Project Title: Insurance Policy Management System

1. Overview

The **Insurance Policy Management System (IPMS)** is a web-based application that allows administrators and customers to manage insurance-related operations effectively. The system is modular and follows the **MVC architecture** for flexibility and scalability, making it compatible with both **Java (Spring MVC)** and **.NET (ASP.NET Core MVC)** frameworks.

The core modules include:

1. **Policy Management** – Handles creation, updates, and management of insurance policies.
2. **Claims Processing** – Facilitates claim submission, processing, and settlement.
3. **Customer Management** – Manages customer profiles and interactions.
4. **Premium Calculation** – Calculates premiums dynamically based on risk factors.
5. **User Management** – Manages user authentication, authorization, and profiles.

2. Assumptions

1. The application will be deployed locally during development using a relational database (e.g., MySQL or MS SQL).
2. Security mechanisms will include role-based authentication.
3. ORM frameworks (Hibernate for Java or Entity Framework for .NET) will handle database interactions.
4. No containerization will be used for local deployment.

3. Module-Level Design

3.1 Policy Management Module

Purpose: Handles operations for policy lifecycle management.

- **Controller**:
    - PolicyController
        - createPolicy(policyData)
        - updatePolicy(policyId, policyData)
        - getPolicyDetails(policyId)
        - deletePolicy(policyId)

- **Service**:
    - PolicyService
        - Validate policy data.
        - Enforce business rules.
- **Model**:
    - **Entity**: Policy
        - Attributes:
            - policyId (PK)
            - policyType (VARCHAR)
            - coverageAmount (DECIMAL)
            - premiumAmount (DECIMAL)
            - validityStartDate (DATE)
            - validityEndDate (DATE)

3.2 Claims Processing Module

Purpose: Facilitates insurance claim management.

- **Controller**:
    - ClaimsController
        - submitClaim(claimData)
        - processClaim(claimId, status)
        - getClaimDetails(claimId)
- **Service**:
    - ClaimsService
        - Validate claims against policy terms.
- **Model**:
    - **Entity**: Claim
        - Attributes:
            - claimId (PK)
            - policyId (FK)
            - claimAmount (DECIMAL)

        - claimStatus (ENUM)
        - submissionDate (DATE)
        - settlementDate (DATE)

3.3 Customer Management Module

Purpose: Manages customer profiles and data.

- **Controller**:
    - CustomerController
        - addCustomer(customerData)
        - updateCustomer(customerId, customerData)
        - getCustomerDetails(customerId)
- **Service**:
    - CustomerService
        - Handle customer lifecycle operations.
- **Model**:
    - **Entity**: Customer
        - Attributes:
            - customerId (PK)
            - name (VARCHAR)
            - email (VARCHAR)
            - phone (VARCHAR)
            - address (VARCHAR)

3.4 Premium Calculation Module

Purpose: Calculates premiums dynamically based on customer and policy factors.

- **Controller**:
    - PremiumController
        - calculatePremium(policyId, customerId)
- **Service**:
    - PremiumService
        - Apply algorithms to calculate risk-adjusted premiums.

- **Model**:
    - **Entity**: PremiumCalculation
        - Attributes:
            - calculationId (PK)
            - policyId (FK)
            - customerId (FK)
            - basePremium (DECIMAL)
            - adjustedPremium (DECIMAL)

3.5 User Management Module

Purpose: Manages authentication and role-based access control.

- **Controller**:
    - UserController
        - registerUser(userData)
        - loginUser(username, password)
        - getUserProfile(userId)
- **Service**:
    - UserService
        - Manage user credentials and roles.
- **Model**:
    - **Entity**: User
        - Attributes:
            - userId (PK)
            - username (VARCHAR)
            - password (VARCHAR, Encrypted)
            - role (ENUM)

4. Database Schema

4.1 Table Definitions

1. **Policy Table**

```
CREATE TABLE Policy (
    policyId INT PRIMARY KEY AUTO_INCREMENT,
    policyType VARCHAR(100),
    coverageAmount DECIMAL(10, 2),
    premiumAmount DECIMAL(10, 2),
    validityStartDate DATE,
    validityEndDate DATE
);
```

2. **Claim Table**

```
CREATE TABLE Claim (
    claimId INT PRIMARY KEY AUTO_INCREMENT,
    policyId INT,
    claimAmount DECIMAL(10, 2),
    claimStatus ENUM('PENDING', 'APPROVED', 'REJECTED'),
    submissionDate DATE,
    settlementDate DATE,
    FOREIGN KEY (policyId) REFERENCES Policy(policyId)
);
```

3. **Customer Table**

```
CREATE TABLE Customer (
    customerId INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    email VARCHAR(100),
    phone VARCHAR(15),
    address TEXT
);
```

4. **PremiumCalculation Table**

```
CREATE TABLE PremiumCalculation (
    calculationId INT PRIMARY KEY AUTO_INCREMENT,
    policyId INT,
    customerId INT,
    basePremium DECIMAL(10, 2),
    adjustedPremium DECIMAL(10, 2),
    FOREIGN KEY (policyId) REFERENCES Policy(policyId),
    FOREIGN KEY (customerId) REFERENCES Customer(customerId)
);
```

5. **User Table**

```
CREATE TABLE User (
   userId INT PRIMARY KEY AUTO_INCREMENT,
   username VARCHAR(50) UNIQUE,
   password VARCHAR(255),
   role ENUM('ADMIN', 'USER')
);
```

5. Local Deployment Details

1. **Environment Setup:**
    - Install JDK 17 or .NET SDK 7.0.
    - Install MySQL or SQL Server.
    - Configure application server (Tomcat for Java, Kestrel for .NET).
2. **Deployment Steps:**
    - Clone the repository.
    - Configure database connection strings in application.properties (Java) or appsettings.json (.NET).
    - Run migration scripts to create the database schema.
    - Build and run the application locally.

6. Conclusion

This document provides a comprehensive low-level design for the **Insurance Policy Management System**, ensuring modularity, security, and compatibility for development in **Spring MVC** or **ASP.NET Core MVC**.