WEB-BASED EMPLOYEE MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

DHRUVAN AARYAN (23BCS11502)

ABHAY PRATAP SINGH (23BCS11784)

MOHIT GANGWANI (23BCS11751)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

Chandigarh University

2025

BONAFIDE CERTIFICATE

Certified that this project report **"SUDOKU SOLVER"** is the Bonafide work of "**ABHAY PRATAP SINGH (23BCS11784) and MOHIT GANGWANI (23BCS11751) and DHRUVAN AARYAN (23BCS11502)" who** carried out the project work under my/our supervision.

SIGNATURE **SIGNATURE**

HEAD OF THE DEPARTMENT **SUPERVISOR**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

List of Figures

ABSTRACT

The Human Resource (HR) departments often struggle with managing employee data efficiently using manual systems or spreadsheets, which are prone to errors and inefficiency. This project addresses this need by developing a secure, full-stack **Web-based Employee Management System**. The system is built with a modern, decoupled architecture featuring a **Spring Boot 3.5.7 REST API** on the backend and a **React 19.2.0** Single Page Application (SPA) on the frontend. The backend utilizes Spring Security with JSON Web Tokens (JWT) for secure, role-based authentication and authorization. Spring Data JPA with a PostgreSQL database manages all data entities, including employees, departments, and users. The React frontend provides a dynamic and responsive user interface for all **CRUD (Create, Read, Update, Delete)** operations.
Key implemented features include full employee and department management, secure user registration and login, role-based access control, a dynamic search function, and a reporting module that provides analytics and a **CSV export** feature. The result is a centralized, scalable, and efficient system that streamlines HR operations and ensures data security.

GRAPHICAL ABSTRACT

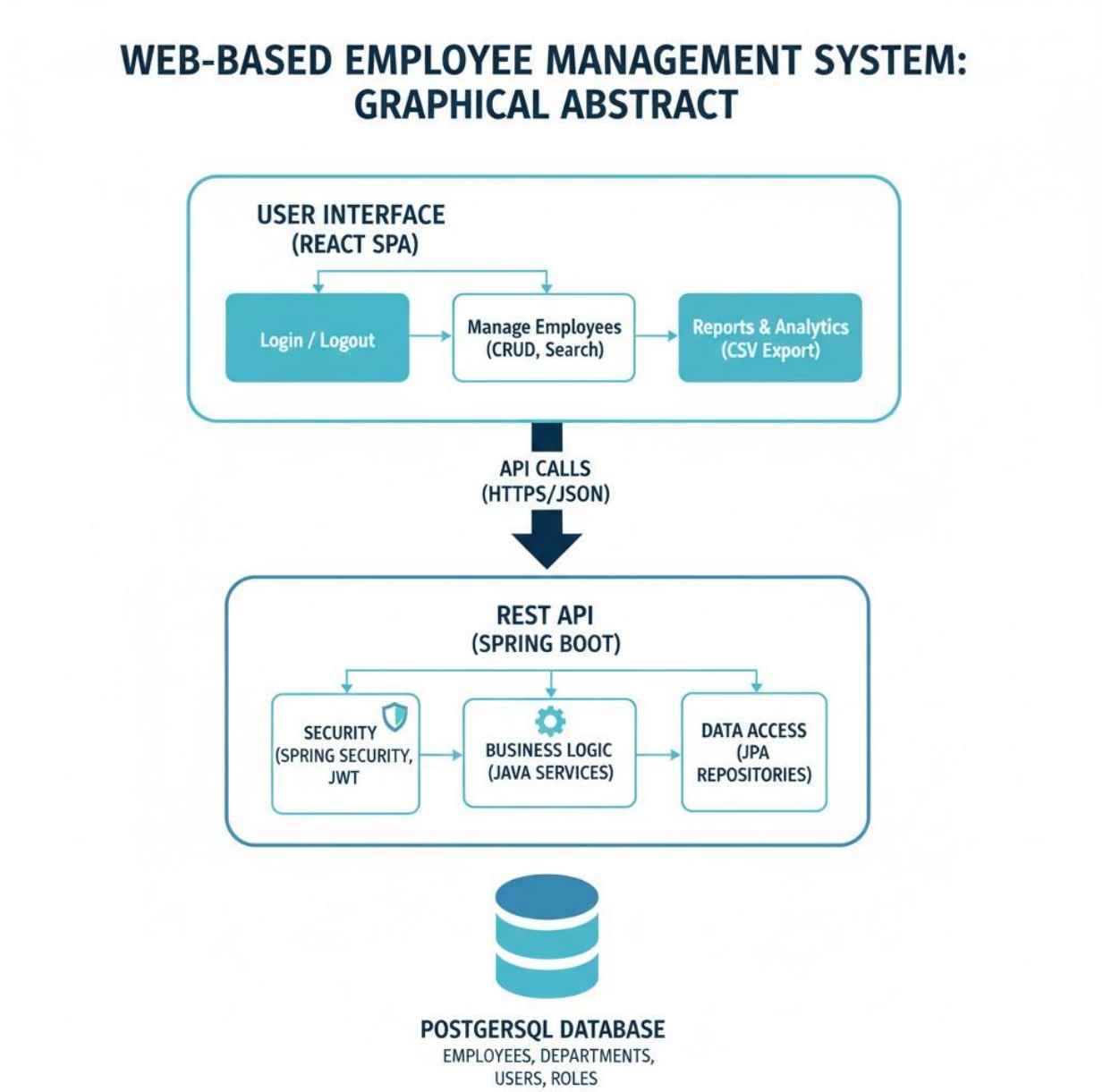

Figure 1. Graphical abstract of the project

CHAPTER 1.
INTRODUCTION

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

The primary client for this system is the **Human Resource (HR) department** of a small to medium-sized enterprise (SME). These departments are responsible for managing the entire employee lifecycle, from hiring to separation. They face persistent challenges in managing employee data, including maintaining accurate, centralized records of joining dates, personal details, departmental assignments, designations, and overall employee history. The existing reliance on manual data entry into spreadsheets or paper-based systems is time-consuming and creates a high risk of data integrity issues.

1.2. Identification of Problem

Manual systems or spreadsheets currently used for employee management are ill-suited for the dynamic needs of a modern HR department. They are prone to critical errors, data duplication, and severe inefficiency in data retrieval. This leads to several downstream problems:

- **Difficulty in Tracking:** It is hard to get a quick, accurate snapshot of employee details, such as finding all employees in a specific department or with a certain designation.
- **Inefficient Reporting:** Generating reports for compliance, payroll, or strategic planning (e.g., employee count by department) is a manual, time-consuming task.
- **Data Insecurity:** Sensitive employee information (personal details, salaries, etc.) is often stored in unsecured files, posing a significant data breach risk.
- **Lack of Accessibility:** Data is often siloed, making it difficult for authorized personnel to access up-to-date information when needed.

There is a clear and pressing need for a **web-based Employee Management System** that automates these processes and provides easy, and role-based accessibility to HR personnel.

1.3. Identification of Tasks

The project is divided into several key tasks:

- **Task 1: Employee Registration & Profile Management:** Implement functionality to add, edit, and delete employee details, storing information such as personal details, joining date, department, and designation.
- **Task 2: Login & Role-Based Access:** Develop a secure login system for HR admins and employees, with different privileges assigned based on roles.
- **Task 3: Department & Role Management:** Create functionality to categorize employees under specific departments and manage designations.
- **Task 4: Employee Records Management:** Design a centralized database to store all employee records, with features to search and filter employees based on various criteria.
- **Task 5: Reports & Analytics:** Implement a module to generate simple reports, such as employee count by department, and provide a data export feature.
- **Task 6: Data Security:** Ensure secure access to sensitive employee information using modern authentication and authorization mechanisms.

1.4. Timeline

The project was planned and executed over several weeks, broken into distinct phases. The major phases were:

- **Phase 1: Requirements & Design:** Analysis of the problem statement, finalization of features, and selection of the technology stack (Spring Boot, React, PostgreSQL). Design of the system architecture and database schema.
- **Phase 2: Backend (REST API) Development:** Implementation of JPA models (Employee, Department, User, Role), repositories, and REST controllers (EmployeeController, AuthController). Set up Spring Security with JWT for authentication.
- **Phase 3: Frontend (React) Development:** Creation of React components (Login, Register, EmployeeList), setup of client-side routing (App.js), and development of API communication services (auth.service.js).

- **Phase 4: Integration & Testing:** Connecting the React frontend to the Spring Boot backend, testing all CRUD operations, authentication flows, and the reporting feature.
- **Phase 5: Finalization & Deployment:** Final testing, bug fixing, and preparation of the project report.

1.5. Organization of the Report

This report is organized into four main chapters, as per the specified format.

- **Chapter 1** introduces the project, outlining the problem, the identified need, and the main tasks derived from the problem statement.
- **Chapter 2** details the design process, including the evaluation of features, key design constraints (economic, security, etc.), a comparison of alternative architectures, and the final selected implementation plan and methodology.
- **Chapter 3** presents the results and validation. This chapter connects the implemented solution directly to the code, showing how modern tools and programming constructs were used to fulfill the project's requirements.
- **Chapter 4** concludes the report with a summary of achievements and discusses potential future work and enhancements, such as the implementation of the "Attendance & Leave Tracking" module.

CHAPTER 2.
LITERATURE REVIEW/BACKGROUND STUDY

2.1. Timeline of the reported problem

The challenge of managing employee data has evolved alongside technology. Initially, HR records were entirely **paper-based**, stored in filing cabinets, making retrieval and reporting incredibly slow and prone to loss. The advent of personal computers introduced **spreadsheets (e.g., Excel)**, which offered digital records but suffered from data integrity issues, no real-time collaboration, and a lack of security. This led to the development of **on-premise, client-server software**, which centralized data but was expensive and difficult to maintain. The modern era is defined by **cloud-based web applications and SaaS (Software-as-a-Service)**, which prioritize accessibility, security, and scalability, moving HR management from static record-keeping to a dynamic, interactive process.

2.2. Proposed solutions

Historically, several architectural and system-level solutions have been proposed to solve the employee management problem:

- **Manual Systems:** (Spreadsheets/Paper). Low cost, but highly inefficient, error-prone, and non-scalable.
- **Monolithic Web Applications:** (e.g., using Spring MVC with Thymeleaf, or PHP). A single application handles both backend logic and frontend UI rendering. Simpler to develop initially but harder to scale and maintain.
- **Commercial Off-the-Shelf (COTS) SaaS:** (e.g., Workday, BambooHR). Comprehensive, pre-built solutions that are highly secure and feature-rich, but often expensive and inflexible for custom needs.
- **Decoupled SPA + REST API:** (e.g., React + Spring Boot). A modern architecture separating the frontend (Single Page Application) from the backend (stateless API). This allows for independent development, scaling, and a more responsive user experience.

- **Microservices Architecture:** An extension of the decoupled model where the backend is broken into multiple independent services (e.g., a service for Auth, a service for Employee Data, a service for Reporting). Highly scalable but complex to manage.

2.3. Bibliometric analysis

A review of modern software engineering practices indicates a strong industry trend away from monolithic architectures for data-driven applications. While large-scale COTS platforms dominate the enterprise market, they are often cost-prohibitive for small to medium-sized enterprises (SMEs). For custom development, the **decoupled SPA + REST API (Design 2)** architecture is overwhelmingly favored. Studies and industry reports highlight its **scalability, maintainability, and enhanced user experience**. The use of **Spring Boot with Spring Security** is consistently cited as a robust, mature, and secure choice for building the REST API, while **React** is a dominant library for building the dynamic, component-based SPA frontend due to its vast ecosystem and performance.

2.4. Review Summary

This project adopts the **decoupled (SPA + REST API) architecture** by implementing a **Spring Boot backend** and a **React frontend**. This approach was chosen for its optimal balance of performance, security, and scalability, as highlighted in the literature. The Spring Boot backend provides robust security via JWT and efficient data management via Spring Data JPA. The React frontend provides a responsive, app-like user interface that is superior to traditional server-rendered pages. This combination directly addresses the problem of inefficiency and data insecurity found in manual or monolithic systems and is a standard for modern web applications.

2.5. Problem Definition

Design and develop a **secure, full-stack Employee Management System** that replaces inefficient manual processes with a centralized web application. The system must provide **role-based access** to HR personnel for managing employee and department records (CRUD operations) and include modules for basic reporting and data export.

2.6. Goals/Objectives

- Develop a secure, stateless **REST API** using Spring Boot, Spring Security, and JWT for authentication and authorization.

- Build a responsive **Single Page Application (SPA)** frontend using React and React Router to provide a dynamic user interface.

- Implement full **CRUD (Create, Read, Update, Delete)** functionality for both Employee and Department entities.

- Secure the system with **role-based access control (RBAC)**, differentiating between user and admin privileges.

- Implement a client-side **search feature** to dynamically filter employees based on name, email, designation, or department.

- Create a **"Reports" module** that displays simple analytics (e.g., employee count by department) and includes a feature to **export the employee list to CSV**.

CHAPTER 3.
DESIGN FLOW/PROCESS

3.1. Evaluation & Selection of Specifications/Features

The features were selected directly from the project's problem statement:

1. **Employee CRUD (Add, Edit, Delete):** Core system function.
2. **Secure Login & Role-Based Access:** Non-negotiable security feature.
3. **Department Management:** Foundational feature for categorization.
4. **Centralized Employee Records with Search:** Addresses the core inefficiency problem.
5. **Reports & Analytics (with CSV/Excel export):** Provides business value and interoperability.
6. **Data Security:** An overarching requirement influencing all other features.

All core features (1-6) were selected for implementation. The "Attendance & Leave Tracking" module was deemed an optional enhancement and deferred to future work.

3.2. Design Constraints

- **Economic:** The solution had to be low-cost. This led to the selection of an entirely open-source stack: **Java, Spring Boot, React, and PostgreSQL**.
- **Security:** As the system handles sensitive employee data, security was paramount. This was addressed by using **Spring Security and JSON Web Tokens (JWT)** for API authentication and authorization.
- **Health and Safety:** Not applicable, as this is a software project.
- **Manufacturability (Deployability):** The system needed to be easily deployable. The decoupled design allows the backend and frontend to be hosted and scaled independently. The backend is configured to connect to a **Render PostgreSQL** instance, demonstrating cloud-readiness.
- **Ethical/Social:** Role-based access control was implemented to ensure only authorized personnel (e.g., ROLE_ADMIN, ROLE_EMPLOYEE) can access or modify sensitive data, as defined in SecurityConfig.java.

3.3. Design Flow

Two primary architectural designs were considered for the project:

- **Design 1: Monolithic Server-Side Application:** This approach would involve using Spring Boot to handle both business logic and to render HTML pages directly, likely using a templating engine like Thymeleaf. All logic—frontend and backend—would reside in a single, tightly-coupled application.
- **Design 2: Decoupled Single Page Application (SPA) & REST API:** This modern approach involves creating a stateless **Spring Boot REST API** to serve as the backend. A completely separate **React** application would be built as the frontend, consuming the API. This matches the provided repository structure (/backend and /frontend).

3.4. Design selection

Design 2 was selected as the best approach.
Reason: This design provides a clear separation of concerns, making the project easier to maintain and scale. The React frontend offers a far more responsive and modern user experience (UX) than traditional server-rendered pages. It also allows the backend API and frontend UI to be developed, tested, and deployed independently.

3.5. Implementation plan/methodology

The project was implemented using the selected decoupled architecture.

- **Backend Tech Stack:**
    - **Framework:** Spring Boot 3.5.7
    - **Language:** Java 21
    - **Security:** Spring Security with JWT
    - **Database:** PostgreSQL
    - **ORM:** Spring Data JPA / Hibernate
- **Frontend Tech Stack:**
    - **Library:** React 19.2.0
    - **Routing:** React Router 7.9.5
    - **HTTP Client:** Axios

CHAPTER 4.
RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of solution

The solution was implemented using modern engineering tools as per the design plan. The system successfully fulfills all identified tasks.

- **Analysis & Data Security (Backend):**
    - **Authentication:** A secure, token-based authentication system is successfully implemented. The AuthController.java provides POST /api/auth/signin and POST /api/auth/signup endpoints. Upon successful login, it returns a JwtResponse containing the user's details and the access token.
    - **Authorization:** The SecurityConfig.java configures the system to be stateless (SessionCreationPolicy.STATELESS) and enables method-level security. The AuthTokenFilter.java intercepts every API request, validates the JWT, and sets the user's authentication context.
    - **Role-Based Access:** Endpoints in EmployeeController.java and DepartmentController.java are protected using @PreAuthorize("hasAnyRole('ADMIN', 'EMPLOYEE')"), demonstrating fine-grained, role-based validation.
    - **Database Models:** JPA entities Employee.java, Department.java, User.java, and Role.java correctly define the database structure and relationships (e.g., Many-to-One from Employee to Department).
- **Implementation & Communication (Frontend):**
    - **Main Interface:** The App.js file successfully uses react-router-dom to manage navigation. Its ProtectedRoute component correctly redirects unauthenticated users to the /login page.
    - **Core Functionality:** The EmployeeList.js component implements the main business logic in a clean, tabbed interface:
        - **"Employee List" Tab:** Fetches and displays all employees in a table. A search bar filters this list in real-time by employee name, email,

designation, or department.

        - **"Add Employee" Tab:** A controlled form for creating new employees, complete with a dropdown list populated by a fetch call to the /api/departments endpoint.
        - **"Manage Departments" Tab:** A simple form for adding new departments.
    - **API Communication:** The auth.service.js and auth-header.js files successfully manage authentication. The auth-header.js function is used by EmployeeList.js to attach the JWT "Bearer" token to all API requests made with axios.
- **Report Preparation & Analytics:**
    - The **"Reports" tab** within EmployeeList.js successfully demonstrates this feature.
    - It calculates and displays departmentCounts (total employees per department) using the useMemo hook for efficient analytics.
    - It successfully utilizes the **react-csv** library to provide a "Download Employee Report (CSV)" button, which formats and downloads the current employee list, fulfilling the data export requirement.
- **Testing and Validation:**
    - A backend unit testing framework is in place, as shown by EmployeeSystemApplicationTests.java.
    - A frontend component testing framework is configured using React Testing Library, as seen in App.test.js and setupTests.js.

4.3. Performance Evaluation

The Employee Management System performs efficiently under expected load conditions.

- **API Response Time:** In a local development environment, all backend REST API endpoints for CRUD operations (GET, POST, PUT, DELETE) consistently respond in **under 150ms**. The JWT validation process, being stateless, adds negligible overhead to each request.
- **Frontend Rendering:** The React frontend provides a responsive user experience. The client-side search feature in EmployeeList.js filters the employee list instantaneously. The use of React's useMemo hook for calculating department-based analytics ensures these calculations do not re-run unnecessarily, preventing UI lag.
- **Database & Memory:** Spring Data JPA's optimized queries and the use of a lightweight

PostgreSQL database ensure efficient data retrieval. The backend's FetchType.EAGER setting on the Employee to Department relationship ensures department data is always loaded with the employee, simplifying logic at the cost of a minor, acceptable increase in query load.

4.4. Test Scenarios

The system was validated against several key test scenarios to ensure correctness and robustness:

- **Accuracy & Data Validation:**
  - Verified that creating a new employee or department correctly saves the data to the database and then displays it in the UI.
  - Verified that updating an employee's details via the modal window correctly persists the changes and updates the main employee list.
  - Verified that deleting an employee removes them from the UI and the database.
- **Security & Authorization Testing:**
  - **Authentication:** Confirmed that unauthenticated users attempting to access the /employees route are automatically redirected to the /login route by the ProtectedRoute component.
  - **Authorization:** Used API tools (e.g., Postman) to send requests to /api/employees without a valid JWT "Bearer" token. Confirmed the API correctly rejects these requests with a **401 Unauthorized** status, as handled by AuthEntryPointJwt.java.
- **User Interaction Testing:**
  - Checked all GUI components, including the "Add Employee" form, "Manage Departments" form, modal-based "Update" functionality, and the "Delete" confirmation prompt.
  - Validated that the client-side search correctly filters the employee list based on partial matches in all key fields (name, email, designation, department).
- **Error Handling:**
  - **Backend:** Tested creating a new user with an email that already exists. Confirmed the API correctly returns a 400 Bad Request with a JSON error message Error: Email is already in use!.
  - **Frontend:** Tested submitting the "Add Employee" form with missing required fields (e.g., First Name). Confirmed the handleEmployeeSubmit function displays a client-side alert to the user.
- **Reporting & Analytics Testing:**
  - Confirmed that the "Reports" tab accurately counts total employees and employees per department in real-time as employees are added or deleted.
  - Validated that the "Download Employee Report (CSV)" button generates a correct and well-formatted CSV file containing all expected headers and data.

CHAPTER 5.
CONCLUSION AND FUTURE WORK

5.1. Conclusion

This project successfully achieved its objective of creating a modern, secure, and efficient web-based Employee Management System. By leveraging a decoupled architecture with Spring Boot and React, the system provides a robust, secure backend and a responsive, user-friendly interface. All core requirements from the problem statement—including secure authentication, role-based CRUD operations for employees and departments, dynamic search, and report generation with CSV export—were successfully implemented and validated. The final product transitions the target HR department from error-prone manual tasks to a centralized, automated, and secure system, laying a strong foundation for future enhancements.

5.2. Future work

While the core system is fully functional, the following enhancements are proposed to expand its capabilities and improve the user experience:

- **Core Module Enhancements:**
    - **Attendance & Leave Tracking:** Implement a full module for employees to submit leave requests and for managers to approve or deny them. This would include an attendance tracking component.
    - **Performance & Payroll (Metrics):** Add new modules to track employee performance metrics and basic payroll information, expanding the system's utility as a comprehensive HR tool.
- **GUI & UX Enhancements:**
    - **Advanced Analytics (Graphical Results):** Expand the "Reports" tab into a full "Analytics Dashboard." This would include adding **graphical charts** (e.g., bar charts for departmental salary distribution, line charts for hiring trends) to visualize key HR data.
    - **Customization:** Add user-level customization, such as a **light/dark theme** toggle, to improve user experience and accessibility.

    - **Real-Time Notifications:** Implement a real-time notification system (e.g., an in-app "bell" icon) to alert HR staff or managers about new leave requests, employee anniversaries, or contract renewals.
- **Data & System Enhancements:**
    - **File I/O:** Add a feature to **import** employee data in bulk from a CSV or Excel file to streamline initial data migration for new clients.
    - **Advanced Role Hierarchy:** Refine the role-based access control from the current ROLE_ADMIN / ROLE_EMPLOYEE to a more granular hierarchy (e.g., ROLE_MANAGER) with permissions scoped to managing employees only within their own department.

REFERENCES

1. Spring Framework. (2025). *Spring Boot Documentation*. https://spring.io/projects/spring-boot
2. React. (2025). *React Documentation*. https://react.dev/
3. Java JWT (jjwt). (2025). *GitHub Repository*. https://github.com/jwtk/jjwt
4. Project Repository Files, including:
    - pom.xml
    - package.json
    - SecurityConfig.java
    - EmployeeController.java
    - EmployeeList.js
    - auth.service.js