AUTHENTICATION MICROSERVICE

Technical Design

OVERVIEW:

Its primary purpose is to manage user identities, securely validate user credentials, and issue authentication tokens to enable secure access to various parts of the system.

ARCHITECTURE:

Programming Language: Java (17)

Framework: Spring Boot (Java)

Database: MySQL/AWS

Authentication: OAuth 2.0 for secure authentication.

HIGH-LEVEL ARCHITECTURE:

RESTful API endpoints for communication.

It interacts with other microservices such as the order Microservice,

Order Microservice, and Authentication Microservice.

FUNCTIONALITY

Endpoints:

User Registration

- Endpoint: `/api/auth/register`
- Method: POST
- Request Body:
  - Username
  - Email
  - Password
- Response:
  - User ID
  - Username
  - Email
  - Token

User Login

- Endpoint: `/api/auth/login`
- Method: POST
- Request Body:
    - Email
    - Password
- Response:
    - User ID
    - Username
    - Email
    - Token

User Logout

- Endpoint: `/api/auth/logout`
- Method: POST
- Request Headers:
    - Authorization: Bearer <token>

Get User Information by ID

- Endpoint: `/api/auth/user/{id}`
- Method: GET
- Request Headers:
    - Authorization: Bearer <token>
- Response:
    - User ID
    - Username
    - Email
    - Profile Information
    - Roles

Update User Profile Information

- Endpoint: `/api/auth/update`
- Method: PUT
- Request Headers:
    - Authorization: Bearer <token>
- Request Body:
    - Updated profile information (e.g., Username, Email, Password)
- Response:
    - Success message

Request Password Reset

- Endpoint: `/api/auth/reset-password`
- Method: POST
- Request Body:
    - Email
- Response:

        - Success message
        - Reset Token (sent via email)

Validate User Authentication Token

- Endpoint: `/api/auth/validate-token`
- Method: POST
- Request Headers:
    - Authorization: Bearer <token>
- Response:

Security Considerations:

- Use HTTPS for secure communication.
- Store passwords securely using hashing algorithms (e.g., bcrypt).
- Implement token-based authentication (JWT) for session management.
- Rate limit login attempts to prevent brute force attacks.
- Implement proper input validation to prevent injection attacks.

Error Handling:

- Return appropriate HTTP status codes for different scenarios (e.g., 200 for success, 400 for bad request, 401 for unauthorized, 404 for not found).
- Provide detailed error messages in the response body for debugging during development but limit the information returned in production.

TESTING

Implement unit testing with code coverage

integration tests (if Required).

end-to-end tests to ensure the reliability and stability of the microservice**.**

DOCUMENTATION:

Create comprehensive API documentation using tools like Swagger or OpenAPI.

DEPLOYMENT:

Utilize continuous integration and continuous deployment (CI/CD) pipelines for automated testing and deployment.

Additional Considerations:

- Implement middleware for authentication to validate tokens on protected routes.
- Consider role-based access control (RBAC) for managing user roles and permissions.
- Implement email verification for user registration.