**Velicheti Durga Sekhar Deepak**  
*Java Full Stack React Batch 3*

**Payroll Management System – System Summary**

The Payroll Management System centralizes employee information, salary computation,  
leave management, attendance tracking, and reporting. It eliminates manual errors, improves  
compliance, and provides a secure role-based portal for both administrators and employees.

**Problem Statement**

Traditional payroll management relies heavily on manual processes, which often lead to  
salary miscalculations, compliance issues, delays in disbursing pay, and lack of transparency.  
Employees face difficulties in tracking their salaries, deductions, and leave balances.  
Administrators struggle with managing large sets of employee data, tax rules, and reporting.

The Payroll Management System addresses these problems by automating salary calculation,  
integrating attendance and leave records, ensuring data accuracy, and providing secure  
access for employees and administrators.

**Scope of the System**

**Roles**

* **Admin** – manages employee records, processes payroll, oversees attendance and leave, generates reports, and monitors compliance.
* **Employee** – views salary details, downloads payslips, applies for leave, and checks leave status.

**Security**

* Spring Security with JWT for authentication and authorization.
* BCrypt for password encryption.
* CORS configured for frontend-backend communication.
* Role-based access: Admin has full access to CRUD operations, while Employees can access only their own data and limited functions.

**Project Development Guidelines**

**Backend (Spring Boot + MySQL)**

* Tech stack: Java 17, Spring Boot 3.x, Spring Security + JWT, JPA/Hibernate, MySQL, Swagger.
* Key modules: Authentication, Employee Management, Payroll, Attendance, Leave Management, Reports.
* Configurations: datasource url → payroll\_db, ddl-auto=update, JWT secret + expiration, Swagger at /swagger-ui/index.html.

**Frontend (React + Vite + Tailwind + React Router)**

* Tech stack: React 18, Vite 5, Axios, jwt-decode, Tailwind CSS.
* Flow: Login/Register → JWT stored → Role inferred → Navigate to Employee/Admin dashboard.

**The 6 Core Modules (Implemented)**

1. Authentication & Users
2. Employee Management
3. Payroll & Salary Processing
4. Attendance & Leave Management
5. Reports & Dashboards
6. Security & Role Management

**Extended API Guidelines**

Base URL: /api/v1  
Auth: Authorization: Bearer <jwt>  
Swagger: /swagger-ui/

**AuthController**

* POST /auth/login – Public
* POST /auth/register – Public

**EmployeeController**

* GET /employees – Admin only
* POST /employees – Admin only
* PUT /employees/{id} – Admin only

**PayrollController**

* POST /payroll/calculate – Admin only
* GET /payroll/{id} – Protected (Admin/Employee)
* GET /payroll/{id}/payslip – Employee

**LeaveController**

* POST /leaves/apply – Employee
* GET /leaves – Admin/Employee
* PUT /leaves/{id}/approve – Admin only

**AttendanceController**

* POST /attendance/mark – Employee
* GET /attendance/{id} – Admin/Employee

**ReportsController**

* GET /reports/payroll – Admin
* GET /reports/attendance – Admin
* GET /reports/leaves – Admin

**Database Guidelines (Conceptual)**

Entities:

* Employee (id, name, email, password, role, department, salaryBase)
* Payroll (id, employeeId, basicSalary, deductions, netSalary, date)
* Leave (id, employeeId, startDate, endDate, type, status)
* Attendance (id, employeeId, date, status)
* User (id, email, password, role)

Relationships:

* Employee ↔ Payroll: One-to-Many
* Employee ↔ Leave: One-to-Many
* Employee ↔ Attendance: One-to-Many
* User ↔ Role: Many-to-One

**Non-Functional Requirements**

* Security: JWT + BCrypt + CORS.
* Performance: Optimized queries for reports.
* Reliability: Transactional salary + leave updates.
* Scalability: Microservice-ready.
* Auditability: Activity logs maintained.

**UX Guidelines → Implementation**

* Consistency: Tailwind CSS styling.
* Simplicity: Intuitive navigation.
* Feedback: Notifications for leave, payroll.
* Error Handling: Backend codes + frontend validation.

**Execution Notes**

**Backend**

1. Run MySQL.
2. mvn clean package -DskipTests
3. java -jar target/\*.jar or mvn spring-boot:run
4. Swagger → http://localhost:8080/swagger-ui/index.html

**Frontend**

1. npm install
2. npm run dev → http://localhost:5173
3. VITE\_API\_BASE\_URL=http://localhost:8080/api/v1

**Key Challenges and Resolutions**

* Day 1 – Setup: Fixed MySQL datasource + ddl-auto, enabled CORS.
* Day 2 – Security: Corrected JWT filter + Spring Security config, Axios interceptor.
* Day 3 – Payroll: Fixed salary deductions, added payslip PDF export.