Employee Management System Documentation

Table of Contents

Introduction

An online program called the Employee Management System (EMS) was created to help businesses handle employee data more effectively. The EMS offers a simplified method for managing a range of employee-related duties, such as generating, reading, updating, and deleting employee data, in a time when efficient human resource management is essential for company success.

Built on a modern tech stack, the EMS features a Java Spring Boot backend and a React frontend. The Spring Boot framework facilitates the construction of RESTful APIs, providing for seamless interaction with a MySQL database with Spring Data JPA. This enables secure and efficient data storage and retrieval, which is vital for ensuring data integrity.

React's component-based architecture makes it possible to create a dynamic and responsive user interface on the front end. The EMS improves the user experience by enabling real-time updates without requiring complete page reloads through the usage of Axios for API calls.

Role-based access control in the system's design guarantees that managers and HR staff, among other user roles, have the proper access to sensitive data. Workflows are streamlined and security is improved by this capability.

In addition to the core staff management features, the EMS can be extended to incorporate more complex features like performance monitoring and attendance control. All things considered, companies seeking to improve data accuracy, expedite personnel management processes, and foster a positive work environment must have a personnel management system.

Technologies Used

Backend: Java, Spring Boot, Spring Data JPA, Maven

Frontend: React, Vite, Axios, Bootstrap, NPM, JavaScript

Database: MySQL

API: Postman Client

Development Tools:

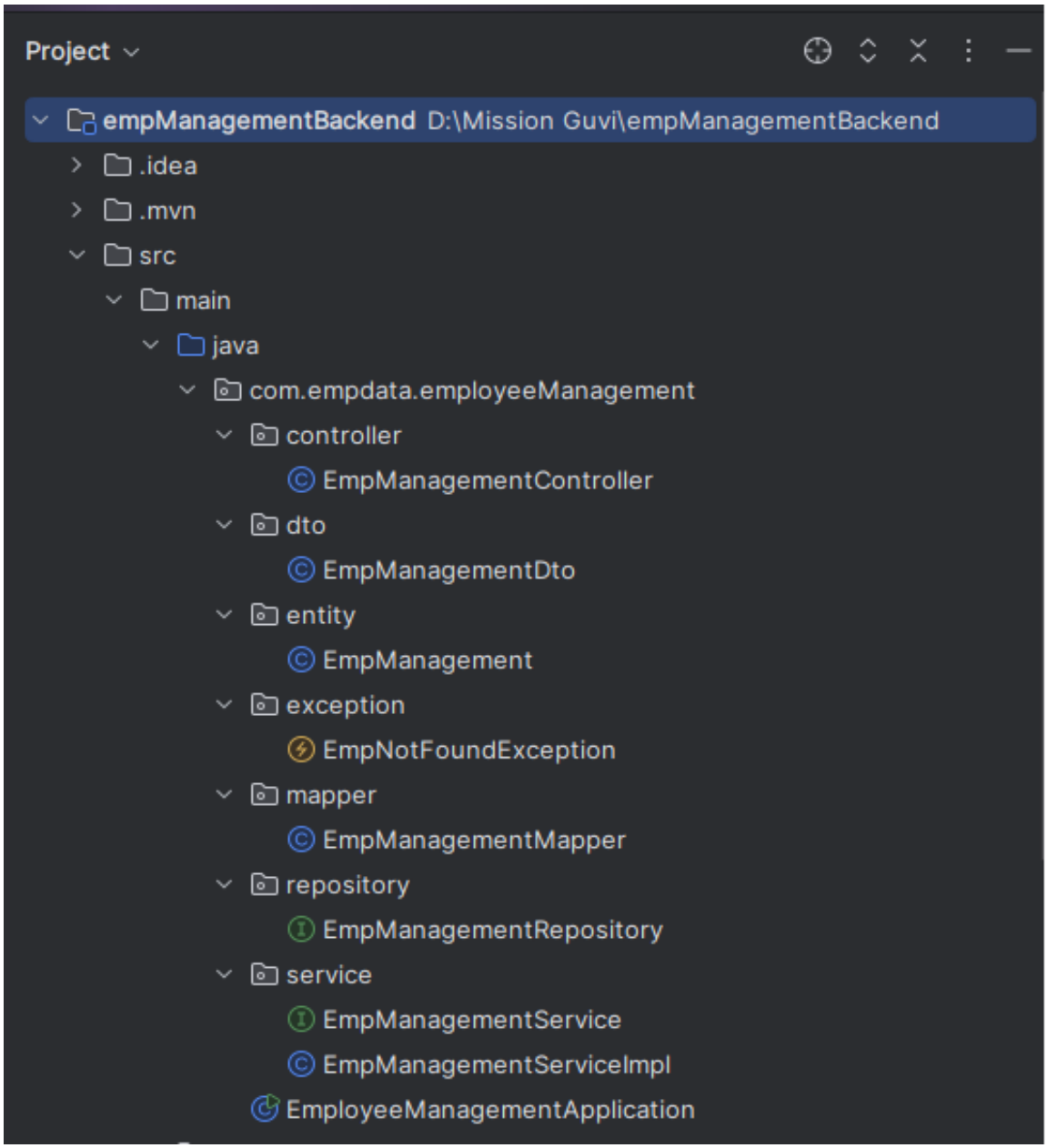
- Backend: IntelliJ IDEA
- Frontend: Visual Studio Code

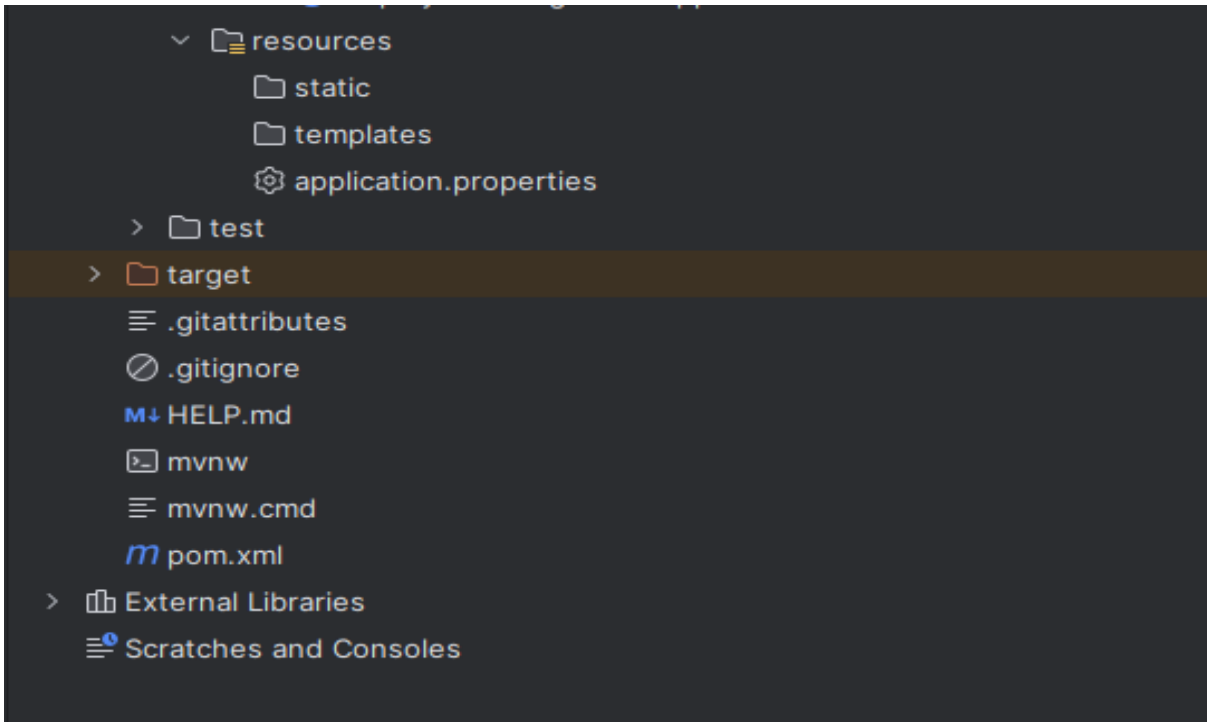
Project Structure

Spring Boot Structure with Dependencies

- Spring Boot Starter Web
- Spring Data JPA
- MySQL driver
- Lombok

Backend Setup





Core Components and Packages

This section details the purpose of the key packages within src/main/java/com/empdata/employeeManagement/.

1. controller

* Purpose: This package contains classes responsible for handling incoming HTTP requests from clients. Controllers act as the entry point to the application's business logic.

* Key File: EmpManagementController.java

  * This class likely defines REST endpoints for operations related to employee management (e.g., creating, retrieving, updating, deleting employees).

2. dto

* Purpose: This package holds Data Transfer Objects (DTOs). DTOs are simple classes used to transfer data between different layers of the application (e.g., between the controller and service layers) or between the application and the client. They help in defining the structure of data exchanged. DTO is frequently used for validation in Java Applications.

* Key File: EmpManagementDto.java

  * Represents the data structure used for transferring employee information.

3. entity

* Purpose: This package contains JPA Entity classes. These classes map to database tables and represent the data model of the application.

* Key File: EmpManagement.java

  * This class is likely mapped to an "employees" table in the database and defines the fields and relationships of an employee record.

4. exception

* Purpose: This package is for custom exception classes. Defining custom exceptions helps in handling specific error scenarios in a structured way, improving the clarity and maintainability of error handling.

* Key File: EmpNotFoundException.java

  * A custom exception likely thrown when an employee record is not found.

5. mapper

* Purpose: This package contains classes responsible for mapping data between different object types, specifically between DTOs and Entity objects. Using mappers helps in separating the mapping logic from the core business logic. Mapper is often used for error handling in java.

* Key File: EmpManagementMapper.java

  * This class provides methods to convert EmpManagementDto objects to EmpManagement entities and vice-versa.

6. repository

* Purpose: This package represents the Data Access Layer. It contains interfaces that extend Spring Data JPA repositories, providing methods for performing CRUD (Create, Read, Update, Delete) operations on the EmpManagement entity without writing boilerplate code.

* Key File: EmpManagementRepository.java

  * This interface extends a Spring Data JPA repository (e.g., JpaRepository) and provides methods for interacting with the EmpManagement entity in the database.

7. service

* Purpose: This package contains the business logic of the application. The service layer orchestrates operations by interacting with repositories and potentially other services. It defines the core functionalities related to employee management.

* Key Files:

  * EmpManagementService.java: An interface defining the contract for employee management operations.

  * EmpManagementServiceImpl.java: The implementation of the EmpManagementService interface, containing the actual business logic.

8. EmployeeManagementApplication.java

* Purpose: This is the main class that bootstraps and launches the Spring Boot application. It contains the main method.

9. Resources (src/main/resources)

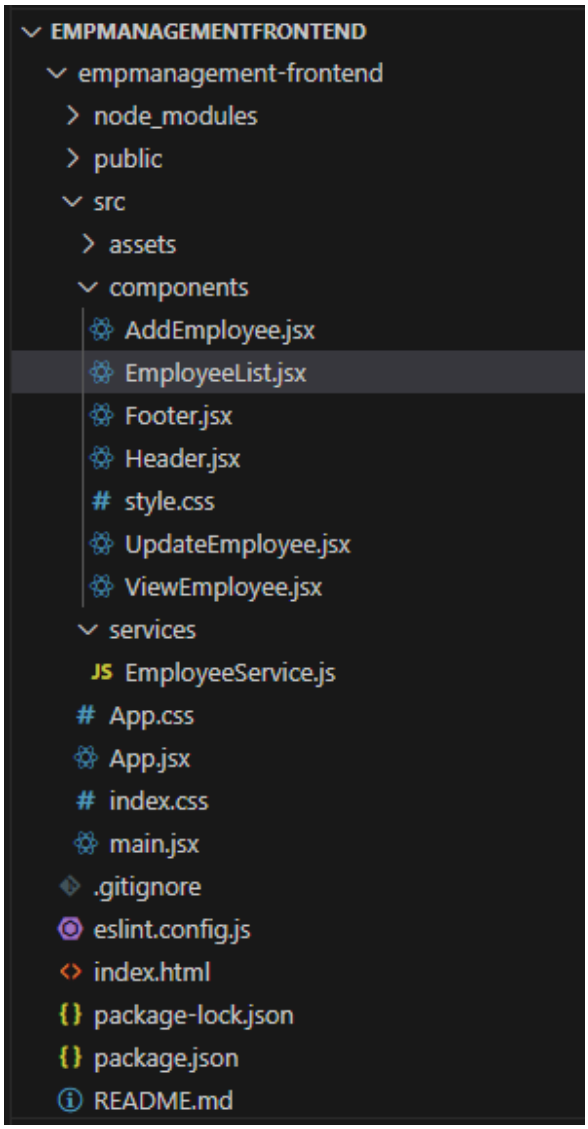
* Purpose: This directory is used for storing configuration files (e.g., application.properties or application.yml), static resources, and templates.

* static/: Typically used for placing static web resources that can be served directly by the application (e.g., HTML, CSS, JavaScript files if this were a web application, though for a backend it might be less common unless serving static documentation or a simple landing page).

10. Dependencies (pom.xml)

The pom.xml file (not explicitly shown but implied) located in the project root manages the project's dependencies (e.g., Spring Boot starters, database drivers, testing libraries) and build configuration using Maven.

Frontend Setup



Core Components and Directories
This section details the purpose of the key directories and files within the project.

1. src/ This directory contains all the source code for the React application.

- assets/:
    - Purpose: This directory is intended for storing static assets used by the application, such as images, icons, or fonts.

- components/:
    - Purpose: This directory houses reusable React components that make up the user interface. Each file typically represents a distinct UI element or a section of a page.
    - Key Files:
        - AddEmployee.jsx: Component for adding a new employee.
        - EmployeeList.jsx: Component for displaying a list of employees.
        - Footer.jsx: Component for the application's footer.
        - Header.jsx: Component for the application's header.
        - style.css: Stylesheet potentially containing styles specific to components within this directory (or perhaps a global component stylesheet).
        - UpdateEmployee.jsx: Component for updating an existing employee's details.
        - ViewEmployee.jsx: Component for displaying the details of a single employee.
- services/:
    - Purpose: This directory contains modules responsible for handling communication with the backend API. This separation keeps API logic distinct from UI components.
    - Key File: EmployeeService.js:
        - Contains functions for making API calls related to employee operations (e.g., fetching employees, adding an employee, deleting an employee).
- App.jsx:
    - Purpose: The main application component, often serving as the root of the component tree. It might handle routing and layout.
- App.css:
    - Purpose: Stylesheet for the App component, potentially containing global or layout-specific styles.
- index.css:
    - Purpose: Global stylesheet for the entire application.
- main.jsx:
    - Purpose: The entry point of the React application. This file typically renders the root App component into the index.html file.

2. public/

- Purpose: This directory contains static assets that are served directly by the web server without being processed by the build pipeline. It typically includes the main index.html file and assets like the favicon.

3. node_modules/

- Purpose: This directory is created by npm or yarn and contains all the project's dependencies. It should not be manually modified or committed to version control.

4. Configuration Files

- .gitignore:
    - Purpose: Specifies files and directories that should be ignored by Git (e.g., node_modules, build artifacts).
- eslint.config.js:
    - Purpose: Configuration file for ESLint, a linter used to identify and report on patterns found in ECMAScript/JavaScript code, helping maintain code quality and consistency.
- vite.config.js:
    - Purpose: Configuration file for Vite, the build tool. This file is used to configure how Vite builds, serves, and optimizes the frontend application. It can include settings for plugins, proxying API requests, build options, etc.

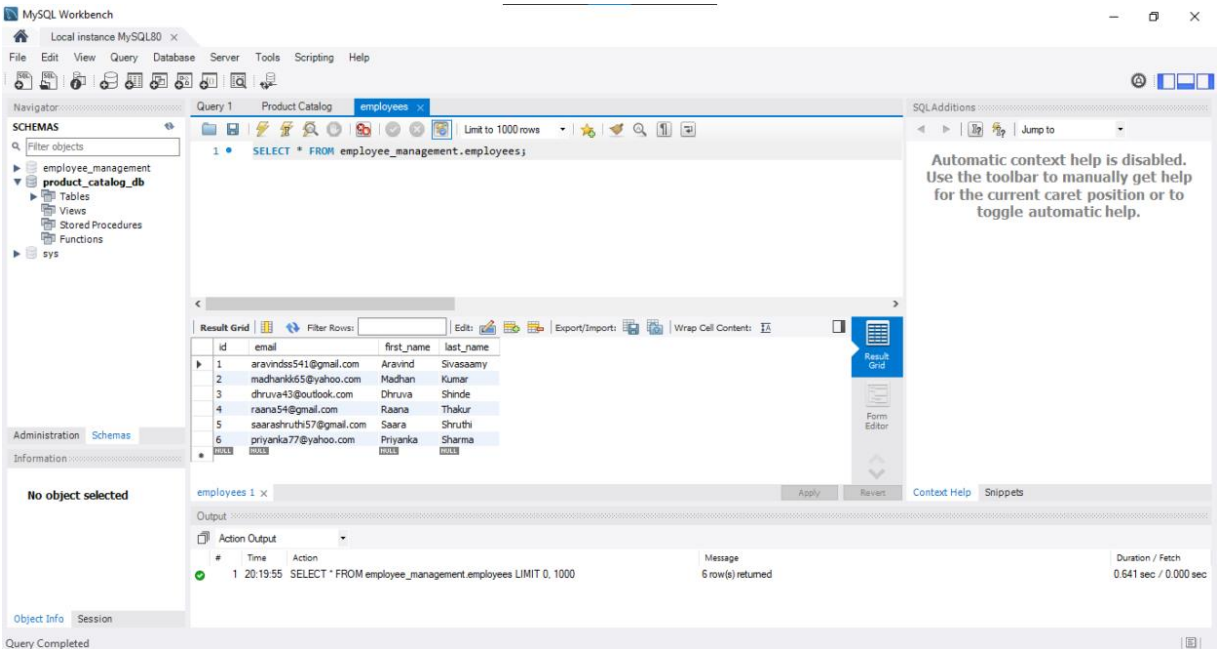
5. Project Management

- package.json:
    - Purpose: Contains metadata about the project, including its name, version, dependencies (libraries the project needs), and scripts for tasks like starting the development server, building the project, and running tests.
- package-lock.json:
    - Purpose: Records the exact versions of all dependencies, including nested dependencies. This ensures that everyone working on the project uses the same dependency versions.
- README.md:
    - Purpose: A markdown file that typically provides a description of the project, instructions on how to set it up and run it, and any other important information for users or contributors.

Database Structure

Employees Table

Column	Type	Description
id	long	Employee unique identifier
first_Name	String	First Name of the employee
last_Name	String	Last Name of the employee
email	String	Employee's email address



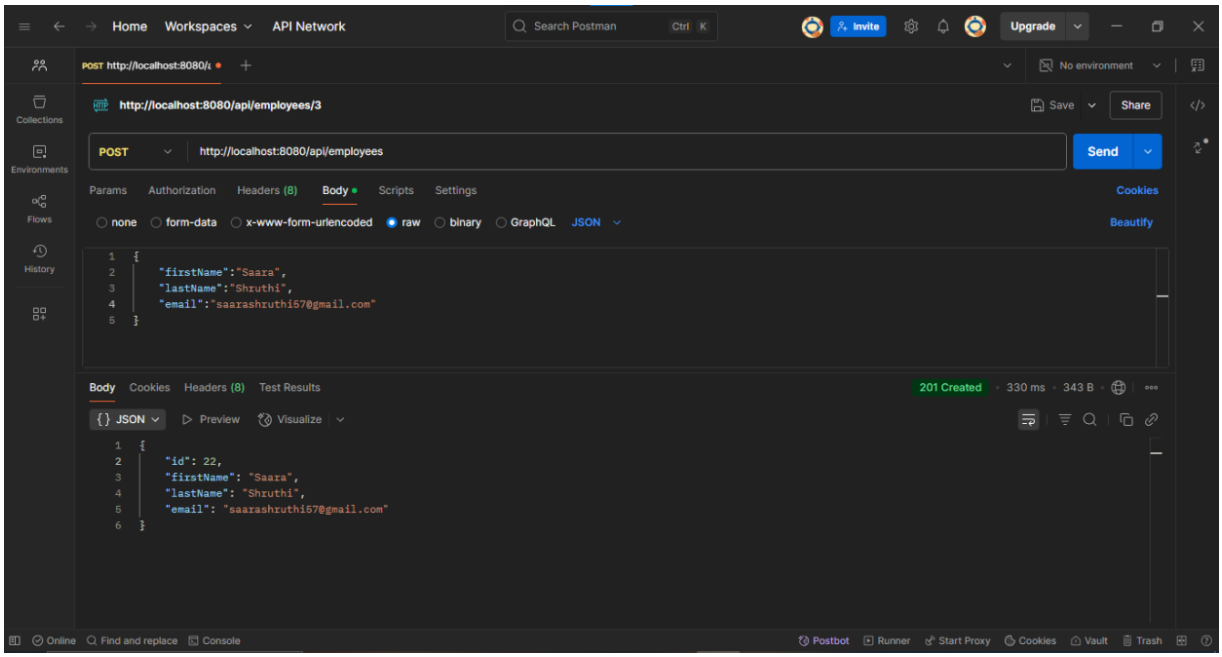
API Endpoints

1.Create Employee

Method: 'POST'

Endpoint: '/employees'

Description: Creates a new employee record.

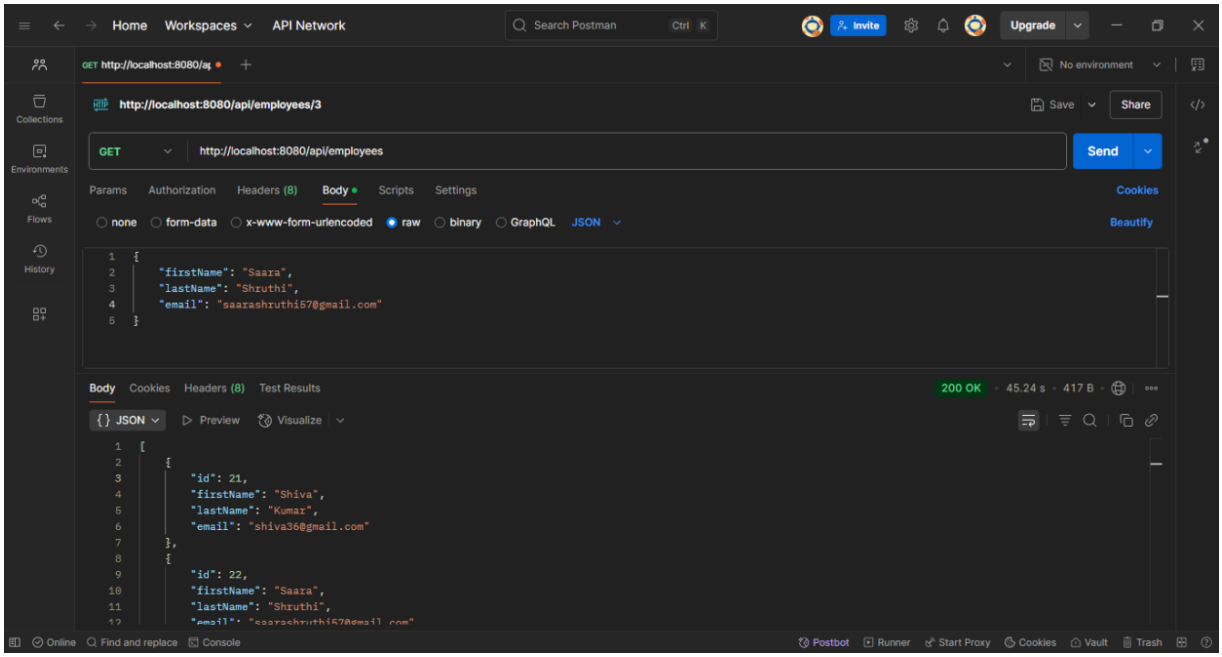


2. Get All Employees

Method: ‘GET’

Endpoint: ‘/employees’

Description: Retrieves a list of all employees



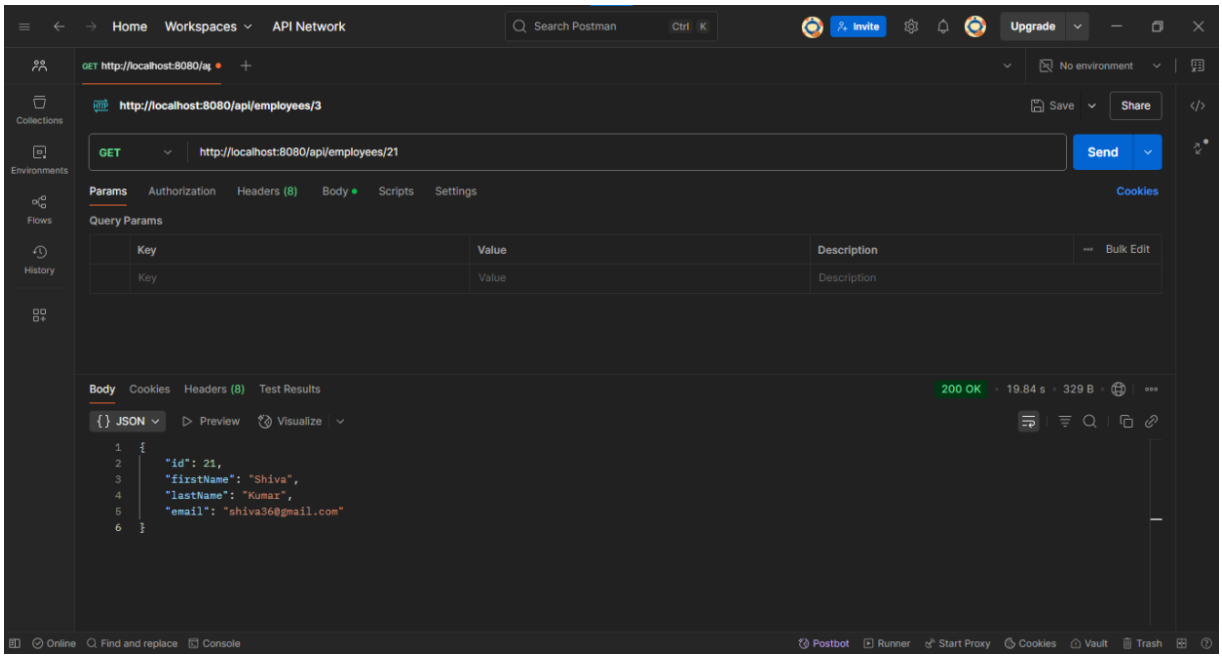
3. Get Employee by ID

Method: ‘GET’

Endpoint: ‘/employees/{id}’

Description: Retrieves a single employee record by ID.

Path Parameters: ‘id’: The ID of the employee to retrieve.



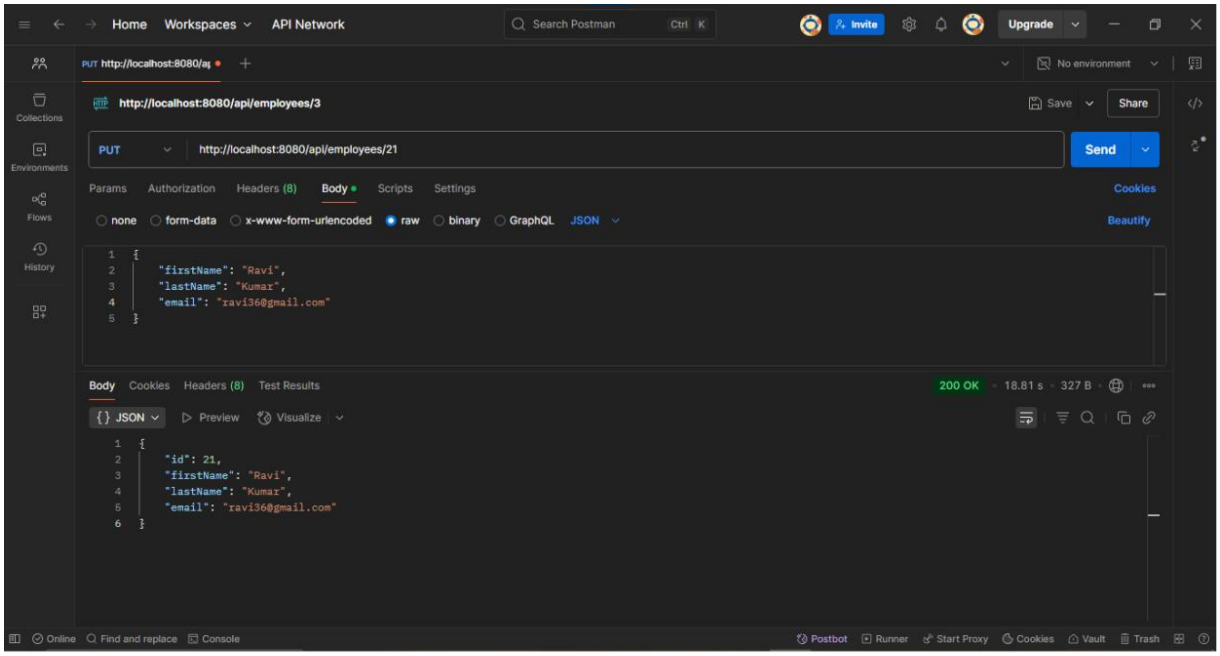
4. Update Employee

Method: ‘PUT’

Endpoint: ‘/employees/{id}’

Description: Updates an existing employee record.

Path Parameters: ‘id’: The ID of the employee to retrieve.



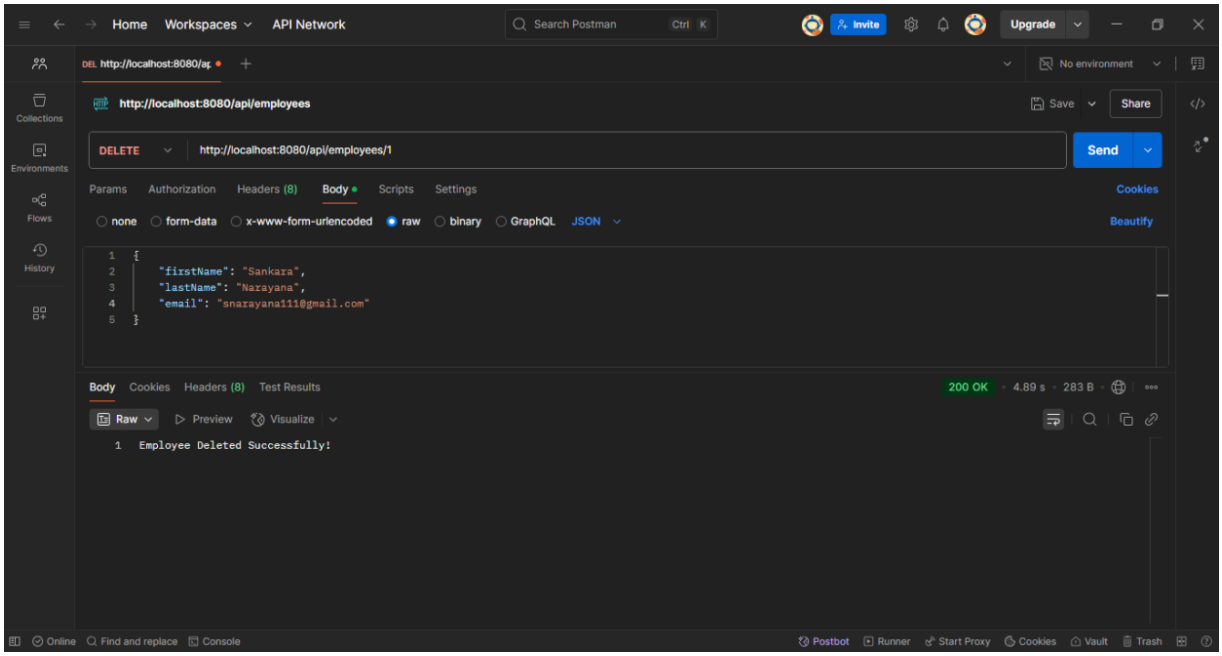
5. Delete Employee

Method: ‘DELETE’

Endpoint: ‘/employees/{id}’

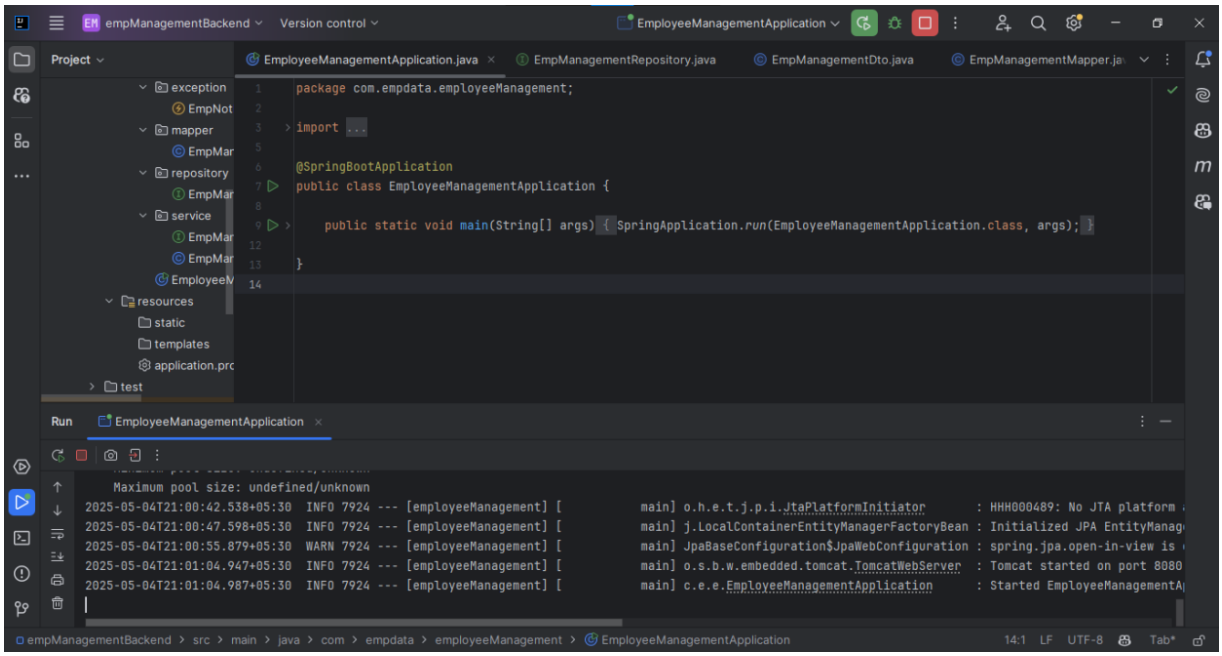
Description: Deletes an employee record.

Path Parameters: ‘id’: The ID of the employee to retrieve.

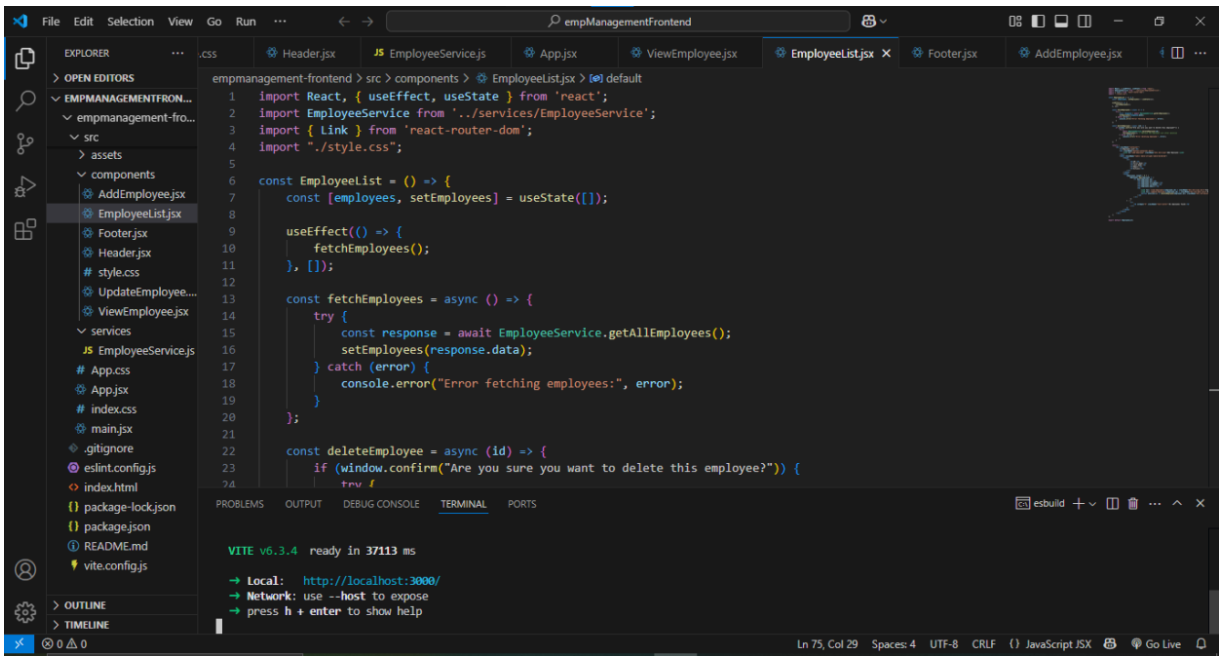


Running the Application

Backend Initialization from Intellij IDEA

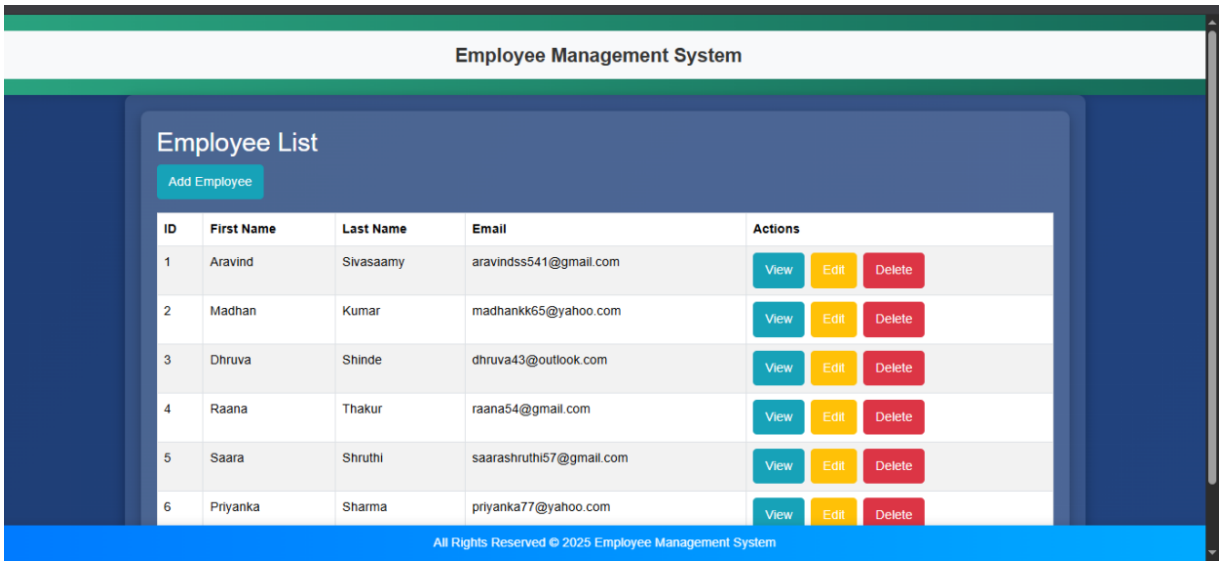


Frontend Initialization from Visual Studio Code

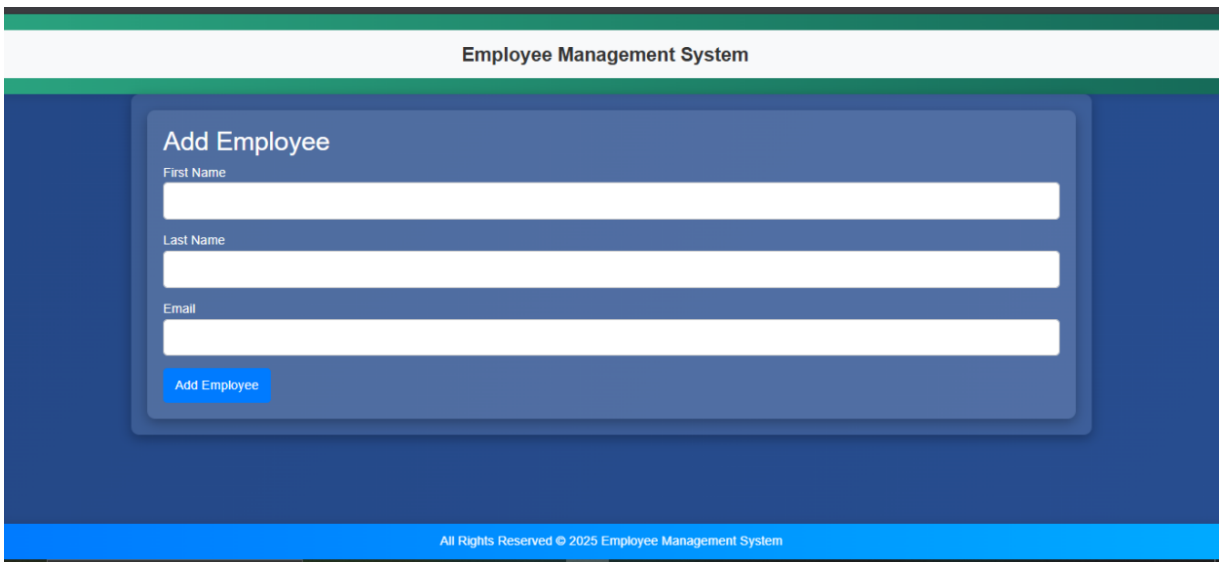


After starting the application, it linked via localhost:8080 and runs in localhost:3000

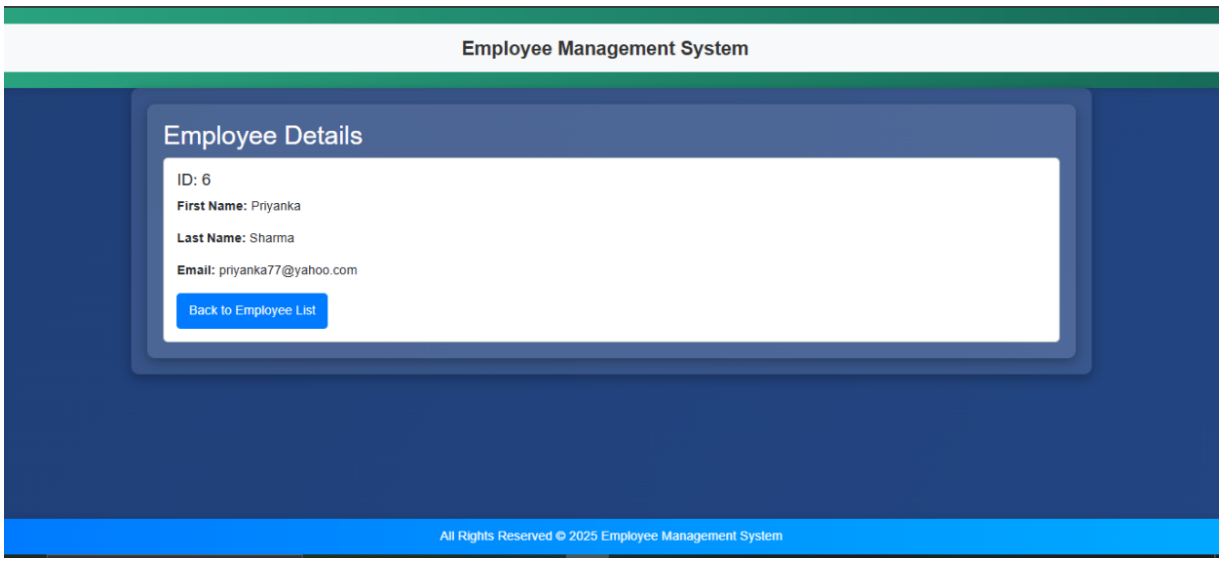
Home Page



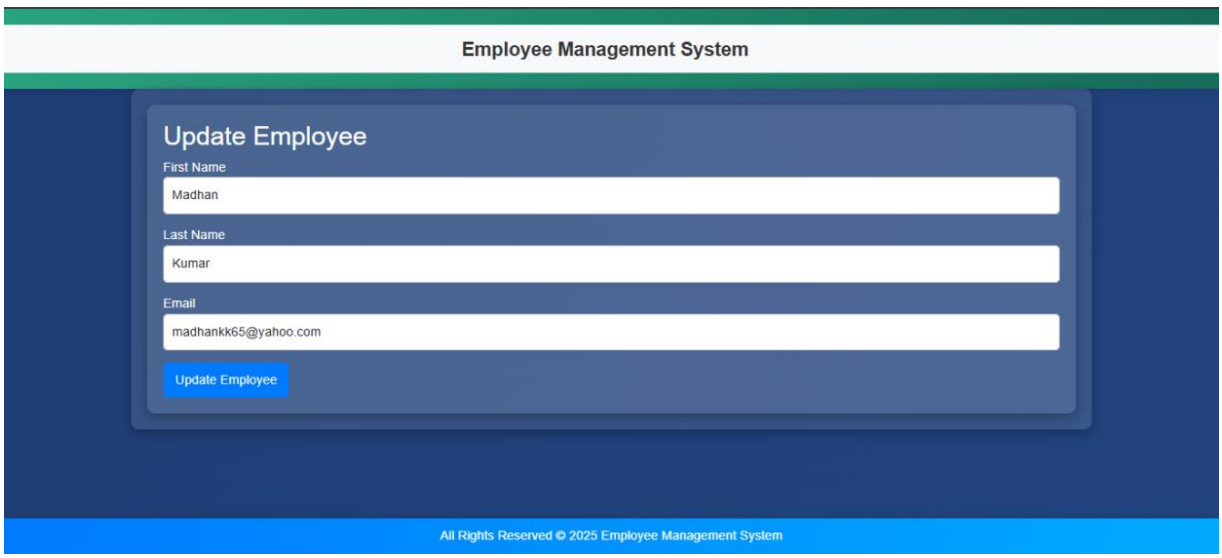
Add Employee Page



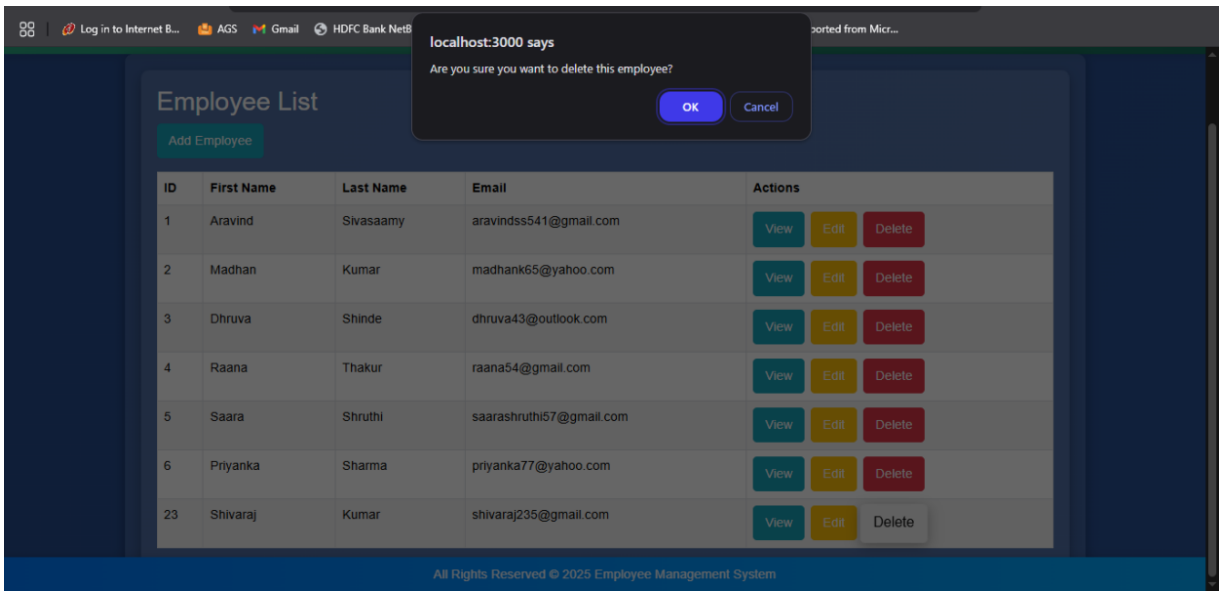
View Employee Page



Edit Employee Page



Delete Employee Page



All functions are working perfectly and data which I've entered is stored in MySQL Database. CSS codes are working and getting a colorful web application.

Conclusion

The Employee Management System (EMS) is a comprehensive solution designed to streamline the management of employee data within organizations. By leveraging modern technologies such as Java Spring Boot for the backend and React for the frontend, the EMS provides a robust and user-friendly interface for managing employee records efficiently.

This documentation serves as a guide for developers and users to understand the architecture, functionalities, and API endpoints of the EMS. The clear organization of the project structure, along with detailed API documentation, ensures that developers can easily integrate, test, and extend the system as needed.

Key features of the EMS include:

- CRUD Operations: The system allows users to create, read, update, and delete employee records, ensuring that all employee data is easily accessible and manageable.
- RESTful API: The well-defined API endpoints facilitate seamless communication between the frontend and backend, enabling real-time data updates and interactions.
- User -Friendly Interface: The React-based frontend provides a responsive and intuitive user experience, making it easy for users to navigate and perform tasks.
- Scalability: The modular architecture of the EMS allows for easy scalability, enabling organizations to add new features and functionalities as their needs evolve.

By implementing the EMS, organizations can improve their employee management processes, enhance data accuracy, and foster a more organized and efficient workplace. This documentation will be continuously updated to reflect any changes or enhancements made to the system, ensuring that users always have access to the most current information.

Thank you for exploring the Employee Management System. I hope this documentation helps you effectively utilize the system and contributes to your organization's success in managing employee data.