Payroll Management System

---

1. Project Overview

The Payroll Management System is a **web-based application** designed to automate and streamline employee management, payroll processing, and leave management for an organization.

It provides functionalities based on user roles: **Admin** and **Employee**.

- **Backend**: Spring Boot, Spring Security, MySQL
- **Frontend**: React with Bootstrap
- **Testing Tools**: Postman, Swagger

---

2. Features

2.1 Admin Role

- **Employee Management**: Add, update, delete, and view employee details.
- **Payroll Processing**: Generate monthly salary based on employee and job details.
- **Leave Management**: Approve or reject employee leave requests.
- **Reports**: Generate month-wise and department-wise salary reports.
- **Departments & Jobs**: Define departments and job roles with base salaries.

2.2 Employee Role

- **Profile Management**: View personal details.
- **Leave Requests**: Apply for leave and track status.
- **Salary Slip**: View monthly salary slips.

---

3. Technology Stack

3.1 Backend

- **Framework**: Spring Boot 3.5.5
- **Security**: Spring Security with JWT authentication
- **Database**: MySQL
- **Build Tool**: Maven
- **Testing**: Postman, Swagger UI
- **Dependencies**:
    - Spring Boot Starter Web

- Spring Boot Starter Data JPA
- Spring Boot Starter Security
- Springdoc OpenAPI
- JJWT (JWT implementation)
- Jakarta Validation API

3.2 Frontend

- **Framework**: React 19.1.1
- **UI Library**: Bootstrap 5.3.8, React Bootstrap 2.10.10
- **Form Handling & Validation**: Formik & Yup
- **Charts & Reports**: Recharts
- **PDF Generation**: jsPDF
- **HTTP Requests**: Axios
- **JWT Handling**: jwt-decode
- **Routing**: react-router-dom 7.8.2
- **Icons**: react-icons 5.5.0

---

4. System Architecture

- **Authentication**: JWT-based token authentication
- **Role-Based Access Control**: Separate routes and features for Admin and Employee
- **Data Persistence**: Hibernate JPA manages ORM between backend entities and MySQL
- **Reporting & Analytics**: Backend generates payroll and department cost reports, visualized in frontend using Recharts

---

5. Database Design

Database Name: payroll_db

5.1 Tables

- **users** – Stores login credentials and roles
- **employees** – Stores employee details, job, and department info
- **departments** – Stores department names and descriptions
- **jobs** – Stores job roles and base salaries
- **leaves** – Stores leave requests and status

- **payroll_runs** – Stores monthly payroll run details
- **payroll_items** – Stores payroll details for individuals and departments

5.2 Relationships

- One-to-one: user ↔ employee
- One-to-many: department ↔ employees
- One-to-many: job ↔ employees
- One-to-many: employee ↔ leaves
- One-to-many: employee ↔ payroll_runs and payroll_items

---

6. API Documentation

The backend exposes **RESTful APIs**, secured with **JWT authentication**, with access controlled by role.

6.1 Leave Controller

- PUT /api/v1/leaves/{leaveId}/status – Update leave status
- GET /api/v1/leaves – Retrieve all leave requests
- POST /api/v1/leaves – Submit a new leave request
- GET /api/v1/leaves/employee/{employeeId} – Get leave requests for a specific employee

6.2 Job Controller

- PUT /api/v1/jobs/{id} – Update a job role
- DELETE /api/v1/jobs/{id} – Delete a job role
- GET /api/v1/jobs – List all job roles
- POST /api/v1/jobs – Create a new job role

6.3 Employee Controller

- GET /api/v1/employees/{id} – Get details of a specific employee
- PUT /api/v1/employees/{id} – Update employee info
- DELETE /api/v1/employees/{id} – Remove employee record
- GET /api/v1/employees – List all employees
- POST /api/v1/employees – Add a new employee

6.4 Department Controller

- PUT /api/v1/departments/{id} – Update department
- DELETE /api/v1/departments/{id} – Delete department

- GET /api/v1/departments – Retrieve all departments
- POST /api/v1/departments – Create a new department

6.5 User Controller

- POST /api/v1/users – Register a new user
- GET /api/v1/users/me – Retrieve logged-in user details

6.6 Payroll Controller

- GET /api/v1/payroll/runs – Retrieve payroll runs
- POST /api/v1/payroll/runs – Create a new payroll run
- POST /api/v1/payroll/runs/{id}/process – Process payroll run
- POST /api/v1/payroll/runs/{id}/lock – Lock payroll run
- GET /api/v1/payroll/runs/{id}/items – Get payroll items of a run
- GET /api/v1/payroll/runs/my/{year}/{month} – Get payroll for employee

6.7 Auth Controller

- POST /api/v1/auth/login – Authenticate and obtain JWT

6.8 Reports Controller

- GET /api/v1/reports/payroll/summary – Payroll summary reports
- GET /api/v1/reports/department-cost – Department cost reports

---

7. Application Configuration

The **application.properties** file was used to configure:

- **Database connection** (URL, username, password, dialect)
- **JPA behavior** (schema update, SQL logging)
- **Swagger/OpenAPI settings** (API docs enabled, Swagger UI enabled)
- **JWT settings** (secret key and token expiration time)

---

8. Project Setup

8.1 Backend

1. Clone repository
2. Open in Eclipse (or any Spring Boot IDE)
3. Configure MySQL credentials in application.properties
4. Run mvn clean install to build

5. Start the Spring Boot application

8.2 Frontend

1. Navigate to frontend directory
2. Run npm install to install dependencies
3. Run npm start to launch React development server

---

9. Day-wise Progress (with Problems & Solutions)

Day 1 – Friday, 29th August 2025

Work Done:

- Understood problem statement & listed features
- Planned modules for Admin & Employee
- Set up backend project with MySQL + Spring Boot
- Created **UserController** & **EmployeeController**
- Designed entities (Jobs, Departments, Leaves, PayrollRuns, PayrollItems)

Problems & Solutions:

- **Entity Mapping Issues** → Infinite recursion during JSON serialization
  Fixed using @JsonBackReference, @JsonManagedReference, @JsonIgnore.
- **Employee Basic Pay not updating** → Fixed in EmployeeService.java by setting job basic pay explicitly.
- **403 Errors in Swagger** → Solved by adding JWT Bearer token support in Swagger config.
- **Swagger UI dependency issues** → Fixed Maven dependency with springdoc-openapi.

Day 2 – Saturday, 30th August 2025

Work Done:

- Defined entity relationships (One-to-One, One-to-Many, Many-to-One)
- Built repositories, services, and controllers for **Jobs, Departments, Leaves**
- Used **DTOs** for clean data transfer
- Implemented **JWT authentication** with role-based access
- Made app stateless by disabling default sessions

Problems & Solutions:

- **Confusion between @RequestParam & @RequestBody** → Learned correct usage, fixed Leave module API & frontend calls.
- **JWT not validating** → Fixed filter chain execution order.
- **Mismatch between backend & frontend API calls** → Adjusted controller signatures and Axios requests.
- **Role access issues** → Restricted employee from accessing admin APIs with proper security config.

Day 3 – Sunday, 31st August 2025

Work Done:

- Developed **Payroll Module** (Payroll Run + Payroll Items)
- Created payroll processing & salary slip APIs
- Populated database with admin/employee users
- Set up frontend with **React + Axios + JWT**
- Built pages for Admin & Employee
- Integrated API calls, aligned frontend-backend communication

Problems & Solutions:

- **Infinite recursion in Payroll entities** → Solved with JSON annotations.
- **Cascade delete not working** → Used cascade + orphan removal + FK constraints.
- **CORS errors** → Enabled allowed origins in Spring config.
- **Axios not sending token** → Implemented Axios interceptors with localStorage.
- **Token expiration handling** → Added auto-logout & redirect to login.