Comprehensive Project Documentation: EMS (Employee Management System)

> **Execution Plan**: This document serves as the master record for the project logic. We will systematically detail every module, class, and function below.

Table of Contents

1. Project Overview

The **Enterprise Employee Management System (EMS)** is a full-stack web application designed to streamline HR operations. It features a secure, role-based architecture capable of handling employee data, attendance tracking, leave

management, payroll processing, and recruitment (ATS).

1.1 Architecture

- **Backend**: Spring Boot (Java 21) REST API.
- **Frontend**: React (Vite) Single Page Application (SPA).
- **Database**: MySQL (Relational Data Persistence).
- **Security**: Stateless JWT (JSON Web Token) Authentication with Role-Based Access Control (RBAC).

1.2 Tech Stack & Libraries

- **Backend**:
  - `Spring Boot 3.2.2` : Core framework.
  - `Spring Security` : Authentication & Authorization.
  - `Spring Data JPA` : Hibernate/ORM for MySQL interactions.
  - `jjwt (0.12.3)` : JWT generation and validation.
  - `OpenPDF (1.3.30)` : PDF Generation for Payslips.
  - `Apache PDFBox (3.0.0)` : Resume Parsing for ATS.
  - `Lombok` : Boilerplate code reduction.
- **Frontend**:
  - `React 19` : UI Library.
  - `Vite` : Build tool.
  - `React Bootstrap 2.10` : UI Components & Grid System.
  - `Axios` : HTTP Client.
  - `Chart.js 4` : Data Visualization.
  - `React Router Dom 7` : Client-side routing.

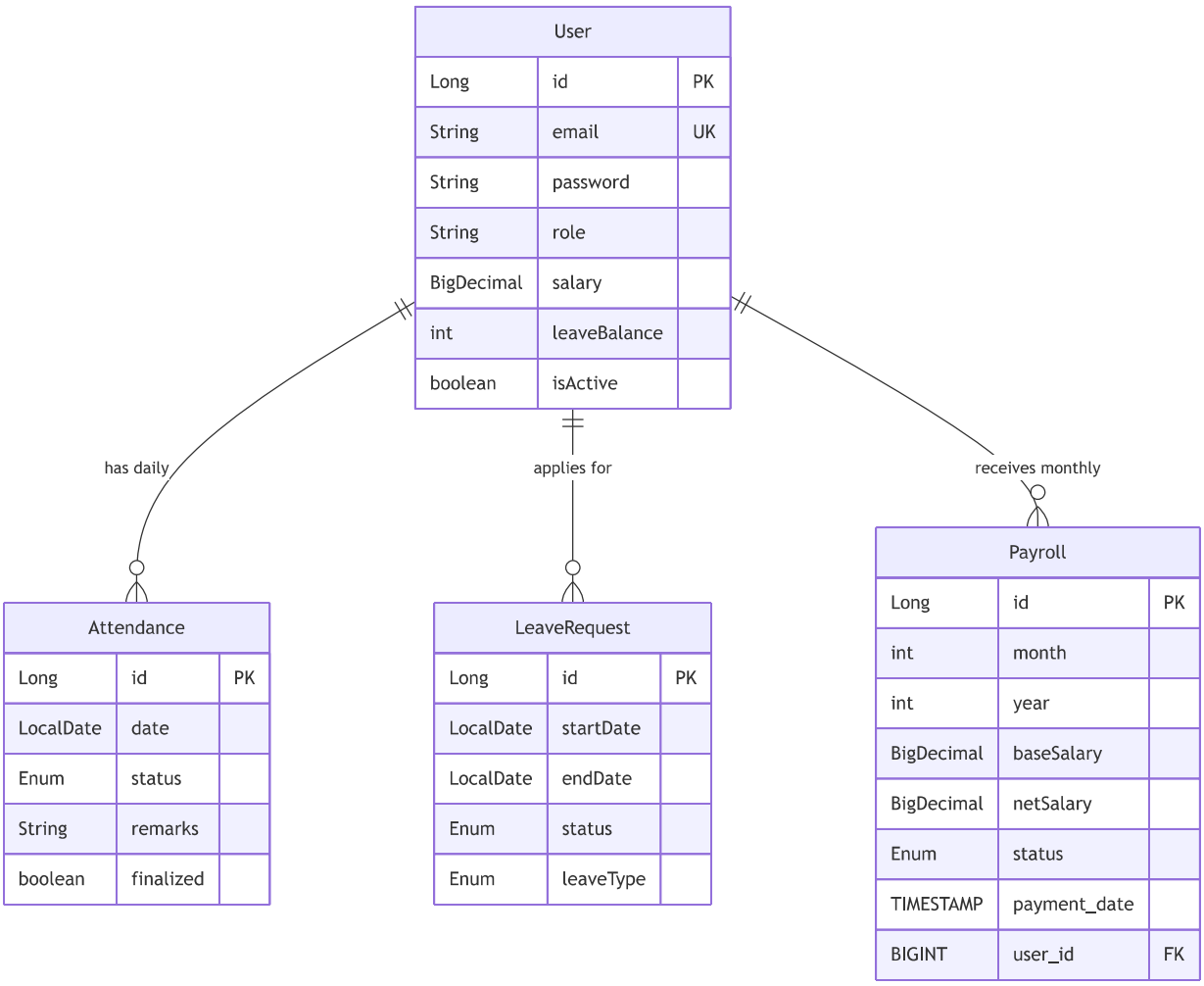
1.3 Key Features

- **Role-Based Access**: Specialized views for ADMIN, HR, and EMPLOYEE.
- **Live Payroll Simulation**: Realistic mock bank transfer steps with verification.
- **PDF Payslips**: Auto-generated downloadable salary slips.
- **Smart Attendance**: Visual tracking of Present/Absent/Half-Day status.
- **Resume Parsing**: ATS scanner to extract skills and email from PDF resumes.

1.4 project Structure

2. Backend Logic Analysis

2.0 Database Schema (ER Diagram)



2.1 Authentication & Security

This module handles user identity verification and access control. It follows a stateless architecture using JWT.

2.1.1 Security Configuration (`SecurityConfig.java`)

- **Status**: Configuration Bean.
- **Logic**:
  - **CSRF**: Disabled (standard for stateless REST APIs).
  - **CORS**: Configured to allow requests from `http://localhost:5173` (Frontend) with all methods (GET, POST, PUT, DELETE).
  - **Session Management**: Set to `STATELESS` (Server does not store session IDs).
  - **Public Endpoints**: `/api/auth/**` (Login), `/swagger-ui/**` , `/v3/api-docs/**` .
  - **Role-Based Restrictions**:
    - `/api/admin/**` -> Requires `ROLE_ADMIN` .
    - `/api/hr/**` -> Requires `ROLE_HR` .
    - `/api/employee/**` -> Requires `ROLE_EMPLOYEE` .
  - **Filter Chain**: Adds `JwtAuthenticationFilter` before the standard `UsernamePasswordAuthenticationFilter` .
- **Key Beans**:

- `PasswordEncoder` : Uses `BCryptPasswordEncoder` for hashing.
- `AuthenticationManager` : Manages the auth process.

2.1.2 JWT Utility (`JwtUtils.java`)

- **Purpose**: Token Generation and Validation.
- **Key Functions**:
  - `generateToken(UserDetails)` : Creates a token with `HS256` signature. Embeds the user's **Role** (stripped of `ROLE_` prefix) into the claims.
  - `extractUsername(token)` : detailed parsing of the `sub` claim.
  - `isTokenValid(token, userDetails)` : Checks signature integrity and expiration time.

2.1.3 User Details Service (`CustomUserDetailsService.java`)

- **Purpose**: Bridges Spring Security with the Database.
- **Logic**:
  - Loads user by `email` from `UserRepository` .
  - **Constraint**: Only loads users where `isActive = true` .
  - **Mapping**: Converts our internal `Role` enum (ADMIN) to Spring's authority format (`ROLE_ADMIN`).

2.1.4 Auth Controller (`AuthController.java`)

- **Endpoints**:
  - `POST /api/auth/login` : Accepts `LoginRequest` (email, password). Authenticates via `AuthenticationManager` . Returns JWT.
  - `POST /api/auth/verify-password` : **(New)** Checks the password of the *currently logged-in user*. Used for confirming sensitive actions (e.g., Payroll disbursement). Uses `SecurityContextHolder` to get the current user, then attempts re-authentication.

2.2 Employee Management

This module handles the lifecycle of employee records, including profile management and role assignments.

2.2.1 Data Model (`User.java`)

- **Key Fields**:
  - `email` : Unique identifier for login.
  - `role` : Enum (`ADMIN` , `HR` , `EMPLOYEE`).
  - `isActive` : Boolean flag for soft-deletion.
  - `leaveBalance` : Tracks remaining leave days.
  - `salary` : Base salary for payroll calculations.
  - `profileImage` : URL path to locally stored image.
  - `createdAt/updatedAt` : Auto-managed audit timestamps.

2.2.2 Service Logic (`EmployeeService.java`)

- **Search**: `searchEmployees(query)` checks if the query matches the name or department. Filters out inactive users by default.
- **Creation**: `createUser(user)` verifies that the email is unique before saving. Encodes the password using BCrypt.

- **Updates**: `updateUser(id, details)` selectively updates mutable fields. Password is updated only if a non-blank value is provided.
- **Image Upload**: `uploadProfileImage` saves files to a local `uploads` directory and updates the user's `profileImage` field with the accessible URL.
- **Soft Delete**: `softDeleteEmployee` sets `isActive = false`, preserving historical data (e.g., for past payrolls) while preventing future login/processing.

2.2.3 Controller Endpoints (`EmployeeController.java`)

- `GET /api/employees`: Returns list of active employees. Supports `?search=` parameter.
- `POST /api/employees`: Creates a new user. Requires validation logic.
- `PUT /api/employees/{id}`: Updates existing user details.
- `PUT /api/employees/{id}/soft-delete`: Archive a user.
- `POST /api/employees/{id}/image`: Multipart file upload for profile pictures.

2.3 Attendance Module

This module tracks daily presence and forms the basis for payroll calculation.

2.3.1 Data Model (`Attendance.java`)

- **Composite Key**: Unique constraint on `employee_id + date`.
- **Fields**:
  - `status`: Enum (`PRESENT`, `ABSENT`, `HALF_DAY`, `LEAVE`).
  - `finalized`: Boolean. If true, the record is locked and cannot be edited.
  - `leaveRequest`: Link to the approved leave (if status is LEAVE).

2.3.2 Service Logic (`AttendanceService.java`)

- **Monthly Initialization**: `initializeMonth(year, month)` iterates through all days of the given month. For every **Employee**, it checks if a record exists. If not, it creates a default record with status `ABSENT`. This ensures no missing data points for payroll.
- **Modifications**: `updateAttendance` allows HR to manually change status (e.g., correct an Absent to Present). **Barrier**: Throws exception if record is `finalized`.
- **Leave Integration**: `handleLeaveApproval(leave)` is a listener method. When a leave is approved:
  i. Calculates the date range.
  ii. Finds or creates attendance records for those dates.
  iii. Sets status to `LEAVE` and links the `LeaveRequest`.
  iv. Updates remarks to "Leave Approved: [Type]".
- **Finalization**: `finalizeMonth` locks all records for a specific month, signaling they are ready for Payroll processing.

2.3.3 Controller Endpoints (`AttendanceController.java`)

- `GET /api/attendance`: Fetch records for a month/year.
- `POST /api/attendance/init`: Trigger the initialization batch job.
- `PUT /api/attendance/{id}`: Modify a specific day's status.
- `POST /api/attendance/finalize`: Lock the month.

2.4 Leave Management

This module manages employee time-off requests, enforcing policy rules and balance checks.

2.4.1 Data Model (`LeaveRequest.java`)

- **Fields**:
  - `startDate` , `endDate` : Duration of the leave.
  - `leaveType` : Enum (`SICK` , `CASUAL` , `EARNED` , `UNPAID_LEAVE`).
  - `status` : Enum (`PENDING` , `APPROVED` , `REJECTED`).
  - `reason` : Text justification.

2.4.2 Service Logic (`LeaveService.java`)

- **Application (`applyLeave`)**:
  - Validations: Start date must be before end date; cannot be in the past.
  - Balance Check: Ensures `user.leaveBalance >= requestedDays` . Does **not** deduct balance immediately (deduction happens on approval).
  - Result: Saves request with `PENDING` status.
- **Approval Workflow (`updateLeaveStatus`)**:
  - **Logic**:
    a. Retrieves request. Throws error if not PENDING.
    b. **If APPROVED**:
      - Re-validates user balance (in case of concurrent requests).
      - **Deducts Balance**: `user.leaveBalance -= days` .
      - **Updates Attendance**: Calls `attendanceService.handleLeaveApproval` to retrospectively mark those days as on leave in the attendance system.
    c. Updates status to `APPROVED` or `REJECTED` .

2.4.3 Controller Endpoints (`LeaveController.java`)

- `POST /api/leaves` : Apply for leave (User).
- `GET /api/leaves/my` : View personal leave history.
- `GET /api/leaves` : View all pending requests (HR).
- `PUT /api/leaves/{id}/status` : Approve or Reject a request.

2.5 Payroll & Finance

This module handles the complex logic of salary calculation, simulating banking transactions, and generating documentation.

2.5.1 Data Model (`Payroll.java`)

- **Constraint**: Unique record per Employee per Month/Year.
- **Fields**:
  - `baseSalary` : Snapshot of the user's salary at generation time.
  - `payableDays` : Calculated effective working days.
  - `netSalary` : Final amount to be paid.
  - `deductionAmount` : `baseSalary - netSalary` .

- `status` : `GENERATED` (Pending Payment) or `PAID` .

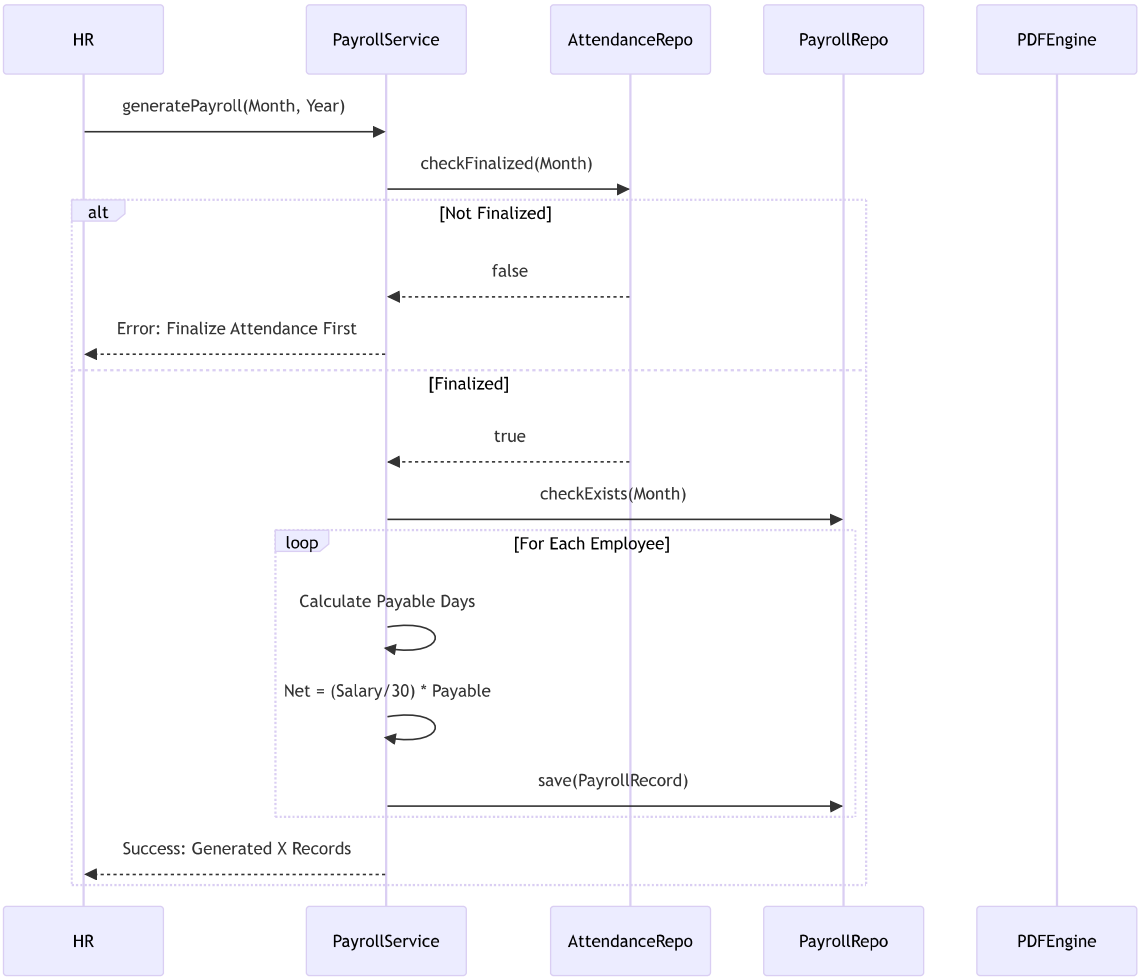
2.5.2 Service Logic (`PayrollService.java`)

- **Generation (`generatePayroll`)**:
  - **Prerequisite**: Attendance for the month must be **Finalized**.
  - **Calculation Algorithm**:
    a. Iterate all active employees.
    b. Fetch attendance records for the month.
    c. **Payable Days Formula**:
      - `PRESENT` = 1.0 day.
      - `HALF_DAY` = 0.5 day.
      - `LEAVE` = 1.0 day (Paid) or 0.0 (Unpaid).
      - `ABSENT` = 0.0 day.
    d. **Salary Formula**:
      - `PerDaySalary = BaseSalary / TotalDaysInMonth`
      - `NetSalary = PerDaySalary * PayableDays`
  - **Result**: Creates a `Payroll` entity with status `GENERATED` .
- **Disbursement (`processPayroll`)**:
  - **Simulation**: Uses `Thread.sleep(2000)` to mimic bank API latency.
  - **Update**: Sets status to `PAID` and records `paymentDate` .
- **PDF Generation (`generatePayslipPdf`)**:
  - **Library**: Uses **OpenPDF** (iText fork).
  - **Content**: Generates a professional header, employee info table, and a detailed Earnings/Deductions breakdown table.

2.5.3 Controller Endpoints (`PayrollController.java`)

- `GET /api/payroll` : View payroll list.
- `POST /api/payroll/generate` : Trigger calculation engine.
- `POST /api/payroll/process` : Trigger batch payment simulation (HR Only).
- `POST /api/payroll/{id}/mark-paid` : Trigger single payment (HR Only).
- `GET /api/payroll/payslip/{id}` : Download generated PDF.

2.5.4 Payroll Calculation Workflow



2.6 Dashboard Analytics

This module aggregates system-wide data to provide real-time insights for HR administrators.

2.6.1 Service Logic (`DashboardService.java`)

- **Data Aggregation (`getStats`)**:
  - **Total Staff**: Queries `UserRepository` for count of users where `isActive = true` .
  - **On Leave Today**: Queries `LeaveRepository` for approved leaves where `today` falls between `startDate` and `endDate` .
  - **Pending Requests**: Counts `LeaveRequests` with status `PENDING` .
  - **Est. Payroll Cost**: Sums the `salary` column of all active users to forecast monthly expense.
  - **Department Distribution**: Groups users by `department` and counts them (used for Pie Charts).

2.6.2 Controller Endpoints (`DashboardController.java`)

- `GET /api/dashboard/stats` : Returns a DTO containing all the above metrics. Secured for `HR` and `ADMIN` roles.

2.7 ATS & Resume Parsing

This feature introduces AI-powered recruitment capabilities, analyzing PDFs to rank candidates against job descriptions.

2.7.1 Service Logic (`ResumeService.java`)

- **Text Extraction**: Uses **Apache PDFBox** (`PDFTextStripper`) to read raw text from uploaded PDF files.
- **AI Analysis**:
  - **Model**: Integrates with **Google Gemini 2.5 Flash** (via REST API).
  - **Prompt Engineering**: Instructs the AI to act as an ATS, comparing the resume text against the target role (e.g., "Software Engineer").
  - **Output JSON**: Requests structured data including `score` (0-100), `missingKeywords` , `summary` , and `recommendation` .
- **Fallback Mechanism**: If the API Key is missing or the external service fails, a local **Mock Analysis** runs. It uses heuristics (checking for keywords like "Experience" or the Role name) to generate a basic score and feedback, ensuring the demo never crashes.

2.7.2 Controller Endpoints (`ResumeController.java`)

- `POST /api/ats/analyze` : Accepts `multipart/form-data` (PDF) and a query param `role` . Returns the analysis result.

2.8 Configuration & Utils

These utility classes handle cross-cutting concerns like data initialization and error handling.

2.8.1 Data Seeder (`DataSeeder.java`)

- **Purpose**: Populates the database with realistic dummy data on startup if empty.
- **Entities Created**:
  - **Admin**: `admin@ems.com / password` (Full Access).
  - **HR**: `hr@ems.com / password` (Payroll/Leave Management).
  - **Employees**: ~15 users with real names (e.g., "Alice Johnson", "Bob Smith"), varied departments, and randomized salaries (60k-150k).
  - **Leave Requests**: Mix of pending future leaves and approved past leaves to populate the dashboard metrics immediately.

2.8.2 Global Exception Handler (`GlobalExceptionHandler.java`)

- **Annotation**: `@ControllerAdvice` – intercepts exceptions from *all* controllers.
- **Handlers**:
  - `IllegalArgumentException` : Returns **400 Bad Request** with the exception message.
  - `Exception` (Generic): Returns **500 Internal Server Error**.
- **Response Format**: standardized JSON `ErrorResponse` (status, message).

3. Frontend Logic Analysis

The frontend is a React application built with functional components and hooks. It uses `react-bootstrap` for styling and layout.

3.1 Core Architecture

3.1.1 Routing & App Structure (`App.jsx`)

- **Router**: Uses `react-router-dom` v6+.

- **Routes**:
  - `/login` : Public access.
  - `/dashboard` , `/employees` , `/payroll` , etc.: Wrapped in `<ProtectedRoute>` .
- **Protection Logic**: Checks if `user` exists in `AuthContext` . If not, redirects to `/login` . If valid, renders the `Layout` .

3.1.2 Authentication Context (`AuthContext.jsx`)

- **State**: `user` object (email, role), `loading` boolean.
- **Initialization**: On load, checks `localStorage` for a JWT token. Decodes it (using `jwt-decode`) to restore the user session.
- **Login**: Sends generic credentials to `/auth/login` , saves the returned token, and updates state.
- **Logout**: Clears token and resets state.

3.1.3 API Client (`axios.js`)

- **Base URL**: `http://localhost:8081/api` .
- **Interceptor**: automatically attaches `Authorization: Bearer <token>` to **every** outgoing request, simplifying component logic.
- **Response Handling**: Global 401 handler (optional but recommended) to auto-logout on token expiration.

3.1.4 Main Layout (`Layout.jsx`)

- **Structure**: Flexbox container.
  - **Sidebar**: Fixed width on left (or collapsible on mobile).
  - **Main Content**: Scrollable area on the right (`flex-grow-1`).
- **Outlet**: Renders the matched child route component (e.g., Dashboard) inside the main content area.

3.2 Authentication (Login)

This module handles the entry point for all users.

3.2.1 Logic (`Login.jsx`)

- **UI**: Split-screen design (Creative Hero Image on Left, Form on Right).
- **State**: Local state for `email` , `password` , `error` .
- **Action (`handleSubmit`)**:
  i. Calls `authContext.login(email, password)` .
  ii. **Redirect Strategy**:
    - If Role contains `EMPLOYEE` -> Navigate to `/profile` .
    - If Role contains `ADMIN` or `HR` -> Navigate to `/dashboard` .
- **Feedback**: Displays error alert on 401 Unauthorized.

3.3 Dashboard

The central hub for HR/Admin analytics.

3.3.1 Logic (`Dashboard.jsx`)

- **Data Loading**: Uses `Promise.all` to fetch `/dashboard/stats` and `/leaves` in parallel.

- **Visualizations**:
  - **KPI Cards**: Total Staff, On Leave Today, Pending Requests, Payroll Cost.
  - **Charts**: Uses `react-chartjs-2` Doughnut chart to show "Department Distribution".
  - **Recent Activity**: A table showing the 5 most recent leave requests (sliced from the full list).
- **Access**: Restricted to `ADMIN` and `HR`.

3.4 Employee Directory

A comprehensive CRUD interface for managing the workforce.

3.4.1 Logic (`EmployeeList.jsx`)

- **Features**:
  - **List View**: Table with Avatar, Name, Email, Dept, Role, Salary.
  - **Search**: Server-side filtering via `?search=` query param.
  - **CSV Export**: Client-side generation of a CSV file from the current data set.
  - **Soft Delete**: "Deactivate" button calls the `soft-delete` endpoint.
- **Modals**:
  - **Add/Edit Modal**: Shared form for creating or updating users. Handles text fields and `file` input for profile pictures.
  - **Image Upload**: uses `FormData` API to send the image as `multipart/form-data` in a separate request after the user creation.

3.5 Attendance Interface

A complex grid interface for tracking daily status.

3.5.1 Logic (`Attendance.jsx`)

- **Filters**: Month, Year, and Day (default: All Days).
- **Initialization**: "Initialize Month" button triggers backend batch creation.
- **Visual Matrix**:
  - **Dropdowns**: Each day/employee cell is editable (unless finalized).
  - **Color Coding**: Present (Green), Absent (Red), Leave (Gray - Disabled).
- **Finalization**: "Finalize Month" locks the entire grid, preventing further edits and enabling Payroll generation.

3.6 Leave Portal

Split into two views based on role.

3.6.1 Apply Leave (`LeaveApply.jsx` - Employee)

- **Form**: Start Date, End Date, Type, Reason.
- **History**: Table showing personal request status (Approved/Rejected/Pending).
- **Validation**: Frontend check to ensure Start < End.

3.6.2 Manage Leaves (`LeaveManage.jsx` - HR)

- **Inbox**: Fetches all `PENDING` requests.
- **Actions**: "Approve" (Triggers balance deduction & attendance update) or "Reject".

- **Visuals**: Shows duration in days (calculated on the fly: `end - start + 1`).

3.7 Payroll Management

The financial command center.

3.7.1 Logic (`Payroll.jsx`)

- **Workflow**:
  i. **Generate**: Triggers calculation engine for the selected month.
  ii. **Review**: Table shows Base Salary, Deductions, and Net Pay.
  iii. **Disburse**: Launch the **Payment Gateway Simulation**.
- **Simulation Modal**:
  - **Auth**: Asks for `Transaction PIN` (Login Password).
  - **Process**: Fake `CONNECTING -> PROCESSING -> SUCCESS` states with a terminal-style log.
- **Payslips**:
  - **View**: Modal with clean HTML layout.
  - **Download**: Hits the PDF endpoint.
  - **Print**: CSS `@media print` rules to hide UI chrome and format only the payslip content.

3.8 Profile & Settings

The personal dashboard for employees.

3.8.1 Logic (`Profile.jsx`)

- **Dual-Column Layout**: Left side for Identity/Stats, Right side for Details/Tabs.
- **Stats**: Fetches user's leave history to calculate `Remaining Balance = Quota (20) - Used`.
- **Tabs**:
  - **My Details**: Read-only view of static info.
  - **My Salary Slip**: Reuses the `<Payroll />` component but passes an `embedded` prop to simplify the view (hiding admin controls).
  - **Security**: Placeholder for password change.
- **Edit Mode**: Modal to update mutable fields (Phone, Address, Bio).

3.9 ATS Scanner

A recruiting tool for HR.

3.9.1 Logic (`AtsScan.jsx`)

- **Input**:
  - **Role Selector**: Dropdown to choose target job (Software Engineer, etc.).
  - **File Input**: Restricts to `.pdf`.
- **Analysis**: Sends `FormData` to `/api/ats/analyze`.
- **Result Display**:
  - **Score Visualization**: React Chart.js Doughnut chart showing the percentage match.
  - **Color Logic**: Green (>80), Yellow (>50), Red (<50).
  - **Skill Gaps**: Renders missing keywords as red badges to highlight deficiencies.

- **AI Summary**: Displays the text feedback returned by Gemini.

4. Setup & Installation

Follow these steps to deploy the application locally.

4.1 Prerequisites

- **Java Development Kit (JDK) 21**: Required for Spring Boot 3.2.
- **Node.js (v18+)**: Required for React/Vite.
- **MySQL Server**: Ensure it's running on port `3306`.
- **Maven**: (Optional, generic wrapper included).

4.2 Database Setup

1. Open MySQL Workbench or CLI.
2. Create the database:

```
CREATE DATABASE ems_db;
```

(Tables will be auto-generated by Hibernate on first run).

4.3 Backend Setup

1. Navigate to `backend/`.
2. Configure `src/main/resources/application.properties`:

```
spring.datasource.url=jdbc:mysql://localhost:3306/ems_db
spring.datasource.username=root
spring.datasource.password=YOUR_PASSWORD
gemini.api.key=YOUR_API_KEY (Optional for ATS)
```

3. Run the application:

```
./mvnw spring-boot:run
```

4.4 Frontend Setup

1. Navigate to `frontend/`.
2. Install dependencies:

```
npm install
```

3. Start the dev server:

```
npm run dev
```

4. Open `http://localhost:5173` in your browser.

4.5 Default Credentials

- **Admin**: `admin@ems.com / password`
- **HR**: `hr@ems.com / password`
- **Employee**: `alice.johnson@ems.com / password`

5. Project Directory Structure

A Roadmap to the codebase files.

5.1 Backend (`/backend`)

```
backend/
├── pom.xml                    # Maven Dependencies
├── src/main/java/com/ems/backend/
│   ├── BackendApplication.java # Entry Point
│   ├── config/
│   │   ├── SecurityConfig.java # CORS, FilterChain, URL Security
│   │   └── DataSeeder.java     # Default User/Data creation
│   ├── controller/             # REST Endpoints
│   │   ├── AuthController.java
│   │   ├── EmployeeController.java
│   │   └── ...
│   ├── model/                  # JPA Entities (DB Tables)
│   │   ├── User.java
│   │   ├── Attendance.java
│   │   └── ...
│   ├── repository/             # DB Logic (Hibernate/SQL)
│   ├── security/               # JWT Logic
│   │   ├── JwtUtils.java
│   │   └── CustomUserDetailsService.java
│   └── service/                # Business Logic
│       ├── AttendanceService.java
│       └── ...
└── src/main/resources/
    └── application.properties  # Config (DB, JWT Secret)
```

5.2 Frontend (`/frontend`)

```
frontend/
├── index.html                 # Entry HTML
├── package.json               # NPM Dependencies
├── src/
│   ├── main.jsx               # React Root
│   ├── App.jsx                # Routing & Layout
│   ├── api/
│   │   └── axios.js           # HTTP Client + Interceptors
│   ├── components/            # Reusable UI
│   │   ├── Sidebar.jsx
│   │   ├── MyNavbar.jsx
│   │   └── UserAvatar.jsx
│   ├── context/
│   │   └── AuthContext.jsx    # Global User State
│   └── pages/                 # Views
│       ├── Login.jsx
│       ├── Dashboard.jsx
│       ├── EmployeeList.jsx
```

```
│       ├── Attendance.jsx
│       ├── Payroll.jsx
│       ├── LeaveApply.jsx
│       └── ...
└── public/                    # Static Assets
```

6. API Reference (Examples)

Standard JSON payloads for key operations.

6.1 Authentication

POST `/api/auth/login`

```
// Request
{
  "email": "admin@ems.com",
  "password": "password"
}

// Response (200 OK)
{
  "token": "eyJhbGciOiJIUzI1NiJ9...",
  "role": "ADMIN",
  "expiration": 86400000
}
```

6.2 Employee Creation

POST `/api/employees`

```
// Request
{
  "name": "John Doe",
  "email": "john@ems.com",
  "password": "securePass123",
  "role": "EMPLOYEE",
  "department": "IT",
  "salary": 75000
}
```

6.3 Leave Application

POST `/api/leaves`

```
// Request
{
  "startDate": "2024-03-10",
  "endDate": "2024-03-12",
  "leaveType": "SICK_LEAVE",
  "reason": "Viral Fever"
}
```

7. Troubleshooting

Common issues and fixes.

7.1 "Permission Denied" on `mvnw`

- **Cause**: Script lacks execution rights.
- **Fix**:

```
chmod +x mvnw
```

7.2 CORS Errors (`Access-Control-Allow-Origin`)

- **Cause**: Frontend running on a port other than `5173`, or Backend security config mismatch.
- **Fix**: check `SecurityConfig.java`:

```
configuration.setAllowedOrigins(List.of("http://localhost:5173"));
```

7.3 "Unknown Database 'ems_db'"

- **Cause**: MySQL DB not created.
- **Fix**: Log into MySQL and run:

```
CREATE DATABASE ems_db;
```

8. Developer Guide: How to Add a Feature

Example: Adding a "Projects" module.

8.1 Backend Steps

1. **Entity**: Create `Project.java` in `com.ems.backend.model`.
   - Add fields (`id`, `name`, `deadline`).
   - Annotate with `@Entity`, `@Data`.
2. **Repository**: Create `ProjectRepository.java` extending `JpaRepository<Project, Long>`.
3. **Service**: Create `ProjectService.java`.
   - Add business logic (e.g., `createProject`, `findAll`).
4. **Controller**: Create `ProjectController.java`.
   - Define endpoints (`@PostMapping`, `@GetMapping`).
   - Annotate with `@RestController`, `@RequestMapping("/api/projects")`.

8.2 Frontend Steps

1. **API**: Add wrapper functions in `api/axios.js` (optional but recommended) or call directly.
2. **Component**: Create `ProjectList.jsx` in `pages/`.
   - Use `useEffect` to fetch data.
   - Render a `Table` or `Card` list.

3. **Route**: Register the page in `App.jsx`.

```
<Route path="/projects" element={<ProtectedRoute><Layout><ProjectList /></Layout></ProtectedRoute>} />
```

4. **Navigation**: Add a link in `Sidebar.jsx`.

9. Future Roadmap

Potential enhancements for Version 2.0.

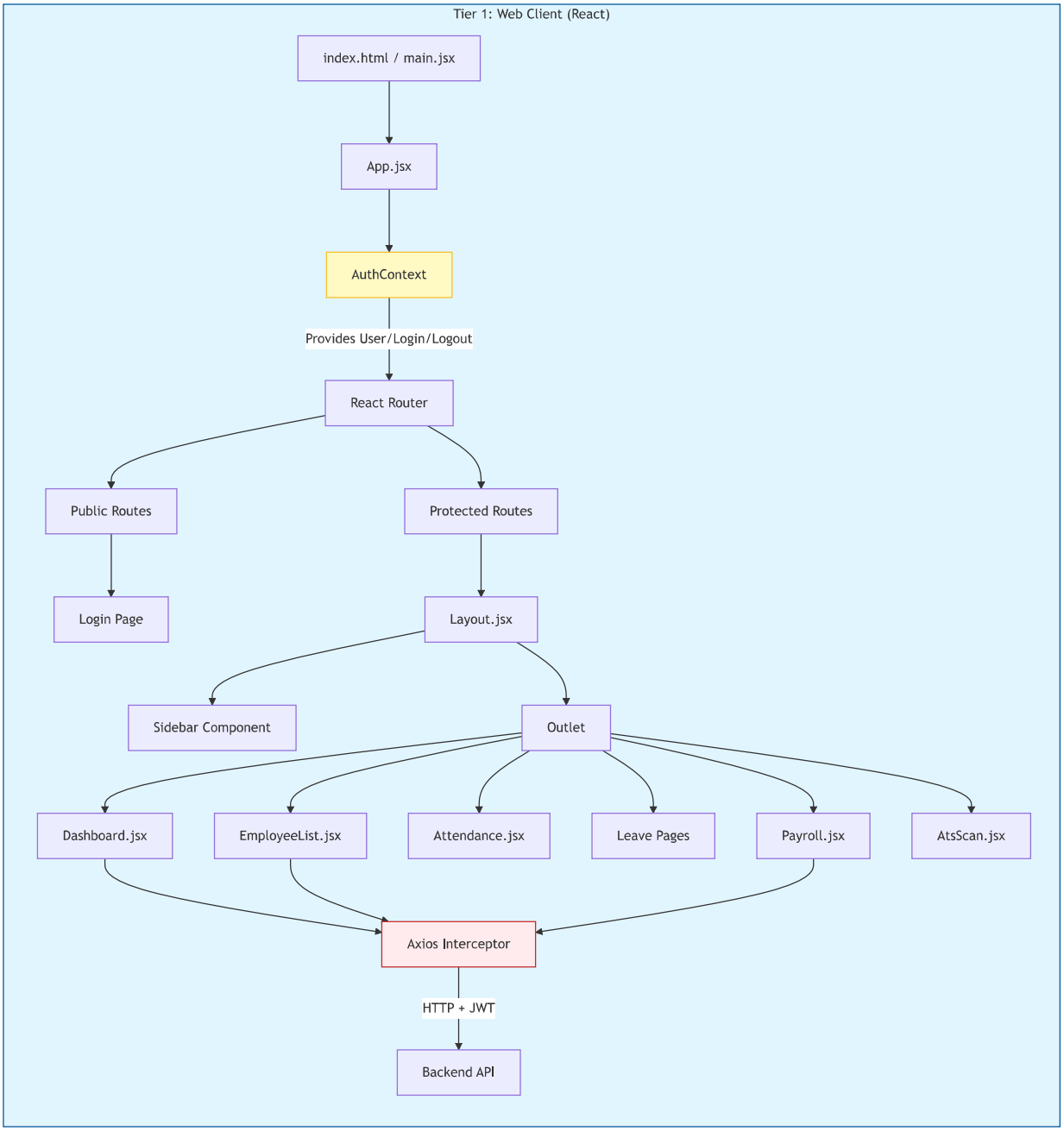
- **Dockerization**: Create `Dockerfile` and `docker-compose.yml` for one-click deployment.
- **Email Notifications**: Integration with JavaMailSender to email payslips and leave updates.
- **Mobile App**: React Native port of the frontend.
- **Redis Caching**: Cache dashboard stats for performance.
- **Unit Tests**: JUnit 5 and React Testing Library coverage.

10. Architecture Diagrams (Tier-wise)

Visualizing the system layers in detail.

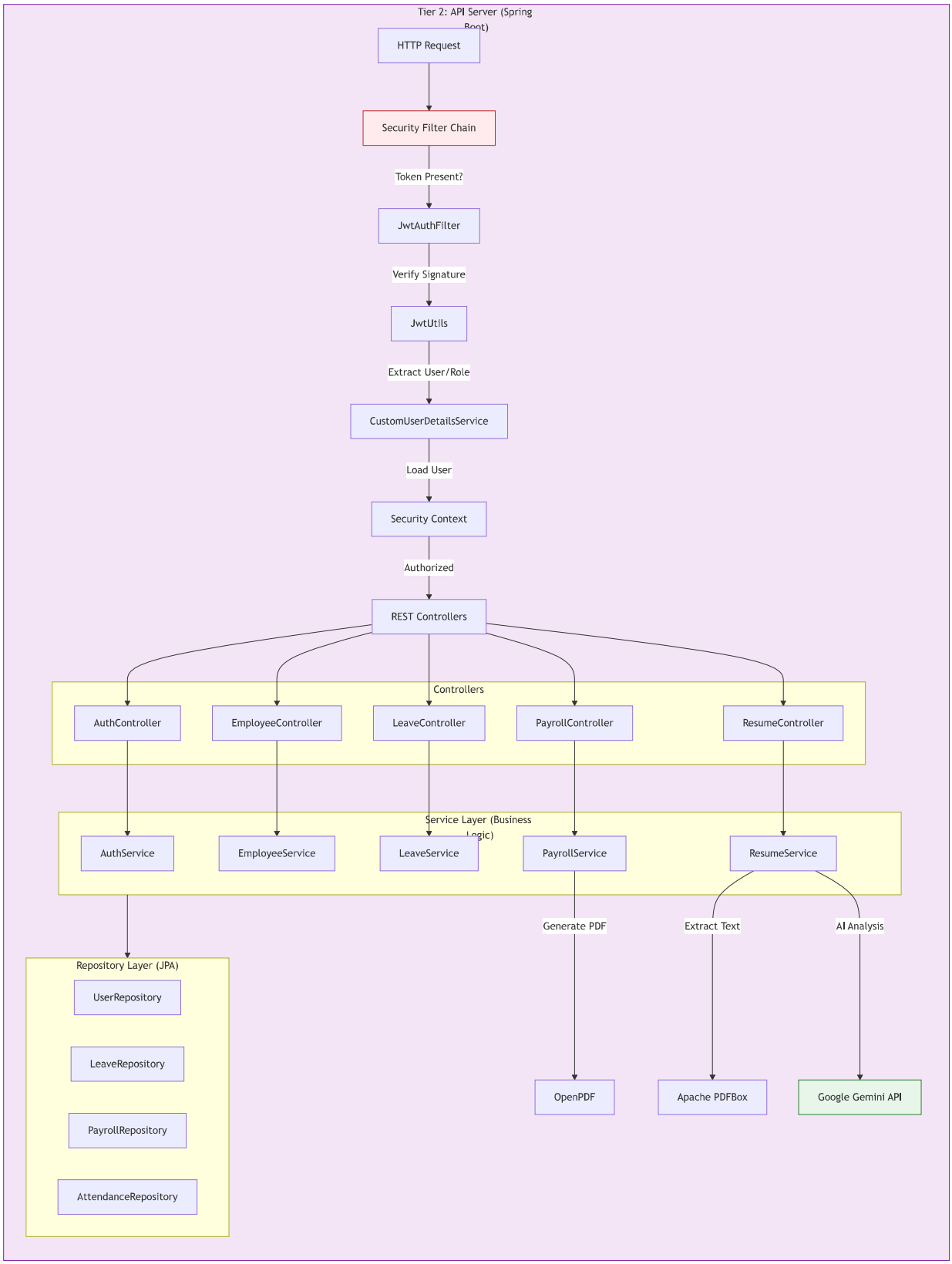
10.1 Tier 1: Presentation Layer (Frontend)

This diagram illustrates the React application structure, focusing on component hierarchy, routing, and state management.



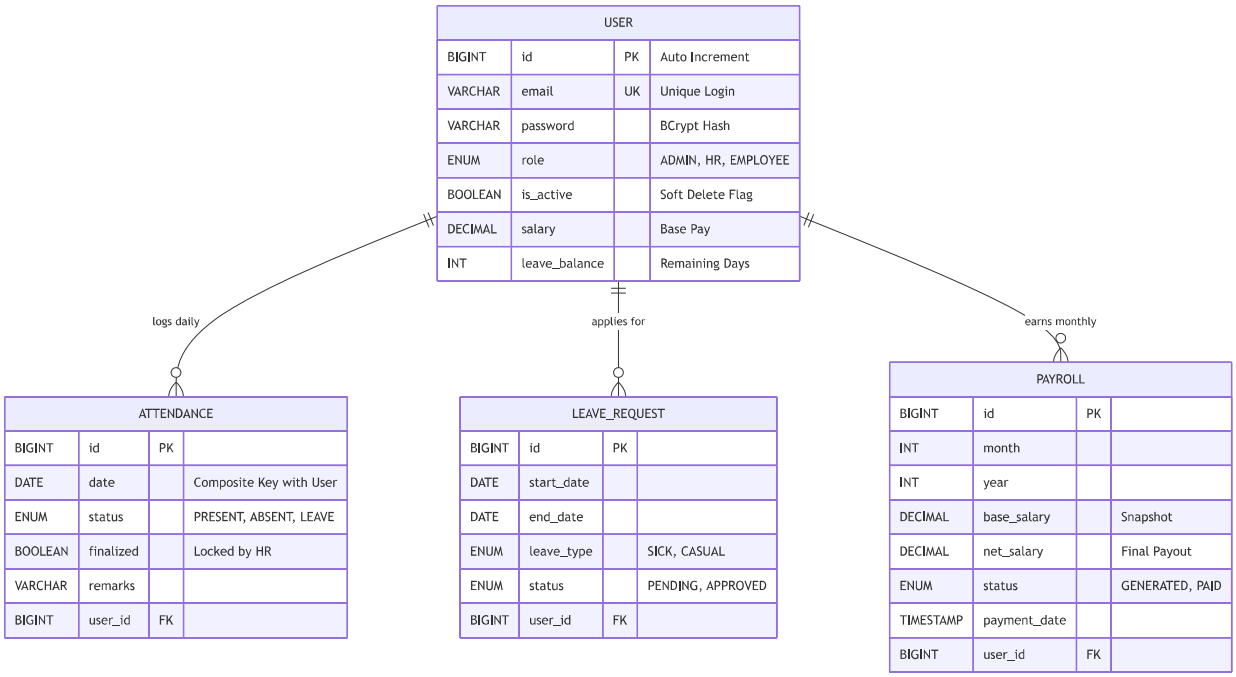
10.2 Tier 2: Business Logic Layer (Backend)

This diagram details the Spring Boot internal architecture, showing how requests flow through Security, Controllers, Services, and Repositories.



10.3 Tier 3: Data Persistence Layer (Database)

This ER Diagram illustrates the MySQL schema, highlighting the central role of the `User` table and its one-to-many relationships with operational data.


USER
BIGINT id PK Auto Increment
VARCHAR email UK Unique Login
VARCHAR password BCrypt Hash
ENUM role ADMIN, HR, EMPLOYEE
BOOLEAN is_active Soft Delete Flag
DECIMAL salary Base Pay
INT leave_balance Remaining Days
logs daily
applies for
earns monthly
ATTENDANCE
BIGINT id PK
DATE date Composite Key with User
ENUM status PRESENT, ABSENT, LEAVE
BOOLEAN finalized Locked by HR
VARCHAR remarks
BIGINT user_id FK
LEAVE_REQUEST
BIGINT id PK
DATE start_date
DATE end_date
ENUM leave_type SICK, CASUAL
ENUM status PENDING, APPROVED
BIGINT user_id FK
PAYROLL
BIGINT id PK
INT month
INT year
DECIMAL base_salary Snapshot
DECIMAL net_salary Final Payout
ENUM status GENERATED, PAID
TIMESTAMP payment_date
BIGINT user_id FK