

FULLSTACK - CRUD - HR MANAGEMENT SYSTEM

This project is a Maven-based application developed using the Spring Boot framework, with integration of MySQL as the database and React as the frontend technology. It aims to provide a robust and scalable solution for building web applications.

Maven:

Maven is a build automation tool widely used in Java projects. It simplifies the build process, manages dependencies, and facilitates project management. Maven uses a declarative XML-based configuration file called `pom.xml` (Project Object Model) to define the project's structure, dependencies, and build configurations.

Spring Boot:

Spring Boot is a powerful framework built on top of the Spring framework that simplifies the development of Java applications. It provides an opinionated approach to configure and deploy Spring applications, reducing the need for manual configuration. With Spring Boot, developers can quickly set up a production-ready application with minimal effort.

MySQL:

MySQL is a popular open-source relational database management system. It provides a robust and scalable platform for storing and retrieving data. In this project, MySQL is used as the backend database to store and manage the HR Management System data.

React:

React is a JavaScript library for building user interfaces. It allows developers to create reusable UI components and efficiently update the user interface when the underlying data changes. In this project, React is used as the frontend technology to build a responsive and interactive user interface.

Project Overview:

The project combines the power of Spring Boot, MySQL, and React to develop a comprehensive HR Management System. It provides functionalities for managing employee data, including creating, updating, and deleting employee records. The system allows users to perform CRUD (Create, Read, Update, Delete) operations on employee data through a user-friendly interface built using React.

Technology Stack:
























The technology stack used in this project includes:

- Maven: Build automation and dependency management tool.
- Spring Boot: Java-based framework for developing backend applications.
- MySQL: Relational database management system for data storage and retrieval.
- React: JavaScript library for building interactive user interfaces.

By leveraging these technologies, the project aims to deliver a seamless and efficient HR Management System that facilitates the management of employee records.

Backend Documentation

Back end project Structure

- Emps [0000]
- ✓  src/main/java
 - ✓  net.prabhu.Emps
 - >  DataInitializer.java
 - >  EmpsApplication.java
 - ✓  net.prabhu.Emps.controller
 - >  EmployeeController.java
 - ✓  net.prabhu.Emps.entity
 - >  Employee.java
 - ✓  net.prabhu.Emps.repository
 - >  EmployeeRepository.java
 - ✓  net.prabhu.Emps.service
 - >  EmployeeService.java
 - >  EmployeeServiceImpl.java
- ✓  src/main/resources
 - >  META-INF
 -  static
 -  templates
 -  application.properties
- >  src/test/java
- >  JRE System Library [JavaSE-17]
- >  Maven Dependencies
- >  target/generated-sources/annotations
- >  JUnit 5
- >  target/generated-test-sources/test-annotations
- >  src
- >  target
 -  HELP.md
 -  mvnw
 -  mvnw.cmd
 -  pom.xml

Project Documentation: Employee.java

This class represents an employee entity in the HR Management System. It contains the following attributes and methods:

Attributes:

- id: Long (auto-generated) - The unique identifier for an employee.
- firstName: String - The first name of the employee.
- lastName: String - The last name of the employee.
- email: String - The email address of the employee.

Methods:

- Constructor: Initializes the Employee object with the provided first name, last name, and email.
- getId(): Retrieves the unique identifier of the employee.
- setId(Long id): Sets the unique identifier of the employee.
- getFirstName(): Retrieves the first name of the employee.
- setFirstName(String firstName): Sets the first name of the employee.
- getLastName(): Retrieves the last name of the employee.
- setLastName(String lastName): Sets the last name of the employee.
- getEmail(): Retrieves the email address of the employee.
- setEmail(String email): Sets the email address of the employee.

Note: The Employee class is annotated with JPA annotations (@Entity, @Table) to map it to the corresponding database table named "employees". It also includes validation annotations (@NotBlank, @Size, @Email) to enforce data integrity and ensure that the attributes meet the specified constraints.

Project Documentation: EmployeeRepository:

1. Introduction:

The EmployeeRepository interface is a component of the HR Management System that provides the data access layer for managing employee entities. It extends the JpaRepository interface, which is part of the Spring Data JPA

framework. The `EmployeeRepository` interface allows for performing various CRUD operations on the `Employee` entities and provides additional methods for querying and manipulating the employee data in the underlying database.

2. Features:

The `EmployeeRepository` interface offers the following features:

CRUD Operations:

Create: Allows for saving new `Employee` entities to the database.

Read: Provides methods for retrieving `Employee` entities based on their unique identifier or other query criteria.

Update: Allows for modifying and updating the attributes of existing `Employee` entities.

Delete: Provides methods for deleting `Employee` entities from the database.

3. Technology Stack:

The technology stack used in the `EmployeeRepository` interface includes:

Java: The programming language used for implementing the repository methods and interfaces.

Spring Data JPA: A part of the Spring Framework that simplifies the implementation of the data access layer using JPA (Java Persistence API).

JpaRepository: An interface provided by Spring Data JPA that includes generic CRUD methods for performing common database operations.

MySQL: The underlying database management system where the `Employee` entities are persisted.

4. Usage:

To use the `EmployeeRepository` interface, you need to inject an instance of it into your service or controller classes. You can then leverage the provided methods to interact with the `Employee` entities in the database.

For example, you can use methods like `save()` to create or update an employee, `findById()` to retrieve an employee by their ID, `findAll()` to retrieve all employees, and `deleteById()` to delete an employee by their ID.

@Repository annotation is used to indicate that the EmployeeRepository interface is a Spring-managed repository component.

Project Documentation: EmployeeService

1. Introduction:

The EmployeeService interface is a component of the HR Management System that defines the contract for handling employee-related operations. It acts as a bridge between the controller layer and the repository layer, providing a set of methods for managing employee entities. The EmployeeService interface declares methods for retrieving, creating, updating, and deleting employee records.

2. Features:

The EmployeeService interface offers the following features:

2.1 Retrieve Employees:

`getAllEmployees()`: Retrieves a list of all employee entities in the system.

`getEmployeeById(Long id)`: Retrieves a specific employee entity based on the provided unique identifier.

2.2 Manage Employees:

`createEmployee(Employee employee)`: Creates a new employee entity with the provided employee details.

`updateEmployee(Long id, Employee employee)`: Updates the details of an existing employee entity identified by the given ID.

`deleteEmployee(Long id)`: Deletes an employee entity from the system based on the provided ID.

3. Usage:

To use the EmployeeService interface, you need to implement it in a service class and provide the required logic for each method. The service class acts

as an intermediary between the controller and the repository, handling the business logic and interactions with the employee repository.

For example, the implementation of the `createEmployee()` method would include validating the employee details, generating an ID, and saving the employee entity using the `EmployeeRepository`.

The `EmployeeService` interface serves as an abstraction layer, enabling loose coupling and separation of concerns in the application architecture.

4. Technology Stack:

The technology stack used in the `EmployeeService` interface includes:

Java: The programming language used for defining the interface and method signatures.

Spring Framework: Provides the underlying framework for dependency injection and inversion of control.

Employee: The entity class representing the employee data structure.

Please note that the implementation of the `EmployeeService` interface will vary depending on the specific business requirements and the application's architecture.

Project Documentation: EmployeeServiceImpl

1. Introduction:

The `EmployeeServiceImpl` class is an implementation of the `EmployeeService` interface in the HR Management System. It provides the actual implementation for the methods defined in the interface, handling the business logic for managing employee entities. The `EmployeeServiceImpl` class interacts with the `EmployeeRepository` to perform CRUD operations on the employee data.

2. Features:

The `EmployeeServiceImpl` class offers the following features:

2.1 Retrieve Employees:

`getAllEmployees()`: Retrieves a list of all employee entities by calling the `findAll()` method of the `EmployeeRepository`.

2.2 Manage Employees:

`getEmployeeById(Long id)`: Retrieves a specific employee entity by calling the `findById()` method of the `EmployeeRepository` with the provided unique identifier.

`createEmployee(Employee employee)`: Creates a new employee entity by calling the `save()` method of the `EmployeeRepository` with the provided employee object.

`updateEmployee(Long id, Employee employee)`: Updates the details of an existing employee entity by checking if the employee with the given ID exists using the `existsById()` method of the `EmployeeRepository`. If the employee exists, the method sets the ID of the provided employee object and calls the `save()` method of the `EmployeeRepository` to save the updated employee details.

`deleteEmployee(Long id)`: Deletes an employee entity by calling the `deleteById()` method of the `EmployeeRepository` with the provided ID.

3. Usage:

The `EmployeeServiceImpl` class is annotated with `@Service`, indicating that it is a Spring-managed service component. The class is constructed with an instance of the `EmployeeRepository`, which is injected using the `@Autowired` annotation.

The `EmployeeServiceImpl` class acts as an intermediary between the controller layer and the repository layer. It implements the business logic for handling employee-related operations and delegates the actual database operations to the `EmployeeRepository`.

4. Technology Stack:

The technology stack used in the `EmployeeServiceImpl` class includes:

Java: The programming language used for implementing the service methods and dependencies.

Spring Framework: Provides the underlying framework for dependency injection and inversion of control.

Employee: The entity class representing the employee data structure.

EmployeeRepository: The repository interface for accessing and manipulating employee data.

Project Documentation: DataInitializer

1. Introduction:

The DataInitializer class is a component in the HR Management System that initializes the database with dummy data. It implements the ApplicationRunner interface from the Spring Boot framework, allowing it to execute code during the application startup. The DataInitializer class is responsible for creating and saving sample Employee entities in the database.

2. Features:

The DataInitializer class offers the following features:

2.1 Data Initialization:

run(ApplicationArguments args): Overrides the run() method from the ApplicationRunner interface. This method is executed when the application starts. It creates sample Employee entities and saves them using the EmployeeRepository.

3. Usage:

The DataInitializer class is annotated with @Component, indicating that it is a Spring-managed component. It is constructed with an instance of the EmployeeRepository, which is injected using the constructor.

During application startup, the run() method of the DataInitializer class is automatically invoked. Inside this method, dummy Employee entities are created with sample data (name and email) and saved using the employeeRepository.save() method.

This ensures that when the application starts, the database is populated with initial data, which can be useful for testing and demonstration purposes.

4. Technology Stack:

The technology stack used in the DataInitializer class includes:

Java: The programming language used for implementing the initialization logic.

Spring Boot: The framework used for developing the application and managing dependencies.

Employee: The entity class representing the employee data structure.

EmployeeRepository: The repository interface for accessing and manipulating employee data.

Please note that the DataInitializer class is executed only during the application startup, and it is not meant for production use. In a production environment, you would typically have a proper data migration strategy and use this class only for development or testing purposes.

Project Documentation: application.properties

1. Introduction:

The application.properties file is a configuration file used in the HR Management System. It contains various settings and properties for configuring the application, including database connection details, Hibernate configuration, server configuration, and CORS (Cross-Origin Resource Sharing) settings.

2. Configuration Properties:

The application.properties file includes the following configuration properties:

2.1 Database Configuration:

spring.datasource.url: Specifies the URL of the MySQL database. In this case, the database is hosted on the localhost machine with port 3306 and named "empsdb".

spring.datasource.username: Specifies the username used to connect to the database. In this case, the username is "root".

spring.datasource.password: Specifies the password used to authenticate the database connection. In this case, the password is "xxxxxxxxxxxx".

2.2 Hibernate Configuration:

`spring.jpa.properties.hibernate.dialect`: Specifies the Hibernate dialect to be used for working with the MySQL database. In this case, the dialect is set to `"org.hibernate.dialect.MySQL8Dialect"`.

`spring.jpa.hibernate.ddl-auto`: Specifies the behavior of Hibernate when creating or updating database tables. In this case, the value is set to `"update"`, which means Hibernate will automatically update the schema based on the entity definitions.

2.3 Server Configuration:

`server.port`: Specifies the port number on which the application will run. In this case, the application will run on port 8080.

2.4 CORS Configuration:

`spring.mvc.cors.allowed-origins`: Specifies the allowed origins for CORS requests. In this case, it is set to `"http://localhost:3000"`, which allows requests from a React app running on localhost with port 3000. Replace this value with the actual origin of your React app.

`spring.mvc.cors.allowed-methods`: Specifies the allowed HTTP methods for CORS requests. In this case, GET, POST, PUT, and DELETE methods are allowed.

`spring.mvc.cors.allowed-headers`: Specifies the allowed headers for CORS requests. In this case, all headers are allowed.

`spring.mvc.cors.allow-credentials`: Specifies whether to allow credentials (e.g., cookies) in CORS requests. In this case, it is set to `true`.

3. Usage:

The `application.properties` file is located in the `src/main/resources` directory of the project. You can modify the values of the configuration properties based on your specific database and server setup.

4. Technology Stack:

The technology stack used in the `application.properties` file includes:

Spring Boot: The framework used for developing the application and managing configurations.

MySQL: The database management system used for storing the HR Management System data.

Hibernate: The ORM (Object-Relational Mapping) framework used for mapping Java objects to database tables and performing database operations.

CORS: A mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the resource originated.

Project Documentation: pom.xml

1. Introduction:

The pom.xml file is an XML file used in Maven-based projects to define project information, dependencies, build configurations, and plugins. In the HR Management System, the pom.xml file is used to manage project dependencies, specify the Java version, and configure the build process.

2. Structure and Key Elements:

The pom.xml file consists of various elements that define project details and configurations. Here are the key elements and their significance:

2.1 Parent:

The <parent> element specifies the parent project from which the current project inherits configurations. In this case, the project inherits configurations from the "spring-boot-starter-parent" with version 2.5.2.

2.2 Project Information:

<groupId>: Specifies the unique identifier for the project's group. In this case, the group ID is "net.prabhu".

<artifactId>: Specifies the unique identifier for the project's artifact. In this case, the artifact ID is "Emps".

<version>: Specifies the version number of the project. In this case, the version is "0.0.1-SNAPSHOT".

<name>: Specifies the name of the project. In this case, the name is "Emps".

<description>: Provides a brief description of the project.

2.3 Properties:

The <properties> section allows you to define project-specific properties. In this case, it sets the Java version to 17.

2.4 Dependencies:

The <dependencies> section lists the project's dependencies. Here are the notable dependencies used in the HR Management System:

spring-boot-starter-data-jpa: Includes the necessary dependencies for working with Spring Data JPA.

spring-boot-starter-web: Includes the dependencies required for building web applications with Spring Boot.

mysql-connector-java: Provides the MySQL database driver to connect with the database.

spring-boot-starter-test: Includes dependencies for testing Spring Boot applications.

hibernate-validator: Provides validation capabilities for Hibernate entities.

jakarta.el-api: Specifies the Jakarta Expression Language (EL) API dependency.

junit-jupiter-api: Includes the JUnit Jupiter API for unit testing.

mockito-core: Provides mocking capabilities for unit testing.

2.5 Build Configuration:

The <build> section defines the build configuration for the project. It includes the spring-boot-maven-plugin, which allows the project to be packaged as an executable JAR file.

3. Usage:

The pom.xml file is located in the root directory of the project. It is used by Maven to manage project dependencies, compile source code, run tests, and build the project.

You can modify the dependencies section to add or remove dependencies based on your project requirements. Additionally, you can configure build-related plugins and properties in the pom.xml file.

4. Technology Stack:

The technology stack used in the pom.xml file includes:

Maven: The build automation tool used for managing dependencies and building the project.

Spring Boot: The framework used for developing the application and managing configurations.

MySQL: The database management system used for storing the HR Management System data.

Hibernate: The ORM (Object-Relational Mapping) framework used for mapping Java objects to database tables and performing database operations.

JUnit: The testing framework used for unit testing.

Mockito: A mocking framework used for creating mock objects in unit testing.

Frontend Documentation

Project Documentation: App.js

The App.js file serves as the entry point for the React application and defines the routing configuration using the React Router library. It determines which components to render based on the specified routes.

1. import Statements:

React is imported from the 'react' library to enable the use of React components.

BrowserRouter, Route, and Routes are imported from the 'react-router-dom' library. These components are used to set up and define the routing configuration.

2. Functional Component: App

The App component is a functional component that represents the root component of the application.

It returns the JSX (JavaScript XML) markup that defines the routing configuration.

3. Router Configuration: <Router>

The <Router> component wraps the entire application and enables routing functionality.

It provides a context for managing the application's routes.

4. Route Configuration: <Routes> and <Route>

The <Routes> component is used to define the routing configuration.

Within the <Routes> component, <Route> components are used to specify individual routes and the components to render for each route.

In the provided code snippet, there is one route defined:

The route with path="/", which maps to the root URL of the application.

The element prop specifies the component to render when the route is matched. In this case, the <EmployeeList> component is rendered when the root URL is accessed.

5. Commented Routes:

There are commented out <Route> components that indicate the possibility of additional routes for other pages, such as a Home page and a NotFound page.

These routes can be uncommented and configured to render the corresponding components for those pages.

This App.js file serves as the central point for defining the routing configuration in the React application. It determines which component to render based on the current URL path, allowing for navigation and rendering of different pages.

Project Documentation: EmployeeList.js

The EmployeeList.js component is responsible for displaying a list of employees, allowing users to update, delete, and add new employees. It utilizes React hooks for managing state and interacts with a RESTful API to perform CRUD operations on employee data.

1. Import Statements:

React is imported from the 'react' library to enable the use of React components.

useState and useEffect are imported from the 'react' library. These hooks are used for managing component state and performing side effects.

axios is imported to handle HTTP requests to the backend API.

The CSS file './EmployeeList.css' is imported to apply styling to the component.

2. Functional Component: EmployeeList

The `EmployeeList` component is a functional component that represents the main component responsible for rendering the employee list and handling user interactions.

It defines multiple state variables using the `useState` hook to manage different aspects of the component's state, such as the list of employees, the current page number for pagination, the ID of the employee being edited, the updated employee data, and the new employee data.

3. `useEffect` Hook: `fetchEmployeeData`

The `fetchEmployeeData` function is defined using the `async` keyword to fetch the employee data from the backend API when the component mounts.

It utilizes the `axios` library to send a GET request to the specified endpoint (`http://localhost:8080/api/employees`).

The retrieved employee data is stored in the `employees` state variable using the `setEmployees` function.

4. `useEffect` Hook: `Logging Employees`

Another `useEffect` hook is used to log the `employees` state variable whenever it changes.

This hook helps monitor and debug the employee data received from the backend API.

5. Event Handlers:

`handleUpdate`: This function is called when the user clicks the "Update" button for a specific employee. It sets the `editingEmployeeId` state variable to the ID of the employee being edited, enabling editing mode for that employee.

`handleInputChange`: This function is called when the user changes the value of an input field while in editing mode. It updates the `updatedEmployee` state variable with the changed field value.

`handleUpdateSubmit`: This function is called when the user clicks the "Submit" button after editing an employee. It sends a PUT request to the backend API with the updated employee data (`updatedEmployee`). The corresponding employee in the `employees` state variable is also updated with the new data.

`handleDelete`: This function is called when the user clicks the "Delete" button for a specific employee. It prompts a confirmation dialog and sends a DELETE request to the backend API to delete the employee if confirmed.

handleAddEmployee: This function is called when the user clicks the "Add Employee" button. It sends a POST request to the backend API with the new employee data (newEmployee). The newly added employee is appended to the employees state variable.

6. Pagination:

The component calculates the pagination values based on the current page number and the number of records per page (recordsPerPage). It determines the range of employees to display based on the current page.

The paginate function is called when the user clicks on a pagination button, updating the currentPage state variable.

7. Conditional Rendering:

If the employees array is empty, a message "No employees found." is rendered to indicate that no employees are available.

If the employees array is not empty, a table is rendered to display the employee data. The table rows are dynamically generated using the map function on the currentRecords array.

If editingEmployeeId is equal to the current employee's ID, input fields are rendered for editing the employee's data. Otherwise, the employee's data is displayed as text.

Based on the editing state, different buttons ("Update", "Cancel", "Delete") are rendered for each employee row.

8. Pagination and Add Employee UI:

Pagination buttons are rendered at the bottom of the table, allowing users to navigate between pages.

An "Add Employee" section is displayed below the pagination buttons, providing input fields for entering the details of a new employee. The "Add Employee" button triggers the handleAddEmployee function to add the new employee to the list.

The EmployeeList.js component integrates with a backend API to fetch, update, delete, and add employee data. It provides a user-friendly interface for managing employees, displaying data in a table format with pagination and allowing for editing and deletion of individual employees.

EMPFRONTEND

> node_modules

> public

▼ src

▼ components

EmployeeList.css

JS EmployeeList.js

App.css

JS App.js

JS App.test.js

index.css

JS index.js

logo.svg

JS reportWebVitals.js

JS setupTests.js

📁 .gitignore

{ } package-lock.json

{ } package.json

📄 README.md