Project: Insurance Management System

1. Introduction

The purpose of this document is to provide a detailed Low-Level Design (LLD) for the Insurance Management System. The system facilitates streamlined insurance operations, including customer management, policy handling, and claim processing. It employs a REST API-based backend architecture and uses either Angular or React for the frontend.

This design is compatible with both Java (Spring Boot) and .NET (ASP.NET Core) frameworks.

2. Module Overview

The project consists of the following modules:

2.1 Customer Management

- Manages customer data, including personal information, contact details, and policy associations.

2.2 Policy Management

- Handles creation, updating, and archiving of insurance policies.

2.3 Claim Processing

- Facilitates filing, reviewing, and approval/rejection of claims.

2.4 Agent Management

- Manages insurance agents, their profiles, and assigned policies.

2.5 Notification and Alerts

- Manages communication with customers and agents via notifications and alerts (e.g., policy renewals, claim status updates).

3. Architecture Overview

3.1 Architectural Style

- **Frontend:** Angular or React
- **Backend:** REST API-based architecture
- **Database:** Relational Database (MySQL/SQL Server)

3.2 Component Interaction

1. **Frontend** communicates with the **REST API backend** via HTTP/HTTPS.
2. **Backend** interacts with the **database layer** for CRUD operations.

3. **Frontend** handles user interactions and renders dynamic views.

4. Module-Wise Design

4.1 Customer Management Module

4.1.1 Features

- Add, update, and delete customer profiles
- Search and view customer details

4.1.2 Data Flow

1. User requests to create/update/view customer details via the frontend.
2. The frontend sends the request to the REST API.
3. Backend processes the request and performs necessary database operations.
4. Response is sent back to the frontend for display.

4.1.3 Entities

- Customer
    - CustomerID
    - Name
    - Email
    - Phone
    - Address

4.2 Policy Management Module

4.2.1 Features

- Create, update, and manage insurance policies
- Link policies to customers and agents

4.2.2 Data Flow

1. Admin/Agent initiates policy-related operations via the frontend.
2. Request is processed by the backend and mapped to database actions.
3. Results are displayed on the frontend.

4.2.3 Entities

- Policy

- PolicyID
- Name
- PremiumAmount
- CoverageDetails
- ValidityPeriod

4.3 Claim Processing Module

4.3.1 Features

- File a claim
- Review and process claims
- Track claim status

4.3.2 Data Flow

1. Customers/Agents file claims via the frontend.
2. Backend validates claim requests and updates the database.
3. Claim status updates are retrieved by users through the frontend.

4.3.3 Entities

- Claim
  - ClaimID
  - PolicyID
  - CustomerID
  - ClaimAmount
  - Status (Filed, Under Review, Approved, Rejected)

4.4 Agent Management Module

4.4.1 Features

- Manage agent profiles
- Assign policies to agents

4.4.2 Data Flow

1. Admin performs agent-related operations via the frontend.
2. Requests are routed to the backend for database interactions.
3. Responses are rendered back to the admin interface.

4.4.3 Entities

- Agent
    - AgentID
    - Name
    - ContactInfo
    - AssignedPolicies

4.5 Notification and Alerts Module

4.5.1 Features

- Notify users of policy updates
- Send alerts for upcoming renewals and claim decisions

4.5.2 Data Flow

1. Backend generates notifications based on triggers (e.g., policy expiration).
2. Notifications are sent via email/SMS or displayed in the frontend interface.

5. Deployment Strategy

5.1 Local Deployment

The system will be deployed in a local environment. Each developer will set up the application on their local machine for testing and development purposes.

5.1.1 Frontend Deployment

- Use local servers like ng serve for Angular or local development servers for React.

5.1.2 Backend Deployment

- Deploy the REST API on local servers using Spring Boot/ASP.NET Core.

5.1.3 Database

- Set up a local instance of the database for development and testing.

6. Database Design

6.1 Tables and Relationships

6.1.1 Customer

- Primary Key: CustomerID
- Foreign Keys: None

6.1.2 Policy

- Primary Key: PolicyID
- Foreign Key: CustomerID

6.1.3 Claim

- Primary Key: ClaimID
- Foreign Keys: PolicyID, CustomerID

6.1.4 Agent

- Primary Key: AgentID
- Foreign Key: None

6.1.5 Notification

- Primary Key: NotificationID
- Foreign Key: CustomerID

7. User Interface Design

7.1 Wireframes

1. Customer Dashboard
2. Policy Management Interface
3. Claim Filing and Tracking Page
4. Agent Management Console
5. Notifications and Alerts Panel

8. Non-Functional Requirements

8.1 Performance

The system must handle up to 100 concurrent users in the local environment.

8.2 Scalability

Designed to be easily scalable if moved to a production-grade environment in the future.

8.3 Security

Data must be secured using appropriate access controls and encryption mechanisms.

8.4 Usability

The user interface must be intuitive and responsive.

9. Assumptions and Constraints

9.1 Assumptions

- The system will run only in a local environment for development and testing.

9.2 Constraints

- No cloud deployment is planned at this stage.
- Developers will use standard IDEs for coding and debugging.