# **Capstone Project 3: Insurance Brokerage Client Servicing Portal**

**Domain: Insurance Brokerage**

## **Problem Statement**

Insurance brokers often struggle to provide clients with a unified view of their policies, claims, and requests because information is dispersed across different insurers. Clients must contact brokers manually for updates, leading to slow servicing and poor customer experience. Brokers also face challenges in managing endorsements, renewals, and client communication effectively.

This project builds an **Insurance Brokerage Client Servicing Portal** where clients can:

* View all policies consolidated across providers.
* Submit and track service requests (e.g., endorsements, renewals).
* Monitor claims filed.
* Receive communication and reminders from the broker.
* Update personal/KYC details.

### **Feature 1: Policy Consolidation**

**Backend**

* GET /api/policies/{userId} → Fetch consolidated client policies.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| policyId | BIGINT (PK) | Policy identifier |
| userId | BIGINT (FK) | Client |
| insurer | VARCHAR(100) | Provider name |
| policyType | VARCHAR(50) | Life, Health, Motor, etc. |
| premium | DECIMAL(10,2) | Premium amount |
| status | VARCHAR(20) | Active, Expired |
| expiry | DATE | Expiry date |

**Frontend**

* PolicyDashboard — Display consolidated policies with filters.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores consolidated policy 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 secured connections.

### **Feature 2: Service Requests**

**Backend**

* POST /api/requests → Submit service request (endorsement, renewal).
* GET /api/requests/{userId} → Track client requests.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| requestId | BIGINT (PK) | Request ID |
| userId | BIGINT (FK) | Client |
| type | VARCHAR(50) | Endorsement, Renewal, Update |
| status | VARCHAR(20) | Pending, In Progress, Completed |
| createdAt | DATETIME | Request creation date |

**Frontend**

* RequestForm — Submit requests.
* RequestTracker — Track status of requests.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores service requests.
* 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 3: Claims Monitoring**

**Backend**

* GET /api/claims/{userId} → List client claims.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| claimId | BIGINT (PK) | Claim ID |
| userId | BIGINT (FK) | Client |
| policyId | BIGINT (FK) | Related policy |
| claimAmt | DECIMAL(10,2) | Claimed amount |
| status | VARCHAR(20) | Pending, Approved, Rejected |
| filedAt | DATETIME | Claim filing date |

**Frontend**

* ClaimList — Client claim history.
* ClaimDetails — Details of a specific claim.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores claims and statuses.
* 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 claims data.
* Optional Azure Service Bus for notifications.

### **Feature 4: Broker Communication**

**Backend**

* POST /api/messages → Send broker-to-client message.
* GET /api/messages/{userId} → Retrieve communications.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| messageId | BIGINT (PK) | Message ID |
| userId | BIGINT (FK) | Recipient |
| content | VARCHAR(500) | Message body |
| sentAt | DATETIME | Timestamp |

**Frontend**

* MessageCenter — Inbox of broker communications.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores broker-client messages.
* 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.

### **Feature 5: Client Profile & KYC**

**Backend**

* GET /api/profile/{userId}
* PUT /api/profile/{userId}

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| userId | BIGINT (PK) | Client ID |
| fullName | VARCHAR(100) | Client name |
| email | VARCHAR(100) | Email |
| phone | VARCHAR(20) | Contact number |
| address | VARCHAR(200) | Address |
| kycId | VARCHAR(50) | PAN/Aadhaar/ID number |

**Frontend**

* ProfileForm — Edit client details.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores client profiles and KYC details.
* 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 encrypted KYC fields.