**Low-Level Design Document for Insurance-backend**

**1. Overview**

The **Insurance-backend** project is a Spring Boot application providing backend services for an insurance-related platform. It includes JWT-based user authentication, data storage with H2 in-memory database, and email notifications using Jakarta Mail.

**2. Architecture**

**Layers:**

1. **Controller Layer**: Handles HTTP requests and responses, routing them to the appropriate service.
2. **Service Layer**: Contains business logic, processing requests, and interacting with the data layer.
3. **Repository Layer**: Interfaces with the H2 database, managing CRUD operations.
4. **Utility Layer**: Provides helper functions, utilities for JWT management, and email processing.

**3. Modules and Components**

**3.1 Authentication Module**

* **Purpose**: Manages user authentication and authorization using JWT tokens.
* **Components**:
  + AuthController: Handles login and registration endpoints.
  + JwtTokenUtil: Utility class for generating, parsing, and validating JWT tokens.
  + JwtRequestFilter: Filters incoming requests to validate JWT tokens.
  + UserDetailsServiceImpl: Implements UserDetailsService to retrieve user information.
* **Flow**:
  + **Login**:
    - User credentials are validated.
    - If valid, a JWT token is generated and returned.
  + **JWT Validation**:
    - Requests are intercepted by JwtRequestFilter.
    - Token is validated, and user details are retrieved for authorized access.

**3.2 User Module**

* **Purpose**: Manages user data, profiles, and roles.
* **Components**:
  + UserController: Exposes endpoints for retrieving, updating user profiles.
  + UserService: Handles business logic for user data.
  + UserRepository: Provides CRUD operations for user data.
  + User Entity: Maps to the users table, storing user details.
* **Database Design**:
  + **Table**: User (cannot be named users due to H2 restriction).
  + **Fields**: userId, username, password, email, role, createdDate.

**3.3 Policy Module**

* **Purpose**: Manages insurance policies for users.
* **Components**:
  + PolicyController: Exposes endpoints for creating, updating, and viewing policies.
  + PolicyService: Contains business logic for policy management.
  + PolicyRepository: Interface for interacting with the policy table.
  + Policy Entity: Represents insurance policy details.
* **Database Design**:
  + **Table**: Policy
  + **Fields**: policyId, userId, policyType, premiumAmount, startDate, endDate, status.

**3.4 Claim Module**

* **Purpose**: Manages insurance claims filed by users.
* **Components**:
  + ClaimController: Handles endpoints for filing and viewing claims.
  + ClaimService: Business logic for processing claims.
  + ClaimRepository: Database operations for claim data.
  + Claim Entity: Represents insurance claim details.
* **Database Design**:
  + **Table**: Claim
  + **Fields**: claimId, policyId, claimAmount, claimDate, status, remarks.

**3.5 Email Notification Module**

* **Purpose**: Sends notification emails to users for registration, claims, and policy updates.
* **Components**:
  + EmailService: Configures Jakarta Mail, prepares and sends email content.
  + EmailTemplates: Template provider for common email formats (registration, claims, policy updates).
* **Flow**:
  + EmailService is called by various modules (e.g., User, Claim).
  + Templates from EmailTemplates are populated and sent using Jakarta Mail.

**3.6 Utilities Module**

* **Components**:
  + PasswordEncoderUtil: Utility class to hash passwords.
  + DateUtil: Date formatting and validation.
  + FileUploadUtil: Manages the file upload folder structure in the main project folder.

**4. API Design**

**4.1 Endpoints**

**Authentication**

* POST /auth/login: Logs in a user, returns a JWT token.
* POST /auth/register: Registers a new user.

**User**

* GET /users/{id}: Fetch user profile by ID.
* PUT /users/{id}: Update user profile.

**Policy**

* POST /policies: Create a new insurance policy.
* GET /policies/{id}: Retrieve policy by ID.
* PUT /policies/{id}: Update policy information.
* DELETE /policies/{id}: Delete policy.

**Claim**

* POST /claims: File a new claim.
* GET /claims/{id}: Retrieve claim by ID.
* PUT /claims/{id}: Update claim status.

**5. Database Design**

**ER Diagram**

* **User** (1) --- (N) **Policy**
* **Policy** (1) --- (N) **Claim**

**Entity Definitions**

1. **User Table**:
   * userId (PK), username, password, email, role, createdDate
2. **Policy Table**:
   * policyId (PK), userId (FK), policyType, premiumAmount, startDate, endDate, status
3. **Claim Table**:
   * claimId (PK), policyId (FK), claimAmount, claimDate, status, remarks

**6. Error Handling**

* **GlobalExceptionHandler**:
  + Handles generic exceptions (e.g., Exception, EntityNotFoundException).
  + Returns standardized error response structure: {timestamp, status, message, details}.

**7. Security**

* **JWT Authentication**: Secures endpoints with JWT validation.
* **Password Encryption**: Uses PasswordEncoderUtil to hash passwords.
* **Role-Based Access Control**: Restricts endpoints based on user roles (Admin, User).

**8. Deployment**

* **Dev Environment**: Localhost with H2 in-memory database for development.
* **Production Considerations**: Update database to persistent storage, add secure email configurations, ensure JWT secret management