Employee Management Application:

Technologies used:

Front end-**Visual code and react-Vite**

Backend-**Intelliji-Springboot**

Database**- MySQL and MySQL workbench**

To download a frontend project from GitHub using Visual Studio Code, you can follow these steps:

1. **Install Git**: Make sure you have Git installed on your system. You can download and install it from the official website.
2. **Open Visual Studio Code**: Launch Visual Studio Code on your computer.
3. **Open Terminal in Visual Studio Code**:
   * Press **Ctrl + ~** (or **View** > **Terminal**).
   * This will open a terminal window at the bottom of the Visual Studio Code interface.
4. **Navigate to the Desired Directory**: Use the **cd** command to navigate to the directory where you want to clone the project.th/to/your/directory
5. **Clone the Repository**: Use the **git clone** command followed by the URL of the GitHub repository you want to clone. For example:github.com/username/repository.git

Replace **username/repository** with the actual username and repository name of the GitHub project.

1. **Open the Project in Visual Studio Code**:
   * Once the cloning process is complete, you can open the project in Visual Studio Code by either selecting **File** > **Open Folder** and navigating to the cloned directory, or by using the command line:
2. **Install Dependencies (if any)**: If the project has dependencies listed in a **package.json** file, you'll need to install them. Open a new terminal in Visual Studio Code and run:

This will install all the required dependencies listed in the **package.json** file.

1. **Start the Development Server (if applicable)**: Depending on the project setup, you may need to start a development server to view the project in your browser.

**To download a backend project in IntelliJ IDEA, you typically follow these steps:**

1. **Install IntelliJ IDEA**: If you haven't already, download and install IntelliJ IDEA from the official website: <https://www.jetbrains.com/idea/download/>
2. **Open IntelliJ IDEA**: After installation, open IntelliJ IDEA from your applications folder.
3. **Clone the Project**: If the project is hosted on a version control system like Git, you'll need to clone it. Here's how to do it:

a. Go to **VCS** menu at the top.

b. Click on **Get from Version Control**.

c. Choose the version control system your project is hosted on (e.g., Git, GitHub, Bitbucket, etc.).

d. Enter the repository URL and specify the directory where you want to clone the project.

e. Click on **Clone**.

1. **Open Existing Project**: If you already have the project downloaded, you can simply open it from IntelliJ IDEA:

a. Go to **File** menu.

b. Click on **Open**.

c. Navigate to the directory where your project is located.

d. Select the project folder and click **OK**.

1. **Wait for IntelliJ to Load the Project**: Once you've opened the project, IntelliJ IDEA will take some time to index and load all the necessary files and dependencies.
2. **Configure Project Settings (if necessary)**: Depending on the project setup, you might need to configure project SDK, language level, project structure, etc. You can do this by going to **File** > **Project Structure**.
3. **Install Dependencies**: If the project relies on external dependencies managed by build tools like Maven or Gradle, IntelliJ will typically recognize them and prompt you to download them automatically. Otherwise, you might need to manually configure dependencies.
4. **Build the Project**: After setting up the project, you may need to build it to ensure that all dependencies are resolved and the project is compiled successfully. You can do this by going to **Build** > **Build Project** or by pressing **Ctrl + F9** (Cmd + F9 on Mac).
5. <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
   </dependency>  
     
   <dependency>  
    <groupId>com.mysql</groupId>  
    <artifactId>mysql-connector-j</artifactId>  
    <scope>runtime</scope>  
   </dependency>  
   <dependency>  
    <groupId>org.projectlombok</groupId>  
    <artifactId>lombok</artifactId>  
    <optional>true</optional>  
   </dependency>  
   <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-test</artifactId>  
    <scope>test</scope>  
   </dependency>

**JWT TOKEN:**

To login using a JWT (JSON Web Token), these are steps:

1. **Authentication:** Authenticate the user using their credentials (username/email and password).
2. **Generate Token:** Upon successful authentication, generate a JWT containing relevant user information and any additional data necessary (e.g., user roles, permissions, etc.).
3. **Token Sending:** Send the JWT to the client (usually in the response body or headers).
4. **Client Storage:** Store the JWT securely on the client-side, typically in local storage or a cookie.
5. **Token Sending with Subsequent Requests:** Include the JWT in the headers of subsequent requests to the server for protected resources.
6. **Token Validation:** On the server-side, validate the JWT for each incoming request to ensure that it has not been tampered with and that it is still valid.
7. **Access Control:** Use the information within the JWT to determine whether the user has permission to access the requested resource.

Back End:

Dependency :

**Jwt Token,Spring security:**

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-api</artifactId>  
 <version>0.12.5</version>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-impl</artifactId>  
 <version>0.12.5</version>  
 <scope>runtime</scope>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-jackson</artifactId>  
 <version>0.12.5</version>  
 <scope>runtime</scope>

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

**Setting up a backend for a Employment application using Spring Boot involves several steps:**

A. **Create a new Spring Boot project:**

* Use Spring Initializr (<https://start.spring.io/>) to create a new Spring Boot project.
* Choose the required dependencies :
* Spring Web,
* Spring Data JPA (for database interaction),
* LOMBOK
* MYSQL Driver
* B. **Define entities and data models:**
* Create entity classes for Bus, Booking, Passenger, and any other relevant entities.
* Use annotations such as **@Entity**, **@Id**, **@GeneratedValue**
* -**Passenger**
* **-Bus**
* **-Booking**
* ., to define the data models and relationships between entities.
* Define relationships such as one-to-many (e.g., a Bus has multiple Bookings) using annotations like **@OneToMany**, **@ManyToOne**, etc.
* In our project:
* @ManyToMany mapping between user and roles.

C. **Implement RESTful APIs using Spring Boot's controllers:**

* Create Spring Boot controller classes to handle various endpoints for bus booking requests, user authentication, and data retrieval.
* Use annotations such as **@RestController**, **@RequestMapping**, **@GetMapping**, **@PostMapping**, etc., to define endpoints and HTTP methods.
* Implement methods within the controllers to perform actions like creating bookings, retrieving bus schedules, authenticating users, etc.

D. **Set up a database:**

* Choose a database (e.g., MySQL) to store bus schedules, booking information, and user data.
* Configure database connection properties in the **application.properties** file:
* spring.datasource.username=root  
  spring.datasource.password=root3306  
  spring.jpa.show-sql: true  
  spring.jpa.hibernate.ddl-auto=update  
  server.port=8083  
    
  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

Use Spring Data JPA repositories or MongoDB repositories to interact with the database.

* Define database tables/collections corresponding to your entities, and let Spring Boot handle the schema creation/update using Hibernate or MongoDB's schema-less nature.

Insert MySQL comment for many-many mappings 🡪 user- roles table

|  |
| --- |
| create database ema;  use ema;  insert into users(email,name,password,username)values("admin@gmail.com","admin","$2a$10$F.NlXYddsdivrBWoC.6L4uoF9XB89s5ZWLJcqZozR6IFbEQkR9awW","admin");  insert into users(email,name,password,username)values("john@gmail.com","user","$2a$10$d.8g5aedjLEwkjyM8G6TG.FcPrw3/vNSlSUa1ZbVaQx9kFaj2yn8q","john");  insert into roles(name)values("ROLE\_ADMIN");  insert into roles(name)values("ROLE\_USER");  insert into users\_roles values(1,1);  insert into users\_roles values(1,2);  insert into users\_roles values(2,2); |

**Front end:**

|  |
| --- |
| **Note**:  If my application throws AXIOS ERROR-403,you can directly go to Browser🡪Inspect-🡪Application🡪clear Local storage and session storage.So that,AXIOS ERROR will be resolved. |

Terminals:

npm create vite@latest employee\_management

npm install axios –library

npm install react-router-dom

npm install bootstrap

**URL:**

**Front Page**: http://localhost:3000

**Sign up**: <http://localhost:3000/reg>

New User can sign up.

**Login**: <http://localhost:3000/login>

**ADMIN:**

**Username**: admin

Password: admin

**User:**

Registered User(by Sign Up)

ADMIN:

<http://localhost:3000/list> 🡪 It displays all the employees list.

1.Add employee: <http://localhost:3000/add-emp>

It allows admin to add the employees.

2.update: [http://localhost:3000/update-emp/{empId}](http://localhost:3000/update-emp/%7bempId%7d)

3.delete:It deletes on refreshing.

User:

<http://localhost:3000/list> 🡪 It displays all the employees list.

View: [http://localhost:3000/view/{empId}](http://localhost:3000/view/%7bempId%7d)

To view the particular Employee.

**Note**:Once entered as admin,please log out and enter as user.

**Logout**:

After logout,you can go to login to go to Admin Component.

Because,once loggedout,both local & session storage will be cleared.

isAuth & isAdmin:

It displays update and delete for admin.

It displays view for users.

**Validation at Front End:**

 try {

      if (name === "") {

        alert("name should be filled out")

        return false;

      } else if (username === "") {

         alert("username should be filled out")

        return false;

      } else if (email === "") {

         alert("email should be filled out")

        return false;

      } else if (password === "") {

         alert("password should be filled out")

        return false;

If the given input is empty ,it wont get registered.

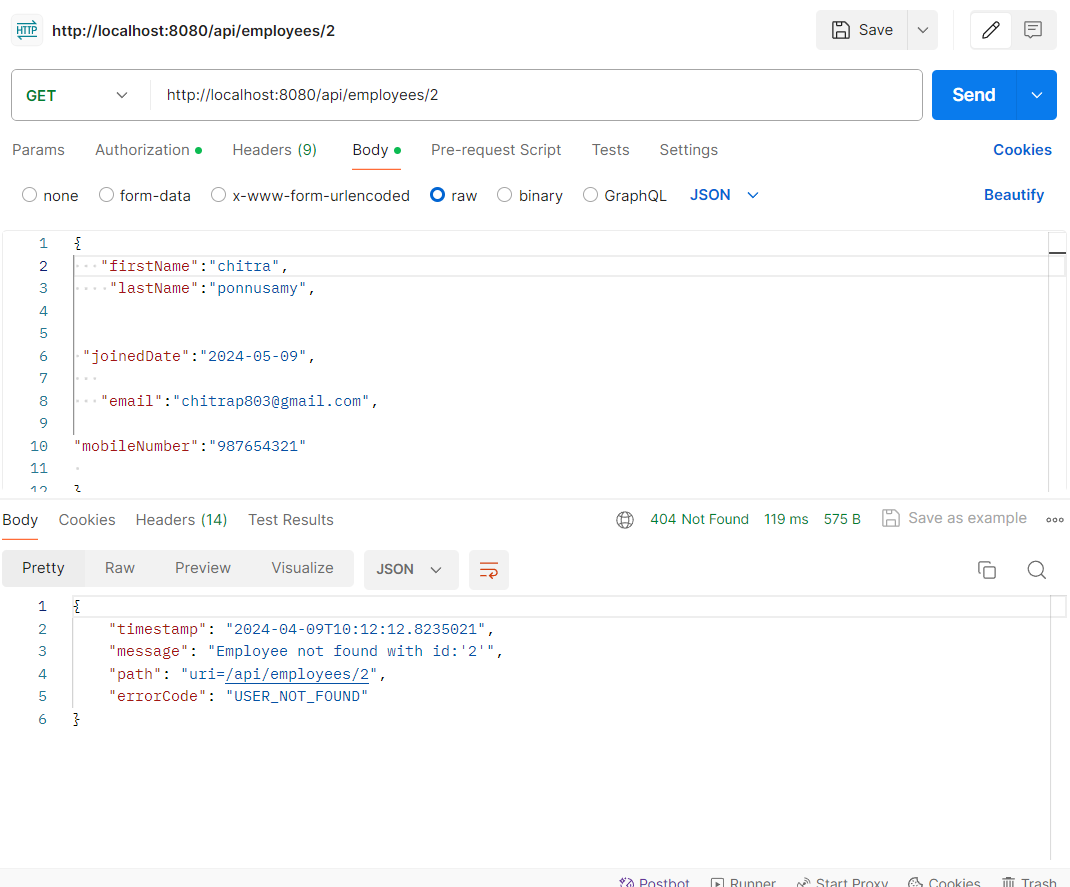
It accepts only unique email id of registered user.

If username is invalid ,it wont login the page.

**Error Handling at Back End:**

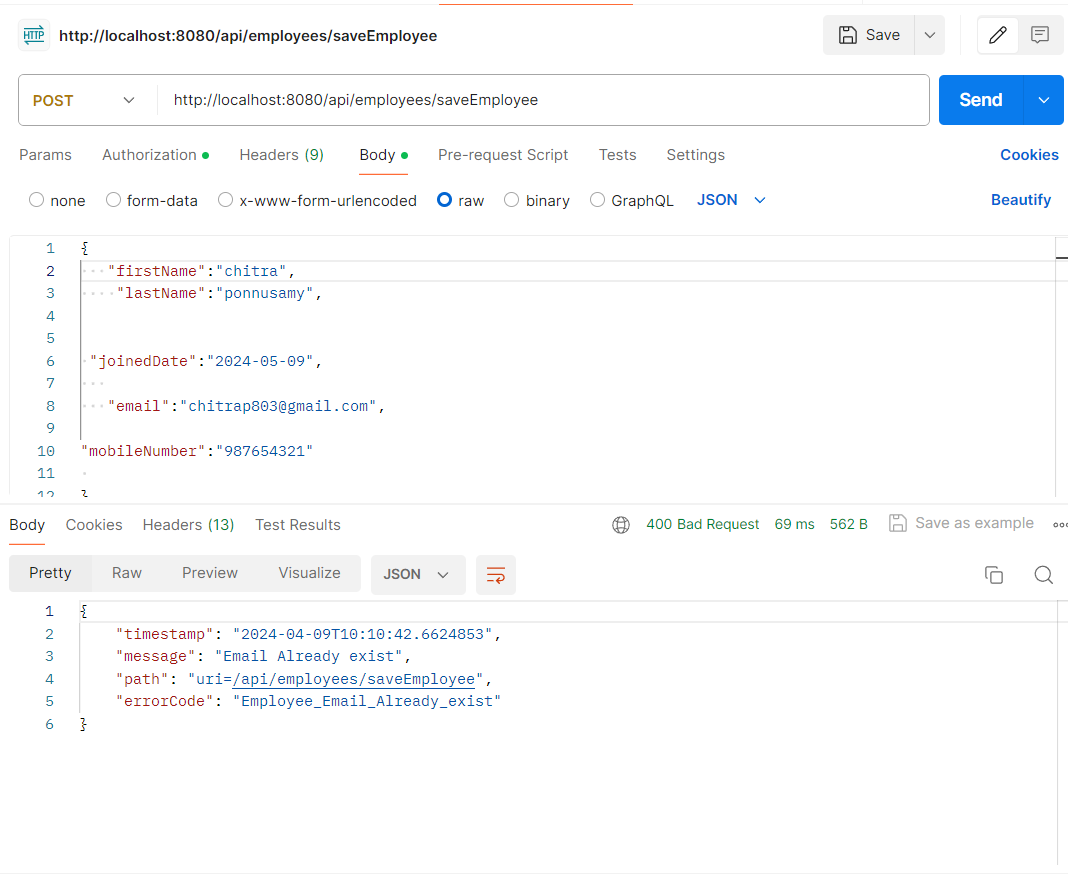
**1.ResourceNotFoundException(404)**

**When invalid Id is given,this exception occurs.**



**2.Email Already Exist exception(400)**

**When user registered with same emailId.**



**3.Validation at Back end:**

**@NotEmpty**

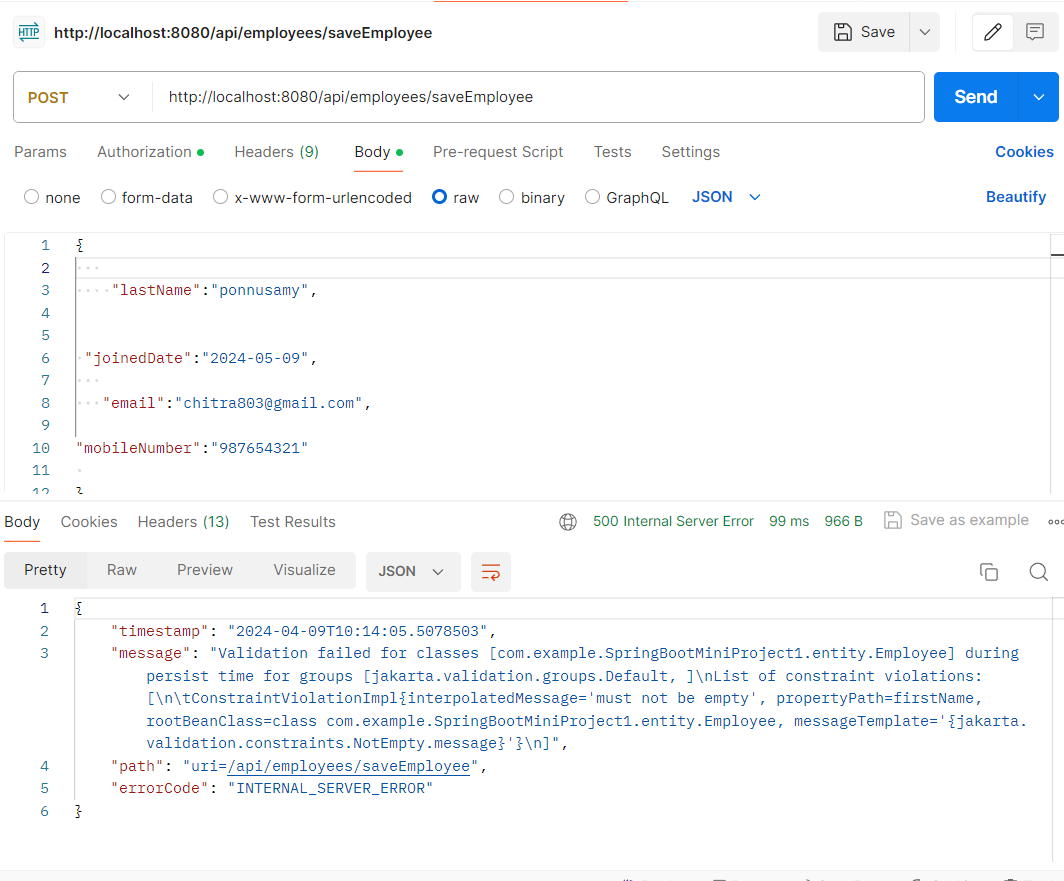
**@Email**

**@ExceptionHandler**

**Global Exception Handler class**

**To handle the exceptions by using custom error details class.**

public class Employee {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long empId;  
 @NotEmpty  
 private String firstName;  
 @NotEmpty  
 private String lastName;  
  
  
 @JsonDeserialize(using = LocalDateTimeDeserializer.class)  
 @JsonSerialize(using = LocalDateSerializer.class)  
  
 @JsonFormat(shape = JsonFormat.Shape.*STRING*, pattern = "yyyy-mm-dd")  
 private LocalDate joinedDate;  
 @NotEmpty  
 @Email  
 private String email;  
@NotEmpty  
 private String mobileNumber;



**API ENDPOINTS:**

**POST**: <http://localhost:8080/api/employees/saveEmployee>

**Get**: <http://localhost:8080/api/employees>

**GetById**: [http://localhost:8080/api/employees/{empId}](http://localhost:8080/api/employees/%7bempId%7d)

**Put**: [http://localhost:8080/api/employees/{empId}](http://localhost:8080/api/employees/%7bempId%7d)

**Delete**: [http://localhost:8080/api/employees/{empId}](http://localhost:8080/api/employees/%7bempId%7d)

**Login**: <http://localhost:8080/api/auth/login>

**Register:** http://localhost:8080/api/auth/register

**Unit test cases:**

**Run the test case for REPOSITORY,SERVICES (by mocking the repository)**

**By adding h2 dependency:**

dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
</dependency>