Tecnológico de Monterrey

LLM DOCKER MICROSERVICES

Implementación de una Arquitectura de Microservicios Integrando un LLM

Construcción de software y toma de decisiones (Gpo 501)

Diego Alejandro Espejo López A01638341

Julieta Carolina Arteaga Legorreta A0637444

Marcela Beatriz De La Rosa Barrios A01637239

Pablo Heredia Sahagún A01637103

Santos Alejandro Arellano Olarte A01643742

12/06/2024

LLM DOCKER MICROSERVICES

Introducción y objetivo

El objetivo de este proyecto es diseñar e implementar una arquitectura de microservicios que integre un modelo de lenguaje grande (LLM). La arquitectura debe cumplir con cuatro atributos de calidad esenciales: modificabilidad, comprobabilidad, seguridad y capacidad de despliegue, basados en las directrices del libro "Software Architecture with Python" de Anand Balachandran Pillai.

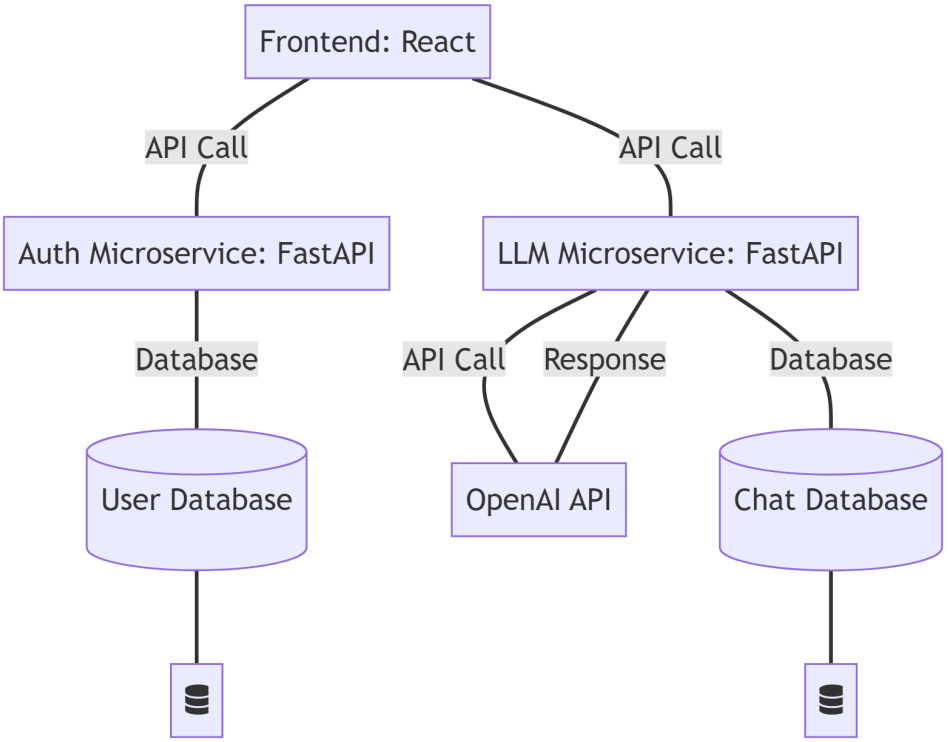
Principios de arquitectura de microservicios y su aplicación en un LLM

Tomando como referencia el libro mencionado se pudieron identificar y aplicar los siguientes principios de arquitectura de microservicios en la implementación de un LLM:

- **Descomposición en Servicios Independientes:**
  - Principio y aplicación en LLM: El sistema se descompone en microservicios como autenticación, frontend y LLM, donde cada uno de ellos maneja una parte específica de la funcionalidad global.
- **Alta Cohesión y Bajo Acoplamiento:**
  - Principio y aplicación en LLM: Los microservicios de autenticación, frontend y LLM están diseñados para ser autónomos. El microservicio de autenticación se encarga únicamente de la autenticación y autorización de usuarios con su base de datos, el LLM se encarga de la funcionalidad y la lógica del chat junto con su conexión a su base de datos y el frontend presenta la interfaz de usuario.
- **Comunicación a través de APIs Bien Definidas:**
  - Principio y aplicación en LLM: El LLM expone su funcionalidad a través de APIs REST, permitiendo que otros microservicios interactúen con él de manera uniforme y predecible.
- **Independencia en el Ciclo de Vida:**
  - Principio y aplicación en LLM: Los microservicios se contenerizada utilizando Docker, lo que facilita el despliegue independiente y la escalabilidad según las necesidades de cada componente.

- **Gestión Independiente de Datos:**
    - Principio y aplicación en LLM: Los microservicios de autenticación y chat utilizan sus propias bases de datos. La base de datos del servicio de autenticación almacena información de usuarios y roles, mientras que la base de datos del servicio de chat almacena el historial de conversaciones.
- **Automatización y Orquestación del Despliegue:**
    - Principio y aplicación en LLM: Se utiliza Docker para contenerizar cada microservicio

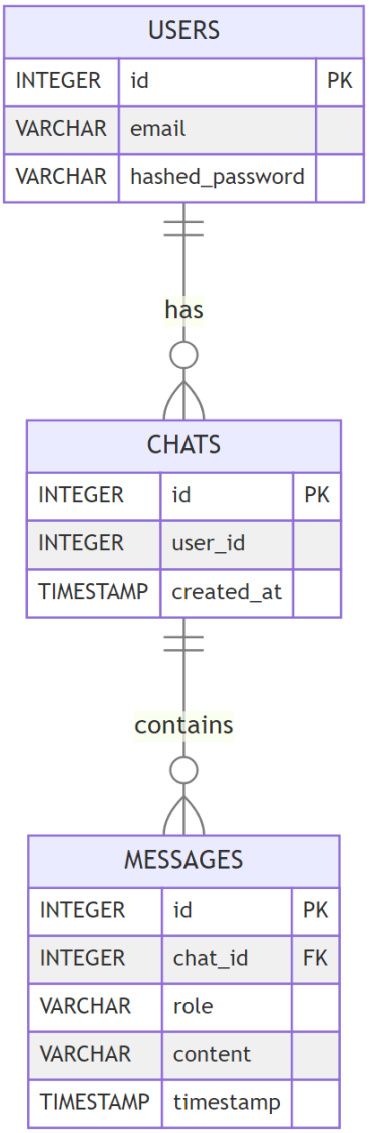
Diagrama de la arquitectura



Implementación de Seguridad: Autenticación y Autorización

- **Tecnología**: FastAPI, JWT
- **Implementación**:
    - Configuración de autenticación básica utilizando email y contraseña.
    - Uso de JWT para la generación y validación de tokens de acceso.
    - Implementación de endpoints para registro, inicio de sesión y validación de tokens.

Diagrama entidad relación de las bases de datos.



Users_table.sql

```
CREATE TABLE IF NOT EXISTS users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    hashed_password VARCHAR(255) NOT NULL
);
```

Chats_tables.sql

```
CREATE TABLE IF NOT EXISTS chats (
    id SERIAL PRIMARY KEY,
    user_id INTEGER,
    created_at TIMESTAMP WITHOUT TIME ZONE
DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS messages (
    id SERIAL PRIMARY KEY,
    chat_id INTEGER REFERENCES chats(id),
    role VARCHAR NOT NULL,
    content VARCHAR,
    timestamp TIMESTAMP WITHOUT TIME ZONE
DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (chat_id) REFERENCES
chats(id)
);
```

Guía de Despliegue: Contenerización - README

- **Tecnología:** Docker
- **Implementación:** Cada microservicio está contenido en su propio directorio y se despliega como un contenedor Docker independiente.
- **La aplicación consta de tres microservicios principales:**
    - Servicio de Autenticación (auth-service): Maneja el registro, inicio de sesión y validación de tokens JWT para los usuarios.
    - Servicio de Chat (chat-service): Permite a los usuarios crear chats y enviar mensajes, con integración a OpenAI para generar respuestas automáticas.

    - Servicio Frontend (frontend-service): Una aplicación React que sirve como la interfaz de usuario para interactuar con los servicios de autenticación y chat.
- **Github:** **https://github.com/Pablo389/llm-docker-microservices/tree/master**
- **Prerrequisitos**
    - Docker
    - Docker Compose
- **Paso 1: Clonar el Repositorio**

```
git clone https://github.com/tuusuario/llm-docker-microservices.git
cd llm-docker-microservices
```

- **Paso 2: Crear el Archivo .env**
    - Crea un archivo .env en la raíz del proyecto con las siguientes variables de entorno:

```
POSTGRES_USER=tu_usuario_postgres
POSTGRES_PASSWORD=tu_contraseña_postgres
SECRET_KEY=tu_clave_secreta
ACCESS_TOKEN_EXPIRE_MINUTES=30
OPENAI_API_KEY=tu_clave_api_openai
```

- **Paso 3: Desplegar la Arquitectura con Docker Compose**
    - Ejecuta el siguiente comando para iniciar todos los servicios:

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

      Esto hace lo siguiente:

    - Levanta dos bases de datos PostgreSQL, una para el servicio de autenticación y otra para el servicio de chat.
    - Construye y despliega los contenedores para auth-service, chat-service y frontend-service.
    - Configura las bases de datos usando los scripts SQL proporcionados en sql-scripts.

- **Servicios y Puertos**
    - auth-service: Disponible en http://localhost:8000
    - chat-service: Disponible en http://localhost:8001
    - frontend-service: Disponible en http://localhost:300

- **Comandos Adicionales**

Detener los servicios:

```
docker-compose down
```

Ver logs de los servicios:

```
docker-compose logs -f
```