Project Title: Vehicle Insurance Management System

1. Overview

The **Vehicle Insurance Management System** is designed to provide a comprehensive solution for managing vehicle insurance policies, claims, and customer interactions. It allows policyholders to purchase, update, and renew their policies, while enabling agents and adjusters to process claims efficiently. The system follows the **MVC architecture** and is designed to be compatible with both **Java (Spring MVC)** and **.NET (ASP.NET Core MVC)** frameworks.

The system is divided into five core modules:

1. **Policy Management** – Manages vehicle insurance policies.
2. **Claim Management** – Handles the claim submission, approval, and tracking process.
3. **Payment Management** – Manages payments for policies and claims.
4. **Customer Support** – Provides support for customers regarding their policies and claims.
5. **User Authentication & Authorization** – Manages user authentication and access control.

2. Assumptions

1. The system will be deployed locally using a relational database (MySQL or SQL Server).
2. The system will use role-based authentication for securing sensitive data and ensuring access control.
3. The application will use ORM tools like Hibernate (Java) and Entity Framework (ASP.NET Core) for data management.
4. Payment processing will be simulated using mock data for local deployment.
5. The system will use a file system or cloud storage for document management.

3. Module-Level Design

3.1 Policy Management Module

Purpose: This module handles the creation, management, and updating of vehicle insurance policies.

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

        - deletePolicy(policyId)

- **Service:**
    - PolicyService
        - Contains business logic for creating, updating, and retrieving policies.
- **Model:**
    - **Entity:** Policy
        - Attributes:
            - policyId (PK)
            - policyNumber (VARCHAR)
            - policyholderId (FK)
            - vehicleId (FK)
            - coverageAmount (DECIMAL)
            - policyStatus (ENUM: ACTIVE, INACTIVE, EXPIRED)
            - createdDate (DATE)

3.2 Claim Management Module

Purpose: Handles the submission, processing, and approval of claims submitted by policyholders.

- **Controller:**
    - ClaimController
        - submitClaim(claimData)
        - getClaimDetails(claimId)
        - updateClaimStatus(claimId, status)
        - getAllClaimsByPolicy(policyId)
- **Service:**
    - ClaimService
        - Handles business logic for claim processing.
- **Model:**
    - **Entity:** Claim
        - Attributes:
            - claimId (PK)

        - policyId (FK)
        - claimAmount (DECIMAL)
        - claimDate (DATE)
        - claimStatus (ENUM: PENDING, APPROVED, REJECTED)
        - adjusterId (FK)

3.3 Payment Management Module

Purpose: This module is responsible for handling payments made by policyholders for policies and claims.

- **Controller:**
    - PaymentController
        - makePayment(paymentData)
        - getPaymentDetails(paymentId)
        - getPaymentsByPolicy(policyId)
- **Service:**
    - PaymentService
        - Handles the logic for processing payments.
- **Model:**
    - **Entity:** Payment
        - Attributes:
            - paymentId (PK)
            - policyId (FK)
            - amount (DECIMAL)
            - paymentDate (DATE)
            - paymentStatus (ENUM: SUCCESS, PENDING, FAILED)

3.4 Customer Support Module

Purpose: Manages customer support tickets, assisting policyholders and claimants with their concerns.

- **Controller:**
    - SupportController
        - createTicket(ticketData)

        - getTicketDetails(ticketId)
        - resolveTicket(ticketId)
        - getAllTicketsByUser(userId)
- **Service:**
    - SupportService
        - Manages the business logic for ticket creation and resolution.
- **Model:**
    - **Entity:** SupportTicket
        - Attributes:
            - ticketId (PK)
            - userId (FK)
            - issueDescription (TEXT)
            - ticketStatus (ENUM: OPEN, RESOLVED)
            - createdDate (DATE)

3.5 User Authentication & Authorization Module

Purpose: Handles user login, registration, and role-based access control for secure operations.

- **Controller:**
    - AuthController
        - registerUser(userData)
        - loginUser(username, password)
        - logoutUser()
        - getUserProfile(userId)
- **Service:**
    - AuthService
        - Manages user authentication, registration, and role management.
- **Model:**
    - **Entity:** User
        - Attributes:
            - userId (PK)

- username (VARCHAR)
- password (VARCHAR, Encrypted)
- role (ENUM: ADMIN, AGENT, CLAIM_ADJUSTER, POLICYHOLDER)
- email (VARCHAR)

4. Database Schema

4.1 Table Definitions

1. **Policy Table**

```
CREATE TABLE Policy (
  policyId INT PRIMARY KEY AUTO_INCREMENT,
  policyNumber VARCHAR(50),
  policyholderId INT,
  vehicleId INT,
  coverageAmount DECIMAL(10, 2),
  policyStatus ENUM('ACTIVE', 'INACTIVE', 'EXPIRED'),
  createdDate DATE,
  FOREIGN KEY (policyholderId) REFERENCES User(userId),
  FOREIGN KEY (vehicleId) REFERENCES Vehicle(vehicleId)
);
```

2. **Claim Table**

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

3. **Payment Table**

```
CREATE TABLE Payment (
  paymentId INT PRIMARY KEY AUTO_INCREMENT,
  policyId INT,
  amount DECIMAL(10, 2),
  paymentDate DATE,
  paymentStatus ENUM('SUCCESS', 'PENDING', 'FAILED'),
  FOREIGN KEY (policyId) REFERENCES Policy(policyId)
);
```

4. **SupportTicket Table**

```
CREATE TABLE SupportTicket (
  ticketId INT PRIMARY KEY AUTO_INCREMENT,
  userId INT,
  issueDescription TEXT,
  ticketStatus ENUM('OPEN', 'RESOLVED'),
  createdDate DATE,
  FOREIGN KEY (userId) REFERENCES User(userId)
);
```

5. **User Table**

```
CREATE TABLE User (
  userId INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) UNIQUE,
  password VARCHAR(255),
  role ENUM('ADMIN', 'AGENT', 'CLAIM_ADJUSTER', 'POLICYHOLDER'),
  email VARCHAR(100)
);
```

6. **Vehicle Table**

```
CREATE TABLE Vehicle (
  vehicleId INT PRIMARY KEY AUTO_INCREMENT,
  vehicleNumber VARCHAR(50),
  vehicleMake VARCHAR(50),
  vehicleModel VARCHAR(50),
  vehicleYear INT,
  ownerId INT,
  FOREIGN KEY (ownerId) REFERENCES User(userId)
);
```

5. Local Deployment Details

1. **Environment Setup:**
   - Install JDK 17 or .NET SDK 7.0.
   - Install MySQL or SQL Server locally.
   - Use Tomcat (Java) or Kestrel (ASP.NET Core) for local server hosting.
2. **Deployment Steps:**
   - Clone the repository to the local machine.
   - Configure the database connection settings in application.properties (Java) or appsettings.json (.NET).
   - Execute SQL scripts to set up the database schema.

- Build and run the application locally.

6. Conclusion

This **Vehicle Insurance Management System** provides a detailed low-level design, breaking down the application into five key modules: **Policy Management**, **Claim Management**, **Payment Management**, **Customer Support**, and **User Authentication & Authorization**. The database schema ensures the proper organization of data while the modular design supports scalability and maintainability. The system is compatible with both **Spring MVC** and **ASP.NET Core MVC** frameworks and can be deployed locally during development.