación, edición y eliminación de publicaciones, así como la funcionalidad de "likes".
- **Frontend de la aplicación (Afrodita)**: Desarrollado en React con Zustand para la gestión del estado, permitiendo a los usuarios interactuar con la plataforma de manera intuitiva.

La base de datos utilizada es PostgreSQL, y se han implementado pruebas unitarias para garantizar la calidad del código. Además, se ha empleado Docker para contenerizar los servicios y facilitar su despliegue en distintos entornos.

Este documento proporciona una guía completa sobre la instalación, configuración y uso de la aplicación, detallando los requisitos previos, los comandos necesarios para ejecutar los servicios y una descripción de la arquitectura y el flujo de la aplicación. También se incluye documentación de la API generada con Swagger, permitiendo una mejor comprensión de los endpoints disponibles.

2. Instalación y Configuración

2.1. Requisitos Previos

Antes de comenzar, asegúrese de tener instalados los siguientes componentes:

- Docker y Docker Compose
- Node.js y npm
- Java 17 y Maven
- PostgreSQL
- Git

2.2. Clonación del Repositorio

Ejecute el siguiente comando para clonar el proyecto:

```
git clone https://github.com/JuanFelipeJaramillo20/red_social.git
cd red_social
```

2.3. Configuración de Variables de Entorno

Para el correcto funcionamiento de la aplicación, es necesario definir ciertas variables de entorno. En el backend, estas variables se encuentran principalmente en los archivos de configuración de Spring Boot (‘application.properties‘ o ‘application.yml‘), mientras que en el frontend se recomienda el uso de un archivo ‘.env‘.

2.3.1. Backend (Atenea y Hermes)

En los microservicios Atenea (autenticación y usuarios) y Hermes (publicaciones y ”likes”), la configuración de las variables de entorno se realiza en los archivos ‘src/main/resources/application.proper de cada servicio.

Ejemplo de configuración en **Atenea**:

```
# Configuración de la base de datos en Atenea
spring.datasource.url=jdbc:postgresql://localhost:5432/atenea
spring.datasource.username=usuario
spring.datasource.password=contraseña

# Configuración del JWT
jwt.secret=clave_secreta
```

Ejemplo de configuración en **Hermes**:

```
# Configuración de la base de datos en Hermes
spring.datasource.url=jdbc:postgresql://localhost:5432/hermes
spring.datasource.username=usuario
spring.datasource.password=contraseña
```

En entornos de producción, se recomienda establecer estas variables como variables de entorno del sistema en lugar de incluirlas en los archivos de configuración.

2.3.2. Frontend (Afrodita)

En el frontend desarrollado en React, se requiere un archivo ‘.env‘ en la raíz del proyecto para configurar las variables de entorno necesarias. Ejemplo:

```
# Archivo .env en Afrodita (frontend)
VITE_API_BASE_URL=http://localhost:8081
VITE_AUTH_API=http://localhost:8080
```

Para que estas variables sean reconocidas por Vite (el bundler utilizado en el proyecto), los nombres de las variables deben comenzar con ‘VITE‘.

2.3.3. Docker y Contenedores

Si se ejecutan los microservicios en contenedores Docker, se pueden definir las variables de entorno en el archivo ‘docker-compose.yml‘:

```
services:
  atenea:
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/atenea
      - SPRING_DATASOURCE_USERNAME=usuario
      - SPRING_DATASOURCE_PASSWORD=contraseña
      - JWT_SECRET=clave_secreta
  hermes:
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/hermes
      - SPRING_DATASOURCE_USERNAME=usuario
      - SPRING_DATASOURCE_PASSWORD=contraseña
```

En este caso, las variables se definen directamente en el archivo de configuración de Docker Compose para facilitar el despliegue.

2.4. Levantando la Aplicación

Para iniciar todos los microservicios y la base de datos, ejecute:

```
docker-compose up --build
```

3. Arquitectura del Sistema

La aplicación sigue una arquitectura de **microservicios**, donde cada servicio tiene una responsabilidad específica y se comunica a través de API REST. Esto permite escalabilidad, mantenimiento independiente y facilidad de integración con otros sistemas.

La arquitectura está compuesta por los siguientes microservicios:

- **Atenea:** Servicio de autenticación y gestión de usuarios.
- **Hermes:** Servicio de publicaciones y gestión de "likes".
- **Afrodita:** Interfaz de usuario.

A continuación, se describe cada uno de estos componentes en detalle.

3.1. Atenea: Servicio de Autenticación y Usuarios

Atenea es el microservicio encargado de la autenticación y gestión de usuarios. Implementa autenticación basada en **JWT (JSON Web Tokens)**, permitiendo que los usuarios inicien sesión y gestionen su perfil de manera segura.

- **Tecnologías:** Spring Boot, Spring Security, JWT, PostgreSQL.
- **Responsabilidades:**
  - Registro e inicio de sesión de usuarios.
  - Generación y validación de tokens JWT.
  - Gestión del perfil del usuario (nombre, avatar, correo electrónico, etc.).
- **Base de datos:** Utiliza PostgreSQL para almacenar la información de los usuarios.

3.2. Hermes: Servicio de Publicaciones y Likes

Hermes es el microservicio responsable de gestionar las publicaciones de los usuarios, así como el sistema de "likes". Los usuarios pueden crear publicaciones, ver las de otros usuarios y reaccionar a ellas.

- **Tecnologías:** Spring Boot, JPA (Hibernate), PostgreSQL.
- **Responsabilidades:**
  - Creación, edición y eliminación de publicaciones.
  - Contabilización y gestión de "likes.[en] las publicaciones.
  - Consulta de publicaciones paginadas.
- **Base de datos:** Utiliza PostgreSQL para almacenar publicaciones y "likes".

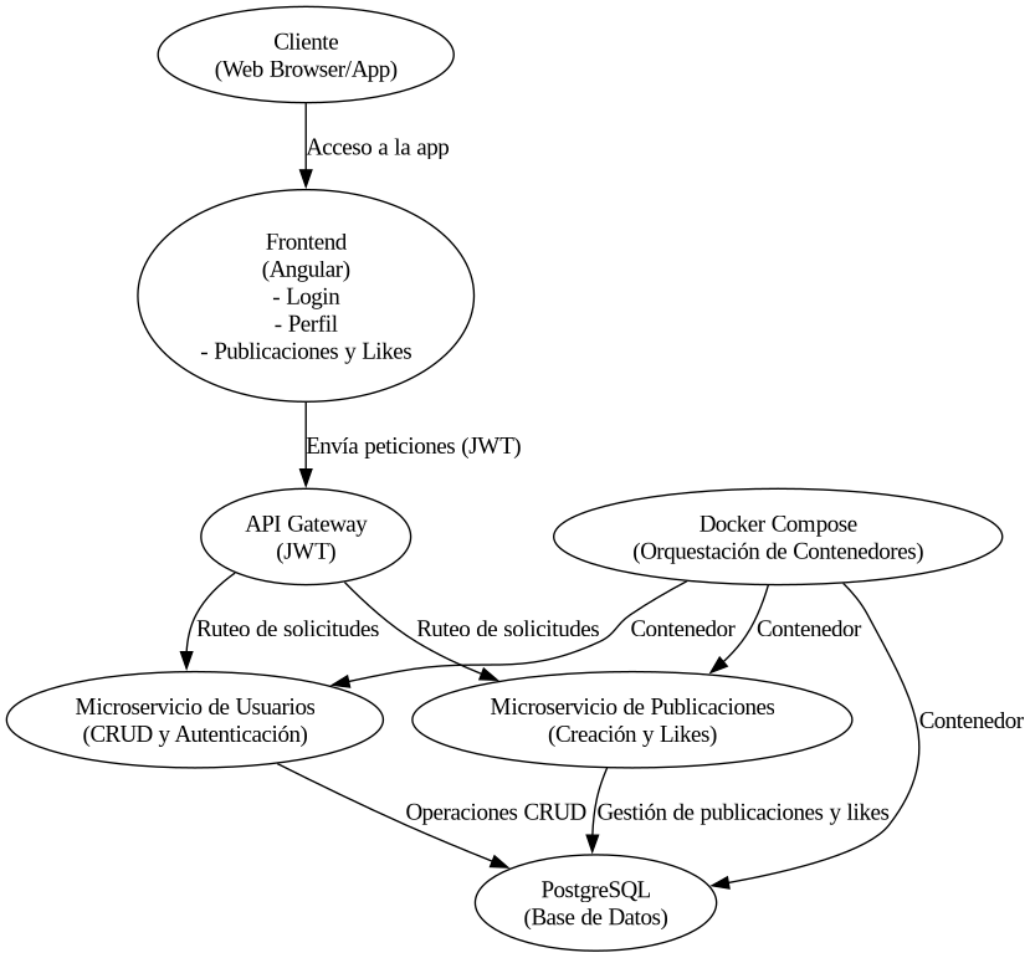
3.3. Afrodita: Interfaz de Usuario

Afrodita es la aplicación web del usuario final. Está desarrollada con **React** y utiliza **Zustand** para la gestión del estado global.

- **Tecnologías:** React, TypeScript, Zustand, Vite, Tailwind CSS.
- **Responsabilidades:**
  - Pantalla de inicio de sesión y autenticación.
  - Visualización y gestión del perfil del usuario.
  - Listado de publicaciones con la posibilidad de dar "like".

3.4. Comunicación entre Microservicios

Los microservicios se comunican a través de API REST. **Atenea** genera tokens JWT que son utilizados por **Hermes** para autenticar peticiones. Afrodita consume ambas APIs para mostrar datos a los usuarios.



3.5. Manejo de Estado y Seguridad

Para garantizar la seguridad y la consistencia de los datos, se utilizan las siguientes estrategias:

- **JWT para autenticación segura.**
- **Paginación y validaciones en el backend** para evitar sobrecarga de datos.
- **Gestión eficiente del estado en React con Zustand.**

4. Documentación de la API

La API está documentada utilizando **Swagger**, lo que facilita la exploración y prueba de los endpoints. A través de esta documentación interactiva, los desarrolladores pueden enviar peticiones, revisar parámetros y analizar respuestas.

Para acceder a la documentación de la API en un entorno local, visite:

`http://localhost:8081/swagger-ui.html`

4.1. Endpoints Principales

A continuación, se presentan algunos de los endpoints clave de la aplicación.

4.1.1. Obtener todas las publicaciones

Descripción: Obtiene una lista paginada de publicaciones con su información asociada.

`GET /api/posts/all?page=0&size=10`

Respuesta esperada:


```
{
  "content": [
    {
      "id": 1,
      "content": "Publicación de prueba",
      "createdBy": "testuser",
      "createdAt": "2025-03-20T12:00:00",
      "updatedAt": "2025-03-20T12:30:00",
      "likedByUser": true,
      "likeCount": 3
    }
  ],
  "pageable": {
    "pageNumber": 0,
    "pageSize": 10
  },
  "totalElements": 1,
  "totalPages": 1
}
```


4.1.2. Obtener publicaciones de un usuario específico

Descripción: Recupera todas las publicaciones creadas por un usuario en particular.

`GET /api/posts/user/{username}?page=0&size=10`

Respuesta esperada:

```
{
  "content": [
    {
      "id": 2,
      "content": "Otra publicación de prueba",
      "createdBy": "testuser",
      "createdAt": "2025-03-20T14:00:00",
      "updatedAt": "2025-03-20T14:30:00",
      "likedByUser": false,
      "likeCount": 1
    }
  ],
  "pageable": {
    "pageNumber": 0,
    "pageSize": 10
  },
  "totalElements": 1,
  "totalPages": 1
}
```


4.1.3. Dar "like.ª una publicación

Descripción: Permite que un usuario reaccione con un "like.ª una publicación.

```
POST /api/likes/{postId}
Headers:
  Authorization: Bearer <TOKEN>
```

Respuesta esperada:


```
{
  "message": "Like registrado correctamente",
  "likeCount": 4
}
```


4.1.4. Eliminar una publicación

Descripción: Permite al creador de una publicación eliminarla.

```
DELETE /api/posts/{postId}
Headers:
  Authorization: Bearer <TOKEN>
```

Respuesta esperada:


```
{
  "message": "Publicación eliminada correctamente"
}
```

4.2. Colección de Postman

Para facilitar las pruebas de la API, se incluye una **colección de Postman** con los endpoints utilizados en este proyecto. La colección contiene:

- Endpoints de autenticación y usuarios.
- Endpoints de gestión de publicaciones y "likes".
- Parámetros de ejemplo y tokens de autenticación preconfigurados.

Esta colección puede importarse directamente en Postman para realizar pruebas y validar el funcionamiento de los servicios.

Instrucciones para usar la colección:

1. Abra Postman y seleccione la opción *Importar*.
2. Cargue el archivo `red_social.postman_collection.json` ubicado en la carpeta del proyecto.
3. Configure las variables de entorno como el token JWT si es necesario.
4. Ejecute las peticiones para probar la API.

Nota: Asegúrese de que los microservicios están corriendo antes de realizar las pruebas en Postman.

5. Funcionamiento de la Aplicación

Esta sección describe las funcionalidades principales de la aplicación y su flujo de uso.

5.1. Inicio de Sesión

Los usuarios pueden acceder a la plataforma mediante un sistema de autenticación con **JWT (JSON Web Tokens)**. Para iniciar sesión, deben ingresar sus credenciales en la pantalla de inicio de sesión.

- Si las credenciales son correctas, el servidor devuelve un **token JWT**, que se almacena en el navegador del usuario y permite acceder a las demás funcionalidades de la aplicación.
- Si las credenciales son incorrectas, se muestra un mensaje de error al usuario.

Flujo del inicio de sesión:

1. El usuario ingresa su nombre de usuario y contraseña.
2. El frontend envía la solicitud al servicio de autenticación (**Atenea**).
3. Si las credenciales son correctas, se genera un token JWT y se almacena localmente.
4. El usuario es redirigido a la pantalla de publicaciones.

5.2. Visualización de Publicaciones

La pantalla principal muestra todas las publicaciones de los usuarios con un diseño en cuadrícula. Cada publicación incluye:

- El nombre del usuario que la creó.
- La fecha y hora de publicación.
- El contenido de la publicación (texto e imagen, si aplica).
- El contador de "likes".
- Un botón para dar o quitar "like".

El usuario puede interactuar con las publicaciones de la siguiente manera:

- **Dar "like"**: Al hacer clic en el botón de "like", la publicación aumenta su contador de reacciones.
- **Quitar "like"**: Si el usuario ya ha dado "like", puede hacer clic nuevamente para eliminarlo.
- **Ver cambios en tiempo real**: Los cambios en la cantidad de "likes"se reflejan inmediatamente en la interfaz.

5.3. Perfil de Usuario

Cada usuario tiene acceso a su perfil, donde puede ver y modificar su información personal.
Opciones disponibles en el perfil:

- Ver su nombre, correo electrónico y fecha de registro.
- Actualizar su nombre completo y foto de perfil.
- Ver todas sus publicaciones organizadas en formato de cuadrícula.
- Eliminar publicaciones propias.

Para editar el perfil, el usuario debe hacer clic en el botón **."Editar Perfil"**, lo que permitirá actualizar los datos y guardar los cambios en la base de datos.

6. Conclusión y Futuras Mejoras

Se ha desarrollado una red social funcional con una arquitectura basada en microservicios, utilizando **Spring Boot**, **React** y **PostgreSQL**. Se han implementado características esenciales como:

- Autenticación de usuarios con JWT.
- Creación, visualización y gestión de publicaciones.
- Funcionalidad de "likesçon actualización en la interfaz.
- Persistencia de datos en una base de datos PostgreSQL.
- Documentación de la API con Swagger.

6.1. Futuras Mejoras

Si bien la aplicación es completamente funcional, hay varias mejoras que pueden implementarse en versiones futuras para optimizar la experiencia del usuario:

- **Comentarios en publicaciones:** Agregar la posibilidad de que los usuarios puedan comentar en las publicaciones de otros, permitiendo interacciones más dinámicas.
- **Notificaciones en tiempo real:** Implementar WebSockets para actualizar el feed de publicaciones y notificar a los usuarios cuando reciben un "like.º un comentario.
- **Subida de imágenes para publicaciones y perfil:** Actualmente, las imágenes deben proporcionarse mediante URLs. Se puede mejorar permitiendo que los usuarios suban imágenes directamente a la plataforma.
- **Búsqueda y filtrado de publicaciones:** Agregar un sistema de búsqueda que permita encontrar publicaciones específicas o filtrar por usuario, fecha o popularidad.
- **Sistema de seguidores:** Implementar la opción de seguir a otros usuarios y ver solo sus publicaciones en un feed personalizado.
- **Modo oscuro:** Añadir soporte para cambiar entre modo claro y oscuro en la interfaz de usuario.

Estas mejoras permitirán hacer que la red social sea más interactiva y atractiva para los usuarios.