**Payroll Management System**

**Prepared By:** Rayudu Hema sai ratna sri

**Table of Contents**

1. Introduction
2. Problem Statement
3. Scope of the System
4. System Architecture & Design
5. Technology Stack
6. Backend Modules
7. Frontend Modules
8. Database Design
9. API Documentation
10. Security Implementation
11. Code Samples
12. UX Guidelines
13. Testing Strategy
14. Challenges Faced
15. Future Enhancements
16. Summary / Conclusion
17. References & Appendices
18. Appendix A: Sample Salary Slip

**1. Introduction**

Payroll plays a vital role in organizations by ensuring accurate and timely compensation of employees. Traditional payroll practices based on manual or spreadsheet systems are inefficient, error-prone, and difficult to secure.

The Payroll Management System (PMS) automates:

* Employee salary processing.
* Leave and attendance integration.
* Role-based dashboards for Admin and Employee.
* Data security using JWT authentication.

This documentation is written for developers and system architects. It explains:

* Architecture and design principles.
* Backend and frontend modules.
* Database schema and APIs.
* Code snippets (Spring Boot, React.js, SQL).
* Testing approaches, challenges, and future enhancements.

**2. Problem Statement**

Manual payroll introduces inefficiencies and risks:

* Miscalculations in deductions and bonuses.
* Delays in salary processing.
* Lack of real-time employee access to salary slips.
* Unauthorized data access and poor security.
* Scalability limitations for growing organizations.

**3. Scope of the System**

**Admin Role**

* Manage employees (CRUD).
* Approve/reject leave requests.
* Define departments and job roles.
* Run payroll and track history.
* Generate reports.

**Employee Role**

* View/update profile.
* Apply for leave.
* View salary slips.

**Security**

* JWT authentication.
* Role-based access.
* Encrypted password storage.

**4. System Architecture & Design**

The PMS follows a **three-tier architecture**:

1. **Frontend (React.js)** – User interface, dashboards, API calls.
2. **Backend (Spring Boot)** – REST APIs, business logic, authentication.
3. **Database (MySQL)** – Stores employees, payroll, and leave records.

**Flow:**

[Browser] → [React.js Frontend] → [Spring Boot Backend] → [MySQL Database]

**5. Technology Stack**

* **Frontend:** React.js, React Router, Axios, Bootstrap.
* **Backend:** Spring Boot, Spring Security, Hibernate/JPA.
* **Database:** MySQL.
* **Authentication:** JWT.
* **Version Control:** Git/GitHub.
* **Documentation:** Swagger UI.

**6. Backend Modules**

**Authentication Module**

* Login, JWT token generation, role assignment.

**Employee Module**

* Add, update, delete, list employees.

**Payroll Module**

* Calculate net salary.
* Maintain salary history.

**Leave Module**

* Apply leave (employee).
* Approve/reject (admin).

**Sample Code: Payroll Controller (Spring Boot)**

@RestController

@RequestMapping("/api/v1/payroll")

public class PayrollController {

@Autowired

private PayrollService payrollService;

@PostMapping("/runs/{id}/process")

public ResponseEntity<?> processPayroll(@PathVariable Long id) {

payrollService.processPayroll(id);

return ResponseEntity.ok("Payroll processed successfully!");

}

@GetMapping("/my/{year}/{month}")

public ResponseEntity<PayrollDTO> getMyPayroll(

@PathVariable int year,

@PathVariable int month,

Principal principal) {

return ResponseEntity.ok(

payrollService.getPayrollForEmployee(principal.getName(), year, month)

);

}

}

**7. Frontend Modules**

**Modules**

1. Authentication Page (login).
2. Admin Dashboard (manage employees, payroll, leave).
3. Employee Dashboard (profile, salary slip, leave).
4. Role-based routing.

**Sample Code: Employee Dashboard**

import React, { useEffect, useState } from "react";

import axios from "axios";

function EmployeeDashboard() {

const [salarySlip, setSalarySlip] = useState(null);

useEffect(() => {

axios.get("/api/v1/payroll/my/2025/09", {

headers: { Authorization: `Bearer ${localStorage.getItem("token")}` }

}).then(res => setSalarySlip(res.data));

}, []);

return (

<div>

<h2>Employee Dashboard</h2>

{salarySlip ? (

<pre>{JSON.stringify(salarySlip, null, 2)}</pre>

) : (

<p>Loading salary slip...</p>

)}

</div>

);

}

export default EmployeeDashboard;

**8. Database Design**

**Entities**

* Users
* Employees
* Payroll
* Leaves

**SQL Schema**

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) UNIQUE,

password VARCHAR(255),

role ENUM('ADMIN', 'EMPLOYEE')

);

CREATE TABLE employees (

employee\_id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

department VARCHAR(50),

designation VARCHAR(50),

salary DECIMAL(10,2),

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);

CREATE TABLE payroll (

payroll\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

basic\_salary DECIMAL(10,2),

deductions DECIMAL(10,2),

bonus DECIMAL(10,2),

net\_salary DECIMAL(10,2),

pay\_date DATE,

FOREIGN KEY (employee\_id) REFERENCES employees(employee\_id)

);

CREATE TABLE leaves (

leave\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

start\_date DATE,

end\_date DATE,

leave\_type ENUM('SICK','CASUAL','PAID'),

status ENUM('PENDING','APPROVED','REJECTED'),

FOREIGN KEY (employee\_id) REFERENCES employees(employee\_id)

);

**9. API Documentation**

**Authentication**

POST /api/v1/auth/login

Request: { "username": "john", "password": "secret" }

Response: { "accessToken": "...", "user": { "id": 1, "role": "EMPLOYEE" } }

**Employees**

GET /api/v1/employees

Headers: Authorization: Bearer <token>

Response: [ { "id": 1, "name": "John Doe", "salary": 50000 } ]

**Payroll**

POST /api/v1/payroll/runs/{id}/process

Response: "Payroll processed successfully!"

**10. Security Implementation**

* JWT Authentication.
* Role-based access.
* Passwords hashed with BCrypt.

**11. Code Samples**

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

Optional<Employee> findByUserUsername(String username);

}

function useAuth() {

const token = localStorage.getItem("token");

return token ? true : false;

}

**12. UX Guidelines**

* Consistent layouts.
* Clear confirmation messages.
* Input validation.
* Error prevention.

**13. Testing Strategy**

1. Unit Tests (JUnit, Mockito).
2. Integration Tests (Spring MockMVC).
3. UI Tests (Jest).
4. End-to-End Testing.

**14. Challenges Faced**

* JWT expiry handling.
* CORS issues between React & Spring Boot.
* Payroll atomic transactions.
* Deployment configs.

**15. Future Enhancements**

* Tax calculation integration.
* Biometric attendance.
* Mobile app support.

**16. Summary / Conclusion**

The Payroll Management System integrates **Spring Boot, React, MySQL, and JWT** into a scalable, secure, and transparent payroll solution. It eliminates manual errors and enables both admin and employees to interact with payroll data efficiently.

**17. References & Appendices**

* Spring Boot Documentation
* React.js Documentation
* MySQL Reference
* JWT Specification RFC 7519

**18. Appendix A: Sample Salary Slip**

| **Field** | **Details** |
| --- | --- |
| Employee ID | EMP102 |
| Name | John Doe |
| Department | IT |
| Designation | Software Engineer |
| Pay Date | 30-Sep-2025 |
| Basic Salary | ₹50,000 |
| Deductions | ₹5,000 |
| Bonus | ₹2,000 |
| **Net Salary** | **₹47,000** |