# **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.

# **Capstone Project 2: Risk Exposure Tracking & Reporting System**

**Domain: Risk Management**

## **Problem Statement**

Consulting firms and insurers need to monitor client risk exposures across categories such as credit, market, operational, and compliance risk. Traditionally, these are tracked via spreadsheets or manual reports, which are prone to errors and lack real-time insights. Without a centralized system, clients cannot assess overall exposure or track mitigation actions, leading to poor decision-making and regulatory gaps.

This project builds a **Risk Exposure Tracking & Reporting System** where clients can:

* Record different categories of risks.
* View aggregated exposures.
* Monitor mitigation actions.
* Generate risk trend reports.
* Manage organizational profiles.

## **Feature 1: Risk Recording**

### **Backend**

* POST /api/risks — Add a new risk record.
* GET /api/risks/{orgId} — Fetch risks by organization.

### **Database Schema**

**Table: Risks**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| riskId | BIGINT (PK) | Unique risk identifier |
| orgId | BIGINT (FK) | Organization owner |
| category | VARCHAR(50) | Credit, Market, Operational, etc. |
| description | VARCHAR(200) | Risk description |
| exposure | DECIMAL(12,2) | Exposure amount |
| status | VARCHAR(20) | Open, Mitigated, Closed |
| createdAt | DATETIME | Creation timestamp |

### **Frontend**

* RiskForm — Add/edit risks.
* RiskList — Display risks with filters by category/status.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores risk records.
* 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 2: Exposure Aggregation**

### **Backend**

* GET /api/exposures/{orgId} — Summarize exposure by category.

### **Database Schema**

**Table: ExposureSummary**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| summaryId | BIGINT (PK) | Summary record ID |
| orgId | BIGINT (FK) | Organization |
| category | VARCHAR(50) | Risk category |
| totalAmt | DECIMAL(12,2) | Total exposure amount |
| lastUpdate | DATETIME | Last updated timestamp |

### **Frontend**

* ExposureDashboard — Chart view by category.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB performs aggregation queries.
* 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 precomputed aggregates.

## **Feature 3: Mitigation Tracking**

### **Backend**

* POST /api/mitigations — Log mitigation action.
* GET /api/mitigations/{riskId} — Fetch actions per risk.

### **Database Schema**

**Table: Mitigations**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| mitigationId | BIGINT (PK) | Unique mitigation identifier |
| riskId | BIGINT (FK) | Related risk |
| action | VARCHAR(200) | Mitigation step description |
| owner | VARCHAR(100) | Person responsible |
| deadline | DATE | Target completion date |
| status | VARCHAR(20) | Open, Completed |

### **Frontend**

* MitigationList — Display and track actions.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores mitigation steps linked to risks.
* 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 4: Risk Trend Reports**

### **Backend**

* GET /api/reports/{orgId} — Generate risk trends by date range.

### **Database Schema**

**Table: RiskReports**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| reportId | BIGINT (PK) | Unique report ID |
| orgId | BIGINT (FK) | Organization |
| period | VARCHAR(20) | Monthly/Quarterly |
| metrics | VARCHAR(500) | JSON/summary metrics |
| createdAt | DATETIME | Report timestamp |

### **Frontend**

* RiskReport — Chart/table visualization of risk trends.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB generates reports.
* 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 integration with Power BI for dashboards.

## **Feature 5: Organization Profiles**

### **Backend**

* GET /api/orgs/{orgId} — Fetch organization details.
* PUT /api/orgs/{orgId} — Update profile.

### **Database Schema**

**Table: Organizations**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| orgId | BIGINT (PK) | Organization ID |
| name | VARCHAR(100) | Organization name |
| sector | VARCHAR(50) | Industry sector |
| region | VARCHAR(50) | Region |
| contact | VARCHAR(100) | Contact person |
| email | VARCHAR(100) | Contact email |

### **Frontend**

* OrgProfile — Manage organization details.

### **Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores organization profiles.
* 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.

# **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.

# **Capstone Project 4: Human Capital Benefits & Analytics Platform**

**Domain: Human Capital Consulting**

## **Problem Statement**

Organizations struggle to manage employee benefits programs, often relying on disparate systems for health, retirement, and wellness benefits. Employees lack transparency about available benefits and their utilization, while HR cannot easily analyze engagement or ROI. This creates dissatisfaction, underutilization of benefits, and higher costs.

This project develops a **Benefits & Analytics Platform** that allows employees to view, enroll in, and provide feedback on benefits, while HR and consultants access dashboards for utilization and cost analysis.

### **Feature 1: Benefit Catalog & Enrollment**

**Backend**

* GET /api/benefits/{empId} — Available benefits.
* POST /api/enrollments — Enroll employee.

**Database Schema**

**Table: Benefits**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| benefitId | BIGINT (PK) | Benefit ID |
| name | VARCHAR(100) | Benefit name |
| type | VARCHAR(50) | Insurance, Wellness |
| description | VARCHAR(200) | Benefit summary |

**Table: Enrollments**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| enrollId | BIGINT (PK) | Enrollment ID |
| empId | BIGINT (FK) | Employee |
| benefitId | BIGINT (FK) | Benefit |
| status | VARCHAR(20) | Active, Cancelled |

**Frontend**

* BenefitCatalog — List benefits.
* EnrollForm — Benefit enrollment.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores benefits and enrollments.
* 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 2: Utilization Tracking**

**Backend**

* GET /api/utilization/{empId} — Track usage of benefits.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| utilizationId | BIGINT (PK) | Usage record ID |
| enrollId | BIGINT (FK) | Related enrollment |
| usedAmount | DECIMAL(10,2) | Amount/units used |
| balance | DECIMAL(10,2) | Remaining allowance |
| lastUsedAt | DATETIME | Last usage date |

**Frontend**

* UtilizationTracker — Employee view of usage.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores utilization 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.
* Optional Power BI dashboards for utilization analytics.

### **Feature 3: Feedback Collection**

**Backend**

* POST /api/feedback — Submit feedback.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| feedbackId | BIGINT (PK) | Feedback ID |
| empId | BIGINT (FK) | Employee |
| benefitId | BIGINT (FK) | Related benefit |
| rating | INT | Rating 1–5 |
| comments | VARCHAR(200) | Employee comments |

**Frontend**

* FeedbackForm — Submit feedback.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores feedback records.
* 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 4: HR Dashboards**

**Backend**

* GET /api/dashboard — Aggregated stats.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| dashboardId | BIGINT (PK) | Dashboard record |
| metric | VARCHAR(50) | Metric name |
| value | DECIMAL(12,2) | Metric value |
| createdAt | DATETIME | Record timestamp |

**Frontend**

* HRDashboard — Charts of benefit utilization.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB aggregates employee 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.
* Optional integration with Azure Power BI for HR analytics.

### **Feature 5: Employee Profiles**

**Backend**

* GET /api/employees/{id} — Fetch employee profile.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| empId | BIGINT (PK) | Employee ID |
| name | VARCHAR(100) | Full name |
| email | VARCHAR(100) | Email |
| department | VARCHAR(50) | Department |
| role | VARCHAR(50) | Role |

**Frontend**

* EmployeeProfile — Profile management.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores employee profiles.
* 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.

# **Capstone Project 5: Client Onboarding & KYC Management System**

**Domain: Insurance Brokerage / Risk Consulting**

## **Problem Statement**

Brokerages must adhere to strict KYC norms to onboard clients. Current processes involve physical documents and email exchanges, delaying onboarding and increasing compliance risks. Without centralized tracking, brokers lack visibility on verification progress.

This project builds a **Client Onboarding & KYC Management System** enabling digital client registration, secure KYC document uploads, verification tracking, automated notifications, and centralized compliance records.

### **Feature 1: Client Registration**

**Backend**

* POST /api/clients/register — Register new client.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| clientId | BIGINT (PK) | Client ID |
| name | VARCHAR(100) | Full name |
| email | VARCHAR(100) | Contact email |
| phone | VARCHAR(20) | Contact phone |
| status | VARCHAR(20) | Registered, Active |

**Frontend**

* RegistrationForm — Client registration UI.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores client registration 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: KYC Document Upload**

**Backend**

* POST /api/kyc/upload — Upload client docs.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| docId | BIGINT (PK) | Document ID |
| clientId | BIGINT (FK) | Related client |
| type | VARCHAR(50) | PAN, Aadhaar, etc. |
| filePath | VARCHAR(200) | Document location |
| status | VARCHAR(20) | Uploaded, Verified |

**Frontend**

* KYCUpload — Document submission UI.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local DB/filesystem stores uploaded KYC docs.
* Frontend runs via npm start and accessed in browser.

**Cloud (Optional):**

* API hosted on Azure App Service.
* Frontend hosted on Azure Static Web Apps.
* Metadata stored in Azure SQL, documents optionally in Azure Blob Storage.

### **Feature 3: Verification Tracking**

**Backend**

* GET /api/verification/{clientId} — Status tracking.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| verificationId | BIGINT (PK) | Verification ID |
| clientId | BIGINT (FK) | Related client |
| verifier | VARCHAR(100) | Assigned verifier |
| status | VARCHAR(20) | Pending, Verified, Rejected |
| reviewedAt | DATETIME | Review date |

**Frontend**

* VerificationStatus — Status dashboard.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores verification logs.
* 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 verification records.

### **Feature 4: Notifications**

**Backend**

* POST /api/notifications — Send notification.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| notificationId | BIGINT (PK) | Notification ID |
| clientId | BIGINT (FK) | Recipient client |
| message | VARCHAR(200) | Notification message |
| sentAt | DATETIME | Sent timestamp |

**Frontend**

* NotificationCenter — View notifications.

**Deployment**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB stores notifications.
* 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 Notification Hubs for alerts.

### **Feature 5: Broker Dashboard**

**Backend**

* GET /api/dashboard — Broker overview of onboarding.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| dashboardId | BIGINT (PK) | Dashboard record |
| clients | INT | Total clients onboarded |
| pending | INT | Clients pending KYC |
| verified | INT | Verified clients |

**Frontend**

* BrokerDashboard — Summarizes onboarding status.

**Deployment**

### **Feature 5: Broker Dashboard**

**Offline:**

* Backend runs via dotnet run.
* Local SQL DB powers dashboard metrics.
* 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.