**Vendor Management Application: Purpose and Scope**

**Purpose**

The **Vendor Management Application** is designed to streamline and automate the tracking of vendor-related data for a company, ensuring efficient management of employee-vendor associations, purchase orders (POs), client contracts, payment schedules, and exit processes.

The primary goals are:

1. **Centralize Vendor & Employee Data** – Maintain records of employees, their associated vendors, and contract details.
2. **Track Client POs & Billing** – Monitor client contracts, PO rates, and validity periods.
3. **Manage Payment Schedules** – Record payable dates and payment status updates.
4. **Facilitate Exit Processes** – Update employee/vendor exit details and archive records.
5. **Ensure Data Accuracy & Security** – Provide role-based access to prevent unauthorized modifications.

**Scope**

The application will cover the following stages of vendor management:

**Stage 1: Employee & Vendor Onboarding**

* **Fields:**
  + Employee Name
  + Date of Joining (DOJ)
  + Vendor Name
  + PO Rate (Agreed rate with the vendor)
  + Skill (Employee’s role/competency)
* **Functionality:**
  + Add, edit, and archive employee-vendor associations.
  + Validate PO rates against vendor contracts.

**Stage 2: Client PO Management**

* **Fields:**
  + Client PO Number
  + Client PO Start & End Date
  + Client Name
  + Client PO Rate (Billing rate to the client)
* **Functionality:**
  + Track active and expired POs with alerts for renewals.
  + Compare vendor PO rates vs. client PO rates for margin analysis.

**Stage 3: Payment Tracking**

* **Fields:**
  + Payment Status (Pending/Processed)
  + Payable Dates
  + Invoice References (if applicable)
* **Functionality:**
  + Generate payment reminders for accounts teams.
  + Log payment confirmations.

**Stage 4: Exit Management & Data Archival**

* **Fields:**
  + Exit Date
  + Reason for Exit (Contract end, termination, etc.)
  + Final Payment Status
* **Functionality:**
  + Archive records while retaining audit trails.
  + Generate reports on vendor/employee turnover.

**Out of Scope**

* Payroll processing (only payment tracking, not execution).
* Vendor performance reviews (can be added later).
* Tax compliance (handled by external finance systems).

**Target Users**

* **HR/Admin Teams** – Manage employee-vendor mappings.
* **Finance Teams** – Track POs and payments.
* **Operations Managers** – Monitor contract timelines.
* **Auditors** – Access historical records for compliance.

**Key Benefits**

✅ **Reduced Manual Errors** – Automated data entry & validation.  
✅ **Improved Compliance** – Audit-ready records of all transactions.  
✅ **Better Financial Control** – Visibility into vendor vs. client billing rates.  
✅ **Scalability** – Supports growing vendor/employee counts.

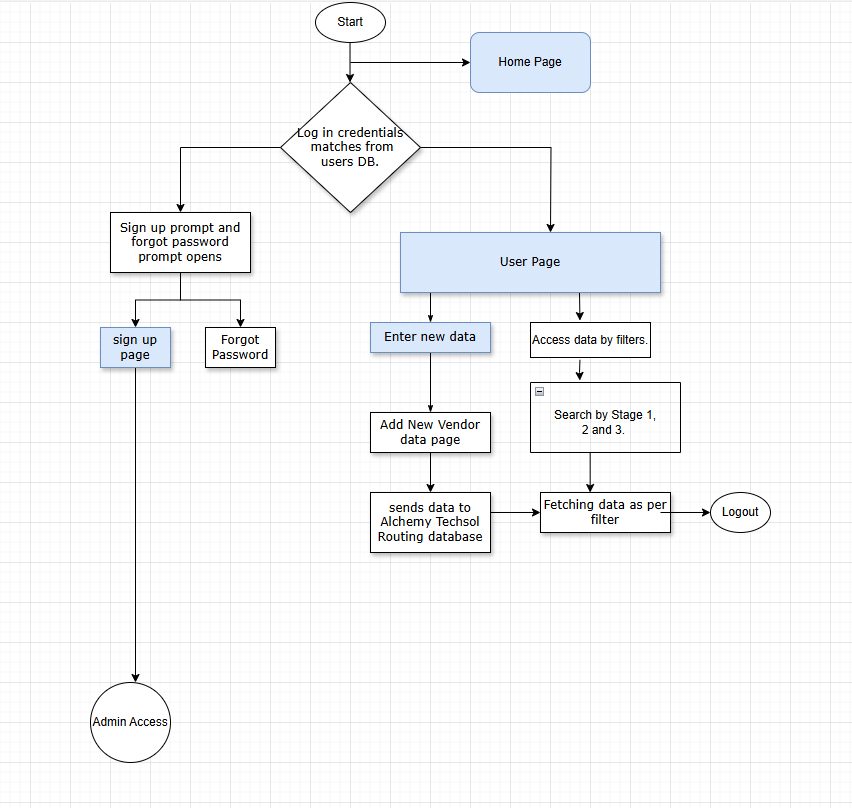
**System** **Architecture**

#### ****Components Breakdown:****

1. **Frontend (Web/Mobile App)**
   * **Tech Stack:** React.js
   * **Features:**
     + Dashboard for HR, Finance, and Operations.
     + Forms for data entry (Employee, Vendor, PO, Payments).
     + Reports & Alerts (e.g., PO expiry reminders).
2. **Backend (Server + APIs)**
   * **Tech Stack:** Node.js (Express) / Python (Django/Flask) / Java (Spring Boot).
   * **Key Modules:**
     + **Auth Service** (Role-based access: Admin, HR, Finance).
     + **Employee-Vendor Manager** (Stage 1).
     + **PO & Client Manager** (Stage 2).
     + **Payment Tracker** (Stage 3).
     + **Exit & Data Archival** (Stage 4).
3. **Database**
   * **Tech Stack:** PostgreSQL
   * **Tables:**
     + Employees (Name, DOJ, Vendor, Skill).
     + Vendors (Vendor Name, PO Rate, Contract Terms).
     + ClientPOs (PO No, Client, Start/End Date, Rate).
     + Payments (Status, Due Date, Invoice Ref).
     + ExitRecords (Exit Date, Reason, Final Payment).
4. **Step 3: Role-Based Access Control (RBAC)**

| **Role** | **Permissions** |
| --- | --- |
| **Admin** | Full access (CRUD all stages). |
| **HR** | Stage 1 (Employee-Vendor mapping). |
| **Finance** | Stage 3 (Payments) + Reports. |
| **Operations** | Stage 2 (Client POs). |

**Control Flow Diagram for Vendor Management App.**



**1. Database Schema (ER Diagram)**

Here’s the **normalized database structure** for your app:

### ****Tables Breakdown:****

| **Table** | **Key Fields** | **Description** |
| --- | --- | --- |
| EMPLOYEE | employee\_id (PK), name, doj, skill | Employee details. |
| VENDOR | vendor\_id (PK), vendor\_name, vendor\_po\_rate | Vendor contracts. |
| CLIENT\_PO | po\_id (PK), client\_name, po\_number, start/end\_date, client\_rate, vendor\_id (FK) | Client POs linked to vendors. |
| PAYMENT | payment\_id (PK), po\_id (FK), status, payable\_date | Payment schedules. |
| EXIT\_RECORD | exit\_id (PK), employee\_id (FK), exit\_date, reason | Archived employee/vendor exits. |

## **2. API Endpoint Specifications**

RESTful APIs with JWT authentication.

### ****2.1 Employee-Vendor Management (Stage 1)****

| **Endpoint** | **Method** | **Description** | **Request Body Example** |
| --- | --- | --- | --- |
| /api/employees | POST | Add employee-vendor mapping. | {name: "John", doj: "2024-01-01", vendor\_id: "V001", skill: "Dev"} |
| /api/employees/{id} | GET | Fetch employee details. | - |
| /api/vendors | POST | Add a new vendor. | {vendor\_name: "ABC Corp", vendor\_po\_rate: 50} |

### ****2.2 Client PO Management (Stage 2)****

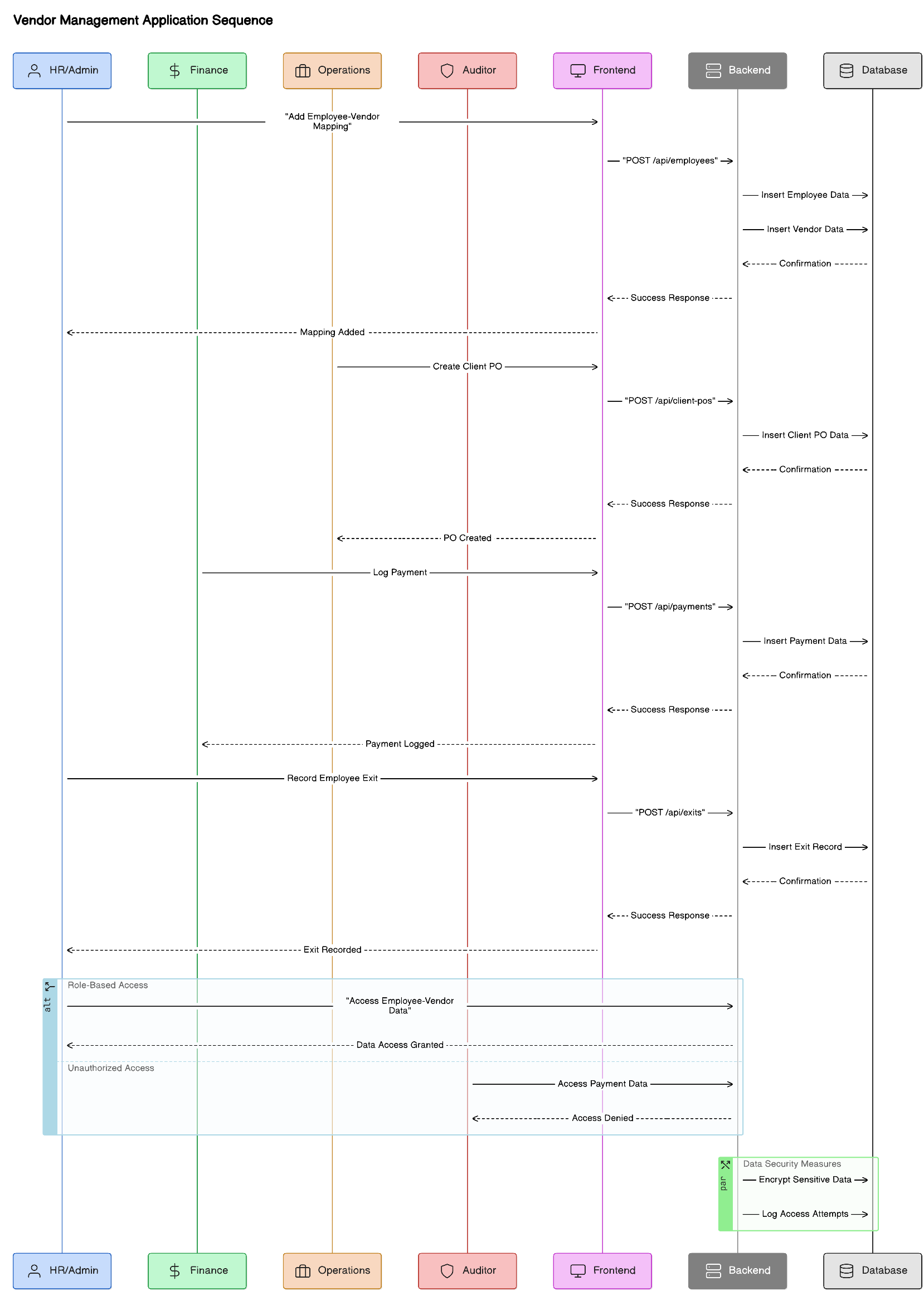
| /api/client-pos | POST | Create a client PO. | {client\_name: "XYZ Ltd", po\_number: "PO1001", vendor\_id: "V001", client\_rate: 70} |  
| /api/client-pos/{id} | PUT | Update PO end date/rate. | {end\_date: "2024-12-31"} |

### ****2.3 Payment Tracking (Stage 3)****

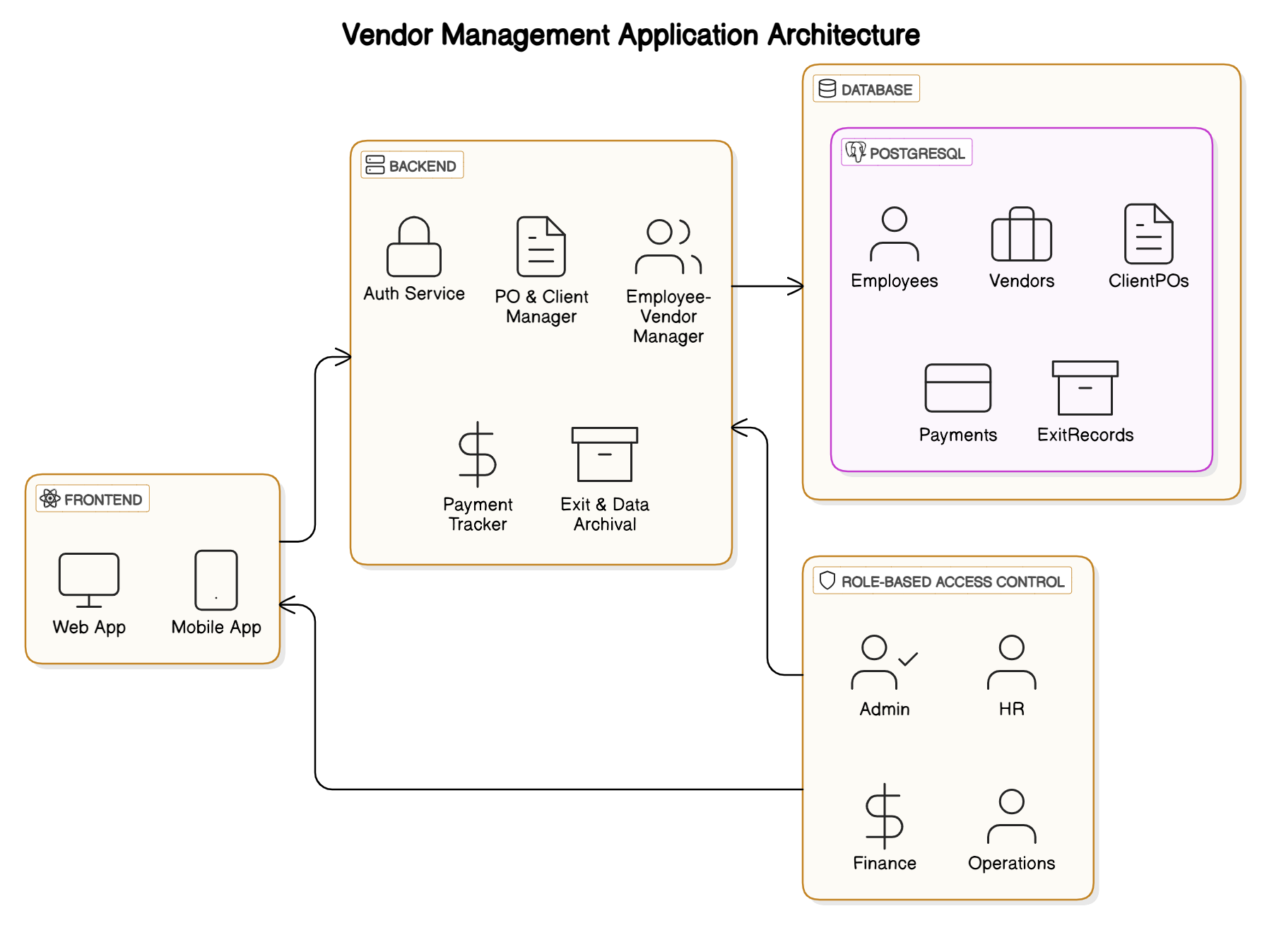
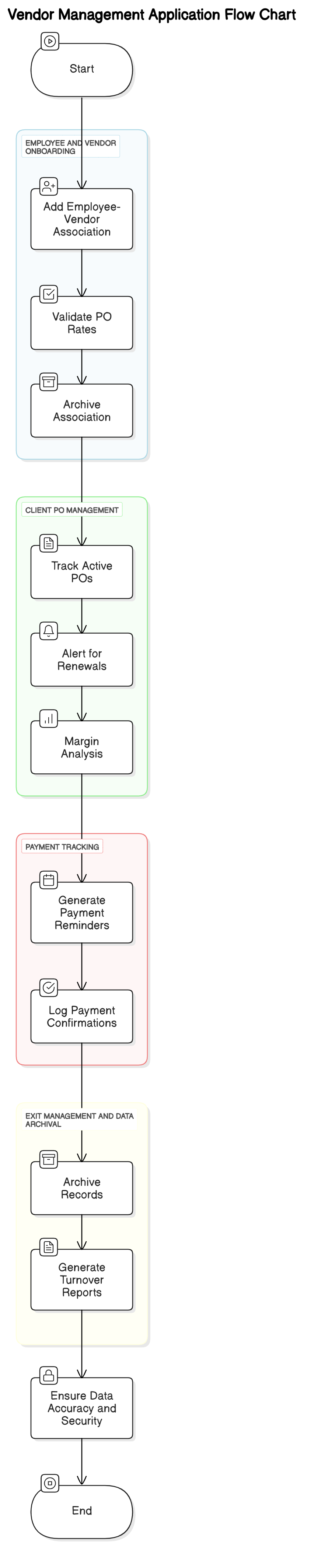
| /api/payments | POST | Log a payment. | {po\_id: "PO1001", status: "Pending", payable\_date: "2024-06-15"} |  
| /api/payments/overdue | GET | List overdue payments. | - |

### ****2.4 Exit Management (Stage 4)****

| /api/exits | POST | Record an exit. | {employee\_id: "E001", exit\_date: "2024-05-20", reason: "Contract End"} |



**Data Flow Diagram**



### ****1. Development Setup****

#### ****1.1 Tech Stack Finalization****

* **Frontend**: React.js (TypeScript)
* **Backend**: Node.js (Express) / Python (Django) / Java (Spring Boot).
* **Database**: PostgreSQL (structured)
* **DevOps**: Docker, AWS/Azure, CI/CD (GitHub Actions/Jenkins).

#### ****1.2 Initialize Repositories****

* Set up **Git** (e.g., GitHub/GitLab) with branches:
  + main (production)
  + dev (staging)
  + Feature branches (e.g., feature/auth, feature/payments).

#### ****1.3 Develop Core Modules****

* **Priority Order**:
  1. **Authentication** (JWT/OAuth).
  2. **Employee-Vendor Mgmt** (Stage 1).
  3. **Client PO Mgmt** (Stage 2).
  4. **Payment Tracking** (Stage 3).
  5. **Exit Mgmt** (Stage 4).

**Summary Checklist**

| **Step** | **Tasks** | **Output** |
| --- | --- | --- |
| **Development** | Code modules, APIs, UI. | Functional application. |
| **Testing** | Unit, integration, UAT. | Bug-free build. |
| **Deployment** | Docker. | Live application. |