Auto Insurance Management System

Technical Design Document

	Prepared By / Last Updated By	Reviewed By	Approved By
Name	Sakshi Lele	Team Lead	
Role			
Signature			
Date			

Table of Contents

1.0 Introduction

1.1 Purpose of this document

The purpose of this document is to provide a comprehensive understanding of the Auto Insurance Management System, outlining its architecture, design, modules, database schema, technology stack, and operational framework. This document serves as a guide for developers, testers, project managers, and stakeholders involved in the system's development, deployment, and maintenance. It aims to ensure alignment of the system objectives with stakeholder requirements and to support the system's scalability, maintainability, and usability.

1.2 Project overview

The Auto Insurance Management System is a full-featured web application developed to automate and manage all critical functionalities of auto insurance services. These functionalities include policy management, claim submission and tracking, payment processing, customer support, and secure user authentication. Designed using the MVC (Model-View-Controller) architectural pattern, the system is compatible with both Java Spring MVC and ASP.NET Core MVC frameworks. It incorporates modern web development standards for a seamless, responsive, and secure user experience.

2.0 Solution Summary

2.1 Scope

The Auto Insurance Management System aims to:

- Provide users (Admins, Agents, Customers) with distinct roles and permissions.
- Facilitate seamless policy creation, viewing, updating, and renewal.
- Enable policyholders to submit insurance claims and track their status.
- Process payments through secure, integrated gateways.
- Offer a ticket-based customer support system for issue resolution.
- Maintain a structured and normalized database to ensure data integrity.
- Ensure cross-browser compatibility and responsive UI.

2.2 Assumptions

- Users will have access to stable internet connections.
- The system will be hosted on a secure, scalable cloud infrastructure.
- Payment gateways and email servers will be fully functional.
- Admin users are responsible for managing agents and monitoring the system.
- All user input will be validated before processing.

2.3 Dependencies

- Backend: Java Spring MVC or ASP.NET Core MVC
- Frontend: HTML, CSS, JavaScript, jQuery, Bootstrap 5
- Database: SQL Server or SQLite
- Development Tools: Visual Studio / IntelliJ, Postman, Git
- APIs: Razorpay/Stripe for payments, SMTP for emails
- Hosting: Azure/AWS (optional)

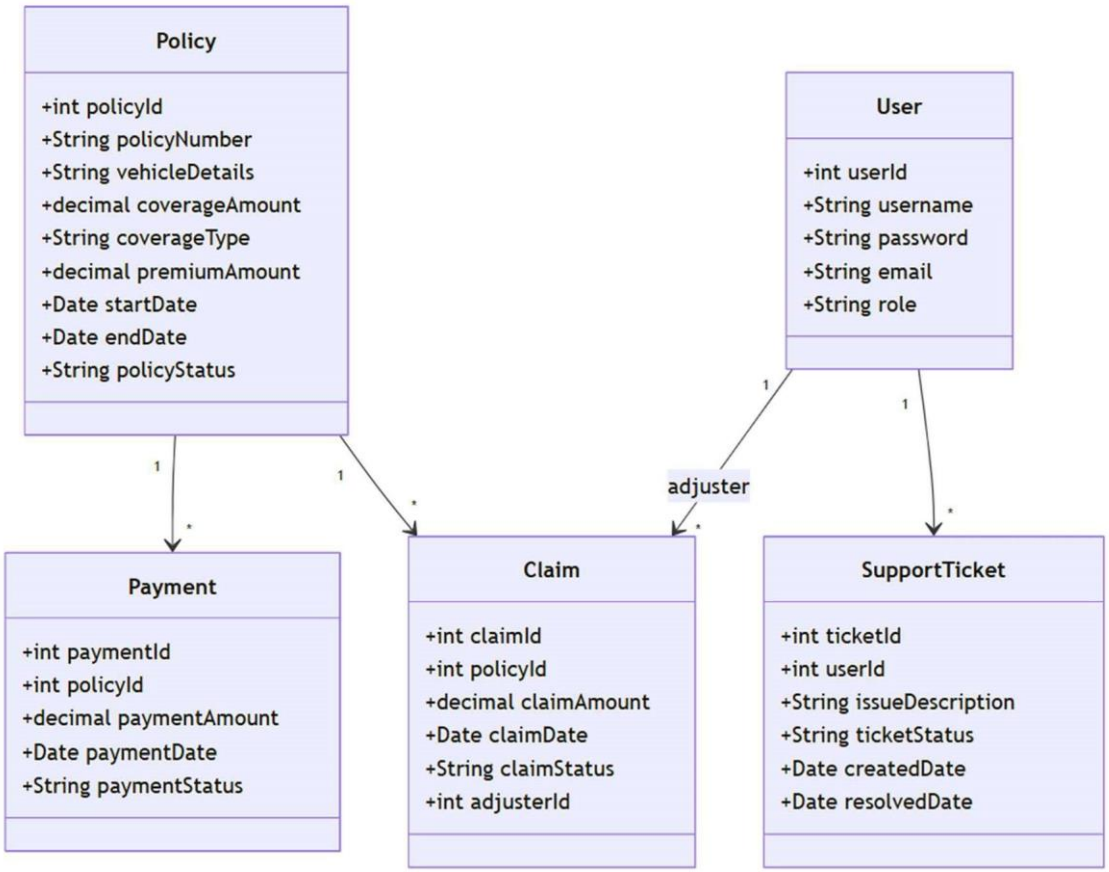
2.4 Risks

- Security Vulnerabilities: Potential data breaches without encryption and access control.
- Downtime: Server failure can lead to service unavailability.
- Data Loss: Risk of data corruption or deletion without proper backups.
- User Errors: Inadequate training may lead to incorrect data entries.
- Regulatory Compliance: Must comply with insurance regulations and data privacy laws.

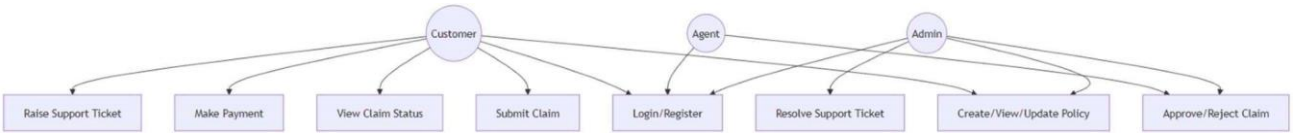
3.0 Schematic Diagram

A schematic, or schematic diagram, is a representation of the elements of a system using abstract, graphic symbols rather than realistic pictures. It gives an overview of overall system.

3.1 Class Diagram

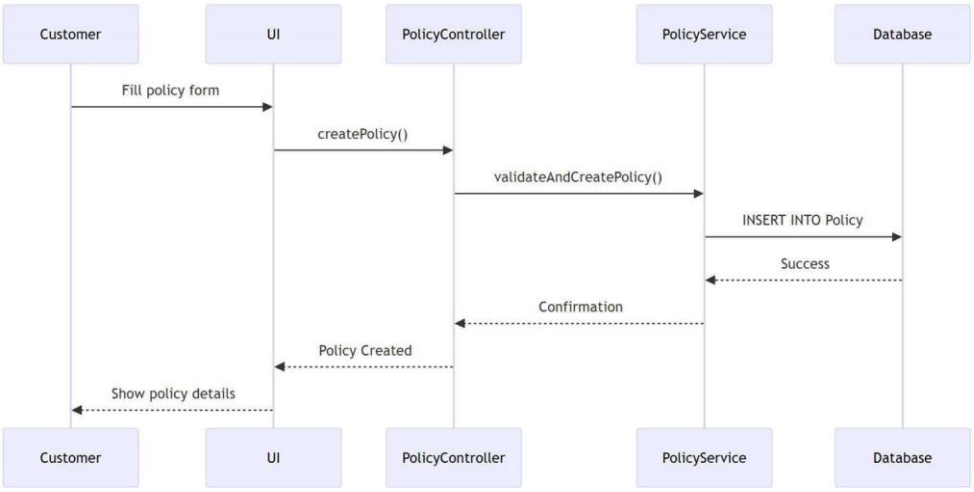


3.2 Use Case Diagram

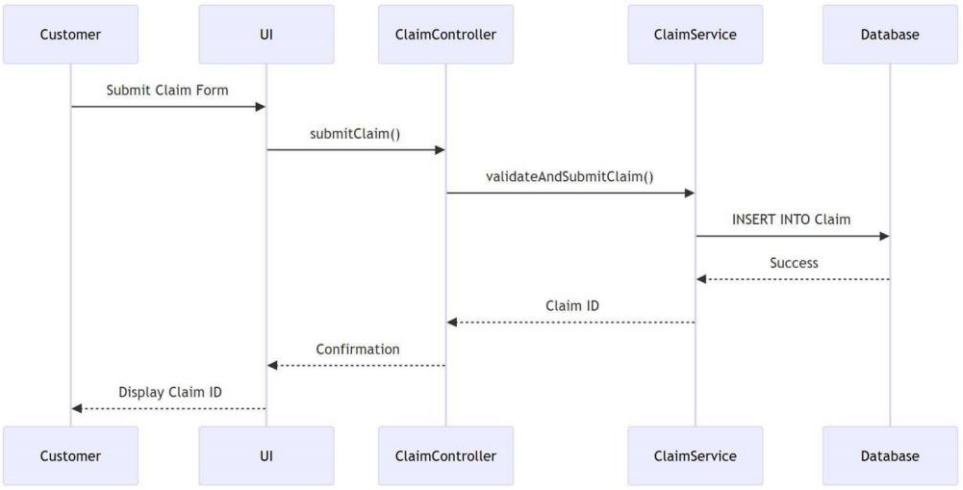


3.3 Sequence Diagram

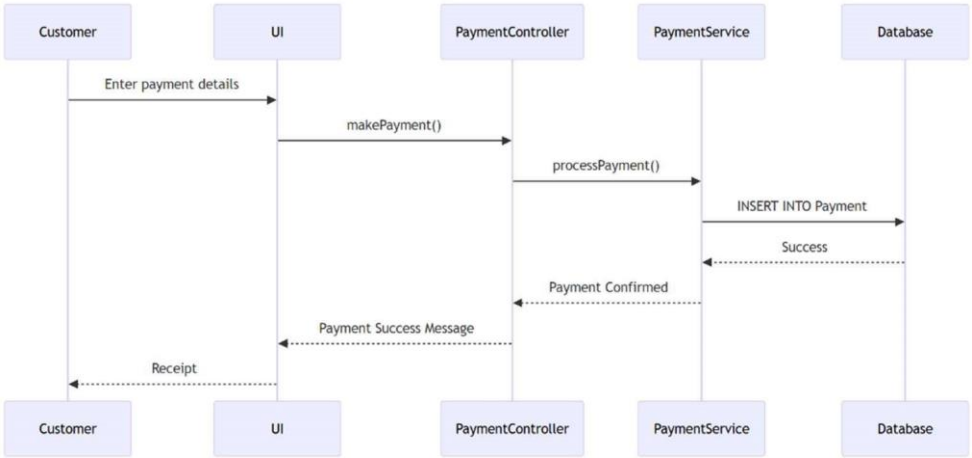
3.3.1 Policy Creation



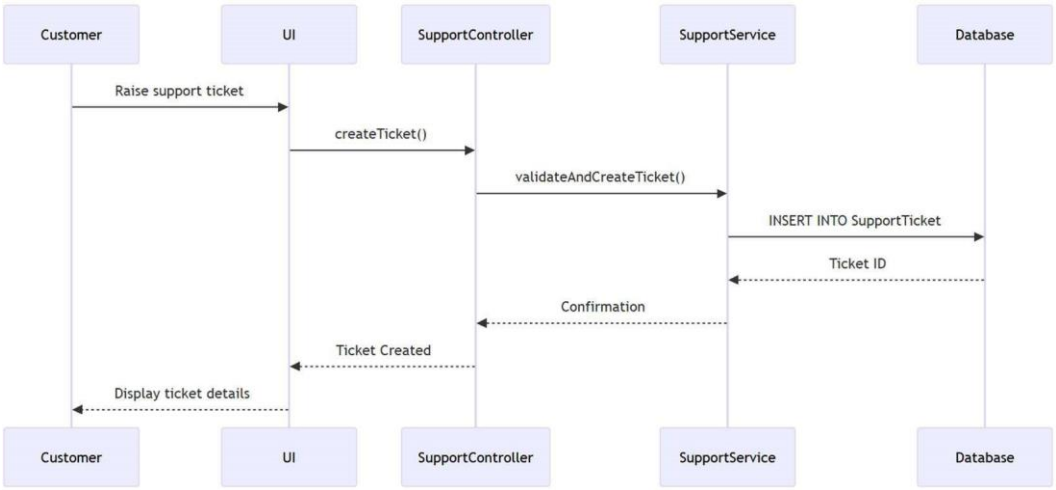
3.3.2 Claim Submission



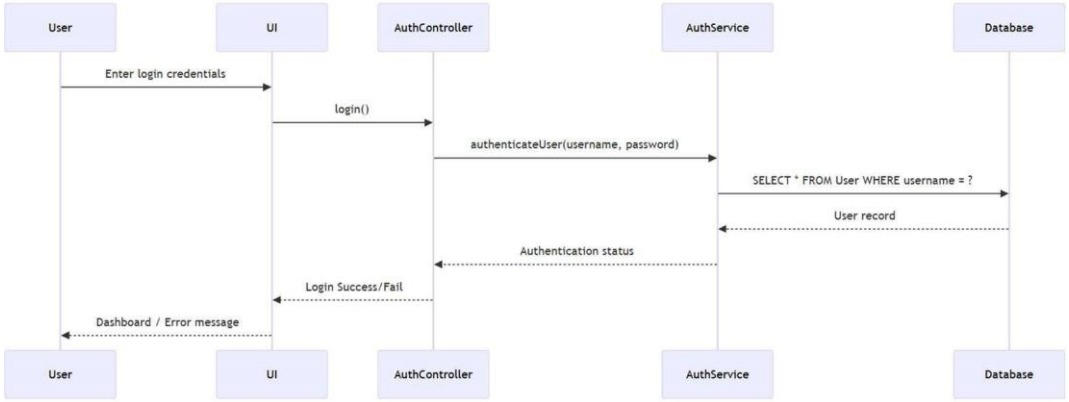
3.3.3 Payment Processing



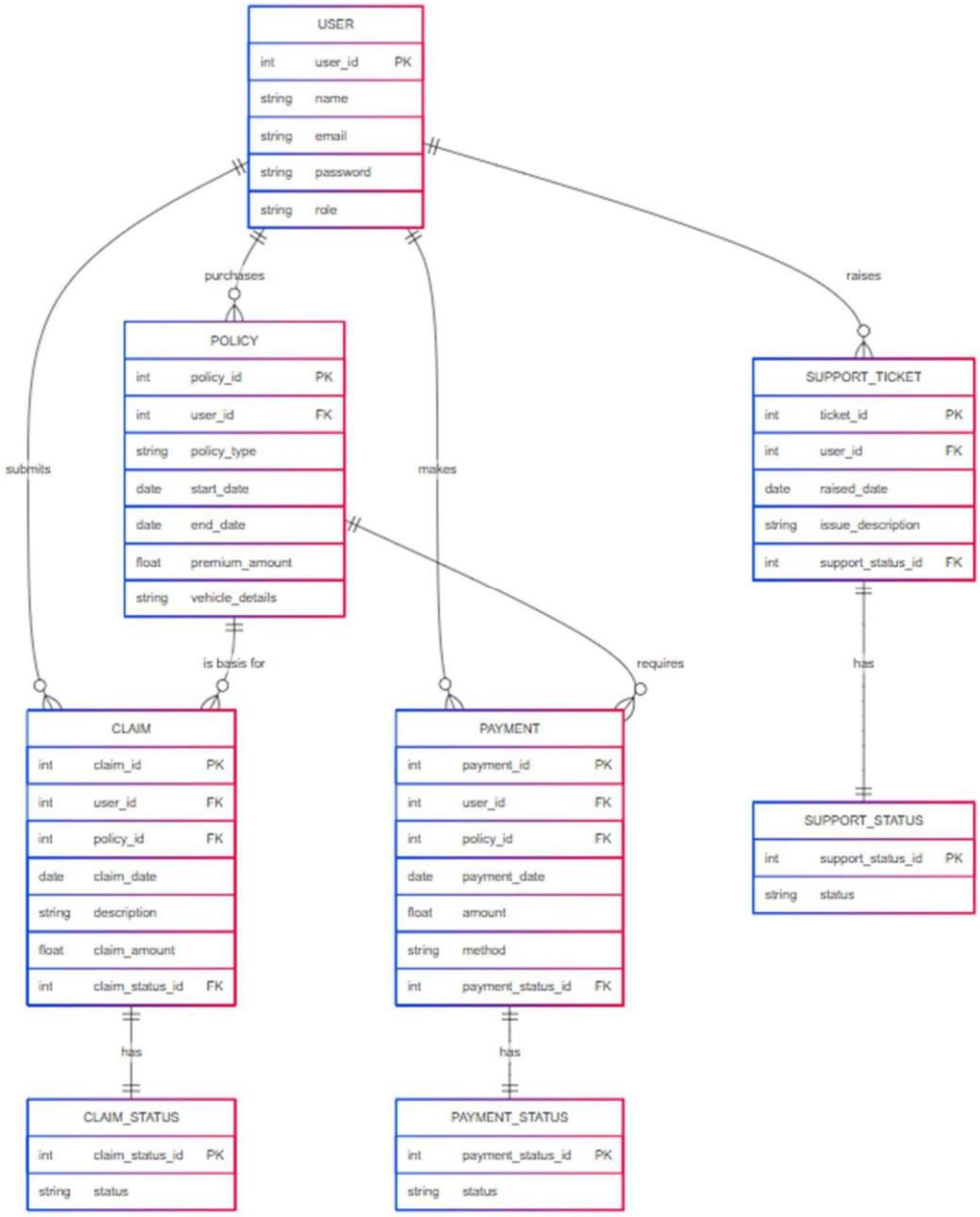
3.3.4 Ticket Raising



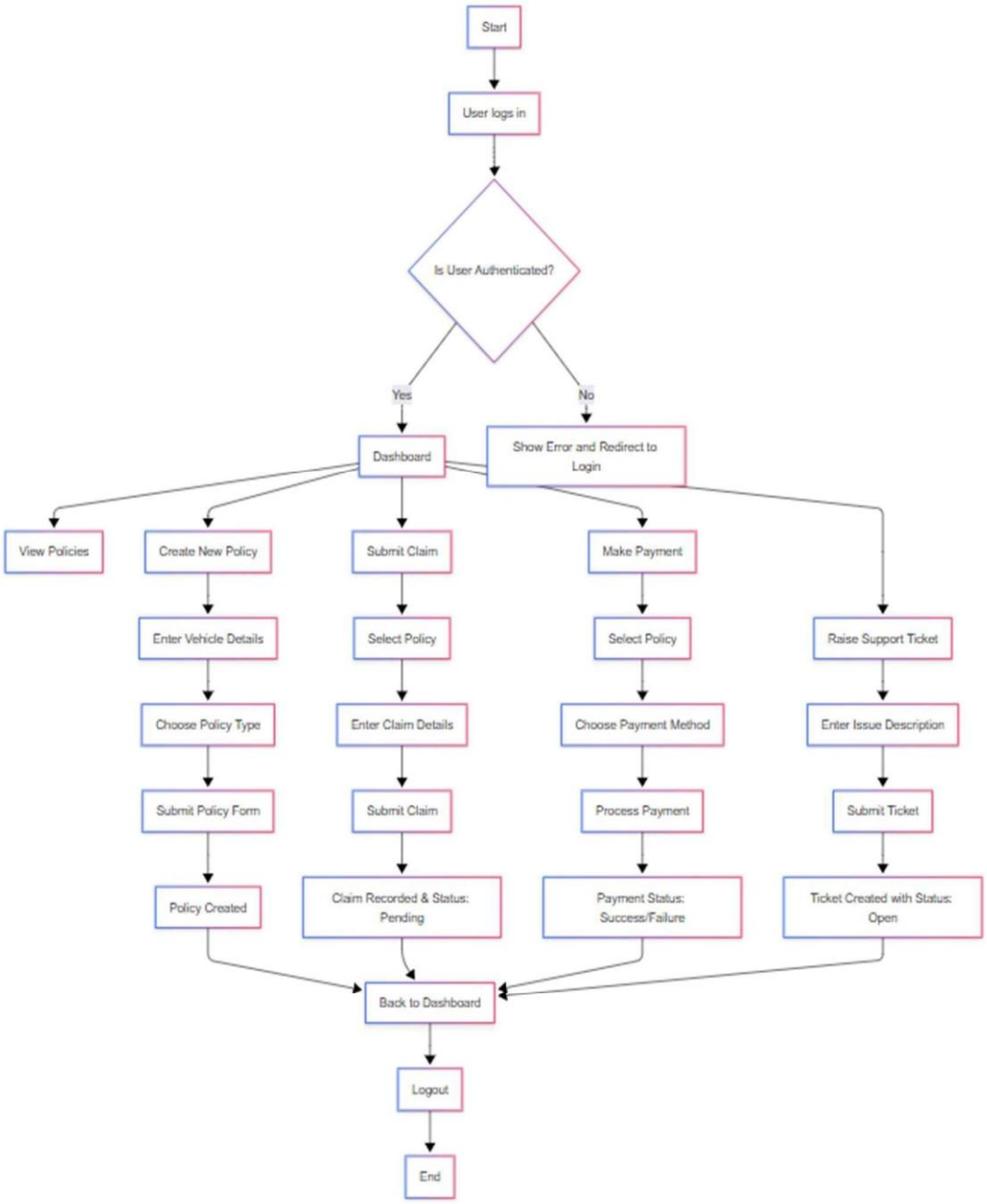
3.3.5 Login



3.4 ER Diagram



3.5 Activity Diagram


Start
User logs in
Is User Authenticated?
Yes
No
Dashboard
Show Error and Redirect to Login
View Policies
Create New Policy
Submit Claim
Make Payment
Raise Support Ticket
Enter Vehicle Details
Select Policy
Select Policy
Choose Policy Type
Enter Claim Details
Choose Payment Method
Enter Issue Description
Submit Policy Form
Submit Claim
Process Payment
Submit Ticket
Policy Created
Claim Recorded & Status: Pending
Payment Status: Success/Failure
Ticket Created with Status: Open
Back to Dashboard
Logout
End

4.0 System Design

4.1 Proposed design

The system follows a layered MVC architecture:

- Client Layer: User accesses system via browser.
- Presentation Layer (View): Displays data using HTML/CSS/Bootstrap.
- Controller Layer: Handles incoming HTTP requests and invokes business logic.
- Service Layer: Contains business logic for processing policies, claims, and payments.
- Data Access Layer (DAO/Repositories): Communicates with the database.
- Database Layer: Stores structured data and enforces relationships.
- External Interfaces: Payment gateways, email APIs.

4.2 Component inventory

- UI Pages: Login, Dashboard, Policies, Claims, Payments, Support
- Controllers:

`UserController`

`PolicyController`

`ClaimController`

`PaymentController`

`SupportController`

- Services:

`AuthenticationService`

`PolicyService`

`ClaimService`

`PaymentService`

`SupportService`

- Repositories/DAOs: Interfaces to interact with the database for each entity
- Security: Role-based authentication and JWT-based session handling

5.0 Database Design

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') )
```

6.0 Appendices

Acronyms	Definitions
MVC	Model-View-Controller, a design pattern for organizing code.

Entity	A class representing a table in the database.
Repository	A data access pattern to abstract database operations.
JWT	JSON Web Token used for secure user authentication.
ORM	Object-Relational Mapping, for example, Entity Framework or Hibernate.
CRUD	Create, Read, Update, Delete operations on database records.

7.0 Terms & Conditions

- This system is to be used solely for internal academic or enterprise use.
- All sensitive data should be encrypted and comply with applicable privacy laws.
- Any misuse or tampering of data is prohibited.
- The project must not be deployed in production without proper security audits.
- The software is provided "as-is" with no express or implied warranties.

8.0 Change Log

Please note that this table needs to be maintained even if a Configuration Management tool is used.

Version Number	Changes made			
V<1.1>	<If the change details are not explicitly documented in the table below, reference should be provided here>			
	Page no	Changed by	Effective date	Changes effected
	6	Sakshi	14-05-25	Added Sequence Diagram
