Project Report: Microservices Architecture with Spring Boot and Java

Overview

This report outlines the development of a microservices-based application using Spring Boot 3.2 and Java 21. The architecture is designed to handle different aspects of a social networking platform, similar to LinkedIn, with distinct services for user management, profile handling, and post interactions.

Microservices Description

Discovery Server

- Port: 8761

- Technology: Eureka

- Purpose: Acts as a service registry to enable service discovery and facilitate inter-service communication.

User Service

- Port: 8080

- Features:
  - User registration and authentication with JWT-based Spring Security.
  - CRUD operations on user data.
- Additional Implementation Details:
  - Swagger UI integration for API documentation, accessible at `/swagger-ui/index.html`.
  - PostgreSQL database running on port 5432.

Profile Service

- Port: 8081
- Features:
  - Manages user profiles, including education, experience, and skills.
  - Communicates with the User service to verify the existence of users.
- Additional Implementation Details:
  - Swagger UI for API documentation.
  - Uses `RestTemplate` for communication with the User service.
  - PostgreSQL database running on port 5434.

Post Service

- Port: 8082
- Features:
  - Handles posts, comments, and likes (both for posts and comments).

  - Interacts with the Profile service to validate profile IDs.

- Additional Implementation Details:

  - Swagger UI for API documentation.

  - PostgreSQL database running on port 5433.

 BFF (Backend For Frontend)

- Port: 8084

- Features:

  - Aggregates and presents data from other microservices to the frontend.

  - Implements the APIs detailed earlier in this conversation.

- **Additional Implementation Details:**

  - Swagger UI for API documentation.

 Deployment and Integration

- Docker Integration:

  - Each service, including databases, is containerized using Docker, with Dockerfiles in each service directory.

  - A `docker-compose` file at the root folder to deploy the databases and services.

- Database Configuration:

  - All services use PostgreSQL databases, running on different ports (5432 for User, 5433 for Post, 5434 for Profile).

Technical Implementations

- Swagger UI Integration:

  - Utilized `springdoc-openapi-starter-webmvc-ui` for integrating Swagger UI, enabling comprehensive API documentation and testing.

- Lombok Library:

  - Used to reduce boilerplate code, especially for model/data objects.

- MapStruct Mapper:

  - Incorporated for efficient mapping between DTOs and entities, enhancing the maintainability and clarity of the code.

Conclusion

The development of this microservice architecture demonstrates the implementation of a scalable and efficient system using modern Java technologies. Each microservice is focused on a specific domain, ensuring separation of concerns and facilitating easier maintenance and scalability. The use of Swagger UI and Docker has streamlined both the documentation process and deployment strategy, making the application robust and developer-friendly.