Employee Management System

Submitted by:

Aditya Kapoor **23BCS12101**

Submitted to: Mr. Deep Prakash

in partial fulfillment for the award of the degree of

Bachelors of Engineering IN

COMPUTER SCIENCE ENGINEERING

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

TABLE OF CONTENTS

1 Abstract

The Employee Management System (EMS) is a web application built with Spring Boot that streamlines HR operations by centralizing employee information and automating routine tasks. It addresses inefficiencies found in manual processes by enabling accurate data capture, secure access control, and quick retrieval of records. The system supports employee registration, department and role management, and report generation, with extensible capabilities for attendance and leave tracking. Using Spring Boot, Thymeleaf, and MySQL, it follows a layered MVC architecture that ensures maintainability and scalability. Role-based authentication safeguards sensitive information and aligns permissions with responsibility. The EMS aims to reduce administrative overhead, improve data integrity, and deliver actionable insights to HR teams and management.

1.1 Problem background (HR data inefficiency)

HR teams often rely on spreadsheets, email threads, and fragmented tools to manage employee information, which leads to errors, duplication, and slow decision-making. As organizations grow, manual processes fail to keep pace with the volume and complexity of data such as joining dates, departmental transfers, designations, and contact details. Retrieving records becomes time-consuming, version control is difficult, and generating accurate reports can be tedious. The lack of centralized systems undermines data consistency, while limited access control increases the risk of unauthorized exposure. These inefficiencies hinder compliance, affect employee experience, and consume valuable time that HR could otherwise invest in strategic initiatives and workforce development.

1.2 Solution overview (web-based EMS using Spring Boot)

The proposed EMS provides a unified, web-based platform that organizes employee profiles, departmental structures, and role hierarchies in a secure and accessible manner. Built with Spring Boot for rapid development and stability, it integrates Spring Data JPA to manage persistence and Thymeleaf for clean, server-side rendered views. The system supports role-based authentication to differentiate privileges between HR administrators and employees, allowing safe updates to personal data while protecting sensitive details. It standardizes CRUD operations for employees, departments, and roles, and offers filtering and reporting features for quick analysis. With modular architecture and RESTful endpoints, the EMS is prepared for future extensions including attendance, leave workflows, and payroll integration.

2 Introduction

This Organizations require reliable systems that support efficient HR operations and accurate record keeping. The Employee Management System consolidates employee details into a single application where administrators can add, update, and search information with confidence. The interface is designed to be intuitive, reducing training effort and minimizing data entry errors. The project uses widely adopted technologies—Spring Boot, MySQL, and Thymeleaf—so it can be deployed and maintained with ease. Data security is enforced using Spring Security, ensuring that access to sensitive information is restricted according to roles. This project aligns with the practical needs of HR teams by delivering a maintainable, scalable solution that improves productivity and strengthens governance.

2.1 Problem Statement

Manual handling of employee data creates operational risks and inefficiencies that directly impact HR performance. Disparate spreadsheets and paper records lack validation, are difficult to audit, and often become inconsistent over time. HR staff spend excessive time searching for information, reconciling duplicate entries, and assembling routine reports. There is no unified mechanism to assign and track departmental roles, nor reliable access controls to protect sensitive data. The absence of a centralized, secure, and user-friendly platform increases the likelihood of errors, slows down administrative processes, and limits organizational insight. A robust web-based system is required to automate record management and ensure data integrity and accessibility.

2.2 Objectives

This project aims to design and implement a centralized Employee Management System that improves data accuracy, accelerates routine HR tasks, and enhances information security. It seeks to provide an intuitive interface for managing employee profiles, including joining dates, departments, and designations, while enabling advanced search and filtering. Another objective is to implement strong authentication and role-based authorization so administrators and employees have appropriate, clearly defined privileges. The system should generate informative reports—such as employee counts by department and tenure analysis—to support informed decision-making. Finally, the architecture should be modular and scalable, allowing seamless integration of future features like attendance, leave workflows, payroll, and analytics.

2.3 Purpose & Scope

The EMS is intended to replace fragmented HR processes with a unified digital solution that simplifies data capture, updating, and analysis. Its purpose is to minimize human error, ensure consistency across records, and provide timely access to reliable information. The scope covers employee CRUD operations, department and role management, authentication with role-based access control, and basic reporting within a Spring Boot web application. While attendance and leave tracking are not mandatory for the core release, the system is designed to accommodate these features without architectural changes. The EMS targets small to medium-sized organizations but can scale to larger deployments with minor optimizations.

2.4 Technologies Used

The system is implemented in Java using the Spring Boot framework to simplify configuration, dependency management, and application startup. Spring Data JPA provides a powerful abstraction for database operations, reducing boilerplate and enabling clean repository patterns. Thymeleaf is used for server-side view rendering, integrating naturally with Spring MVC and form validation. MySQL serves as the relational database due to its reliability, performance, and widespread adoption. Spring Security secures authentication and authorization workflows with role-based access policies. Maven handles build and dependency management, while Git and GitHub (repository: Ghost-Ak24/SPRING-BOOT) support version control and collaboration. Together, these technologies form a robust, maintainable, and industry-aligned solution stack.

3 System Design

The EMS follows a layered, modular design to separate concerns and improve maintainability. At the presentation layer, Spring MVC controllers process HTTP requests and bind data to Thymeleaf templates for dynamic views. A dedicated service layer encapsulates business rules, ensuring consistent validation, transformation, and transaction boundaries. The repository layer manages persistence using Spring Data JPA, mapping entities to relational tables with clear relationships among employees, departments, and roles. The database schema is normalized to reduce duplication and enforce referential integrity. This design enables straightforward testing, clear responsibility boundaries, and effortless evolution of features. The architecture anticipates future modules, ensuring low coupling and high cohesion throughout the codebase.

3.1 Architecture (Spring Boot layered structure: Controller, Service, Repository)

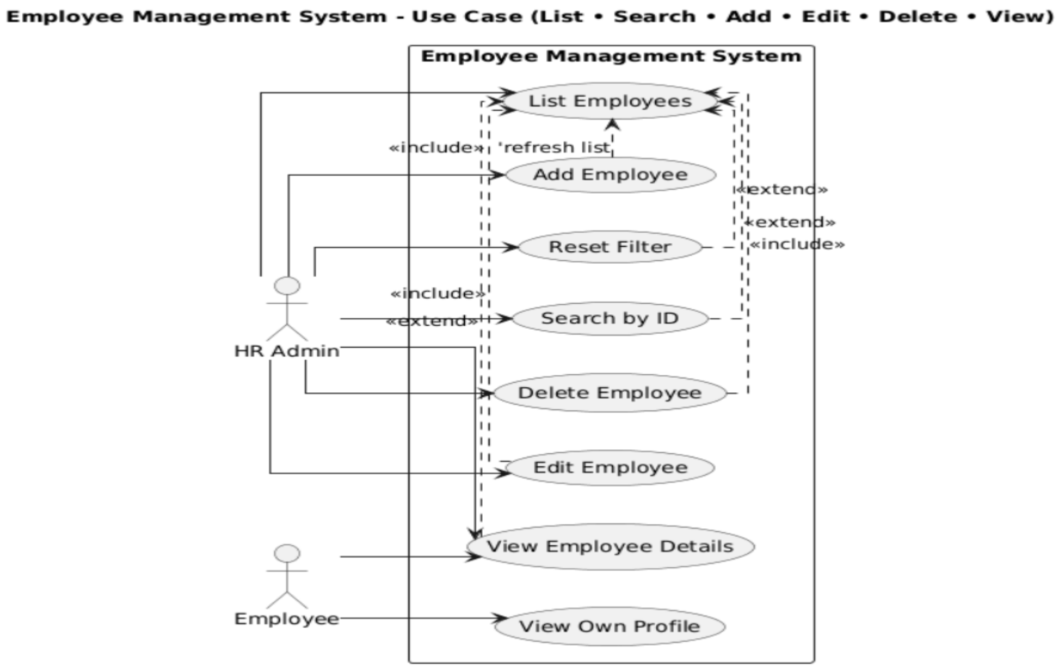
The architecture consists of Controller, Service, Repository, and Entity layers aligned with Spring MVC principles, plus Thymeleaf views. Controllers expose endpoints and orchestrate requests, delegating business logic to services. Services implement validation, rule enforcement, and transaction handling, ensuring that operations remain consistent and secure. Repositories abstract database access with interfaces that Spring Data JPA implements at runtime. Entities define the domain model—Employee, Department, and Role—with annotations for mapping and constraints. Thymeleaf templates render forms, tables, and feedback messages, while Spring Security filters requests and enforces access rules. This composition promotes testability, traceability, and scalability.

3.2 Data Flow Diagram

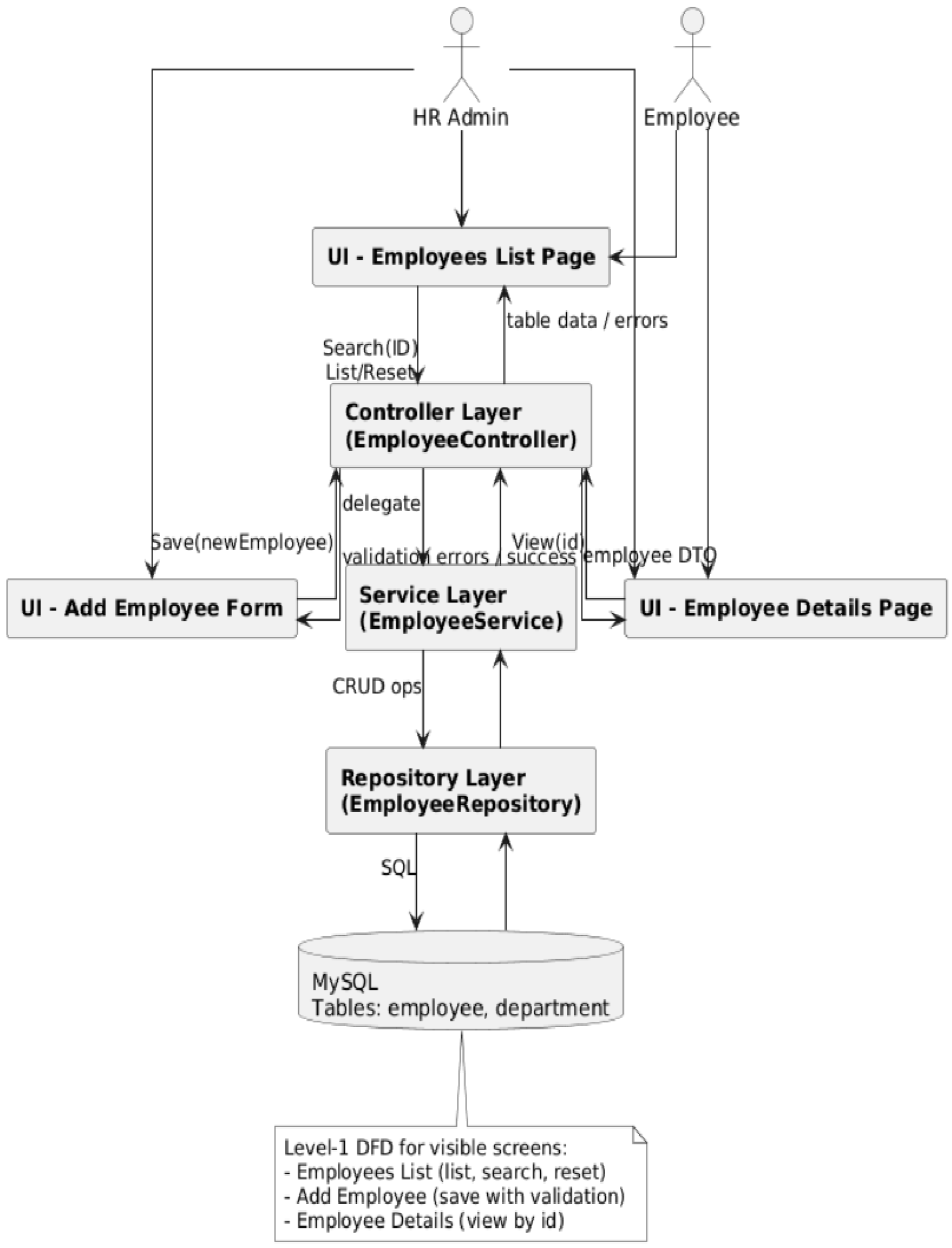
The data flow begins when an authenticated user submits a request via a browser. The application validates inputs, applies business rules, and persists or retrieves information from MySQL through JPA repositories. At Level 0, the EMS is represented as a single process that exchanges data with two external entities: Admin and Employee. At Level 1, subprocesses include authentication, employee management, and reporting. Administrators create or update employee records, assign departments and roles, and trigger report generation. Employees access their profiles and request permitted updates. Each action flows through controllers, services, and repositories before reflecting in views.

3.3Use Case Diagram

The system comprises two primary actors: Admin and Employee. Admins authenticate, manage employee profiles, define departments and roles, and generate analytical reports. Employees authenticate to view their profiles and update permitted fields such as contact details. Optional enhancements include leave requests by employees and approval workflows by admins. The diagram highlights relationships among use cases and actors, emphasizing role-based access boundaries enforced by Spring Security. It also reflects typical CRUD operations and search/filter capabilities that HR users rely upon. By modeling responsibilities and actions succinctly, the use case diagram ensures that functional requirements remain clear and traceable throughout development and testing.



Employee Management System - Data Flow Diagram (List • Search • Add • View)



4.Implementation

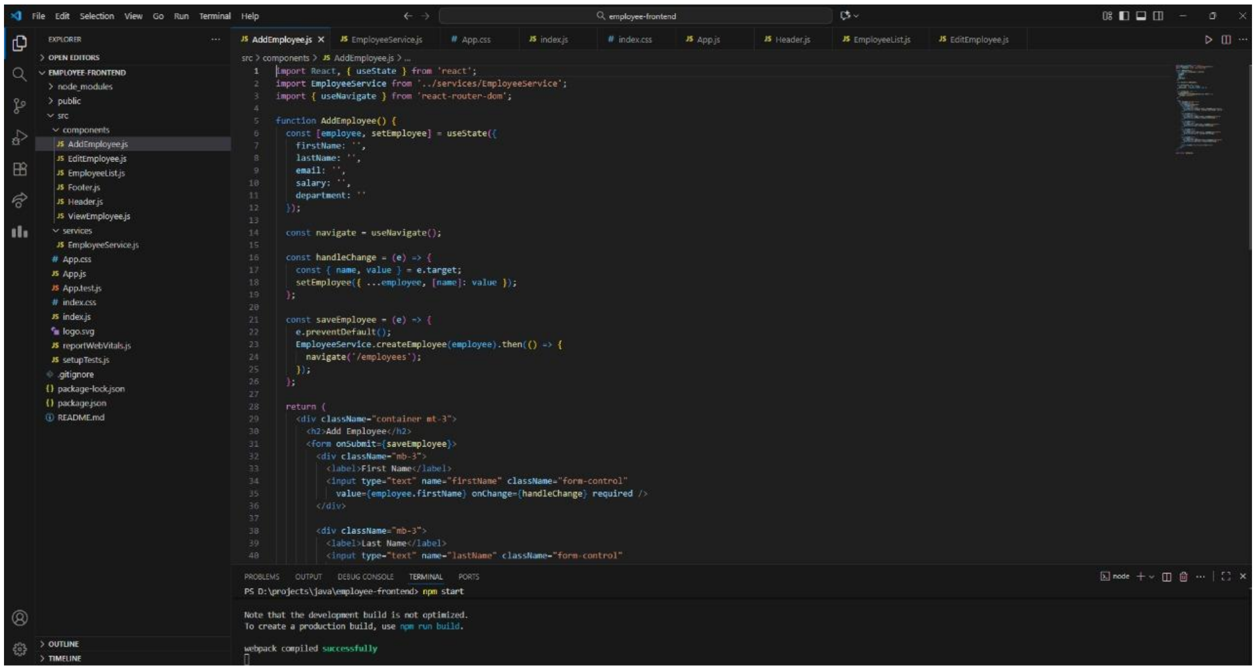
The implementation follows a conventional Spring Boot project structure with packages for controllers, services, repositories, and entities. Configuration is managed through application properties, including database connectivity and Hibernate settings. Controllers handle routing, form submissions, and error feedback. Services encapsulate update logic, enforce required fields, validate formats, and guard against duplicates. Repositories provide CRUD operations and custom query methods for filtering by department, role, or joining date. Thymeleaf templates render accessible forms and tables, leveraging Bootstrap for responsive design. Security configuration defines user roles, login flows, and access restrictions for administrative endpoints. This organized implementation facilitates testing, maintainability, and future extension of the codebase.

4.1 Tools & Technologies

Development uses Java 17 (or compatible LTS), Spring Boot, and Maven for dependency management and builds. IntelliJ IDEA or Eclipse serves as the IDE, with Lombok optionally reducing boilerplate for getters, setters, and constructors. MySQL and MySQL Workbench provide a reliable database and modeling environment, while Postman assists in testing endpoints where REST interactions are present. Git and GitHub (repository: Ghost-Ak24/SPRING-BOOT) support version control, issue tracking, and collaboration. JUnit and Spring Test enable unit and integration testing, ensuring code quality and regression safety. Bootstrap enhances front-end responsiveness within Thymeleaf templates, delivering consistent, user-friendly screens across desktop and mobile devices.

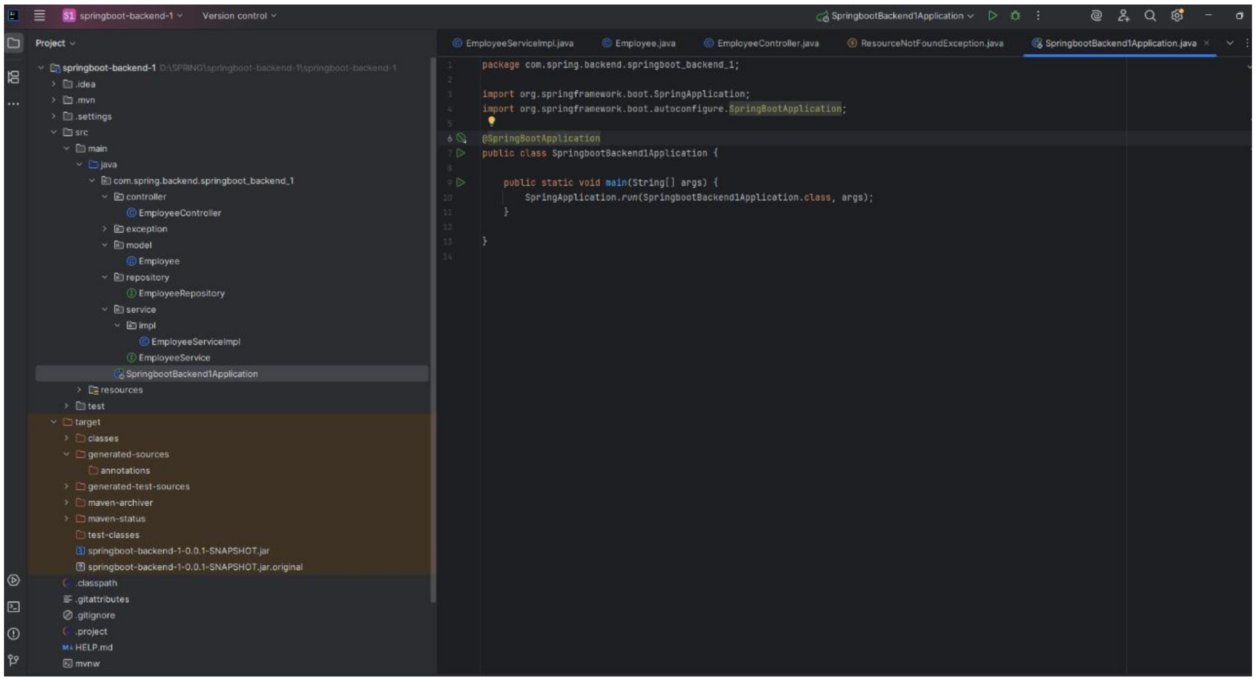
4.2 Code Modules (Employee CRUD, Department, Role Management, Login)

The Employee module supports creating profiles with validated fields, editing attributes like designation or department, listing employees with filters, and deactivating or deleting records when appropriate. Department management enables HR to define departments, assign employees, and maintain consistent naming conventions. Role management establishes designations with permissions mapped to Spring Security roles, ensuring restricted access to administrative functions. Authentication and authorization modules validate credentials and direct users to appropriate views after login. Controllers coordinate these modules, services apply business logic and validation, and repositories abstract persistence. The design isolates responsibilities, making it straightforward to adjust rules, add fields, or extend workflows without disruptive changes.



```
import React, { useState } from 'react';
import EmployeeService from '../services/EmployeeService';
import { useNavigate } from 'react-router-dom';

function AddEmployee() {
  const [employee, setEmployee] = useState({
    firstName: '',
    lastName: '',
    email: '',
    salary: '',
    department: ''
  });

  const navigate = useNavigate();

  const handleChange = (e) => {
    const { name, value } = e.target;
    setEmployee({ ...employee, [name]: value });
  };

  const saveEmployee = (e) => {
    e.preventDefault();
    EmployeeService.createEmployee(employee).then(() => {
      navigate('/employees');
    });
  };

  return (
    <div className="container mt-3">
      <h2>Add Employee</h2>
      <form onSubmit={saveEmployee}>
        <div className="mb-3">
          <label>First Name</label>
          <input type="text" name="firstName" className="form-control"
            value={employee.firstName} onChange={handleChange} required />
        </div>

        <div className="mb-3">
          <label>Last Name</label>
          <input type="text" name="lastName" className="form-control"
```

```
PS D:\projects\java\employee-frontend> npm start

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

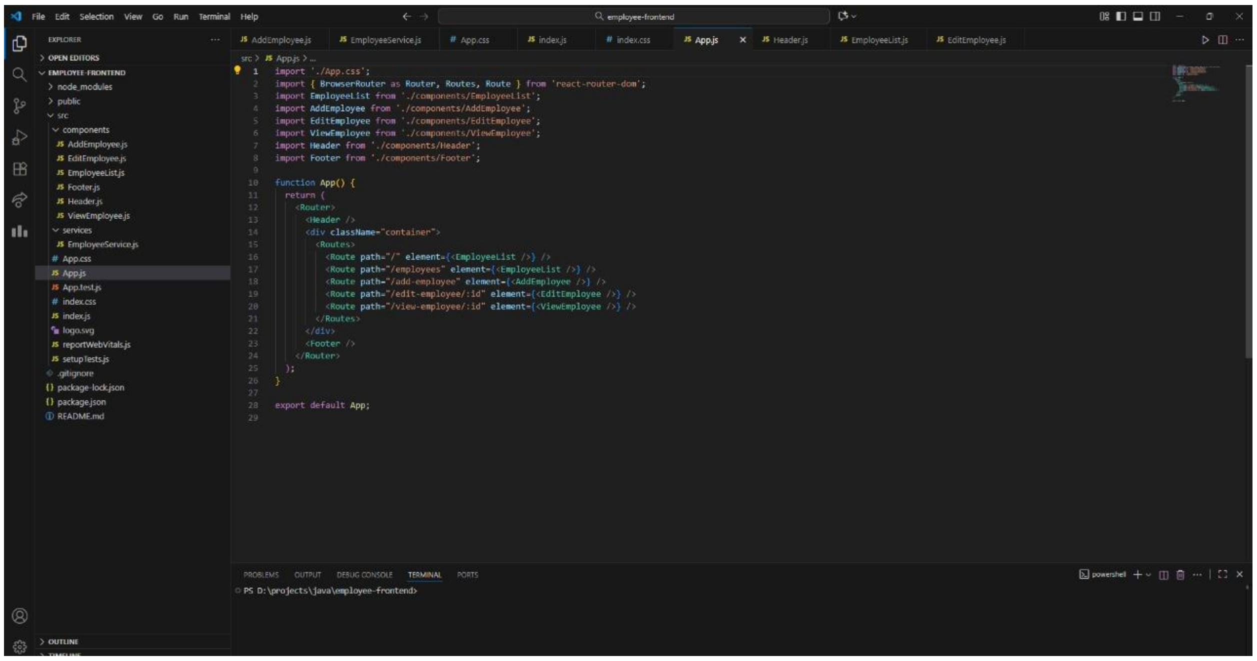


```
package com.spring.backend.springboot_backend_1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringbootBackend1Application {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootBackend1Application.class, args);
    }

}
```

Fig.1 & Fig.2 frontend and backend of project where code is implemented.

4.3 Key Features Implemented

The EMS delivers secure authentication and role-based authorization that aligns user privileges with responsibilities. It centralizes employee records and enforces validation to reduce duplicates and inconsistent data. The application supports fast search and filtering by department, role, and joining date, improving operational responsiveness for HR teams. Department and role configuration is accessible to administrators, providing flexibility as organizations evolve. Reports summarize key indicators such as employee distribution and tenure, assisting with planning and compliance. The system's layered architecture, clear code organization, and reliance on standard frameworks ensure maintainability, while thoughtful UI choices improve user experience and reduce training needs.



```javascript
import './App.css';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import EmployeeList from './components/EmployeeList';
import AddEmployee from './components/AddEmployee';
import EditEmployee from './components/EditEmployee';
import ViewEmployee from './components/ViewEmployee';
import Header from './components/Header';
import Footer from './components/Footer';

function App() {
  return (
    <Router>
      <Header />
      <div className="container">
        <Routes>
          <Route path="/" element={<EmployeeList />} />
          <Route path="/employees" element={<EmployeeList />} />
          <Route path="/add-employee" element={<AddEmployee />} />
          <Route path="/edit-employee/:id" element={<EditEmployee />} />
          <Route path="/view-employee/:id" element={<ViewEmployee />} />
        </Routes>
      </div>
      <Footer />
    </Router>
  );
}

export default App;
```

```
PS D:\projects\java\employee-frontend>
```

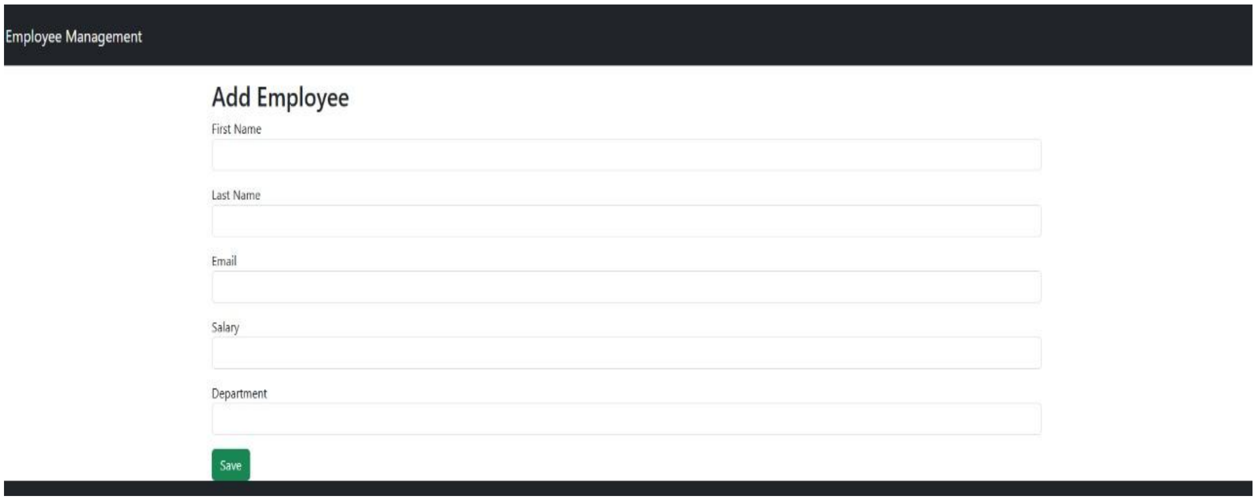
Fig 3. Represent app.js code for frontend in vscode.

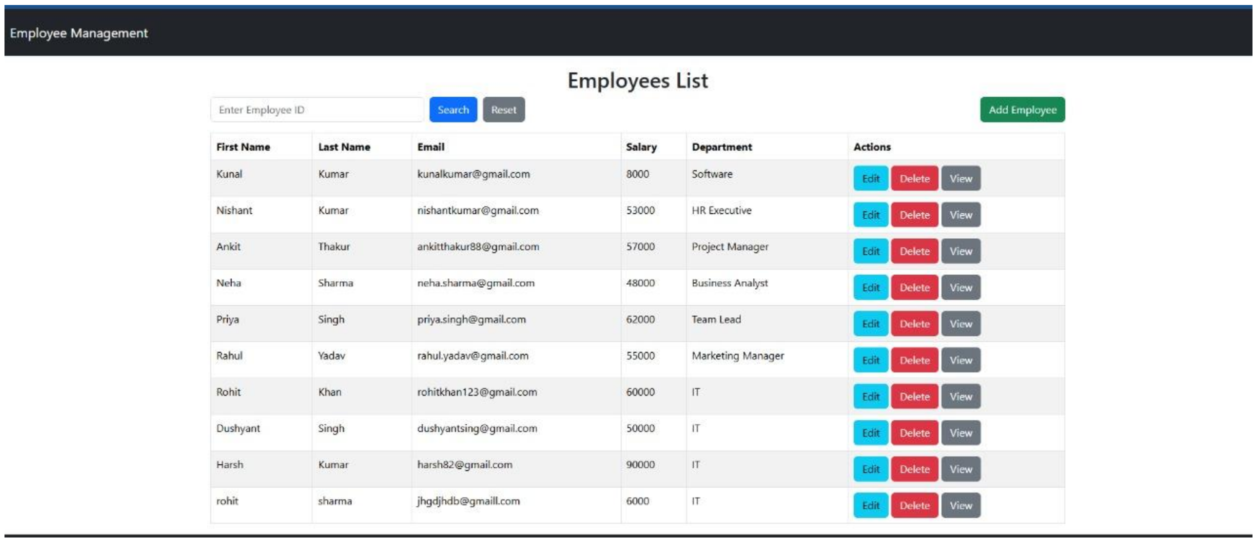
5. Results & Testing

Comprehensive testing validates both functionality and reliability. Unit tests verify business rules, field validations, and service-level logic. Integration tests exercise the full stack—controllers, services, repositories, and database—ensuring correct wiring and transaction boundaries. Manual tests simulate realistic user journeys, including login, profile management, and reporting flows. Observed results confirm that authenticated sessions are enforced, data constraints prevent duplicates, and role-based permissions restrict access appropriately. Database states are consistent after concurrent operations, and error handling provides informative feedback in the UI. Performance remains acceptable for small to medium data sets, and the system proves stable during typical HR workflows**.**

5.1Functional Testing (Add/Edit/Delete Employee)

Functional testing confirms that core CRUD operations behave as expected with valid and invalid inputs. Creating an employee requires mandatory fields such as name, email, department, role, and joining date, with validation messages shown for missing or malformed data. Editing supports updates to designation, department, or contact details while preserving auditability where implemented. Deletion—hard or soft, depending on configuration—removes entries from active listings and upholds referential integrity. Post-operation states are verified both in the UI and database tables. Pagination and search are tested to ensure correct filtering and consistent results. These tests demonstrate robust behavior and reliable data handling.



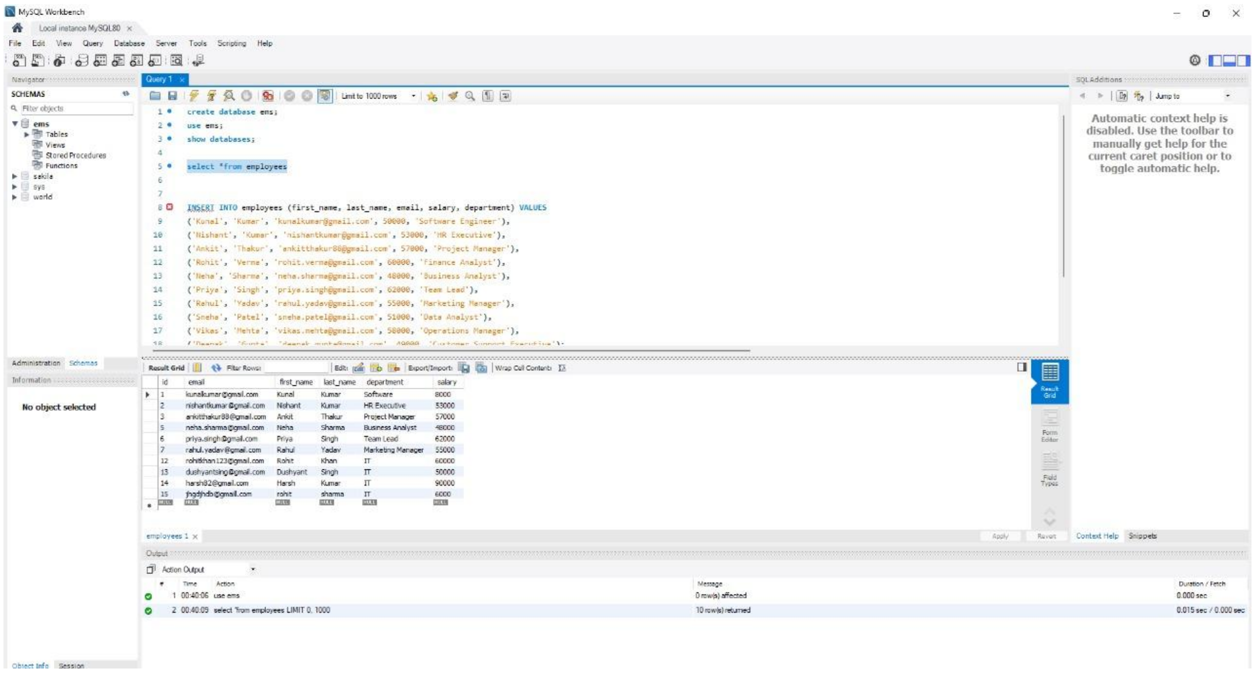


First Name	Last Name	Email	Salary	Department	Actions
Kunal	Kumar	kunalkumar@gmail.com	8000	Software	Edit Delete View
Nishant	Kumar	nishantkumar@gmail.com	53000	HR Executive	Edit Delete View
Ankit	Thakur	ankitthakur88@gmail.com	57000	Project Manager	Edit Delete View
Neha	Sharma	neha.sharma@gmail.com	48000	Business Analyst	Edit Delete View
Priya	Singh	priya.singh@gmail.com	62000	Team Lead	Edit Delete View
Rahul	Yadav	rahul.yadav@gmail.com	55000	Marketing Manager	Edit Delete View
Rohit	Khan	rohitkhan123@gmail.com	60000	IT	Edit Delete View
Dushyant	Singh	dushyantsing@gmail.com	50000	IT	Edit Delete View
Harsh	Kumar	harsh82@gmail.com	90000	IT	Edit Delete View
rohit	sharma	jhgdjhdb@gmaill.com	6000	IT	Edit Delete View

Fig 4,fig 5. In this we can add and check employee list.

5.2Role-based access test

Security tests ensure that admins and employees see only what they are permitted to access. Admin accounts can manage departments, roles, and employee records, while employee accounts are limited to viewing and editing permissible profile fields. Unauthorized access attempts to administrative endpoints redirect to login or return forbidden responses, as configured. Session management is checked for timeout behavior and correct logout flows. Password policies, if configured, are validated against common vulnerabilities. Integration with Spring Security is verified through configuration reviews and end-to-end tests, confirming that URL patterns, method-level annotations, and view controls consistently enforce the intended authorization model across the application.



```sql
create database ems;
use ems;
show databases;

select *from employees


INSERT INTO employees (first_name, last_name, email, salary, department) VALUES
('Kunal', 'Kumar', 'kunalkumar@gmail.com', 50000, 'Software Engineer'),
('Nishant', 'Kumar', 'nishantkumar@gmail.com', 53000, 'HR Executive'),
('Ankit', 'Thakur', 'ankitthakur88@gmail.com', 57000, 'Project Manager'),
('Rohit', 'Verma', 'rohit.verma@gmail.com', 60000, 'Finance Analyst'),
('Neha', 'Sharma', 'neha.sharma@gmail.com', 48000, 'Business Analyst'),
('Priya', 'Singh', 'priya.singh@gmail.com', 62000, 'Team Lead'),
('Rahul', 'Yadav', 'rahul.yadav@gmail.com', 55000, 'Marketing Manager'),
('Sneha', 'Patel', 'sneha.patel@gmail.com', 51000, 'Data Analyst'),
('Vikas', 'Mehta', 'vikas.mehta@gmail.com', 58000, 'Operations Manager'),
('Deepak', 'Gupta', 'deepak.gupta@gmail.com', 49000, 'Customer Support Executive');
```

id	email	first_name	last_name	department	salary
1	kunalkumar@gmail.com	Kunal	Kumar	Software	8000
2	nishantkumar@gmail.com	Nishant	Kumar	HR Executive	53000
3	ankitthakur88@gmail.com	Ankit	Thakur	Project Manager	57000
5	neha.sharma@gmail.com	Neha	Sharma	Business Analyst	48000
6	priya.singh@gmail.com	Priya	Singh	Team Lead	62000
7	rahul.yadav@gmail.com	Rahul	Yadav	Marketing Manager	55000
12	rohitkhan123@gmail.com	Rohit	Khan	IT	60000
13	dushyantsing@gmail.com	Dushyant	Singh	IT	50000
14	harsh82@gmail.com	Harsh	Kumar	IT	90000
15	jhgdjhdb@gmail.com	rohit	sharma	IT	6000
NULL	NULL	NULL	NULL	NULL	NULL

```
#  Time      Action                                 Message               Duration / Fetch
1  00:40:06  use ems                                0 row(s) affected     0.000 sec
2  00:40:09  select *from employees LIMIT 0, 1000   10 row(s) returned    0.015 sec / 0.000 sec
```

Fig 6.In this employee list added to database (my sql).

5.3Reportgenerationvalidation

Validation focuses on accuracy, completeness, and usability of generated reports. Department-wise counts are cross-checked against filtered lists to ensure consistency. Tenure analysis is validated by recalculating durations from joining dates and comparing results with displayed aggregates. Boundary cases—such as empty departments, recent joinees, or inactive employees—are included to confirm robust behavior. Filters applied to reports are tested for correct combinations and predictable results. Performance is assessed by measuring response times for typical data volumes. Export options, if enabled, are reviewed for correct encoding, column alignment, and date formats. The outcome confirms the reliability and decision-readiness of the reporting component.

6.Conclusion

The EMS fulfills its core objective of streamlining HR operations by centralizing records, enforcing validation, and providing secure, role-based access to sensitive data. Its layered architecture supports maintainability and future growth, while the technology choices align with industry standards for scalability and developer productivity. The system improves daily workflows for HR personnel, reduces manual errors, and offers timely insights through reporting. Looking ahead, the application can incorporate attendance tracking, leave workflows, payroll integration, and richer analytics dashboards. With modular design and RESTful endpoints, the EMS is well positioned to integrate with third-party services and evolve alongside organizational needs.

6.1Achivements

This project successfully delivers a production-ready foundation that supports secure authentication, structured employee management, and clear separation of concerns across layers. The team implemented meaningful validation rules, reduced duplication risks, and provided reliable search and filtering capabilities. The design anticipates growth by accommodating new modules without refactoring core components. User experience considerations—such as clean forms, feedback messages, and consistent views—make the system approachable for non-technical users. The codebase adheres to best practices, easing onboarding for future contributors. Collectively, these achievements demonstrate a practical solution that meets real-world HR requirements while showcasing solid engineering discipline and teamwork.

6.2Limitations

While the EMS covers essential HR needs, it does not yet implement attendance tracking, leave approvals, or payroll processing in the core build. Email notifications, audit trails, and fine-grained field-level permissions may be limited or pending depending on configuration. Advanced analytics and charting are not native to the current interface, and exports may be restricted to basic formats. The application targets small to medium data volumes; performance tuning for very large datasets would require indexing strategies and caching. Multi-tenant support and single sign-on are not included by default. These limitations are conscious trade-offs to prioritize core functionality and deliver a stable, extensible baseline.

6.3Future Enhancements (attendance, analytics, email notifications)

Planned enhancements include an attendance module with integration points for biometric devices or authenticated check-ins, plus a leave management workflow with request, approval, and balance tracking. A payroll module could automate salary computation, deductions, and payslip generation. Analytics dashboards with visualizations would surface trends such as attrition, hiring velocity, and departmental composition. Email and in-app notifications could announce joining anniversaries, policy updates, or pending approvals. Data export in Excel and CSV formats, along with API endpoints, would support external reporting and integration. Role-based field permissions, audit logs, and SSO integration would strengthen governance and enterprise adoption.

7.References

- Spring Boot Official Documentation — https://spring.io/projects/spring-boot
- Oracle Java SE Documentation — https://docs.oracle.com/javase
- TutorialsPoint — Spring Boot and Thymeleaf Tutorials
- GitHubRepository:Ghost-Ak24/SPRING-BOOT