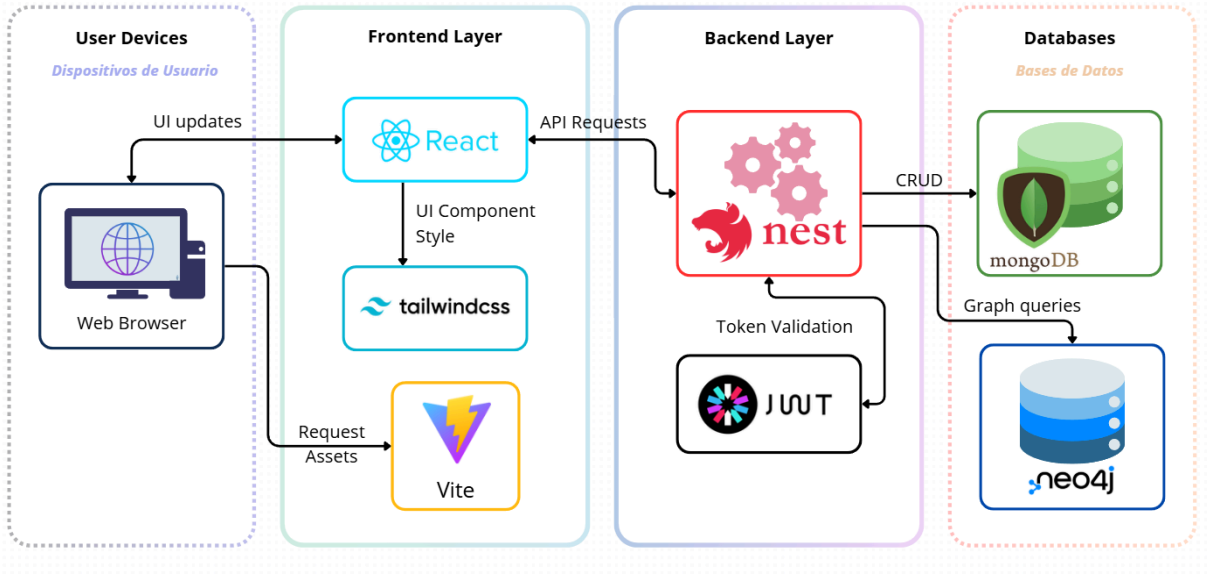
Taller: Red Social con NoSQL

Entrega 1

Diagrama de Arquitectura

Nombre de Equipo: MUSH Connections
Equipo de Trabajo: Maximo Sarno, Emily Volta, Martín Ubilla

Diagrama de Arquitectura



Componentes

React + Vite + TailwindCSS (Frontend)

Es la interfaz con la que interactúan los usuarios. Está desarrollado con React, Vite y TailwindCSS. Desde aquí, los usuarios pueden registrarse, iniciar sesión, subir fotos, añadir reacciones, comentar, etc. El navegador envía peticiones al backend a través de llamadas API (REST o GraphQL).

NestJS (Backend)

Contiene la lógica de negocio de la aplicación. Desarrollado en Node.js con NestJS, este módulo gestiona la autenticación (con tokens JWT), la validación de datos, la subida de imágenes, la administración de publicaciones y la conexión con las bases de datos. Se encarga de procesar las solicitudes provenientes del frontend y devolver las respuestas adecuadas.

Base de Datos MongoDB

Es la base de datos principal, orientada a documentos. Aquí se almacenan los usuarios, posts, comentarios, likes y mensajes. Su flexibilidad en el manejo de datos no estructurados la hace ideal para el contenido multimedia y dinámico de una red social.

Base de Datos Neo4j (con NodeJS)

Es una base de datos de grafos que se encarga de manejar las relaciones sociales de la aplicación, como seguidores, seguidos y sugerencias de nuevos contactos. Gracias a su modelo basado en grafos, permite consultas rápidas y eficientes sobre conexiones entre usuarios, algo fundamental en una red social tipo Instagram/Facebook.

Flujos de Datos

- El **usuario** interactúa desde el **navegador**.
- El **navegador** envía acciones/peticiones.
- Se reciben actualizaciones de UI desde el **frontend**.
- **React** renderiza la interfaz de usuario, muestra los posts, comentarios, etc.
- **Tailwind** se encarga del estilo y componentes visuales.
- **Vite** entrega recursos y assets de manera rápida.
- Desde el **Frontend** se envían peticiones API al **backend**
- **NestJS** recibe las peticiones del Frontend
- Valida la autenticación mediante **JWT**
- Se encarga de la lógica de negocio (crear posts, comentar, seguir usuarios)
- Realiza operaciones **CRUD** en **MongoDB** y consultas de grafos en **Neo4J**
- **MongoDB** maneja la información estructural (usuarios, publicaciones, likes, comentarios, etc).
- **Neo4J** gestiona las relaciones entre usuarios (recomendaciones de amigos, seguidores, seguidos).
- Responden al **backend** con lo solicitado
- **Backend** devuelve los resultados al **Frontend**
- **React** actualiza la UI para el usuario

Ventajas y Razones de la Elección

Escalabilidad

Las sesiones sin estado con **JWT** y las bases de datos **NoSQL** permiten que el sistema **crezca de manera horizontal**, sobrellevando el aumento de contenido y usuarios. **MongoDB** escala naturalmente mediante **sharding** distribuido, mientras que **Neo4j** ofrece capacidades de **clustering** nativo para manejar grandes conjuntos de **datos interconectados**. La arquitectura sin estado del backend permite añadir más instancias de servidores según sea necesario, sin afectar las sesiones de usuarios existentes.

Seguridad

Los **tokens JWT** protegen con eficiencia todos los endpoints de la API, asegurando que **solo usuarios autenticados** puedan **acceder a los recursos**. El sistema puede aplicar almacenamiento seguro de contraseñas mediante algoritmos de **hashing** robustos como **bcrypt**, e incluir autenticación en dos pasos (2FA). La separación entre frontend y backend añade una capa adicional de seguridad, **minimizando los vectores de ataque.**

Rendimiento

La división de responsabilidades entre MongoDB para **operaciones documentales** y Neo4j para **consultas de grafos** optimiza el rendimiento del sistema. MongoDB ofrece **tiempos de respuesta rápidos** para operaciones **CRUD** tradicionales, mientras que Neo4j proporciona un **rendimiento excepcional** para las consultas que involucran **relaciones complejas entre datos**. Esta división mantiene el sistema rápido incluso con grandes conjuntos de datos.

Compatibilidad

El frontend construido con **React y Vite garantiza compatibilidad total** con todos los navegadores modernos, incluyendo **Chrome, Firefox, Safari y Edge**. Las aplicaciones React son **responsivas por diseño y se adaptan perfectamente** a los diferentes dispositivos y tamaños de pantalla. El uso de estándares web abiertos asegura que la aplicación permanezca **accesible y funcional** en las futuras versiones de navegadores.

Simplicidad con Extensibilidad

La arquitectura presentada **cumple con los requisitos esenciales**, y está preparada para **extensiones futuras**. El sistema puede ampliarse más adelante con mecanismos de caché, sistemas de mensajería, notificaciones, micro-servicios adicionales, y más funcionalidades **sin necesidad de cambios fundamentales** en la base ya existente.