Payroll Management System – Capstone Documentation

1) Aim of the Project

Design and implement a secure, role-based **Payroll Management System** that automates payroll computation, streamlines leave workflows, and provides transparent access to salary information for **Admins** and **Employees**.

---

2) Why Do We Need This Project?

Manual payroll is error-prone, slow, and difficult to audit. Organizations struggle with: - Maintaining accurate salary, deduction, and bonus records - Mapping leave/attendance to monthly payroll - Generating compliant, consistent salary slips and reports - Enforcing secure, role-based access to sensitive data

This system replaces ad-hoc spreadsheets and manual approvals with a **centralized, auditable, and secure** platform.

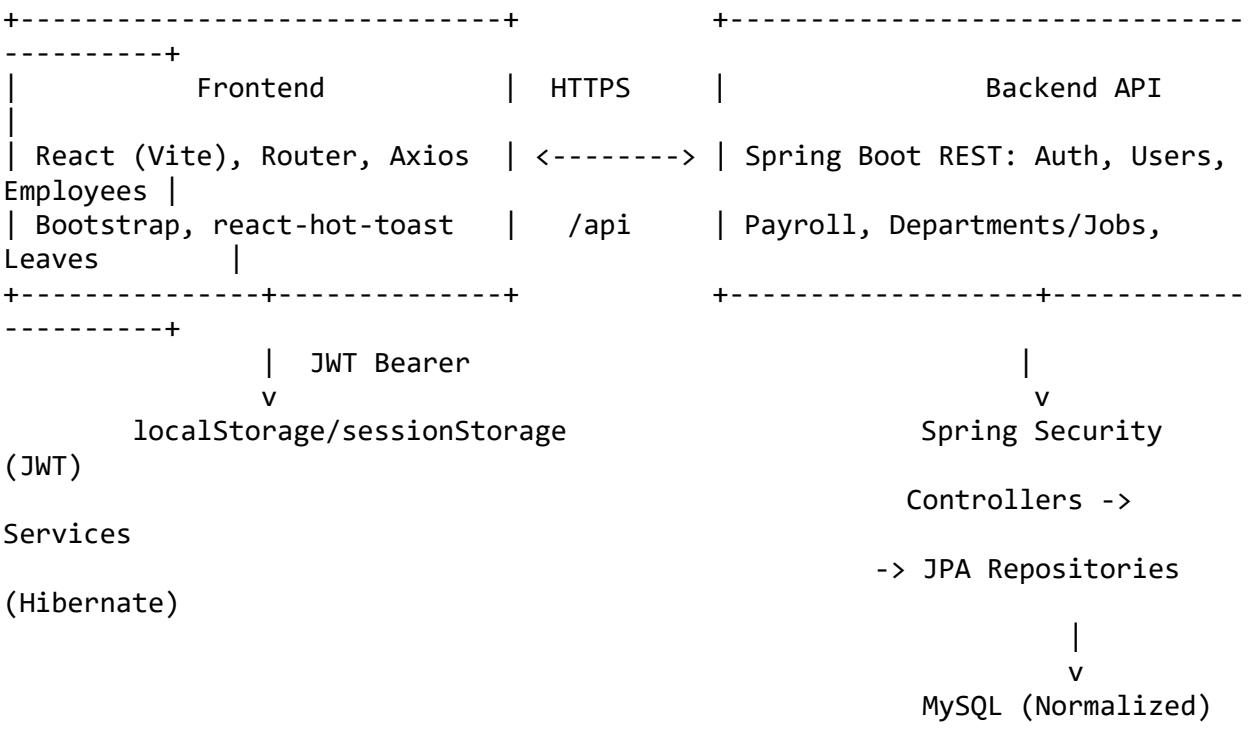
---

3) Benefits

- **Accuracy & Efficiency:** Automated calculations reduce human error and processing time.
- **Role-Based Security:** JWT authentication with Admin/Employee authorization prevents unauthorized access.
- **Transparency:** Employees can self-serve salary slips and leave status; Admins see payroll runs and department costs.
- **Traceability:** Historical salary structures and payroll runs are retained for audits and compliance.
- **Scalability:** Modular services allow future integrations (tax computation, biometric attendance, mobile apps).
- **Better Decisions:** Built-in summaries and department-cost views support operational planning.

4) Architecture & Flow Diagrams

4.1 High-Level Architecture



Key points - JWT issued on login; token is stored client-side and attached as `Authorization: Bearer <token>`. - Spring Security filters validate JWT and enforce **role-based** access (Admin vs Employee). - Business logic lives in **Services**; persistence via **JPA Repositories** (Hibernate) to **MySQL**. - **Swagger UI** documents endpoints for testing and onboarding.

4.2 Primary User Flows (Sequence-Style)

Login & Routing

```
User -> POST /api/v1/auth/login -> {accessToken, user:{id, username, role}}
UI stores token -> decodes role -> routes to Admin or Employee dashboard
Subsequent API calls add Authorization: Bearer <token>
401/403 -> auto-logout + redirect to /login
```

Employee Management (Admin)

```
Admin UI -> POST /api/v1/users (create user + role)
Admin UI -> POST /api/v1/employees (profile & org data)
Admin UI -> GET/PUT/DELETE /api/v1/employees/:id
```

Payroll Run (Admin)

```
Admin UI -> POST /api/v1/payroll/runs {year, month}
Admin UI -> POST /api/v1/payroll/runs/:id/process (computes salaries)
```

```
Admin UI -> POST /api/v1/payroll/runs/:id/lock (freeze results)
Admin UI -> GET  /api/v1/payroll/runs/:id/items (per-employee items)
```

Leave Workflow

```
Employee UI -> POST /api/v1/leave (request)
Admin UI     -> PATCH /api/v1/leave/:id {status: Approved/Rejected}
Employee UI  -> GET   /api/v1/leave/my (track status)
(Optional) Payroll Service factors approved leave into net pay for the month
```

---

5) Tools & Technologies Used

Backend (Spring Boot + MySQL)

- **Java 17**, **Spring Boot** (REST controllers)
- **Spring Security + JWT** for authentication/authorization
- **Spring Data JPA (Hibernate)** for ORM
- **Bean Validation** (`@Valid`, constraints) for request validation
- **MySQL 8** (normalized schema; historical salary structures)
- **Maven** for build/dependency management
- **Swagger/OpenAPI** for live API docs & testing
- **JUnit 5 + Mockito** for unit tests

Frontend (React)

- **React 18** with **Vite**
- **React Router v6** for client-side routing & role-based redirects
- **Axios** with interceptors for JWT headers & 401 handling
- **Bootstrap 5** for responsive UI
- **react-hot-toast** for feedback & notifications
- **jwt-decode** (or equivalent) for extracting role/claims client-side

Tooling & Collaboration

- **Git & GitHub** for version control
- **Postman/Swagger UI** for API exploration

---

6) Challenges & How We Solved Them

1) **JWT & Role-Based Routing (401/403, stale sessions)**
   Solution: Central **AuthProvider** on the frontend stores/refreshes token; Axios interceptor appends `Authorization` header and logs out on 401/403; route guards render Admin/Employee areas conditionally.

2) **CORS & Environment Configuration**
Solution: Server-side CORS config for `http://localhost:5173`; front-end `.env` (e.g., `VITE_API_BASE=http://localhost:8080/api/v1`) consumed by a single Axios instance.

3) **Blank Screen / HMR & Import Issues**
Solution: Align named/default exports (especially hooks), clear caches, reinstall `node_modules`, and avoid OS-specific commands (`rmdir /s /q node_modules` on Windows). Add missing deps (e.g., `axios`) and verify import paths.

4) **Schema & Mapping Integrity (User ↔ Employee)**
Solution: Enforce one-to-one mapping at DB level; use DTOs to decouple API models; validate inputs (`@NotNull`, `@Email`, ranges) to prevent bad data.

5) **Payroll Calculations & Idempotency**
Solution: Encapsulate computation in a service; run calculations inside a transaction; store **payroll runs** and **items** with a locked state to prevent re-processing.

6) **Leave Workflow Consistency**
Solution: Separate Leave Request, Leave Type, and balances; restrict approvals to Admin role; ensure status transitions are validated and auditable.

7) **Discoverability & Onboarding**
Solution: Provide **Swagger UI** and a concise README with environment setup, seed data, and sample requests.

8) **Testing Core Logic**
Solution: Unit tests with **JUnit + Mockito** for services (employee creation, payroll run processing, leave approvals), mocking repositories and asserting side-effects.

---

7) Project Flowcharts

7.1 Admin Role Flow

```
flowchart TD
  A[Admin Login] --> B[Admin Dashboard]
  B --> C[Manage Users/Employees]
  B --> D[Create Payroll Run]
  B --> E[Review Leave Requests]
  C --> C1[CRUD Employees]
  D --> D1[Process Run]
  D1 --> D2[Lock Run]
  D2 --> D3[View Items & Reports]
  E --> E1[Approve/Reject Requests]
  D3 --> F[Department-Cost Summary]
```

7.2 Employee Role Flow

```
flowchart TD
  A[Employee Login] --> B[Employee Dashboard]
  B --> C[View/Update Profile]
  B --> D[Apply for Leave]
  B --> E[View Salary Slip]
  D --> D1[Track Leave Status]
  E --> E1[Net Pay for Month]
```

---

8) Complete API Endpoint Reference

Base URL: `/api/v1` (secured with JWT unless marked Public)

8.1 Authentication & Users

Endpoint	Method	Access	Description
`/auth/login`	POST	Public	Authenticate and return JWT + user role.
`/auth/register`	POST	Public	Register a new user (defaults to Employee).
`/users/me`	GET	Auth	Get the currently logged-in user info.
`/users`	POST	Admin	Create a new user and assign role.
`/users/:id/status`	PATCH	Admin	Activate/deactivate a user account.

8.2 Employees

Endpoint	Method	Access	Description
`/employees`	GET	Admin	List employees (optional filters).
`/employees`	POST	Admin	Create an employee record.
`/employees/:id`	GET	Admin/Self	Get a specific employee profile.
`/employees/:id`	PUT	Admin	Update employee details.
`/employees/:id`	DELETE	Admin	Delete an employee.
`/employees/:id/sal`	GET	Admin	Get salary structure

Endpoint	Method	Access	Description
`ary-structures`			history.
`/employees/:id/salary-structures`	POST	Admin	Assign a new salary structure.

8.3 Departments & Jobs

Endpoint	Method	Access	Description
`/departments`	GET	Admin	List departments.
`/departments`	POST	Admin	Create a department.
`/departments/:id`	PUT	Admin	Update a department.
`/departments/:id`	DELETE	Admin	Delete a department.
`/jobs`	GET	Admin	List job roles.
`/jobs`	POST	Admin	Create a job role.
`/jobs/:id`	PUT	Admin	Update a job role.
`/jobs/:id`	DELETE	Admin	Delete a job role.

8.4 Leave Management

Endpoint	Method	Access	Description
`/leave`	POST	Employee	Create a leave request.
`/leave/my`	GET	Employee	List current user's leave requests.
`/leave`	GET	Admin	List all leave requests.
`/leave/:id`	GET	Admin/Self	Get a specific leave request.
`/leave/:id`	PATCH	Admin	Update status (Approve/Reject).

8.5 Payroll

Endpoint	Method	Access	Description
`/payroll/runs`	POST	Admin	Create a payroll run (year, month).
`/payroll/runs/:id/process`	POST	Admin	Compute salaries for the run.
`/payroll/runs/:id/lock`	POST	Admin	Lock run to prevent changes.
`/payroll/runs/:id/items`	GET	Admin	View all payroll items for a run.
`/payroll/my/:year/:month`	GET	Employee	View own net pay for a period.

8.6 Reports

Endpoint	Method	Access	Description
`/reports/payrollsummary`	GET	Admin	Payroll summary for a period.
`/reports/department-cost`	GET	Admin	Department-wise cost report.

Conclusion

The Payroll Management System delivers a **secure, auditable, and efficient** way to manage employees, payroll, and leaves. By combining JWT-based security, modular Spring services, and a responsive React UI, it **eliminates manual errors**, improves transparency for employees, and equips admins with actionable payroll and department-cost insights. The architecture is scalable for future integrations such as tax rules, attendance systems, and mobile clients.