Project Title: Auto Insurance Management System

1. Overview

The **Auto Insurance Management System** is designed to handle the end-to-end process of managing auto insurance policies, claims, payments, and customer support services. The system provides functionalities for policy creation, claim submission and approval, payment processing, and support ticket management, all underpinned by a **Model-View-Controller (MVC) architecture**.

This design accommodates both **Java (Spring MVC)** and **.NET (ASP.NET Core MVC)** development frameworks. The system is divided into five primary modules:

1. **Policy Management**
2. **Claim Management**
3. **Payment Management**
4. **Customer Support**
5. **User Authentication & Authorization**

2. Assumptions

- The system is to be deployed locally, using MySQL or SQL Server as the database.
- The system will include role-based access for different types of users (e.g., admin, agent, customer).
- ORM tools (Hibernate for Java, Entity Framework for .NET) will be used for database interactions.
- The user interface will be designed to be responsive and easy to navigate.

3. Module-Level Design

3.1 Policy Management Module

Purpose: Responsible for the creation, modification, and retrieval of auto insurance policies.

- **Controller:**
    - PolicyController
        - createPolicy()
        - updatePolicy()
        - getPolicyById()
        - getAllPolicies()
        - deletePolicy()

- **Service:**
  - PolicyService
    - Handles logic for creating, updating, and deleting policies.
    - Retrieves policies based on user queries.
- **Model:**
  - **Policy Entity**
    - Attributes:
      - policyId (PK)
      - policyNumber
      - vehicleDetails
      - coverageAmount
      - coverageType
      - premiumAmount
      - startDate
      - endDate
      - policyStatus (ACTIVE, INACTIVE, RENEWED)

3.2 Claim Management Module

Purpose: Manages claims submitted by customers and their approval/rejection process.

- **Controller:**
  - ClaimController
    - submitClaim()
    - getClaimDetails()
    - updateClaimStatus()
    - getAllClaims()
- **Service:**
  - ClaimService
    - Manages the claim lifecycle, including submission, status updates, and notifications.
- **Model:**

- **Claim Entity**
  - Attributes:
    - claimId (PK)
    - policyId (FK)
    - claimAmount
    - claimDate
    - claimStatus (OPEN, APPROVED, REJECTED)
    - adjusterId (FK)

3.3 Payment Management Module

Purpose: Manages payment processing for policies and claims.

- **Controller:**
  - PaymentController
    - makePayment()
    - getPaymentDetails()
    - getPaymentsByPolicy()
- **Service:**
  - PaymentService
    - Handles payment processing, refunds, and payment status updates.
- **Model:**
  - **Payment Entity**
    - Attributes:
      - paymentId (PK)
      - paymentAmount
      - paymentDate
      - paymentStatus (SUCCESS, FAILED, PENDING)
      - policyId (FK)

3.4 Customer Support Module

Purpose: Provides customer support by managing tickets raised by customers regarding their policies or claims.

- **Controller:**
    - SupportController
        - createTicket()
        - getTicketDetails()
        - resolveTicket()
        - getAllTickets()
- **Service:**
    - SupportService
        - Handles creation, resolution, and tracking of customer support tickets.
- **Model:**
    - **SupportTicket Entity**
        - Attributes:
            - ticketId (PK)
            - userId (FK)
            - issueDescription
            - ticketStatus (OPEN, RESOLVED)
            - createdDate
            - resolvedDate

3.5 User Authentication & Authorization Module

Purpose: Manages authentication and authorization of users to ensure secure access to the system.

- **Controller:**
    - AuthController
        - login()
        - registerUser()
        - logout()
        - getUserProfile()
- **Service:**
    - AuthService
        - Handles user login, registration, and role-based access control.

- **Model:**
  - **User Entity**
    - Attributes:
      - userId (PK)
      - username
      - password (hashed)
      - email
      - role (ENUM: ADMIN, AGENT, CUSTOMER)

4. Database Schema

4.1 Table Definitions

1. **Policy Table**

```
CREATE TABLE Policy (
  policyId INT AUTO_INCREMENT PRIMARY KEY,
  policyNumber VARCHAR(50) NOT NULL,
  vehicleDetails TEXT,
  coverageAmount DECIMAL(10, 2),
  coverageType VARCHAR(100),
  premiumAmount DECIMAL(10, 2),
  startDate DATE,
  endDate DATE,
  policyStatus ENUM('ACTIVE', 'INACTIVE', 'RENEWED')
);
```

2. **Claim Table**

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

3. **Payment Table**

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

4. **SupportTicket Table**

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

5. **User Table**

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

5. Local Deployment Details

1. **Environment Setup:**
   - Ensure that MySQL or SQL Server is installed and configured on the local machine.
   - Install the necessary SDKs (JDK 17 for Java or .NET SDK 7.0 for .NET).
   - The application will be deployed using **Apache Tomcat** (Java) or **Kestrel** (ASP.NET Core) for local server hosting.
2. **Deployment Steps:**
   - Clone the repository from the version control system (GitHub/Bitbucket).
   - Set up the MySQL/SQL Server database with the provided schema.

- Modify the application properties (e.g., application.properties for Java or appsettings.json for .NET) to include database connection details.
- Build and run the application. For Java, deploy using **Apache Tomcat**, and for .NET, deploy using **Kestrel**.

6. Conclusion

The **Auto Insurance Management System** provides a robust design following MVC architecture, divided into functional modules: **Policy Management**, **Claim Management**, **Payment Management**, **Customer Support**, and **User Authentication & Authorization**. The detailed database schema supports efficient storage and management of insurance policies, claims, payments, and support tickets. The system is designed to be deployed locally using a MySQL or SQL Server database.