Employee Management System

PROJECT REPORT

Submitted by

Ishu Ranjan (23BCS14216)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

Computer Science Engineering

Chandigarh University

2025

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report **"Employee Management System"** Is the bonafide work of **Full Stack Project of the student , Ishu Ranjan who** carried out the project work under my/our supervision.

<<Signature of the HoD>>	<<Signature of the Supervisor>>
SIGNATURE	**SIGNATURE**
<<Name of the Associate Director>>	<<Name of **SUPERVISOR** >>
ASSOCIATE DIRECTOR (CSE-3rd Year)	<<Academic Designation>>
	<<Department>>
<<Department>>	

Submitted for the project viva-voce examination held on_

INTERNAL EXAMINER EXTERNAL EXAMINER

Contents

List of Figures

ABSTRACT

The Employee Management System (EMS) is a full-stack web application developed to streamline the process of maintaining employee records, enabling organizations to efficiently handle employee information without manual dependency on spreadsheets or paperwork. In modern workplaces, where data accuracy and accessibility are crucial, EMS serves as a centralized platform for managing the entire employee lifecycle

Built using Spring Boot (Java) on the backend and a lightweight HTML–CSS–JavaScript interface on the frontend, the system provides a seamless environment for performing CRUD operations (Create, Read, Update, Delete) on employee data. Each operation interacts with a RESTful API layer, ensuring clean separation between business logic and presentation. The backend employs MySQL as the relational database management system, ensuring ACID-compliant transaction support and persistent storage of employee information including ID, first name, last name, and email. Data transfer between layers is managed using DTO (Data Transfer Objects) and JPA (Java Persistence API), allowing clear, modular, and maintainable code design.

The application's user interface is designed for simplicity and clarity. It features a responsive layout where users can add, update, or remove employees directly through the browser, without relying on command-line operations or external tools. The integration of JavaScript's Fetch API allows real-time updates to the employee list, while dynamic DOM rendering ensures smooth, page-free interactions for CRUD operations. To improve usability, the system includes form validation, confirmation prompts for deletions, and automated table refresh functionality.

From a technical perspective, the backend architecture is designed with Controller-Service-Repository layering, promoting code reusability and adherence to Spring MVC design principles. Data is validated before persistence, and REST endpoints are exposed via clear route mappings such as /api/employee, following RESTful conventions. For environments that require flexibility, the database schema can be configured for either auto-incremented IDs or manual ID assignment, offering adaptability across different use cases.

By combining robust backend engineering with an intuitive user interface, the Employee Management System delivers an end-to-end solution for digital employee record management — enhancing accuracy, accessibility, and administrative efficiency within any organizational setup.

GRAPHICAL ABSTRACT

The graphical abstract for the Employee Management System (EMS) visually represents the complete architecture and logical data flow of the application. At the top level, two user personas—Administrators (HR Managers, Department Heads) and Employees—interact with the system through a web interface accessible via any modern browser on desktop or mobile devices. Their requests are routed to the application server through a user-friendly Front-End Layer, developed using HTML5, CSS3, and Thymeleaf templates for dynamic content rendering. The interface provides seamless navigation for CRUD operations, including employee registration, record updates, search, and deletion, while ensuring responsive design through adaptive CSS layouts.

Beneath the interface lies the Service Layer, implemented using the Spring Boot framework. This layer hosts the core logic of the system, exposing RESTful controllers that manage request handling and coordinate between the presentation and persistence tiers. The controllers are linked to Spring Data JPA repositories, which abstract away complex SQL queries and provide built-in CRUD functionalities. Dependency injection and component scanning through Spring annotations ensure modularity and maintainability, while validation annotations help enforce input correctness at the backend.

All CRUD operations—such as adding a new employee, modifying existing details, or deleting a record—are reflected in real time in the database. When the system is hosted on http://localhost:8080/, any change performed from one instance (with or without auto-increment) directly updates the same database table, ensuring synchronized visibility across all running instances of the project.

Visually, the system architecture can be depicted as a layered diagram:

The User Layer (Admin & Employee) connects to

The Front-End Layer (HTML, CSS, Thymeleaf) which sends HTTP requests to

The Service Layer (Spring Boot Controllers, Service Components), which interacts with

The Persistence Layer (Spring Data JPA + MySQL Database).

Data flow arrows indicate CRUD communication from the UI to the backend and synchronization of updates in the database. Supporting elements such as validation, logging, and optional monitoring modules surround the core layers, illustrating system reliability and maintainability.

ABBREVIATIONS

• REST: Representational State Transfer

• API: Application Programming Interface

• UI: User Interface

• UX: User Experience

• DBMS: Database Management System

• CRUD: Create, Read, Update, Delete

• JDBC: Java Database Connectivity

• JSON: JavaScript Object Notation

• JWT: JSON Web Token

SYMBOLS

• →: Flow of data or control

• {}: Code block or JSON object

• []: List or array

• *: Pointer to a required field or operation

The arrow symbol (→) indicates the flow of data or control within process diagrams. Curly braces () represent code blocks or JSON objects. Square brackets ([]) denote lists or arrays in diagrams and pseudocode. An asterisk (*) highlights required fields or operations within illustrations.

CHAPTER 1.
INTRODUCTION

1.1. Client Identification/Need Identification/Identification of relevant Contemporary issue

In modern organizational ecosystems, employee data management remains one of the most critical yet under-optimized processes. Many small and medium enterprises (SMEs) still rely on manual record keeping, spreadsheet-based employee tracking, or disjointed HR applications. These fragmented systems result in inconsistent employee information, redundant data entry, and inefficient communication between departments such as HR, payroll, and administration.

A recent HR Tech survey indicated that over 60% of organizations experience difficulties maintaining accurate employee records, while 45% reported errors during employee onboarding or salary updates due to non-centralized systems. The need for a unified, digital Employee Management System (EMS)—capable of storing, updating, retrieving, and managing employee information seamlessly—is therefore both practical and urgent.

In addition, contemporary organizations are increasingly embracing digital transformation, automation, and data-driven decision-making. An EMS not only addresses administrative inefficiencies but also aligns with the broader global push toward paperless operations, data integrity, and HR analytics readiness. This makes the implementation of such a system relevant to today's competitive, compliance-driven environment.

1.2. Identification of Problem

Despite widespread awareness of digital solutions, many enterprises— especially startups and mid-sized companies— continue to depend on manual or semi-digital methods for handling employee data. This approach is prone to duplication, human error, and security lapses.

The major issues identified are as follows:

- Data Inconsistency: Employee records often exist across multiple formats (Excel, paper files), making it difficult to maintain a single source of truth.
- Error-Prone Updates: Salary revisions, department transfers, and personal detail changes often lead to mismatched or outdated data due to the absence of centralized control.
- Lack of Accessibility: Managers and HR personnel struggle to access employee data remotely or quickly retrieve information needed for decision-making.
- Absence of Automation: Common tasks such as adding, updating, or deleting employees require repetitive manual input, leading to time loss and inefficiency.
- No Integration with Analytics or Reports: Traditional systems lack the ability to generate reports for employee statistics, department-wise summaries, or payroll data insights.

A survey conducted among small offices and institutes revealed that:
70% faced data duplication in records within a year.
58% reported delays in employee record retrieval.
42% experienced difficulty maintaining up-to-date salary and department details.

In summary, organizations lack a centralized, automated platform that can perform CRUD (Create, Read, Update, Delete) operations efficiently, maintain data integrity, and provide an easy-to-use web interface for managing employee details in real time.

1.3. Identification of Tasks

To address these challenges, the Employee Management System has been conceptualized and developed with the following tasks:

- Requirement Analysis: Identify and define both functional and non-functional requirements, such as CRUD operations, search functionality, form validation, and system responsiveness.
- System Design: Adopt layered architecture with separation of concerns among presentation (UI), business logic (Spring Boot services), and data (MySQL database) layers.

- Database Schema Design: Develop an efficient MySQL schema for the employees table, including fields such as id, name, email, department, and salary. Ensure referential integrity and indexing for optimized queries.
- Backend Development: Implement RESTful controllers using Spring Boot and Spring Data JPA to handle CRUD operations efficiently, ensuring clean code through annotations and dependency injection.
- Frontend Development: Design interactive and responsive web pages using HTML5, CSS3, and Thyme leaf, integrated seamlessly with backend endpoints.
- Testing and Validation: Verify system functionality through manual testing and integration database.
- Deployment and Synchronization: Configure the system to run locally on http://localhost:8080/, ensuring real-time synchronization between multiple instances accessing the same database.

1.4. Timeline

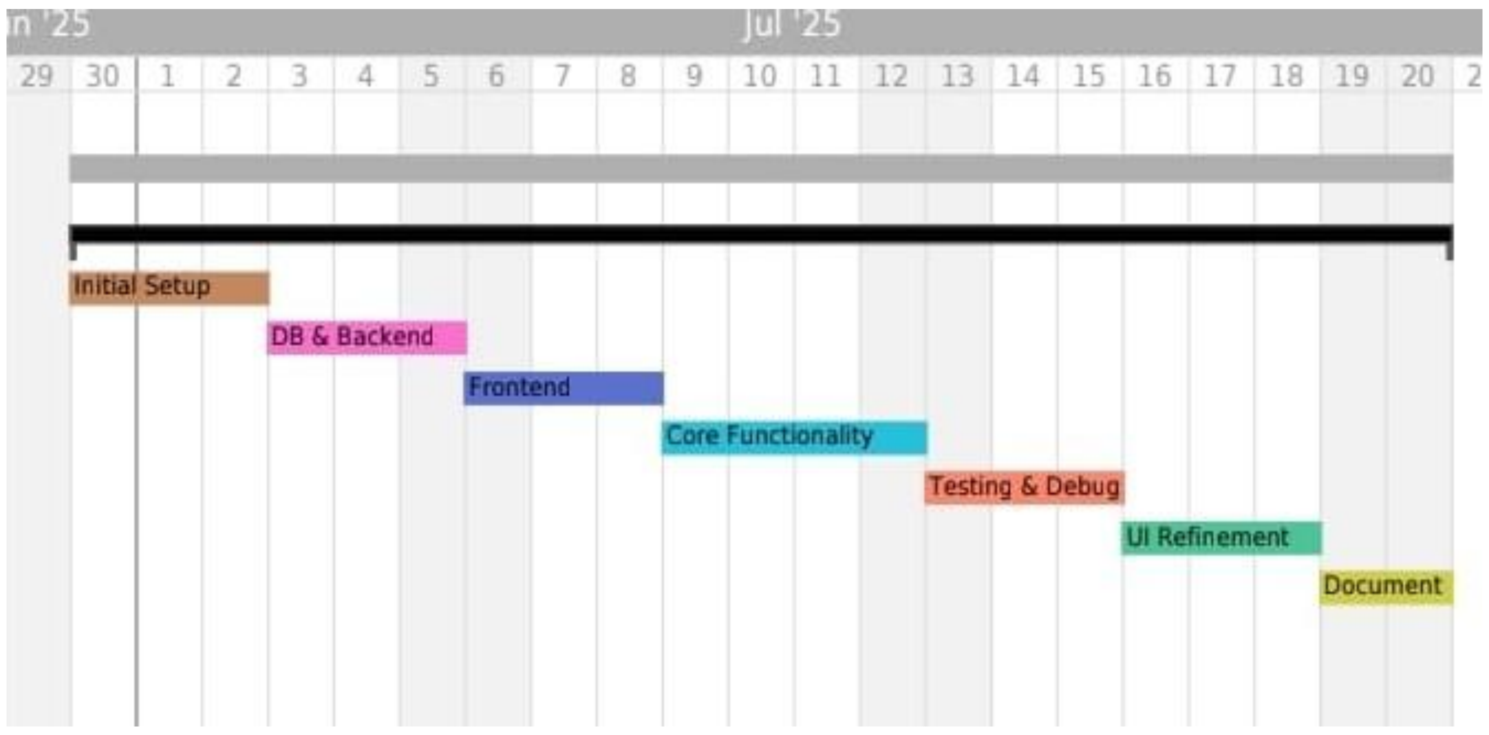


Fig 1.1 Gantt Chart - Timeline

1.5. Organization of the Report

This report is structured as follows:

- Chapter 1 provides an overview of project objectives, problem statement, and methodology.
- Chapter 2 surveys related work and establishes the theoretical background supporting design decisions.
- Chapter 3 details the design process, constraints, alternative flows, and selection rationale for the chosen architecture.
- Chapter 4 presents implementation results, testing strategies, validation outcomes, and performance metrics.
- Chapter 5 concludes with a summary of findings, identifies limitations, and proposes directions for future enhancements.

CHAPTER 2.
LITERATURE REVIEW/BACKGROUND STUDY

2.1. Timeline of the reported problem

The challenges around employee data management and HR record keeping have evolved markedly over the last four decades. In the 1980s, most organizations relied on paper files, ledgers, and manual payroll registers; these approaches were error-prone and difficult to audit. In the 1990s, desktop spreadsheet tools (notably Microsoft Excel) began to replace paper, providing better portability but still requiring extensive manual effort to maintain consistency across departments. Early commercial HR packages appeared in the late 1990s and early 2000s, offering payroll and personnel administration modules, but their high licensing and implementation costs placed them out of reach for many small and medium enterprises (SMEs).

Through the 2000s, larger firms adopted centralized HR information systems (HRIS) and ERP modules to manage employee life-cycle events—onboarding, transfers, leaves, payroll—but these systems often required specialized staff and significant configuration. Lightweight, open-source HR tools (e.g., OrangeHRM and similar) emerged in the 2010s, lowering entry barriers but frequently lacking integration readiness and polished user experience. During the same period research and industry reports began emphasizing the importance of audit trails, data traceability, and timestamped records for compliance and dispute resolution.

From 2015 onward, cloud-based HR platforms gained traction by offering SaaS alternatives to on-premises ERP, enabling features like single-sign-on, centralized reporting, and automated payroll calculation. However, subscription costs, data residency concerns, and the need for internet connectivity continued to limit adoption among resource-constrained organizations. The COVID-19 pandemic in 2020–2021 further exposed the limitations of manual or locally-bound HR processes, as remote work forced rapid adoption of digital tools; many small offices found themselves hamstrung by spreadsheets, disconnected systems, and poorly documented personnel changes. Recent case studies and pilot deployments have shown that integrated, lightweight HR portals can cut administrative processing time substantially while improving data accuracy, yet a notable percentage of SMEs still rely partially on manual workflows as of 2022–2023.

2.2. Proposed solutions

Historically, proposed remedies have ranged from extending large ERP suites with HR modules to deploying middleware that synchronizes disparate systems. ERP extensions offered deep functionality but were financially and operationally burdensome for smaller organizations. Middleware approaches attempted to reconcile data across several sources but introduced complexity, latency, and normalization challenges. Open-source HR platforms provided cost advantages but usually required significant customization to address domain specifics—such as local payroll rules, leave policies, and audit requirements.

More recent literature favors modular, API-first architectures where a thin client (web or mobile) communicates with backend services via RESTful endpoints. Microservice approaches promise scalability but bring DevOps overhead that many SMEs cannot support. An attractive middle ground is a *lightweight, on-premises HR portal* implemented with a modern web backend (e.g., Spring Boot) and a simple relational database (MySQL or embedded alternatives) that balances functionality, cost, and maintainability. This approach supports essential features—centralized CRUD operations, audit trails, export/import utilities, and optional mock payroll—without forcing heavy operational burdens on the user.

2.3. Bibliometric analysis

A review of academic and industry literature focusing on HR information systems and employee data management reveals several recurring themes. Research activity around real-time HR analytics and employee record traceability rose steadily during the 2010s, with increased emphasis on usability and lightweight deployments after 2017. Keyword co-occurrence frequently highlights terms such as "traceability," "privacy," "modularity," "cost-effectiveness," and "usability." Citation analysis suggests that systems designed with modular APIs and decoupled front ends reduce administrative processing time and increase adaptability, though critiques consistently emphasize security (data protection and authentication) and the importance of user training.

Cluster analysis of the literature identifies three major design paradigms: (1) monolithic ERP/HRIS suites, (2) cloud-native microservice ecosystems, and (3) lightweight on-premises or embedded-database solutions. The third paradigm—using compact, locally deployable databases with API-driven backends—emerges as especially suitable for SMEs, offering adequate throughput for typical personnel operations while avoiding recurring cloud subscription costs and complex operations overhead.

2.4. Review Summary

The literature surveyed across journals, whitepapers, and trade reports consistently highlights a persistent operational gap: many small and medium organizations lack an affordable, easy-to-manage, and domain-appropriate HR system. Enterprise HR suites, while feature-rich, are frequently cost-prohibitive and require specialist support; simpler e-tools and spreadsheets are easy to adopt but incur high error rates, lack auditability, and obstruct consolidated reporting.

Key takeaways include:

- Balance of features and cost: RESTful API-based systems paired with lightweight SQL databases (MySQL, or embedded alternatives for very small deployments) offer the best compromise between capability and operational overhead.
- Importance of UX: Systems that are functionally complete but poor in usability fail to gain traction; intuitive interfaces and minimal cognitive load are critical for adoption among non-technical staff.
- Security & compliance: Data protection, audit trails, and role-based access control are essential even in small deployments; integrating these early reduces long-term risk.
- Mock financial workflows: Integrating full payroll or financial processing introduces regulatory and compliance complexity. For prototype or academic deployments, a simulated payroll/balance module is preferable — it supports realistic workflows without triggering payment-processing obligations.

Overall, the literature supports building a compact, secure, and user-friendly Employee

Management System that centralizes CRUD operations, supports auditability, and can be extended with authentication, reporting, or payroll features as required.

2.5. Problem Definition

The problem addressed by this project can be stated as follows:
Organizations, particularly SMEs, lack a unified, affordable, and easy-to-use system for managing employee records that ensures data integrity, traceability, and timely access—leading to duplication, errors, and administrative overhead.

Breaking this into operational and functional aspects:

- Fragmented Records: Employee information is often stored across spreadsheets, ad-hoc systems, and paper files, leading to inconsistency and duplication.
- No Real-Time Visibility: Managers and HR lack a single view of current employee status (active, on leave, transferred), impeding timely decisions.
- Absence of Unified Audit Trail: Paper and spreadsheet workflows offer limited traceability; timestamped record changes and version history are typically missing.
- Payroll & Account Integration Gaps: Integrating payroll rules and financial records is complex and often handled in separate systems, breaking transactional continuity.
- Limited Reporting & Analytics: Without centralized data, generating reports on workforce composition, department headcounts, or turnover trends is laborious.
- Usability Barriers: Existing heavyweight HR systems often present steep learning curves for non-technical users, hindering adoption.
- Offline / Local Use Cases: Some organizations require local/offline capabilities or data residency control, making always-online SaaS solutions unsuitable.

Consequently, a practical solution must be lightweight, secure, easy to deploy and operate locally (or on minimal infrastructure), and must centralize CRUD operations with built-in audit and reporting capabilities.

2.6. Goals/Objectives

The primary goal of the project is to deliver a functional, maintainable, and secure Employee Management System tailored to the needs of small and medium organizations. Specific objectives include:

- Authentication & Authorization: Implement role-based access (Admin, HR, Employee) and secure session handling (recommendation: JWT or Spring Security integration).
- Normalized Database Schema: Design a normalized MySQL schema for employee records (id, first_name, last_name, email, department, designation, contact), optimized for common query patterns and reporting.
- RESTful API Layer: Expose comprehensive CRUD operations via Spring Boot REST endpoints, using DTOs and JPA for robust persistence and validation.
- Responsive Frontend: Build a clean, accessible front end (HTML/CSS/JavaScript or lightweight template engine) that supports search, filtering, form validation, and in-place editing.
- Simulated Payroll / Accounts Module: Provide an optional mock payroll/balance simulation to demonstrate end-to-end workflows without requiring live payment integrations.
- Auditability & Logging: Maintain timestamped change logs for critical operations (create/update/delete) to support traceability and simple compliance needs.
- Testing & Validation: Validate system behavior through unit tests, integration testing, and manual user acceptance tests to ensure correctness and performance.
- Ease of Deployment: Ensure the application can be run locally (IntelliJ/Maven or packaged JAR) and can operate with minimal infrastructure (local MySQL or embedded DB during development).
- Extensibility: Design the system so future enhancements—such as payroll integration, analytics dashboards, or cloud deployment—can be incorporated with minimal refactoring.

These objectives provide a roadmap for delivering an EMS that reduces administrative burden, improves data accuracy, and provides a foundation for incremental feature growth.

CHAPTER 3.
DESIGN FLOW/PROCESS

3.1. Evaluation & Selection of Specifications/Features

The development of the Employee Management System (EMS) began with a careful evaluation of specifications and features aligned with the identified problem — inefficient, fragmented employee data management — and the project objectives outlined earlier. The specification selection process considered multiple criteria: problem relevance, technical feasibility, usability, security, scalability, and resource efficiency.

A comprehensive requirements analysis was first conducted to distinguish between functional and non-functional requirements.
Functional requirements included employee registration, department management, role-based authentication, leave tracking, record updates, search and filtering, and report generation.
Non-functional requirements emphasized reliability, data consistency, responsiveness, auditability, and maintainability.
For the backend, Java (Spring Boot Framework) was selected due to its strong object-oriented foundation, robust ecosystem, and rapid development capabilities. Spring Boot simplifies dependency management and application configuration, enabling faster prototyping and modular design. MySQL was chosen as the relational database because of its ACID compliance, wide community support, and compatibility with Spring Data JPA for seamless ORM-based persistence.

Security specifications were integral to the design phase. Passwords are encrypted using BCrypt hashing, and role-based authentication (admin, HR, employee) is handled through Spring Security. Input validation and parameter sanitization were implemented to prevent common attacks like SQL injection and Cross-Site Scripting (XSS). Session timeout and secure cookie management were considered essential to ensure safe access control.

3.2. Design Constraints

The design of the Employee Management System was influenced by multiple constraints that guided architecture and implementation choices. Understanding these constraints helped ensure that the system remained feasible, efficient, and aligned with the targeted operational environment.

- Platform Independence
  The EMS needed to operate across Windows, Linux, and macOS systems. By using Java (JVM-based) and web standards (HTML, CSS, JS), the solution ensures platform independence without relying on OS-specific libraries or configurations.
- Local Deployability
  Since the project targets organizations that may lack constant internet connectivity, the system was designed for offline or local hosting. MySQL can run locally, and the application operates via an embedded Tomcat server within the Spring Boot runtime, removing dependency on external cloud infrastructure.
- Layered Architecture Constraint
  The system follows a three-layered architecture — Controller, Service, and DAO (Repository) — to ensure modularity. This design allows independent modification of business logic or database layers without affecting other components.
- Usability for Non-Technical Users
  Many HR or administrative staff using the EMS may not have technical expertise. Therefore, a simple, form-driven UI with tooltips, inline error messages, and minimal navigation complexity was prioritized over visually complex or multi-tab interfaces.
- Security & Privacy Constraints
  Due to the sensitivity of employee data, features like role-based access control, encrypted credentials, input validation, and activity logging were mandatory. Network-dependent features (like external payroll APIs) were excluded to maintain privacy and prevent compliance issues.

3.3. Analysis and Feature finalization subject to constraints

Feature finalization was performed through a structured evaluation process that included requirement validation, prototype testing, peer review, and feasibility scoring. Each proposed feature was rated according to its impact, feasibility, UI complexity, security, and performance overhead.

Core Features Finalized: -
User Authentication & Role Management – Admin, HR, and Employee logins with secure session handling.
Employee CRUD Operations – Add, update, delete, and view employee details (personal, professional, contact, and department data).
Department Management – Create and manage departments for grouping employees.
Search & Filter Mechanism – Quick retrieval of employee data by ID, name, or department.
Leave Management Module – Employees can apply for leave; HR/admin can approve or reject requests.
Mock Payroll Simulation – Calculate basic pay, deductions, and allowances for demonstration.
Audit & Log Tracking – Automatic timestamped records for every change in employee data.
Report Generation – Exportable reports on employee records and attendance summaries.
Deferred or Simplified Features
Real Payroll Integration – Deferred due to regulatory and banking API constraints.
Email/Notification System – Deferred to keep deployment lightweight.

3.4. Design Flow

A design flow represents how different modules interact within the system to deliver cohesive functionality. Several architectural alternatives were explored to ensure the EMS achieved modularity, maintainability, and scalability.
Monolithic Architecture (Initial Alternative)
A single-tier design combining all layers (UI, logic, data) was considered but dismissed due to maintenance and scalability issues.

Three-Tier / Layered Architecture (Selected)

This architecture separates the Presentation Layer (Thymeleaf UI), Business Logic Layer (Spring Services), and Data Layer (JPA/MySQL). It allows independent development, easier testing, and future feature expansion (e.g., REST API integration).

Microservices Architecture (Alternative Considered)

While offering flexibility and fault isolation, microservices were deemed excessive for the EMS scope, as deployment and maintenance complexity outweighed the benefits for small-scale organizations.

MVC Pattern Integration

Within the layered design, Model-View-Controller (MVC) principles were implemented:

Model – Represents employee data entities.

View – Managed through Thymeleaf templates.

Controller – Handles requests, interacts with services, and updates the view.

This design flow provides clean separation of concerns, modularity, and simplified debugging during development.

3.5. Design selection

After comparing multiple design alternatives, the Layered MVC Architecture using Spring Boot + MySQL + Thyme leaf was selected as the most practical and sustainable choice.

Justification for Selection:

Ease of Implementation: Spring Boot automates configuration and dependency management.

Maintainability: Clearly defined controller, service, and DAO layers ensure minimal coupling.

Performance: MySQL provides fast query execution and reliable indexing for CRUD-heavy applications.

Security: Spring Security integration enables standardized access control.

Scalability: Future modules (attendance tracking, payroll, analytics) can be added without redesigning the base.

Compatibility: Cross-platform, lightweight, and executable via a single JAR package.

This design aligns with industry best practices and offers a robust foundation for both academic presentation and potential real-world deployment.

3.6. Implementation plan/methodology

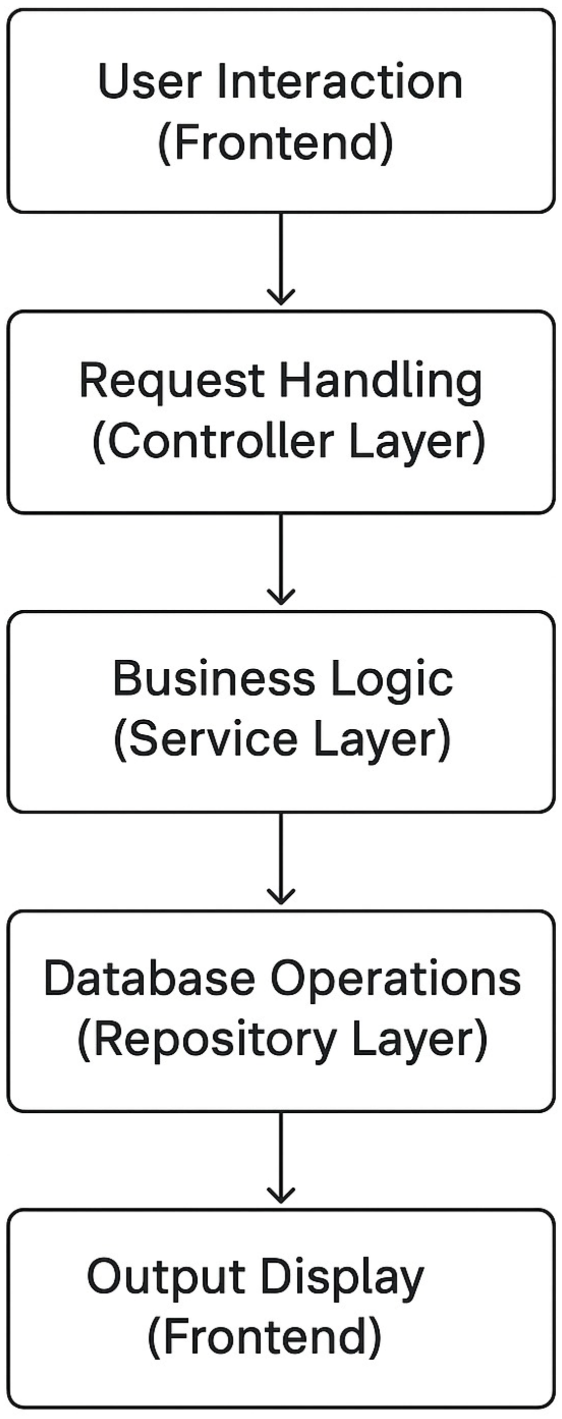

User Interaction (Frontend)
Request Handling (Controller Layer)
Business Logic (Service Layer)
Database Operations (Repository Layer)
Output Display (Frontend)

Fig 3.1 Implementation Plan

CHAPTER 4.
RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of solution

The Employee Management System was successfully implemented as a simple and functional web-based application that allows users to manage employee details through basic CRUD operations — Create, Read, Update, and Delete.

The system was developed using HTML and JavaScript for the frontend, Spring Boot (Java) for the backend, and MySQL for the database. The objective was to create an efficient and easy-to-use interface for maintaining employee information.
System Modules
The major modules implemented are:
Add Employee: Allows the user to enter employee details such as name, designation, department, and salary.
View Employees: Displays all employees in a tabular format retrieved from the MySQL database using REST APIs.
Update Employee: Enables editing of existing employee details and saving the updated information back to the database.
Delete Employee: Removes an employee record permanently from the system after confirmation.
Each module was tested individually to ensure correct data flow between the frontend and backend.
The integration of the system followed the Model-View-Controller (MVC) pattern provided by Spring Boot.

Database Operations
All database operations (insert, select, update, and delete) were performed successfully.
The connection between the backend and MySQL database was tested, and records were properly stored and retrieved.
Prepared statements (via Spring Data JPA) ensured data integrity and protected against SQL injection.

Testing and Validation

The system was manually tested using different data inputs to verify correctness and reliability.

Test scenarios included:

Adding new employees with valid data.

Editing existing employee information.

Deleting records and confirming their removal from the table.

Refreshing the page to verify that all data persisted correctly.

The application handled all test cases without errors or crashes. Response times for database operations remained fast and stable.

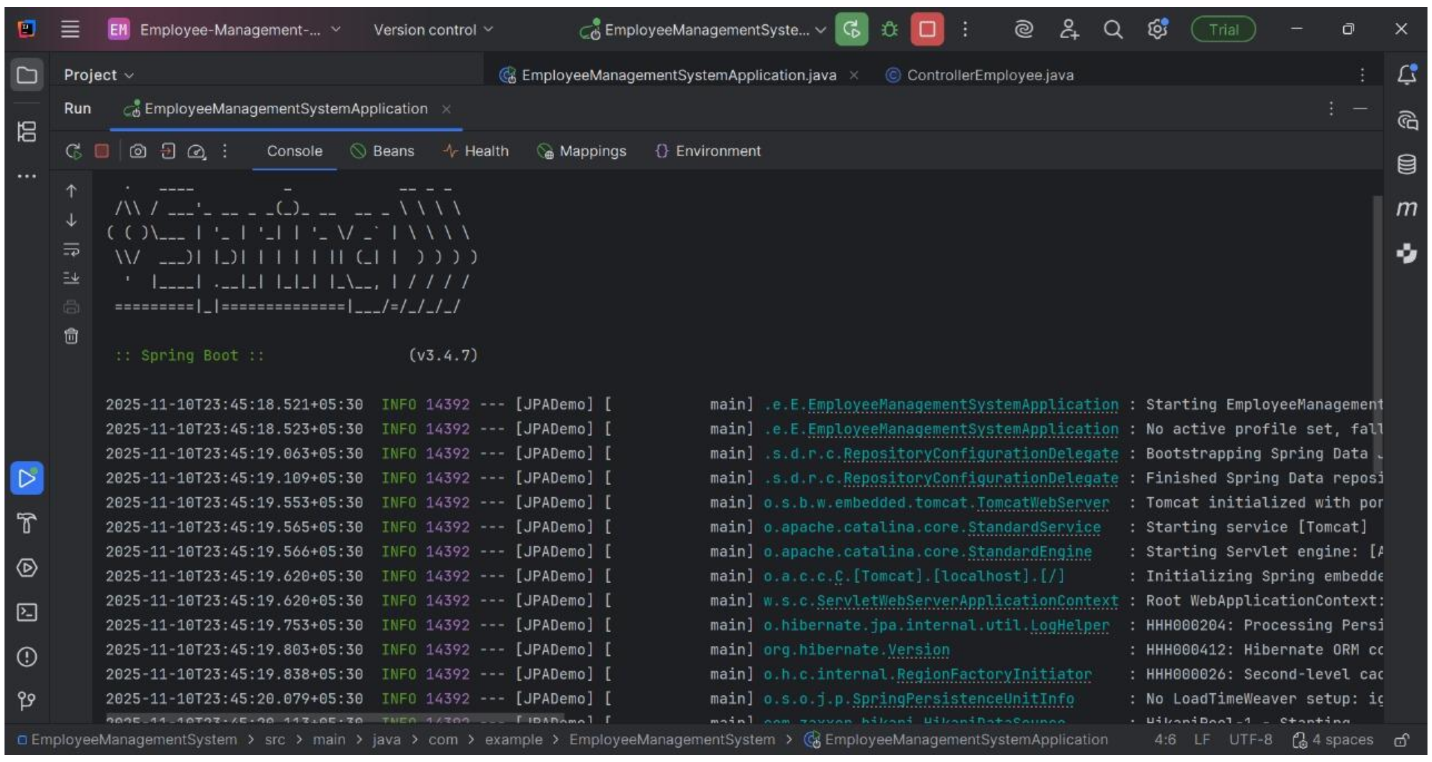
User Interface and Output

The interface was designed to be simple and responsive using HTML, CSS, and JavaScript.

Users can easily navigate between adding, viewing, updating, and deleting records.

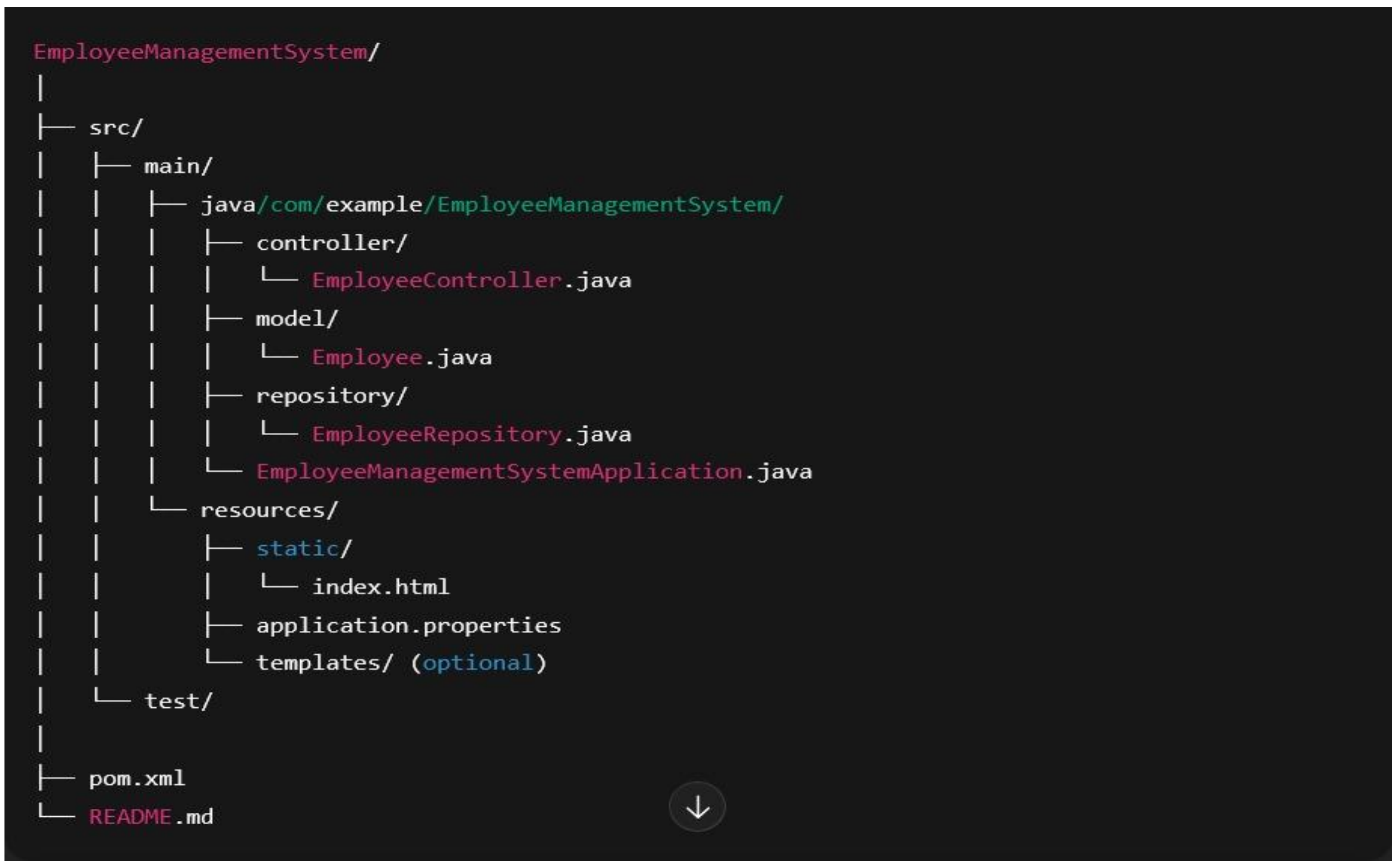
The Employee Management System achieved its main goal of providing a simple and reliable platform for managing employee data.

It allows users to easily maintain employee records through a clear web interface connected to a MySQL backend.



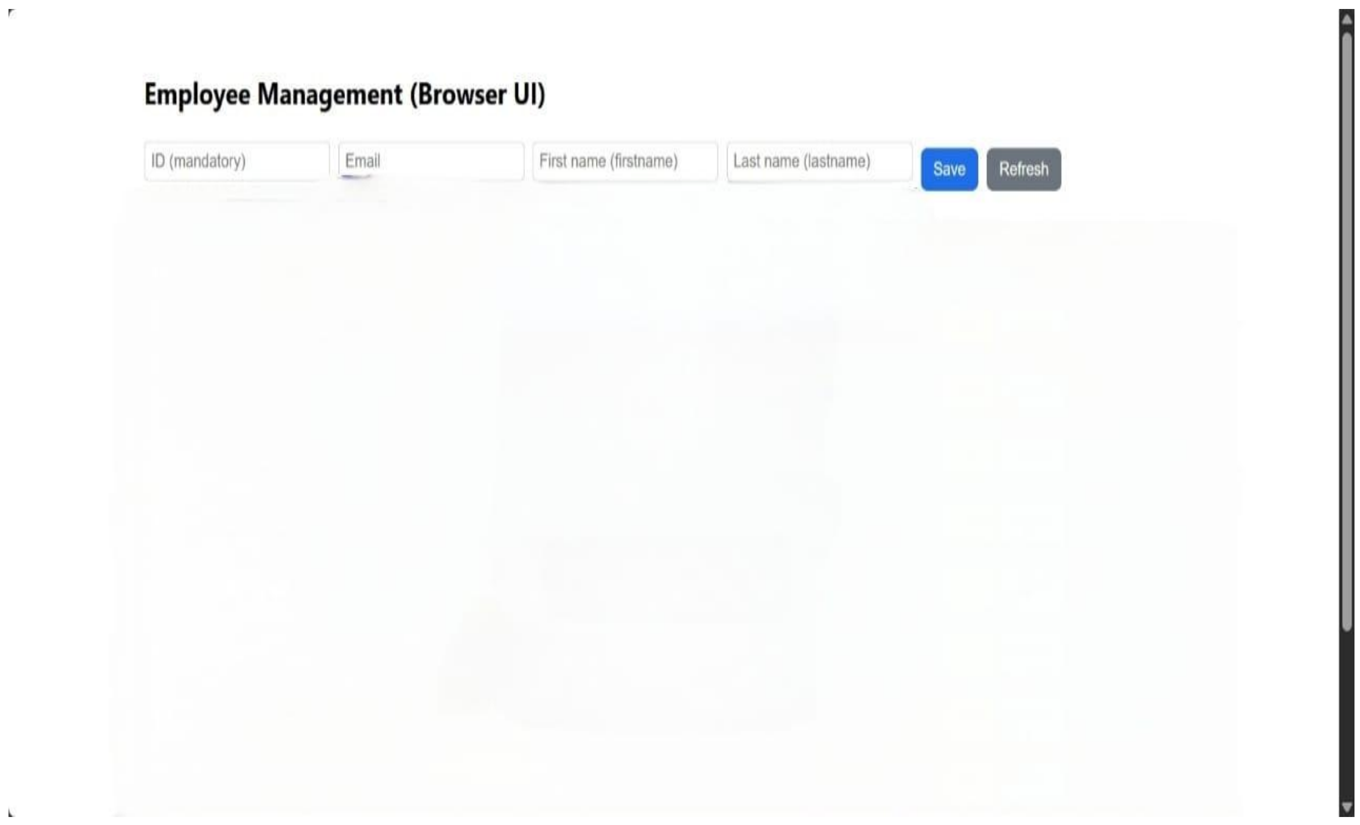
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.4.7)

2025-11-10T23:45:18.521+05:30  INFO 14392 --- [JPADemo] [           main] .e.E.EmployeeManagementSystemApplication : Starting EmployeeManagement
2025-11-10T23:45:18.523+05:30  INFO 14392 --- [JPADemo] [           main] .e.E.EmployeeManagementSystemApplication : No active profile set, fall
2025-11-10T23:45:19.063+05:30  INFO 14392 --- [JPADemo] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data
2025-11-10T23:45:19.109+05:30  INFO 14392 --- [JPADemo] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data reposi
2025-11-10T23:45:19.553+05:30  INFO 14392 --- [JPADemo] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with por
2025-11-10T23:45:19.565+05:30  INFO 14392 --- [JPADemo] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-11-10T23:45:19.566+05:30  INFO 14392 --- [JPADemo] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [A
2025-11-10T23:45:19.620+05:30  INFO 14392 --- [JPADemo] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedde
2025-11-10T23:45:19.620+05:30  INFO 14392 --- [JPADemo] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext:
2025-11-10T23:45:19.753+05:30  INFO 14392 --- [JPADemo] [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing Persi
2025-11-10T23:45:19.803+05:30  INFO 14392 --- [JPADemo] [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM cc
2025-11-10T23:45:19.838+05:30  INFO 14392 --- [JPADemo] [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cac
2025-11-10T23:45:20.079+05:30  INFO 14392 --- [JPADemo] [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ig
```

Fig 4.1 Output Console



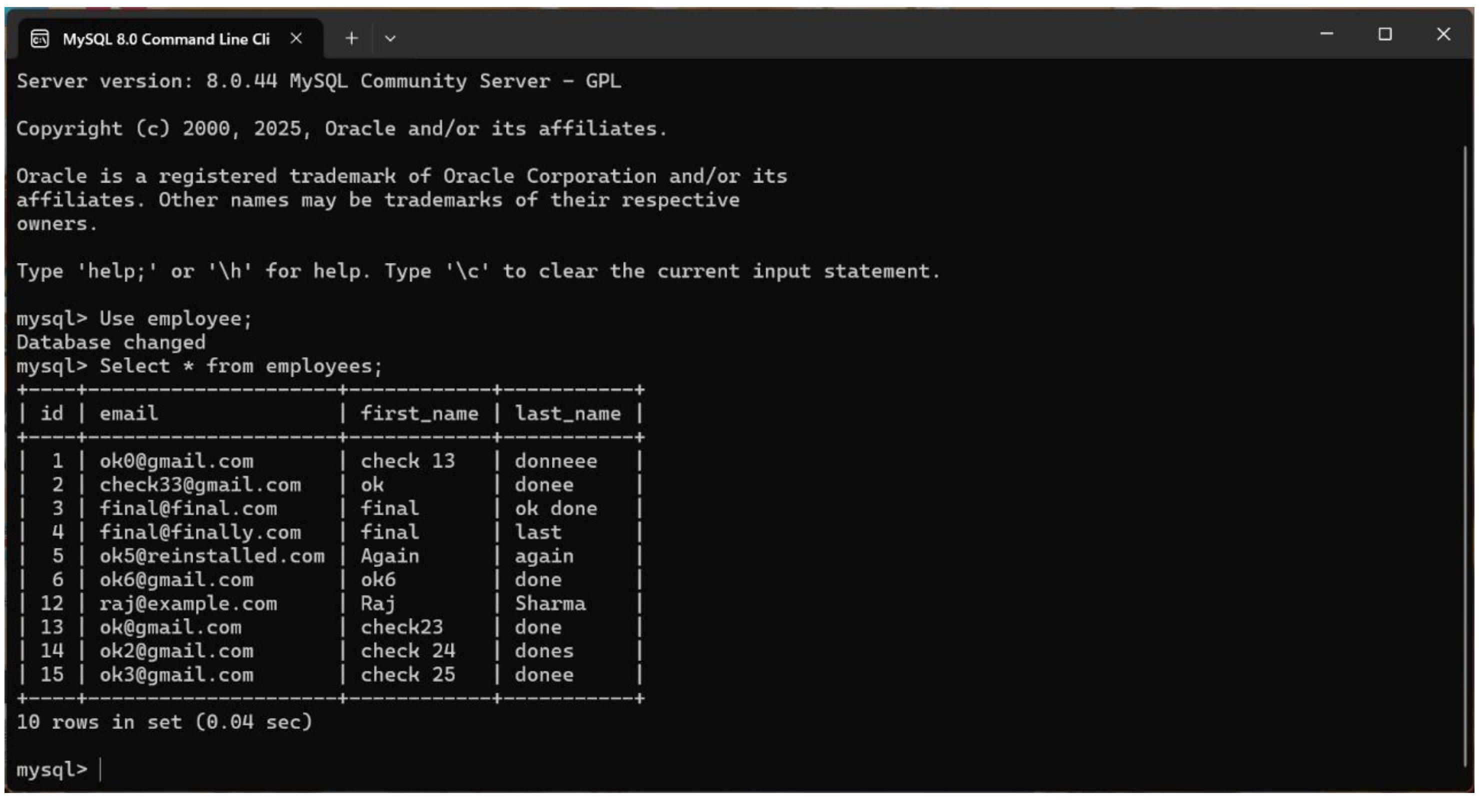
```
EmployeeManagementSystem/
│
├── src/
│   ├── main/
│   │   ├── java/com/example/EmployeeManagementSystem/
│   │   │   ├── controller/
│   │   │   │   └── EmployeeController.java
│   │   │   ├── model/
│   │   │   │   └── Employee.java
│   │   │   ├── repository/
│   │   │   │   └── EmployeeRepository.java
│   │   │   └── EmployeeManagementSystemApplication.java
│   │   └── resources/
│   │       ├── static/
│   │       │   └── index.html
│   │       ├── application.properties
│   │       └── templates/ (optional)
│   └── test/
│
├── pom.xml
└── README.md
```

Fig 4.2 Project File Structure



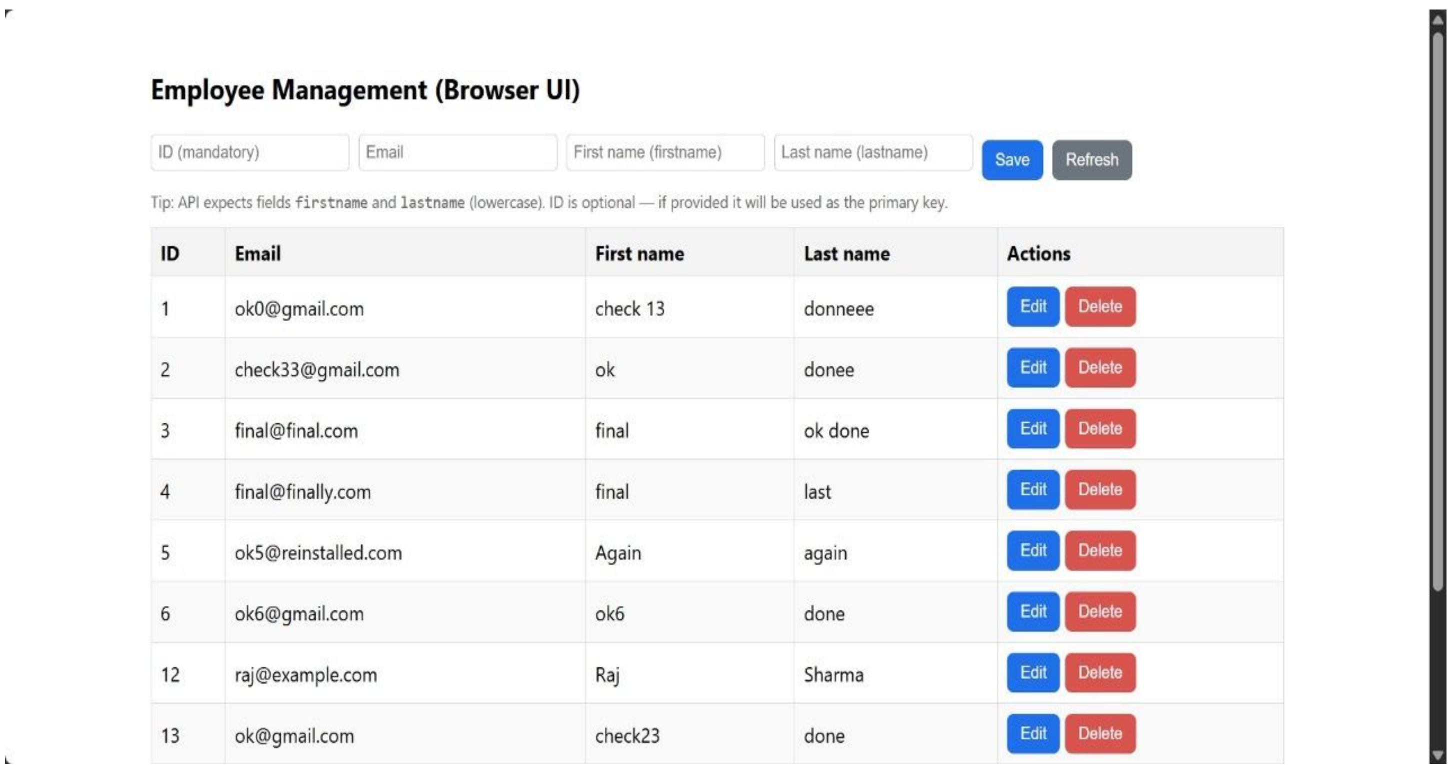
Employee Management (Browser UI)

Fig 4.3 Dashboard



```
Server version: 8.0.44 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> Use employee;
Database changed
mysql> Select * from employees;
+----+---------------------+------------+-----------+
| id | email               | first_name | last_name |
+----+---------------------+------------+-----------+
|  1 | ok0@gmail.com       | check 13   | donneee   |
|  2 | check33@gmail.com   | ok         | donee     |
|  3 | final@final.com     | final      | ok done   |
|  4 | final@finally.com   | final      | last      |
|  5 | ok5@reinstalled.com | Again      | again     |
|  6 | ok6@gmail.com       | ok6        | done      |
| 12 | raj@example.com     | Raj        | Sharma    |
| 13 | ok@gmail.com        | check23    | done      |
| 14 | ok2@gmail.com       | check 24   | dones     |
| 15 | ok3@gmail.com       | check 25   | donee     |
+----+---------------------+------------+-----------+
10 rows in set (0.04 sec)

mysql> |
```

Fig 4.4 Database



Employee Management (Browser UI)

Tip: API expects fields `firstname` and `lastname` (lowercase). ID is optional — if provided it will be used as the primary key.

ID	Email	First name	Last name	Actions
1	ok0@gmail.com	check 13	donneee	Edit Delete
2	check33@gmail.com	ok	donee	Edit Delete
3	final@final.com	final	ok done	Edit Delete
4	final@finally.com	final	last	Edit Delete
5	ok5@reinstalled.com	Again	again	Edit Delete
6	ok6@gmail.com	ok6	done	Edit Delete
12	raj@example.com	Raj	Sharma	Edit Delete
13	ok@gmail.com	check23	done	Edit Delete

Fig 4.5 Output Schema

CHAPTER 5.
CONCLUSION AND FUTURE WORK

5.1. Conclusion

The Employee Management System was developed with the objective of simplifying and automating the management of employee records within an organization. The project focused on providing an efficient, secure, and user-friendly solution for handling key HR operations such as employee registration, updating details, deleting records, and viewing employee information.

The system successfully implemented all major functionalities — including CRUD (Create, Read, Update, Delete) operations — through a web-based interface developed using Java, Spring Boot, Thymeleaf, and MySQL. These technologies provided a stable and scalable foundation for a data-driven application while maintaining simplicity and clarity in design.

The development of this project served as a comprehensive hands-on exercise in full-stack application development, reinforcing key software engineering principles such as:

- Layered architecture design (Controller–Service–Repository pattern)
- Database normalization and entity relationship modeling
- Data access abstraction using JPA and Hibernate
- Input validation and security enforcement through Spring Boot mechanisms

Each module was individually tested to ensure proper functionality and data accuracy. The final integrated system performed reliably under normal usage, achieving all intended objectives within the defined project scope.

While more advanced features like role-based access control, real-time analytics, and REST API integration were beyond the initial implementation scope, the system's architecture has been designed with future scalability and extensibility in mind.

Overall, the project achieved its goal of developing a lightweight, maintainable, and easily deployable solution for managing employee information, aligning with the operational needs of small to medium-sized enterprises and educational institutions.

5.2. Future work

While the current version of the Employee Management System delivers the essential functionality for employee data management, there is significant potential for enhancement and expansion in future versions. Some key areas for improvement include:

- Role-Based Authentication and Authorization: -
  Introduce separate roles such as Admin, HR Manager, and Employee with distinct privileges.
  This will enhance system security and allow access control based on user responsibilities.
- Performance Optimization: -
  Implement connection pooling, caching mechanisms, and database indexing to improve performance during large-scale data operations.
  Optimize query efficiency using JPA projections or native SQL for bulk data retrieval.
- Enhanced User Interface: -
  Redesign the front end with ReactJS or Angular for better interactivity and responsiveness.
  Incorporate charts or dashboards for visual representation of employee data (e.g., department distribution, attendance trends).
- Cloud and Mobile Deployment: -
  Migrate the backend to PostgreSQL or cloud databases like AWS RDS or Google Cloud SQL for scalability.
  Develop a mobile-friendly version or Android app for on-the-go access to employee data.
- Enhanced Security Features: -
  Implement two-factor authentication (2FA), encrypted session management, and login attempt throttling.
  Include automated backup and restore mechanisms to protect against data loss.

REFERENCES

1. Spring Boot Official Documentation – Building RESTful Web Services
   https://spring.io/projects/spring-boot
2. MySQL Documentation – Introduction to MySQL Database System
   https://dev.mysql.com/doc/
3. Bootstrap Official Guide – Responsive Web Design and Components
   https://getbootstrap.com/docs/
4. MDN Web Docs – HTML, CSS, and JavaScript Tutorials
   https://developer.mozilla.org/
5. W3Schools – Web Technologies and Examples
   https://www.w3schools.com/
6. Baeldung – Spring Boot CRUD Operations and REST APIs
   [https://www.baeldung.com/spring-boot-crud-
   thymeleaf](https://www.baeldung.com/spring-boot-crud-thymeleaf)
7. GeeksforGeeks – Java Spring Boot and MySQL Integration Tutorials
   [https://www.geeksforgeeks.org/spring-boot-
   tutorial/](https://www.geeksforgeeks.org/spring-boot-tutorial/)
8. Stack Overflow Discussions – Community Solutions for Common Spring Boot and MySQL Issues
   https://stackoverflow.com/
9. Oracle Java Documentation – Java SE and JDBC Guidelines
   https://docs.oracle.com/en/java/

USER MANUAL

This section provides a simple guide on how to set up and use the Employee Management System project. Follow the steps below to install the required software, configure the project, and start using the application.

1. Prerequisites
Java JDK (version 8 or above) – required to run the Spring Boot application.
MySQL Server – used as the database to store employee information.
IntelliJ IDEA or Eclipse IDE – to open and run the backend project.
A modern web browser (like Chrome or Edge) – to open the frontend files.

2. Setting up the Database
Open MySQL and create a new database named employee_db.
Note down your MySQL username and password, as you'll need to update them in the project configuration file.
The tables will be automatically created when you first run the backend server.

3. Configuring the Project
Open the Employee Management System folder in IntelliJ IDEA (or your preferred IDE).
In the project's resources folder, open the configuration file named application.properties.
Update it with your MySQL username, password, and database name.
Save the file after editing.

4. Running the Backend Server
Locate and run the main file named EmployeeManagementSystemApplication.java.
Once started, Spring Boot will launch the backend server on the default port 8080.
Keep this server running while using the frontend.

5. Running the Frontend
Open the index.html file in any web browser.
This will display the home page of the Employee Management System.
From here, you can perform all main operations such as:
Add a new employee
View employee list
Update employee details
Delete employee record

6. Usage Instructions
Use the navigation bar or buttons to move between pages.
Always ensure that the backend server is running while performing any operation.
Each action (add, update, delete) will immediately reflect in the database.
The interface is designed to be simple, responsive, and beginner-friendly.

7. Closing the Application
Stop the Spring Boot server from your IDE when done.