Payroll Management System – Capstone Documentation

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**.

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

---

Project Objective

Design and deliver a **secure, role-based Payroll Management System** that automates monthly payroll computation, streamlines leave approvals, and provides transparent, self-service access to salary information—reducing manual effort and errors while ensuring auditability and scalability.

Specific objectives

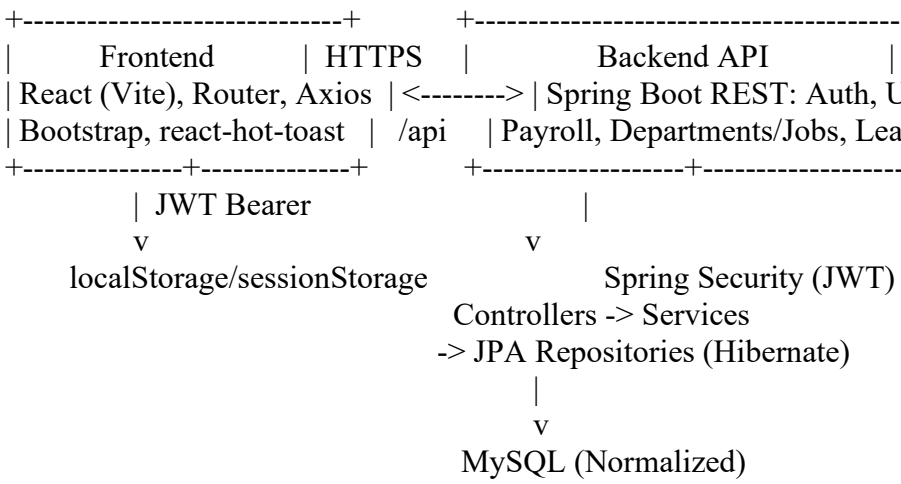
- **Automate payroll:** Calculate net pay from salary structures, deductions, bonuses, and approved leave; produce locked payroll runs and downloadable salary details.
- **Enforce security:** Use JWT authentication and **Admin vs. Employee** authorization; store credentials securely (e.g., BCrypt).
- **Employee & org management:** CRUD for employees, departments, and jobs; maintain **historical salary structures** with effective dates.
- **Leave workflow:** Employees request leave; Admins approve/reject; statuses are tracked and reflected in payroll.
- **Usable frontend:** Build a responsive React UI with role-based routing, clean forms/tables, and clear feedback states
- **Documentation & quality:** Expose APIs via **Swagger/OpenAPI**; validate inputs; handle errors consistently; include baseline unit tests for core services.
- **Reporting:** Provide payroll summaries and department-wise cost views for decision-making.

- **Scalability & maintainability:** Apply modular Spring services, normalized MySQL schema, and a clean API/UX foundation for future integrations (tax, attendance, mobile).

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.

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.

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

---

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

Day-Wise Improvisation

Day 1 – Backend Architecture & Completion

Objectives - Finalize domain model and DB schema (Users, Employees, Departments, Jobs, SalaryStructure, PayrollRun, PayrollItem, LeaveRequest). - Implement secure REST APIs with JWT & role-based access.

Work Completed - Project scaffolding (Spring Boot, Maven, modules & packages). - Entities, DTOs, mappers, repositories (JPA/Hibernate), services, controllers. - Security: Spring Security filters, JWT issuance/validation, password hashing (BCrypt), CORS. - Validation & error handling (@Valid, global exception handler), Swagger/OpenAPI enabled. - Seed & config: application YAML profiles; sample admin/employee for testing.

Outcomes/Artifacts - All core endpoints reachable via Swagger UI; DB tables created & relations verified. - Build successful; unit tests added for critical services.

Issues & Fixes - Lazy loading & JSON recursion → used DTOs and Jackson annotations. - Role mapping consistency → single authority source from JWT claims.

Plan for Day 2 - Systematic endpoint verification (happy paths, validation, authorization).

Day 2 – Endpoint Verification & QA

Objectives - Validate authentication/authorization, CRUD operations, and payroll/leave workflows.

Test Activities - **Auth & Users:** /auth/login, /auth/register, /users/me, role enforcement. - **Employees:** list/create/update/delete; salary-structure history. - **Departments & Jobs:** CRUD and referential integrity. - **Leave:** employee request → admin approve/reject → employee tracking. - **Payroll:** create run → process → lock → list items; idempotency checks. - **Reports:** payroll summary, department cost.

Tools - Swagger UI, Postman collections, MySQL Workbench; JUnit/Mockito for service logic.

Findings & Fixes - CORS mismatch fixed; request validation messages improved; standardized error responses. - Added guards for illegal state transitions; tightened 401/403 handling.

Outcomes/Artifacts - Green test suite, sample Postman environment.

Plan for Day 3 - Build React frontend, integrate APIs, and finalize UI/UX.

Day 3 – Frontend Implementation & UI Design

Objectives - Deliver responsive, role-aware React UI with secure routing and API integration.

Work Completed - Vite + React setup; folder structure for pages, components, hooks, and services. - **AuthProvider** with token storage; Axios instance + interceptors (Bearer token, 401 logout). - **Routing:** React Router v6 with ProtectedRoute; role-based dashboards and redirects. - **Admin Pages:** Employees (list/form), Departments, Jobs, Leave Approvals, Payroll Runs (create/process/lock/items), Reports. - **Employee Pages:** Profile (view/update), Apply Leave & My Leaves, My Pay for {year, month}. - **UI/UX:** Bootstrap 5 layout, tables/forms, empty states, loading indicators, and toast feedback.

Outcomes/Artifacts - End-to-end flows working for Admin and Employee roles; manual sanity checks passed.

Issues & Fixes - HMR/blank screen due to import/export mismatch → aligned named/default exports, reinstalled deps. - Env/config drift → unified .env keys and single Axios base configuration.

Next Steps (Post-Capstone) - Add pagination & filters, exportable reports, audit logs; improve test coverage and CI.

Challenges and solutions

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.

Project Flowcharts

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]
```

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
```

API Endpoint Reference

Base URL: /api/v1
Security: Send Authorization: Bearer <JWT> for all secured endpoints.
1) Authentication & Users

- POST /auth/login — Public — Authenticate user; returns JWT + role.
- POST /auth/register — Public — Register a new user (defaults to EMPLOYEE).
- GET /users/me — Auth — Get the currently logged-in user.
- GET /users — Admin — List users.

- POST /users — Admin — Create user and assign role (ADMIN/EMPLOYEE).
- PATCH /users/:id/status — Admin — Activate/Deactivate a user.

2) Employees
- GET /employees — Admin — List employees (filters optional).
- POST /employees — Admin — Create employee record.
- GET /employees/:id — Admin/Self — Get employee profile.
- PUT /employees/:id — Admin — Update employee details.
- DELETE /employees/:id — Admin — Delete employee.
- GET /employees/:id/salary-structures — Admin — Salary structure history.
- POST /employees/:id/salary-structures — Admin — Assign new salary structure.

3) Departments & Jobs
- GET /departments — Admin — List departments.
- POST /departments — Admin — Create department.
- PUT /departments/:id — Admin — Update department.
- DELETE /departments/:id — Admin — Delete department.
- GET /jobs — Admin — List job roles.
- POST /jobs — Admin — Create job role.
- PUT /jobs/:id — Admin — Update job role.
- DELETE /jobs/:id — Admin — Delete job role.

4) Leave Management
- POST /leave — Employee — Apply for leave.
- GET /leave/my — Employee — View my leave requests/status.
- GET /leave — Admin — List all leave requests.
- GET /leave/:id — Admin/Self — Get a specific leave request.
- PATCH /leave/:id — Admin — Approve/Reject leave.

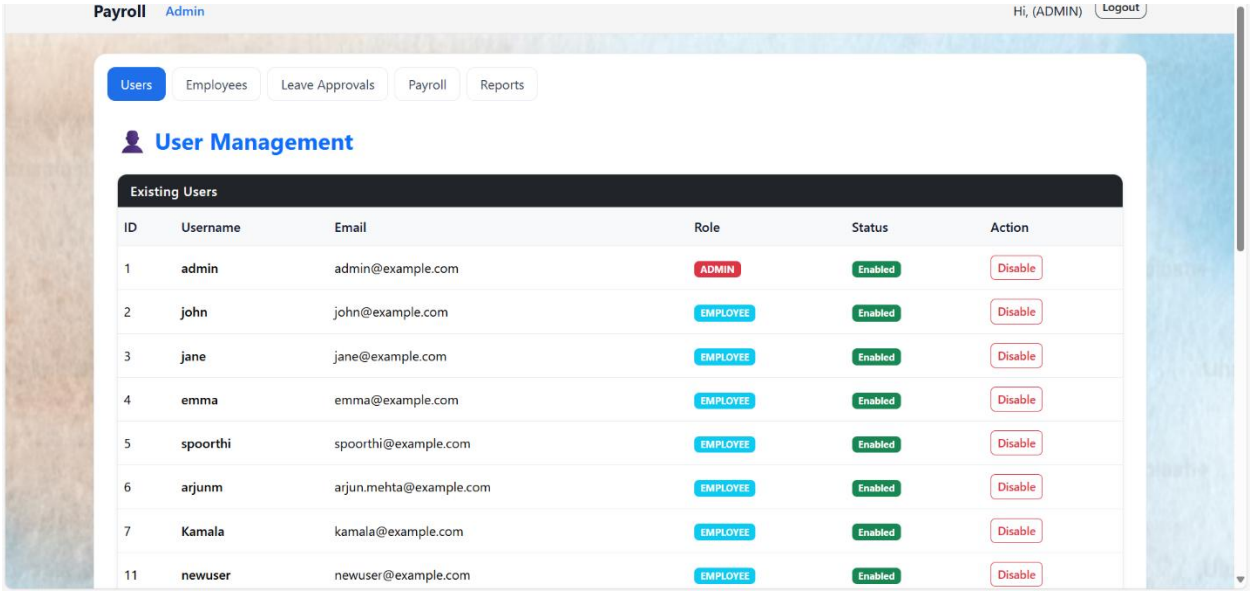
5) Payroll
- POST /payroll/runs — Admin — Create payroll run (year, month).
- POST /payroll/runs/:id/process — Admin — Compute salaries for the run.
- POST /payroll/runs/:id/lock — Admin — Lock run (freeze results).
- GET /payroll/runs/:id/items — Admin — View payroll items for the run.
- GET /payroll/my/:year/:month — Employee — View my net pay for a period.

6) Reports
- GET /reports/payrollsummary — Admin — Payroll summary for a period.
- GET /reports/department-cost — Admin — Department-wise cost report.

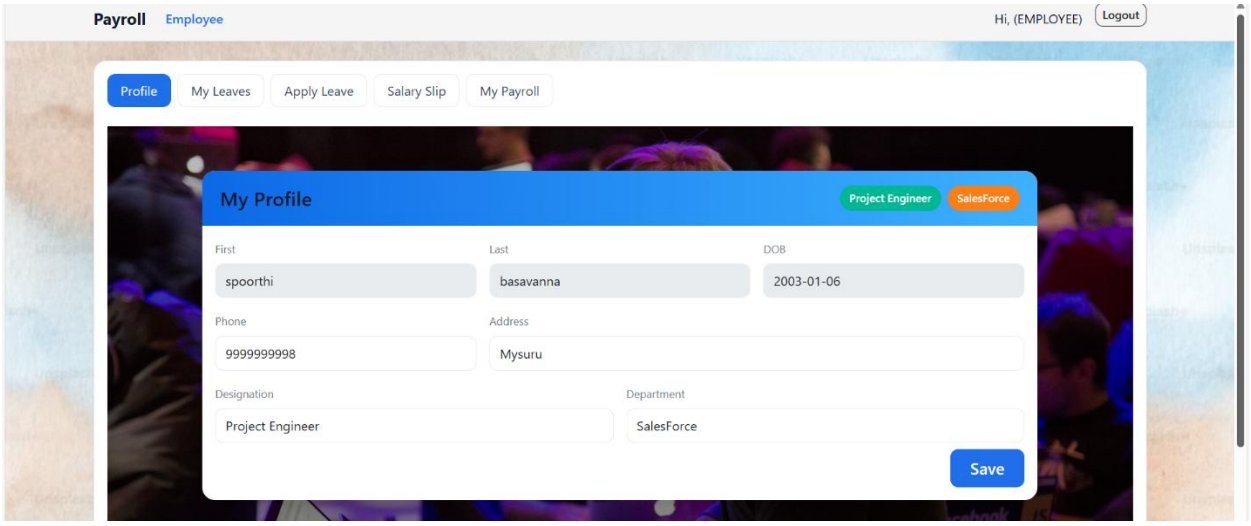
OUT PUT :

Here the images of the admin page and employee page are shown below for reference

Admin Page :



Employee Page :



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.