# **Capstone Project 1: Insurance Premium Payment & Claims Portal**

## **Problem Statement**

Insurance brokers act as intermediaries between clients and multiple insurance providers, but premium collections and claims servicing are often inefficient. Clients may miss premium deadlines because reminders are fragmented or not timely, leading to policy lapses. Claims processing can be slow, with clients having little visibility into claim status. Brokers must manually reconcile premiums and claims across insurers, which is both error-prone and time-consuming.

This project aims to create an **Insurance Premium Payment & Claims Portal** to centralize premium management and claims processing for clients. The portal will allow policyholders to:

* View active policies and upcoming premium schedules.
* Pay premiums securely using multiple payment methods.
* Track their payment and claims history.
* Schedule automatic premium payments to avoid lapses.
* Manage personal and KYC details for compliance.

By digitizing these workflows, the portal reduces missed payments, streamlines claims servicing, and improves the broker’s ability to maintain accurate records across providers.

## **Feature 1: Policy Viewing and Premium Payments**

### **Backend**

* GET /api/policies/{userId} — Retrieve active policies and premium due dates.
* POST /api/premiums/pay — Record a premium payment and update policy status.

### **Database Schema**

**Table: Policies**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| policyId | BIGINT (PK) | Unique policy identifier |
| userId | BIGINT (FK) | Policyholder |
| insurer | VARCHAR(100) | Insurance provider name |
| policyType | VARCHAR(50) | Life, Health, Motor, etc. |
| premiumAmt | DECIMAL(10,2) | Premium amount |
| dueDate | DATE | Next premium due date |
| status | VARCHAR(20) | Active, Lapsed, Cancelled |

**Table: PremiumPayments**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| paymentId | BIGINT (PK) | Unique payment record |
| policyId | BIGINT (FK) | Related policy |
| userId | BIGINT (FK) | Policyholder |
| amount | DECIMAL(10,2) | Amount paid |
| paidAt | DATETIME | Payment timestamp |
| methodId | BIGINT (FK) | Payment method used |
| status | VARCHAR(20) | Success, Failed |

### **Frontend**

* PolicyList — Displays policies with premium status and due dates.
* PayPremium — Workflow to process payment for selected policy.

### **Deployment**

* **Offline**:
  + Backend runs via dotnet run.
  + Local SQL DB stores policies and payments.
  + Frontend runs via npm start and accessed in browser.
* **Cloud (Optional)**:
  + API hosted on Azure App Service.
  + Frontend hosted on Azure Static Web Apps.
  + Database on Azure SQL with secured connections.

## **Feature 2: Payment Method Management**

### **Backend**

* POST /api/payment-methods — Add a new payment method.
* GET /api/payment-methods/{userId} — Retrieve stored methods.

### **Database Schema**

**Table: PaymentMethods**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| methodId | BIGINT (PK) | Unique identifier |
| userId | BIGINT (FK) | Policyholder |
| type | VARCHAR(30) | CreditCard, DebitCard, Bank, UPI |
| maskedNo | VARCHAR(30) | Masked card/account number |
| expiry | DATE | Expiry date if applicable |
| isDefault | BIT | Marks default method |

### **Frontend**

* PaymentMethods — Add, remove, set default payment options.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores payment methods.
* Frontend runs via npm start and accessed in browser.

**Cloud (Optional):**

* API hosted on Azure App Service.
* Frontend hosted on Azure Static Web Apps.
* Database on Azure SQL with encryption for sensitive card/bank data.

## **Feature 3: Premium and Claims History**

### **Backend**

* GET /api/history/{userId} — Retrieve premiums and claims history.
* GET /api/history/{userId}?filter=claim — View only claims.

### **Database Schema**

**Table: Claims**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| claimId | BIGINT (PK) | Unique claim identifier |
| policyId | BIGINT (FK) | Related policy |
| userId | BIGINT (FK) | Policyholder |
| claimAmt | DECIMAL(10,2) | Claimed amount |
| status | VARCHAR(20) | Pending, Approved, Rejected |
| submittedAt | DATETIME | Claim submission timestamp |
| resolvedAt | DATETIME | Claim resolution date (nullable) |

### **Frontend**

* HistoryList — Filterable list for both premiums and claims.
* ClaimDetails — Details of a selected claim.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores premium and claims history.
* Frontend runs via npm start and accessed in browser.

**Cloud (Optional):**

* API hosted on Azure App Service.
* Frontend hosted on Azure Static Web Apps.
* Database on Azure SQL with secured connections.
* Optional integration with Azure Service Bus for claim status updates.

## **Feature 4: Premium Scheduling**

### **Backend**

* POST /api/schedules — Create scheduled premium payment.
* GET /api/schedules/{userId} — Retrieve schedules.

### **Database Schema**

**Table: PremiumSchedules**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| scheduleId | BIGINT (PK) | Unique schedule identifier |
| userId | BIGINT (FK) | Policyholder |
| policyId | BIGINT (FK) | Related policy |
| methodId | BIGINT (FK) | Payment method |
| frequency | VARCHAR(20) | Monthly, Quarterly, Yearly |
| nextRunAt | DATETIME | Next scheduled payment date |
| status | VARCHAR(20) | Active, Paused, Cancelled |

### **Frontend**

* ScheduleList — View and manage scheduled premiums.
* AddSchedule — Create or edit schedules.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores scheduled payments.
* Frontend runs via npm start and accessed in browser.

**Cloud (Optional):**

* API hosted on Azure App Service.
* Frontend hosted on Azure Static Web Apps.
* Database on Azure SQL.
* Optional Azure Functions/WebJobs used to trigger scheduled tasks.

## **Feature 5: User Profile & KYC Management**

### **Backend**

* GET /api/profile/{userId} — Fetch user profile.
* PUT /api/profile/{userId} — Update profile & KYC fields.

### **Database Schema**

**Table: Users**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| userId | BIGINT (PK) | Policyholder ID |
| fullName | VARCHAR(100) | Full name |
| email | VARCHAR(100) | Email |
| phone | VARCHAR(20) | Phone number |
| address | VARCHAR(200) | Address |
| panNo | VARCHAR(20) | PAN (tax ID) |
| aadhaarNo | VARCHAR(20) | Aadhaar/ID number |
| createdAt | DATETIME | Profile creation timestamp |
| updatedAt | DATETIME | Last profile update |

### **Frontend**

* UserProfile — Form to edit personal/KYC details.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores user profiles and KYC data.
* Frontend runs via npm start and accessed in browser.

**Cloud (Optional):**

* API hosted on Azure App Service.
* Frontend hosted on Azure Static Web Apps.
* Database on Azure SQL with encryption for sensitive KYC fields.