Employee Management Application Documentation

Introduction

This document provides an overview and detailed instructions for using the Employee Management Application. The application is designed to manage employee information, including registration and login functionalities for users. employee Management System is open to admins, and regular employees. Among all users, only the admins have all privileges to access all the information of EMS. So the admins will insert, update, remove the employees, departments, generate reports and whereas other users will have limited roles. Once the user's login they can perform few tasks specific to their role.

Technologies Used

- **Backend**: Spring Boot
- **Frontend**: React.js, HTML, CSS
- **Database**: MySQL
- **Authentication**: JWT (JSON Web Tokens)

Project Overview

User Roles

1. **Administrator**: Has full control over the system, including adding new employees, employee details, and viewing all employee information.
2. **Employee**: Can view and update their own details, apply for leave, and view the status of their leave applications.

Modules and Features

Employee Module

The Employee Module handles all functionality related to employee data management. This includes the following CRUD (Create, Read, Update, Delete) operations:

- **Create**: Add new employee records.
- **Read**: View details of all employees (admin) or view personal details (employee).
- **Update**: Modify existing employee information.
- **Delete**: Remove employee records from the system.

For API documentation to Spring Boot application, we need to include the Swagger dependencies and configure it in our project. Swagger, through the OpenAPI specification, allows us to document our APIs effectively and provides a user-friendly UI for testing them.

`http://localhost:8080/swagger-ui/`

This will provide a user-friendly interface where you can view and interact with your API documentation.

Example of API Documentation

Here’s how the Swagger documentation for the `UserController` endpoints might look:

- **POST /api/users/register**: Registers a new user.
  - **Parameters**: User object in the request body.
  - **Responses**:
    - `200 OK`: Successfully registered the user.
    - `400 Bad Request`: Invalid input data.

- **POST /api/users/login**: Authenticates a user and returns a JWT token.
  - **Parameters**: User object in the request body.
  - **Responses**:
    - `200 OK`: Successfully authenticated and returned the JWT token.
    - `401 Unauthorized`: Invalid credentials.

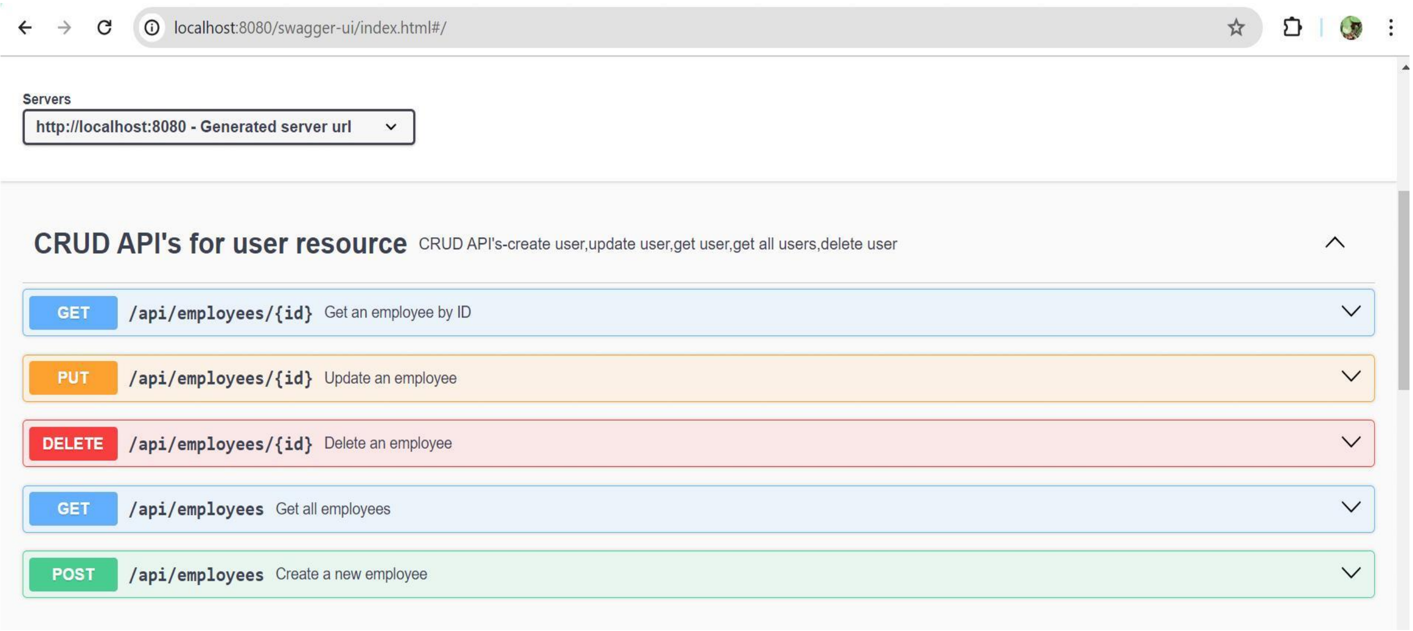
Example `SwaggerConfig` Class

java

This configuration ensures that Swagger scans your specified package for API endpoints and generates the documentation accordingly.

By following these steps, you will have a well-documented API with Swagger integrated into your Spring Boot application. This will help developers understand and interact with your API more effectively.

Swagger:



localhost:8080/swagger-ui/index.html#/

Servers

http://localhost:8080 - Generated server url

CRUD API's for user resource CRUD API's-create user,update user,get user,get all users,delete user

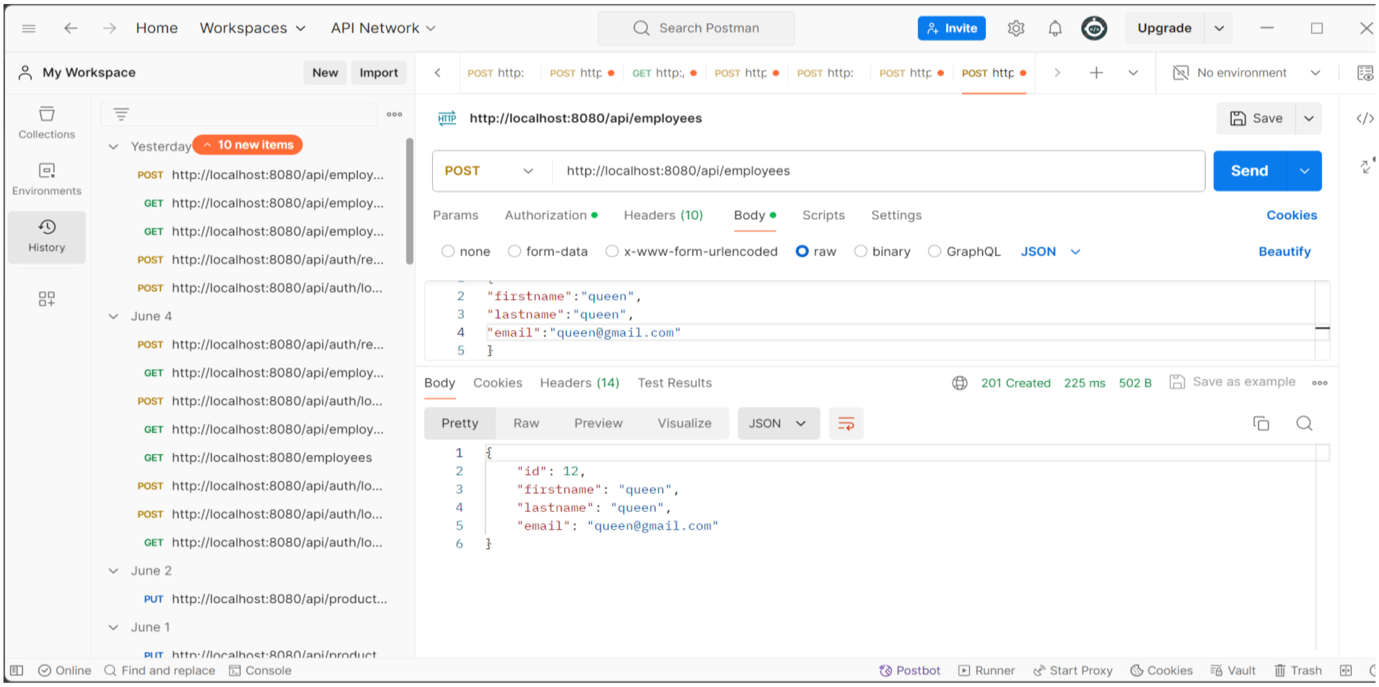
GET `/api/employees/{id}` Get an employee by ID

PUT `/api/employees/{id}` Update an employee

DELETE `/api/employees/{id}` Delete an employee

GET `/api/employees` Get all employees

POST `/api/employees` Create a new employee



Yesterday 10 new items

- POST http://localhost:8080/api/employ...
- GET http://localhost:8080/api/employ...
- GET http://localhost:8080/api/employ...
- POST http://localhost:8080/api/auth/re...
- POST http://localhost:8080/api/auth/lo...

June 4

- POST http://localhost:8080/api/auth/re...
- GET http://localhost:8080/api/employ...
- POST http://localhost:8080/api/auth/lo...
- GET http://localhost:8080/api/employ...
- GET http://localhost:8080/employees
- POST http://localhost:8080/api/auth/lo...
- POST http://localhost:8080/api/auth/lo...
- GET http://localhost:8080/api/auth/lo...

June 2

- PUT http://localhost:8080/api/product...

June 1

- PUT http://localhost:8080/api/product...

http://localhost:8080/api/employees

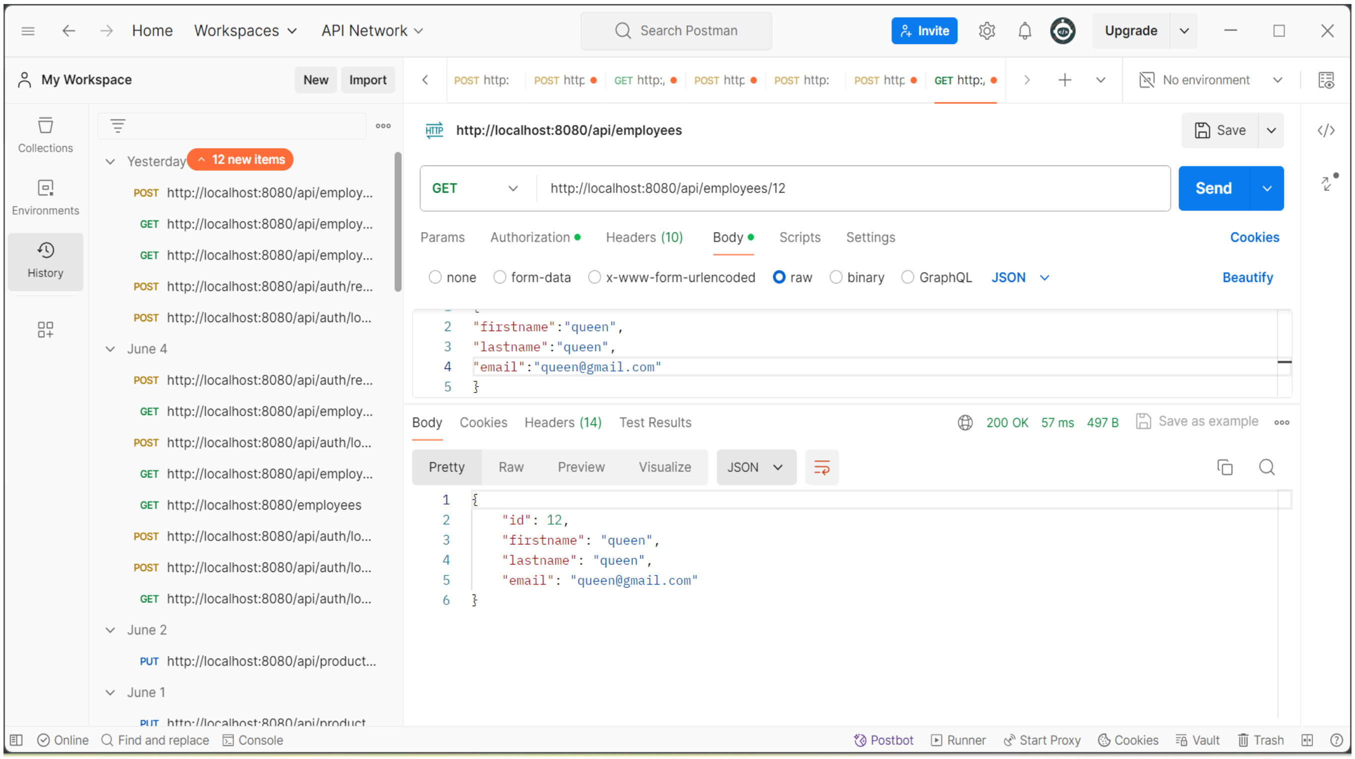
POST http://localhost:8080/api/employees

```
"firstname":"queen",
"lastname":"queen",
"email":"queen@gmail.com"
}
```

Body Cookies Headers (14) Test Results 201 Created 225 ms 502 B

```
{
    "id": 12,
    "firstname": "queen",
    "lastname": "queen",
    "email": "queen@gmail.com"
}
```

GET API:By Id:


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Environments
History
Yesterday
12 new items
POST http://localhost:8080/api/employ...
GET http://localhost:8080/api/employ...
GET http://localhost:8080/api/employ...
POST http://localhost:8080/api/auth/re...
POST http://localhost:8080/api/auth/lo...
June 4
POST http://localhost:8080/api/auth/re...
GET http://localhost:8080/api/employ...
POST http://localhost:8080/api/auth/lo...
GET http://localhost:8080/api/employ...
GET http://localhost:8080/employees
POST http://localhost:8080/api/auth/lo...
POST http://localhost:8080/api/auth/lo...
GET http://localhost:8080/api/auth/lo...
June 2
PUT http://localhost:8080/api/product...
June 1
No environment
Save
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
Beautify
Pretty
Raw
Preview
Visualize
Save as example
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

http://localhost:8080/api/employees

GET http://localhost:8080/api/employees/12

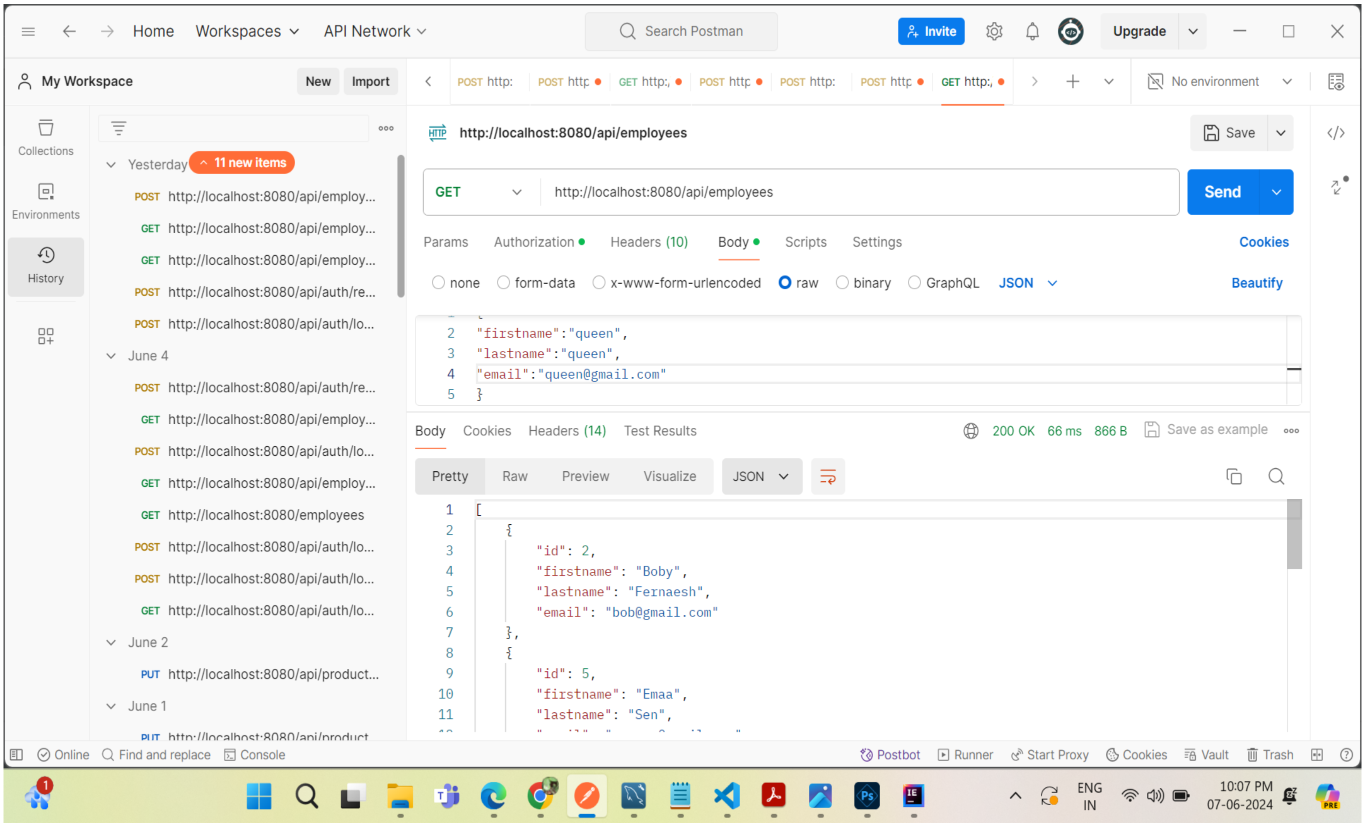
○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
"firstname":"queen",
"lastname":"queen",
"email":"queen@gmail.com"
}
```

Body Cookies Headers (14) Test Results 200 OK 57 ms 497 B

```
{
    "id": 12,
    "firstname": "queen",
    "lastname": "queen",
    "email": "queen@gmail.com"
}
```

GET all employees


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Environments
History
Yesterday
11 new items
POST http://localhost:8080/api/employ...
GET http://localhost:8080/api/employ...
GET http://localhost:8080/api/employ...
POST http://localhost:8080/api/auth/re...
POST http://localhost:8080/api/auth/lo...
June 4
POST http://localhost:8080/api/auth/re...
GET http://localhost:8080/api/employ...
POST http://localhost:8080/api/auth/lo...
GET http://localhost:8080/api/employ...
GET http://localhost:8080/employees
POST http://localhost:8080/api/auth/lo...
POST http://localhost:8080/api/auth/lo...
GET http://localhost:8080/api/auth/lo...
June 2
PUT http://localhost:8080/api/product...
June 1
No environment
Save
Send
Params
Authorization
Headers (10)
Body
Scripts
Settings
Cookies
Beautify
Pretty
Raw
Preview
Visualize
Save as example
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash
ENG
IN
10:07 PM
07-06-2024

http://localhost:8080/api/employees

GET http://localhost:8080/api/employees

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
"firstname":"queen",
"lastname":"queen",
"email":"queen@gmail.com"
}
```

Body Cookies Headers (14) Test Results 200 OK 66 ms 866 B

```
[
    {
        "id": 2,
        "firstname": "Boby",
        "lastname": "Fernaesh",
        "email": "bob@gmail.com"
    },
    {
        "id": 5,
        "firstname": "Emaa",
        "lastname": "Sen",
```

Key Features

- **Role-Based Access Control**: Ensures that only authorized users can perform specific operations. For example, only administrators can modify or delete employee records.

- **Employee Self-Service**: Employees can view and manage their own data, including applying for leave and checking leave status.

Application Setup

Backend (Spring Boot)

1. **Configure MySQL Database**: Ensure MySQL is installed and running. Create a database named `employee_management`.

Application Properties Configuration:
Open
`src/main/resources/application.properties` and configure your MySQL database connection:
properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/employee_management
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

2. Build and Run the Application: Use Maven or your preferred build tool to build and run the Spring Boot application:

3. Defined the entity classes.

4. Created repository interfaces.

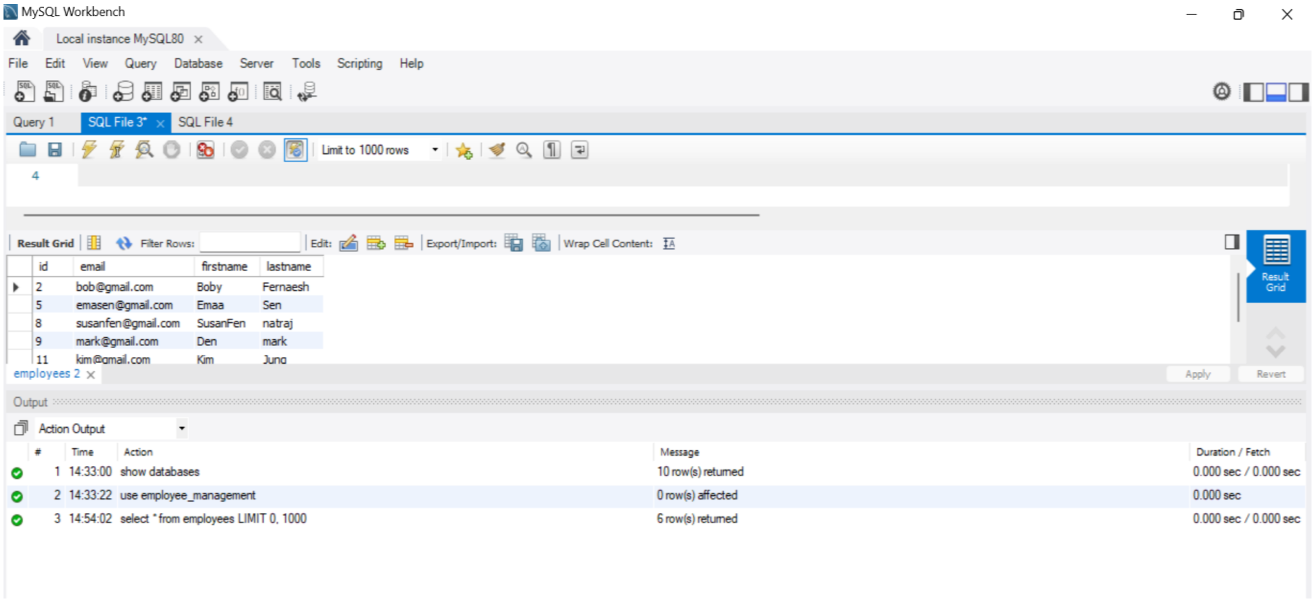
5. Implement service classes.

6. Create controller classes to handle incoming HTTP requests.

7. Use initialization scripts or application logic to insert initial data into the database.

Below, I will provide an example of how to set up and insert content into the database.

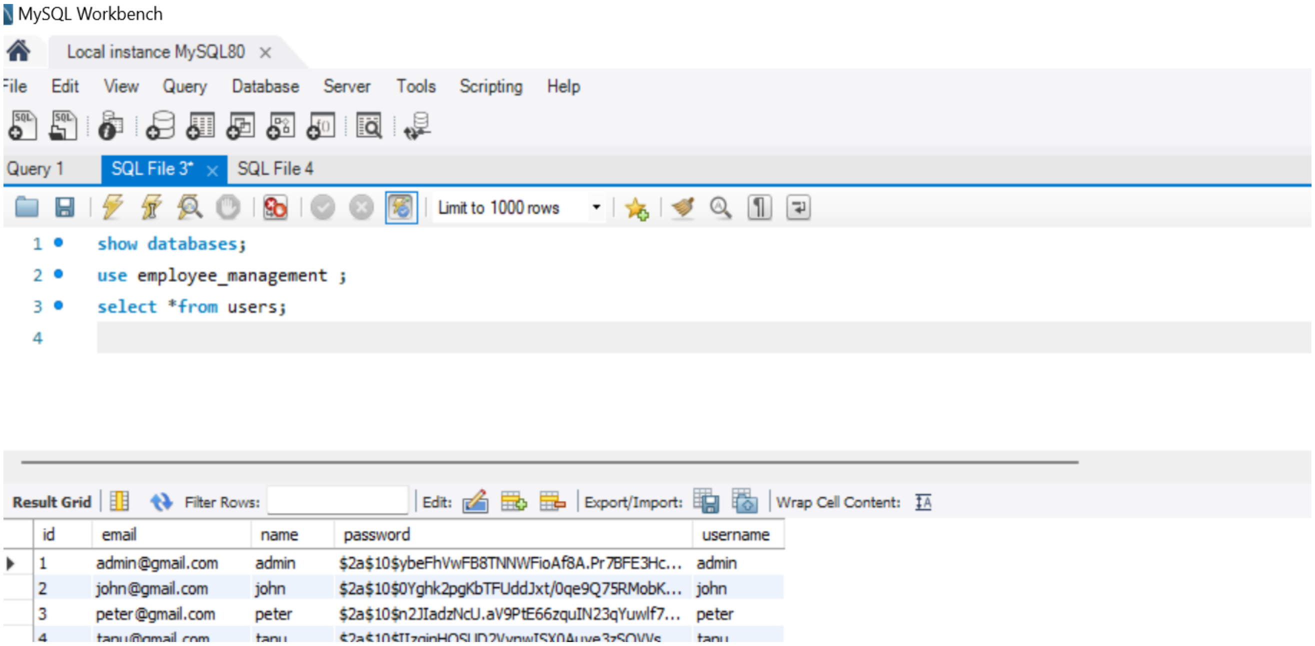
Employee Table



id	email	firstname	lastname
2	bob@gmail.com	Boby	Fernaesh
5	emasen@gmail.com	Emaa	Sen
8	susanfen@gmail.com	SusanFen	natraj
9	mark@gmail.com	Den	mark
11	kim@gmail.com	Kim	Jung

#	Time	Action	Message	Duration / Fetch
1	14:33:00	show databases	10 row(s) returned	0.000 sec / 0.000 sec
2	14:33:22	use employee_management	0 row(s) affected	0.000 sec
3	14:54:02	select * from employees LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec

Users



```
show databases;
use employee_management ;
select *from users;
```

id	email	name	password	username
1	admin@gmail.com	admin	$2a$10$ybeFhVwFB8TNNWFioAf8A.Pr7BFE3Hc...	admin
2	john@gmail.com	john	$2a$10$0Yghk2pgKbTFUddJxt/0qe9Q75RMobK...	john
3	peter@gmail.com	peter	$2a$10$n2JIadzNcU.aV9PtE66zquIN23qYuwlf7...	peter
4	tanu@gmail.com	tanu	$2a$10$IIzqinHOSUD2VvnwISX0Auve3zSOVVs	tanu

Security and Authentication

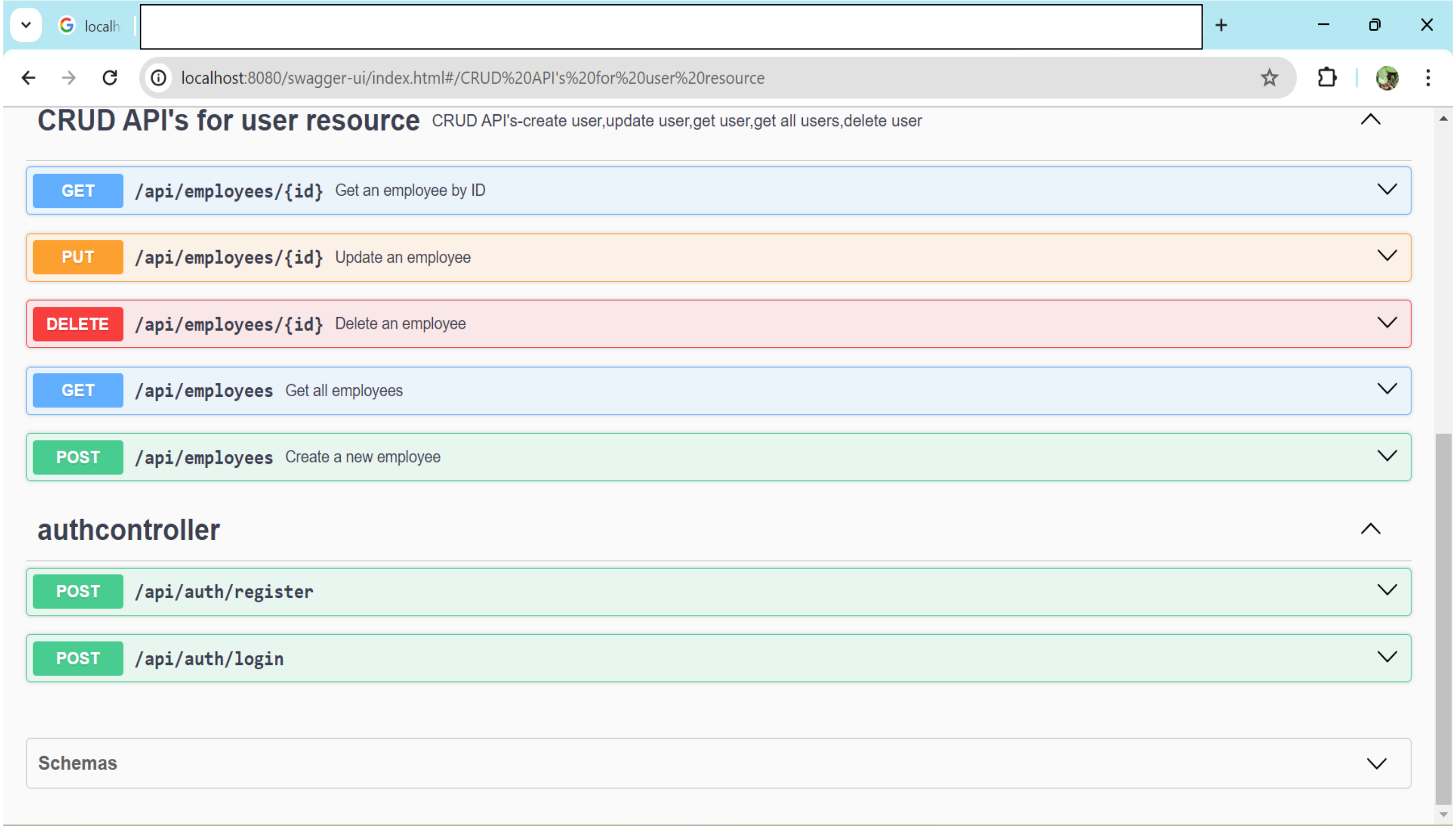
Password Encryption

To enhance security, user passwords are encrypted before storing them in the database using Spring Security’s `BCryptPasswordEncoder`.

JWT Authentication

JWT (JSON Web Tokens) are used to secure access to the application’s endpoints. Here’s how the JWT authentication process is integrated:

1. **User Registration**:
   - A new user (admin or regular) can register by providing a username and password.
   - The password is encrypted before being saved to the database.
2. **User Login**:
   - The user provides their username and password to log in.
   - If the credentials are correct, a JWT is generated and returned to the user.
3. **Accessing Protected Endpoints**:
   - The JWT must be included in the `Authorization` header as `Bearer <token>` for all requests to protected endpoints.



localhost:8080/swagger-ui/index.html#/CRUD%20API's%20for%20user%20resource

CRUD API's for user resource CRUD API's-create user,update user,get user,get all users,delete user

GET `/api/employees/{id}` Get an employee by ID

PUT `/api/employees/{id}` Update an employee

DELETE `/api/employees/{id}` Delete an employee

GET `/api/employees` Get all employees

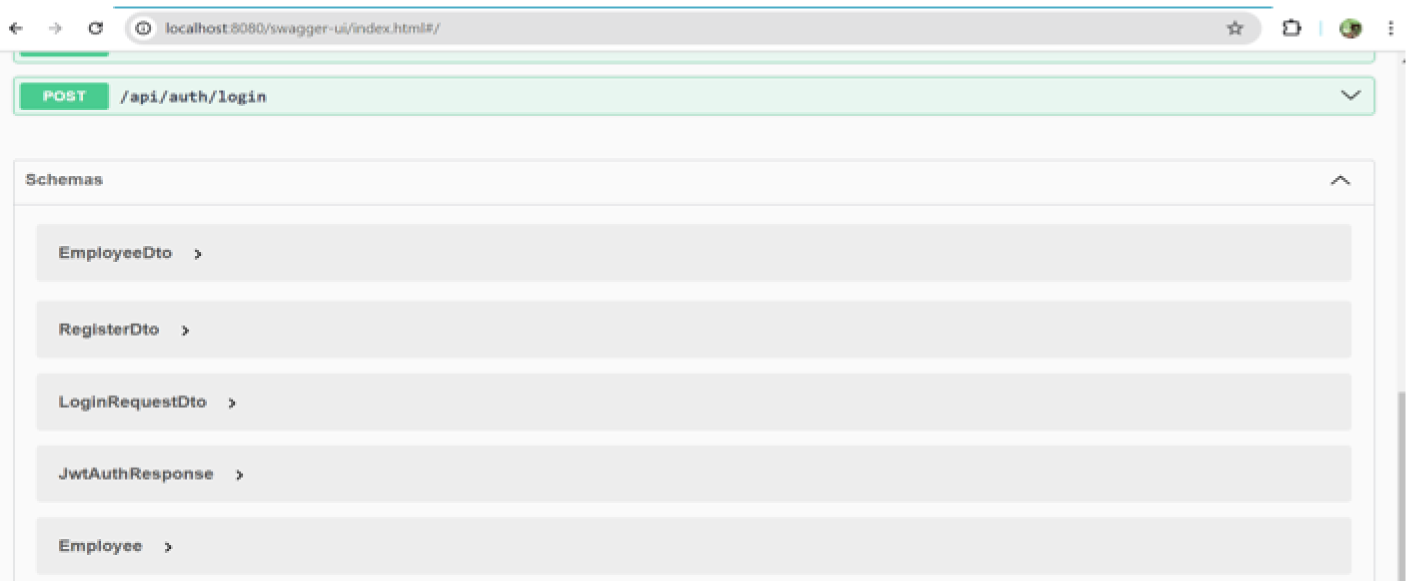
POST `/api/employees` Create a new employee

authcontroller

POST `/api/auth/register`

POST `/api/auth/login`

Schemas



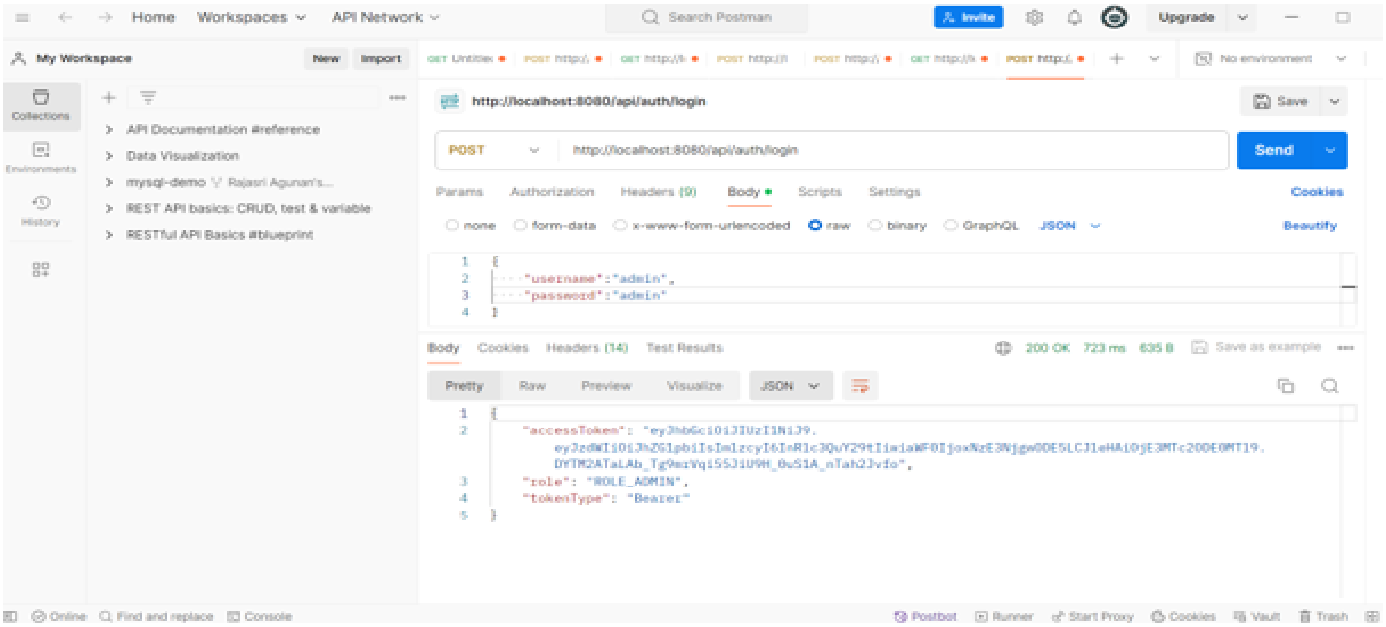
localhost:8080/swagger-ui/index.html#/

POST /api/auth/login

Schemas

- EmployeeDto
- RegisterDto
- LoginRequestDto
- JwtAuthResponse
- Employee

Login Token Generation



http://localhost:8080/api/auth/login

POST http://localhost:8080/api/auth/login

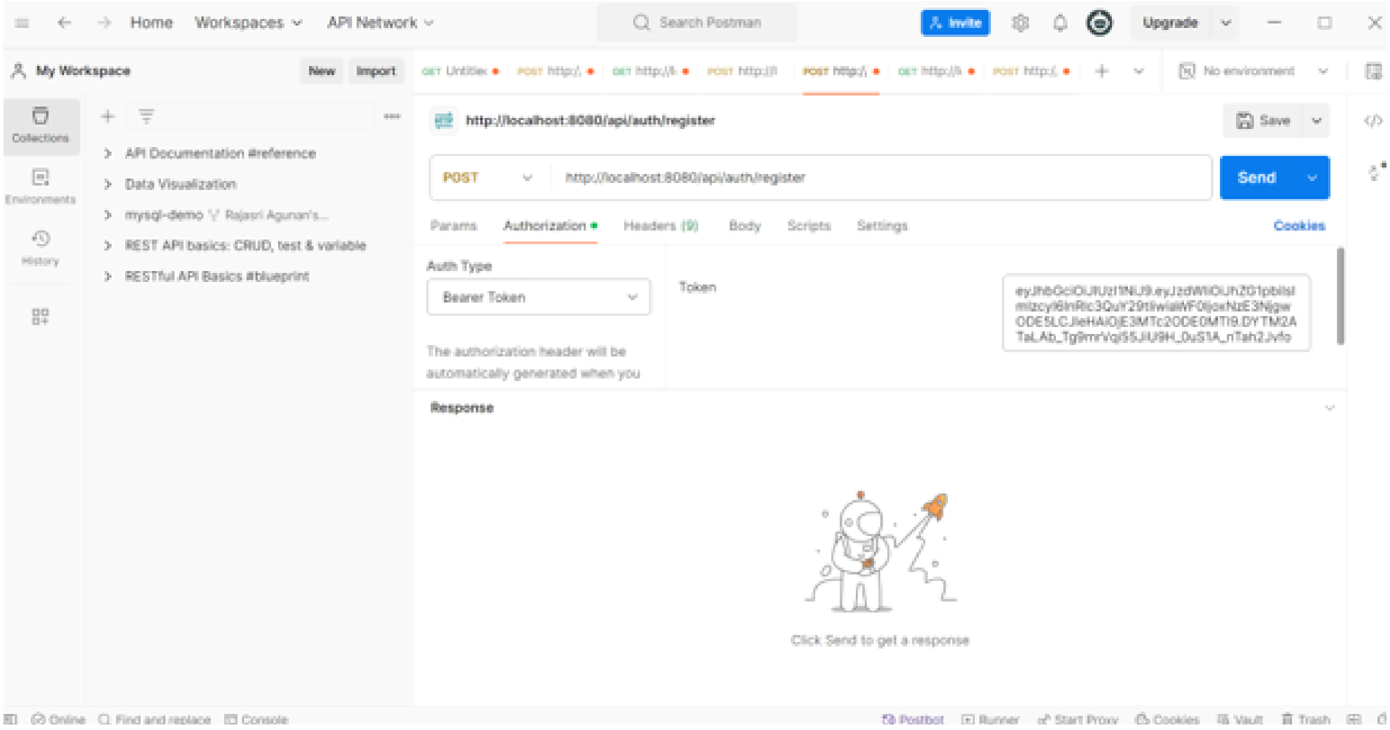
○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "username":"admin",
    "password":"admin"
}
```

200 OK 723 ms 635 B

JSON

```
{
    "accessToken": "eyJhbGciOiJIUzI1NiJ9.
        eyJzdWIiOiJhZG1pbiIsImlzcyI6InRlc3QuY29tIiwiaWF0IjoxNzE3NjgwODE5LCJleHAiOjE3MTc2ODE0MTI9.
        DYTM2ATaLAb_Tg9mrVqi55JiU9H_0uS1A_nTah2Jvfo",
    "role": "ROLE_ADMIN",
    "tokenType": "Bearer"
}
```



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

POST http://localhost:8080/api/auth/register

Auth Type: Bearer Token

The authorization header will be automatically generated when you

Token: eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImlzcyI6InRlc3QuY29tIiwiaWF0IjoxNzE3NjgwODE5LCJleHAiOjE3MTc2ODE0MTI9.DYTM2ATaLAb_Tg9mrVqi55JiU9H_0uS1A_nTah2Jvfo

Click Send to get a response

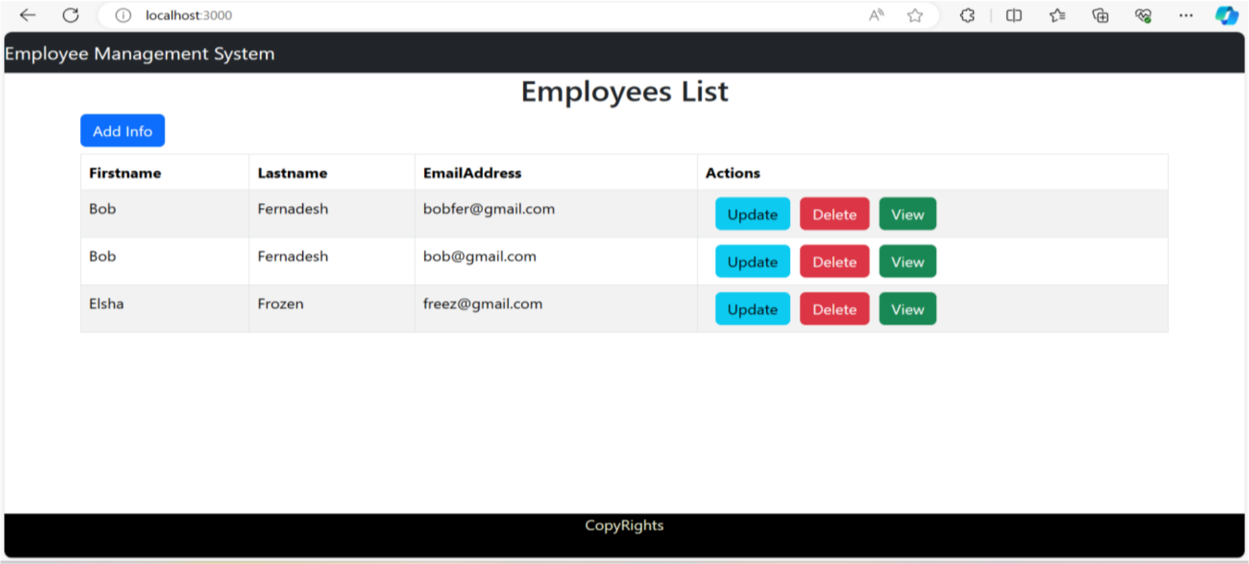
```
cd employee--app
npm install
```

3.Run the React Application: Start the development server:

```
npm run dev
```

The application should now be running on `http://localhost:3000`.

Employee Management Application



Employee Management System

Employees List

Firstname	Lastname	EmailAddress	Actions
Bob	Fernadesh	bobfer@gmail.com	Update Delete View
Bob	Fernadesh	bob@gmail.com	Update Delete View
Elsha	Frozen	freez@gmail.com	Update Delete View

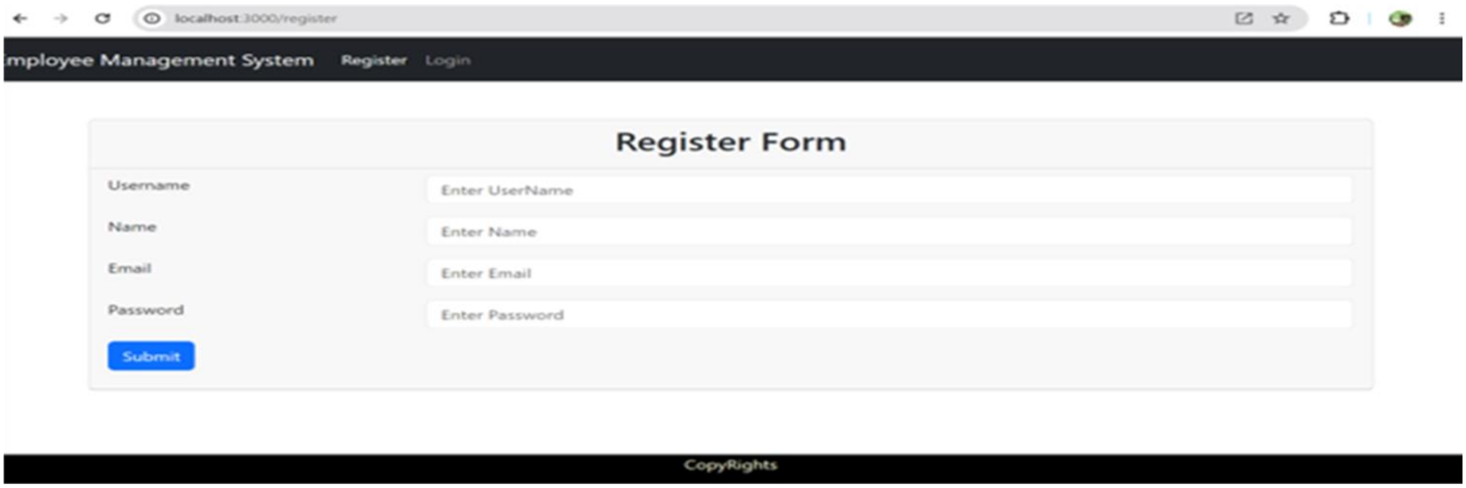
CopyRights

Application Usage

User Registration

1. **Access the Registration Page**: Open your web browser and navigate to `http://localhost:3000/register`.
2. **Fill Out Registration Form**: Enter your desired username and password. Click the "Register" button to create a new account.
3. **Successful Registration**: Upon successful registration, you will be redirected to the login page.

Registration Form to Fill



localhost:3000/register

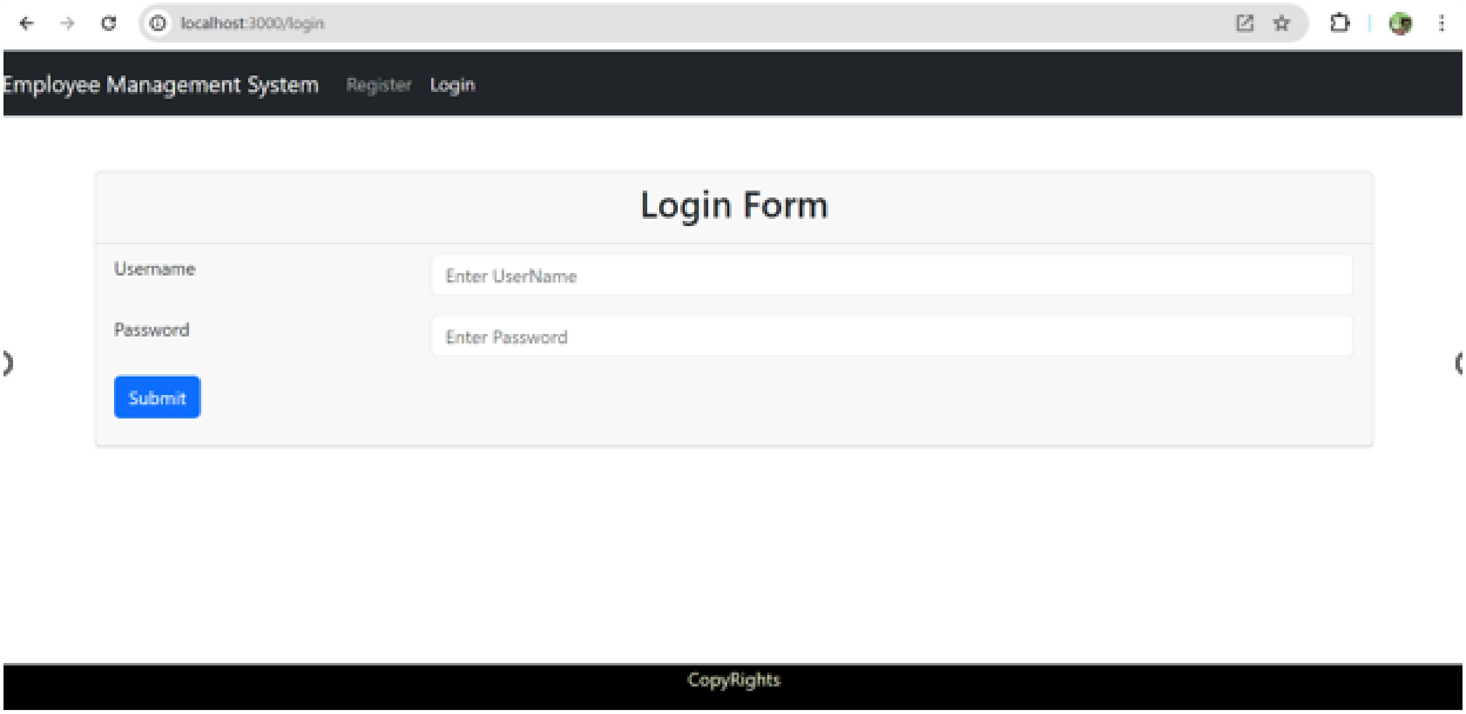
Register Form

Username	Enter UserName
Name	Enter Name
Email	Enter Email
Password	Enter Password

User Login

1. **Access the Login Page**: Navigate to `http://localhost:3000/login`.
2. **Fill Out Login Form**: Enter your username and password. Click the "Login" button to authenticate.
3. **Successful Login**: If the credentials are correct, you will be redirected to the employee dashboard.

User Admin Login Form

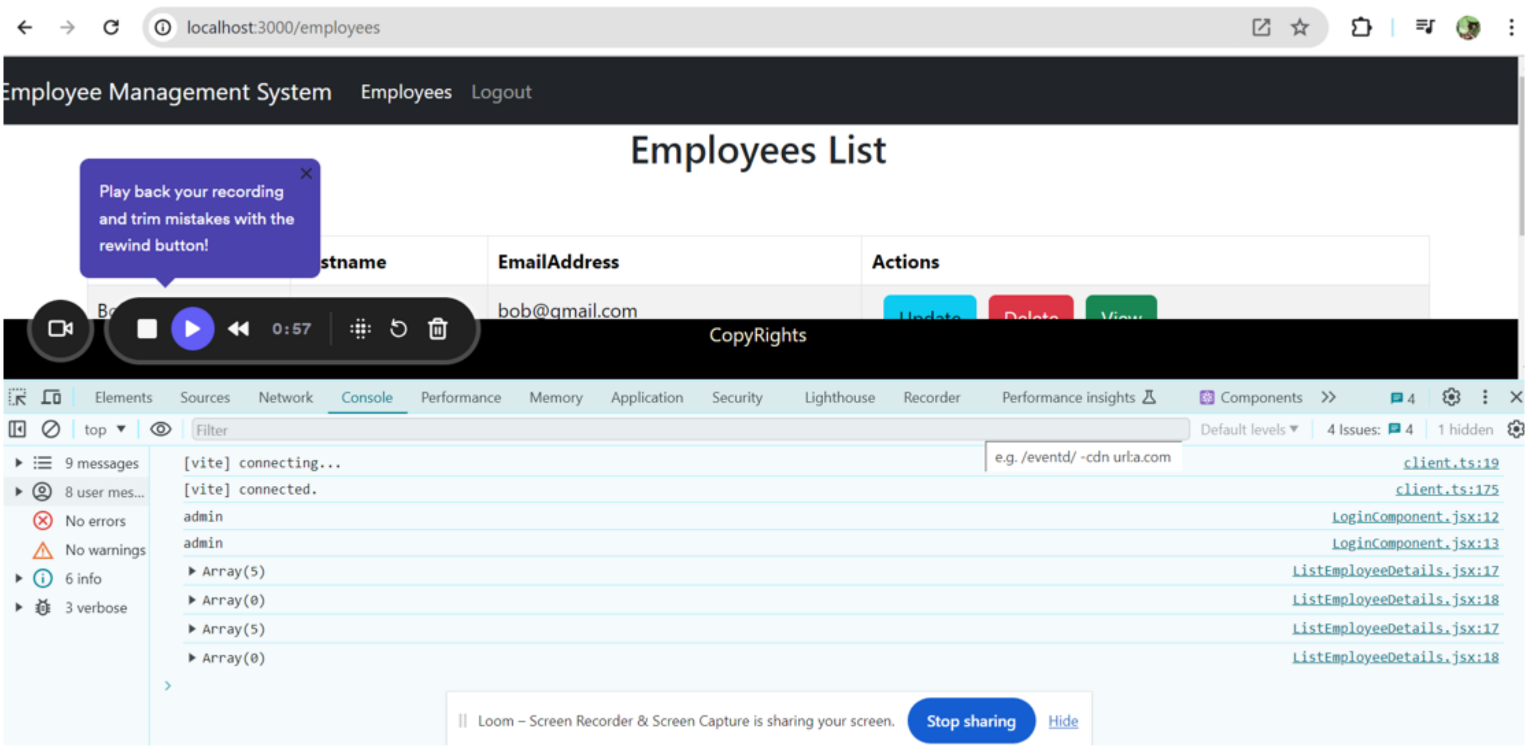


localhost:3000/login

Login Form

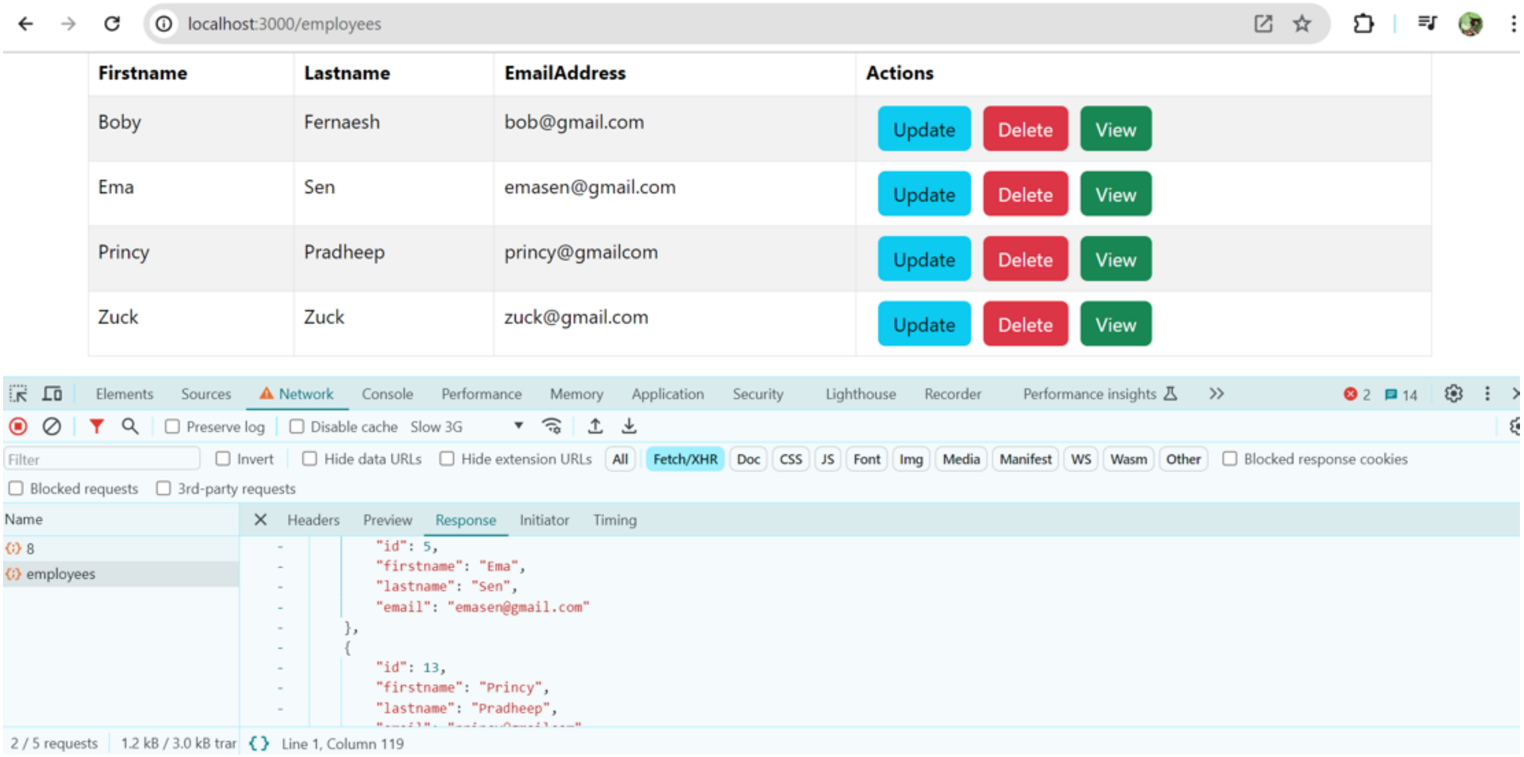
Username	Enter UserName
Password	Enter Password

Login Response



```
[vite] connecting...                                   client.ts:19
[vite] connected.                                      client.ts:175
admin                                                  LoginComponent.jsx:12
admin                                                  LoginComponent.jsx:13
▸ Array(5)                                             ListEmployeeDetails.jsx:17
▸ Array(0)                                             ListEmployeeDetails.jsx:18
▸ Array(5)                                             ListEmployeeDetails.jsx:17
▸ Array(0)                                             ListEmployeeDetails.jsx:18
```

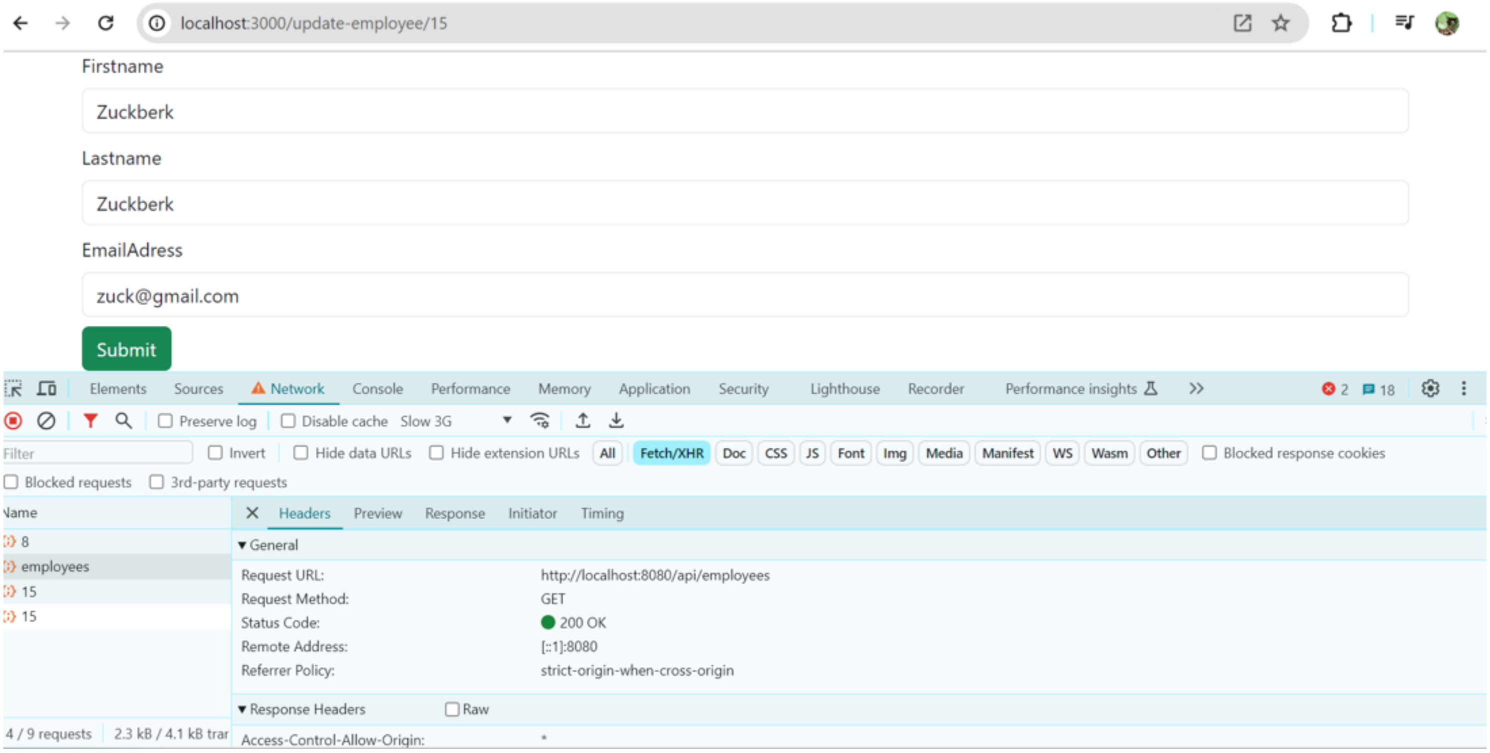
Response



Firstname	Lastname	EmailAddress	Actions
Boby	Fernaesh	bob@gmail.com	Update Delete View
Ema	Sen	emasen@gmail.com	Update Delete View
Princy	Pradheep	princy@gmailcom	Update Delete View
Zuck	Zuck	zuck@gmail.com	Update Delete View

```
        "id": 5,
        "firstname": "Ema",
        "lastname": "Sen",
        "email": "emasen@gmail.com"
    },
    {
        "id": 13,
        "firstname": "Princy",
        "lastname": "Pradheep",
```

Add Form Response


localhost:3000/update-employee/15

Firstname

Zuckberk

Lastname

Zuckberk

EmailAdress

zuck@gmail.com

Submit
Elements Sources Network Console Performance Memory Application Security Lighthouse Recorder Performance insights
□ Preserve log □ Disable cache Slow 3G
Filter □ Invert □ Hide data URLs □ Hide extension URLs All Fetch/XHR Doc CSS JS Font Img Media Manifest WS Wasm Other □ Blocked response cookies
□ Blocked requests □ 3rd-party requests
Name
8
employees
15
15
Headers Preview Response Initiator Timing

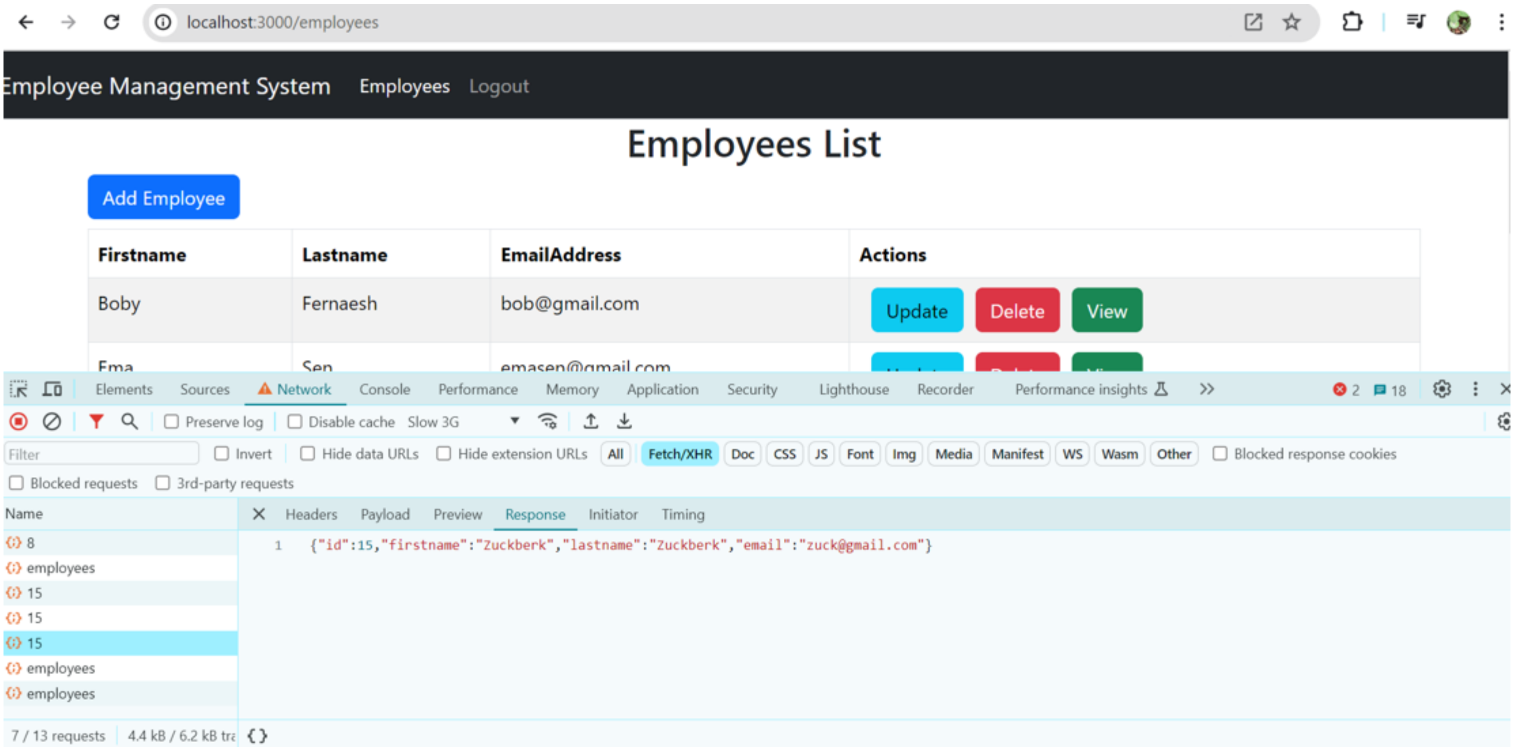
▾ General

Request URL:	http://localhost:8080/api/employees
Request Method:	GET
Status Code:	● 200 OK
Remote Address:	[::1]:8080
Referrer Policy:	strict-origin-when-cross-origin

▾ Response Headers □ Raw

Access-Control-Allow-Origin: *

4 / 9 requests | 2.3 kB / 4.1 kB trar


localhost:3000/employees
Employee Management System
Employees
Logout

Employees List

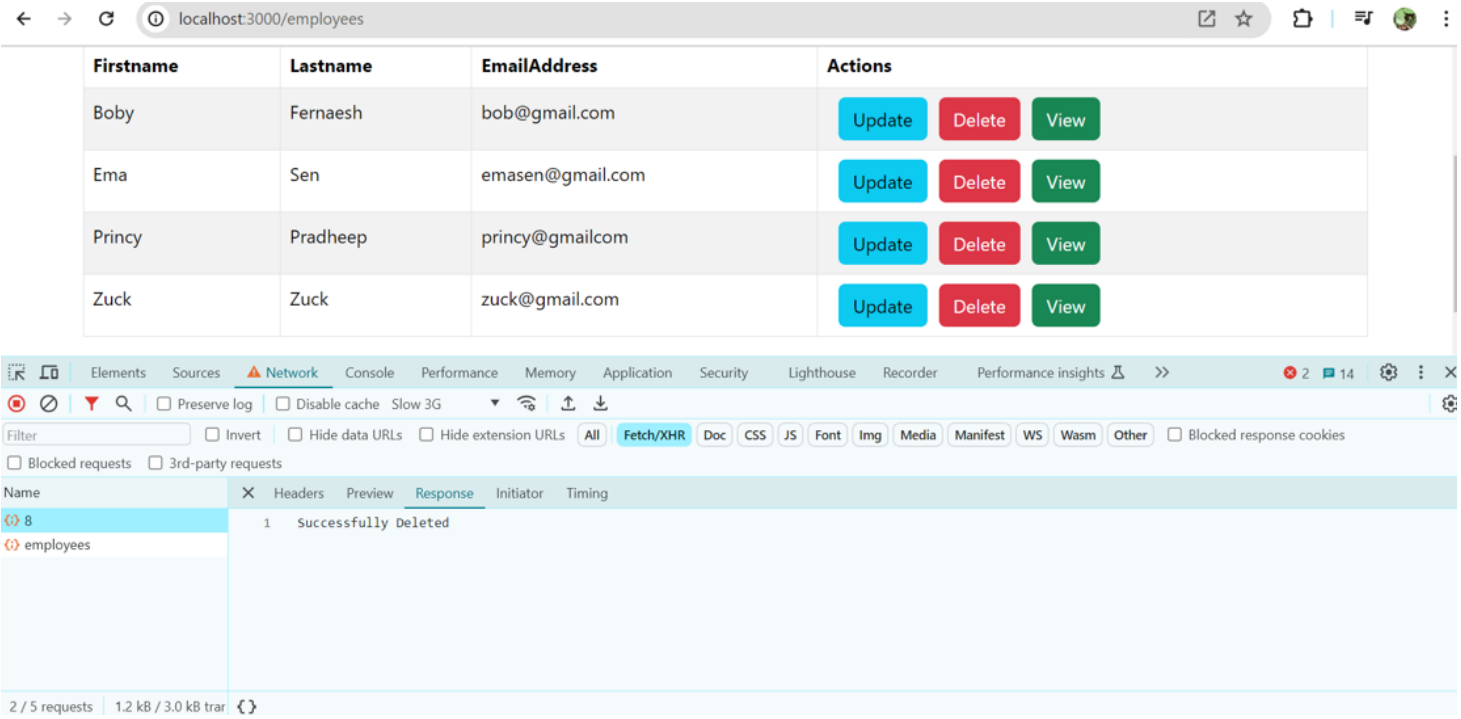
Firstname	Lastname	EmailAddress	Actions
Boby	Fernaesh	bob@gmail.com	Update Delete View
Ema	Sen	emasen@gmail.com	

Elements Sources Network Console Performance Memory Application Security Lighthouse Recorder Performance insights
□ Preserve log □ Disable cache Slow 3G
Filter □ Invert □ Hide data URLs □ Hide extension URLs All Fetch/XHR Doc CSS JS Font Img Media Manifest WS Wasm Other □ Blocked response cookies
□ Blocked requests □ 3rd-party requests
Name
8
employees
15
15
15
employees
employees
Headers Payload Preview Response Initiator Timing

```
{"id":15,"firstname":"Zuckberk","lastname":"Zuckberk","email":"zuck@gmail.com"}
```

7 / 13 requests | 4.4 kB / 6.2 kB tra

Delete Form Response



Firstname	Lastname	EmailAddress	Actions
Boby	Fernaesh	bob@gmail.com	Update Delete View
Ema	Sen	emasen@gmail.com	Update Delete View
Princy	Pradheep	princy@gmailcom	Update Delete View
Zuck	Zuck	zuck@gmail.com	Update Delete View

```
Successfully Deleted
```

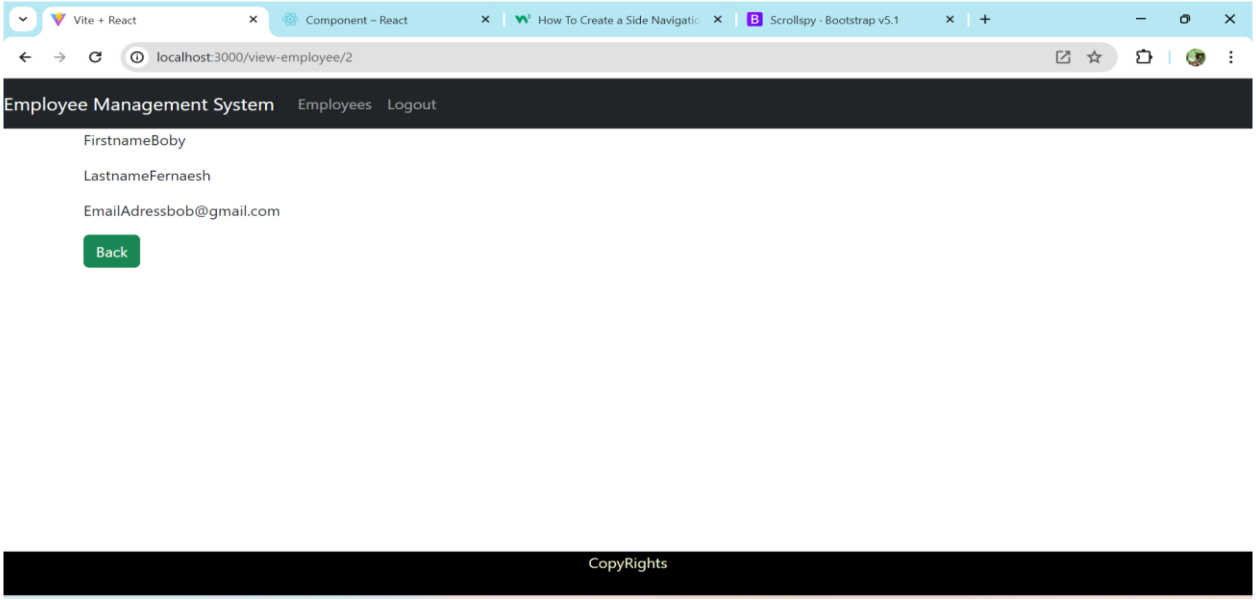
Employee Management

Viewing Employee

1. **Access Employee List**: After logging in, navigate to the employee list page. This page displays a list of all employees.
2. **Employee Details**: Each employee entry

displays key information such as first name, last name, and email.

User View Page


Vite + React
Component – React
How To Create a Side Navigatic
Scrollspy · Bootstrap v5.1
localhost:3000/view-employee/2
Employee Management System
Employees
Logout

FirstnameBoby

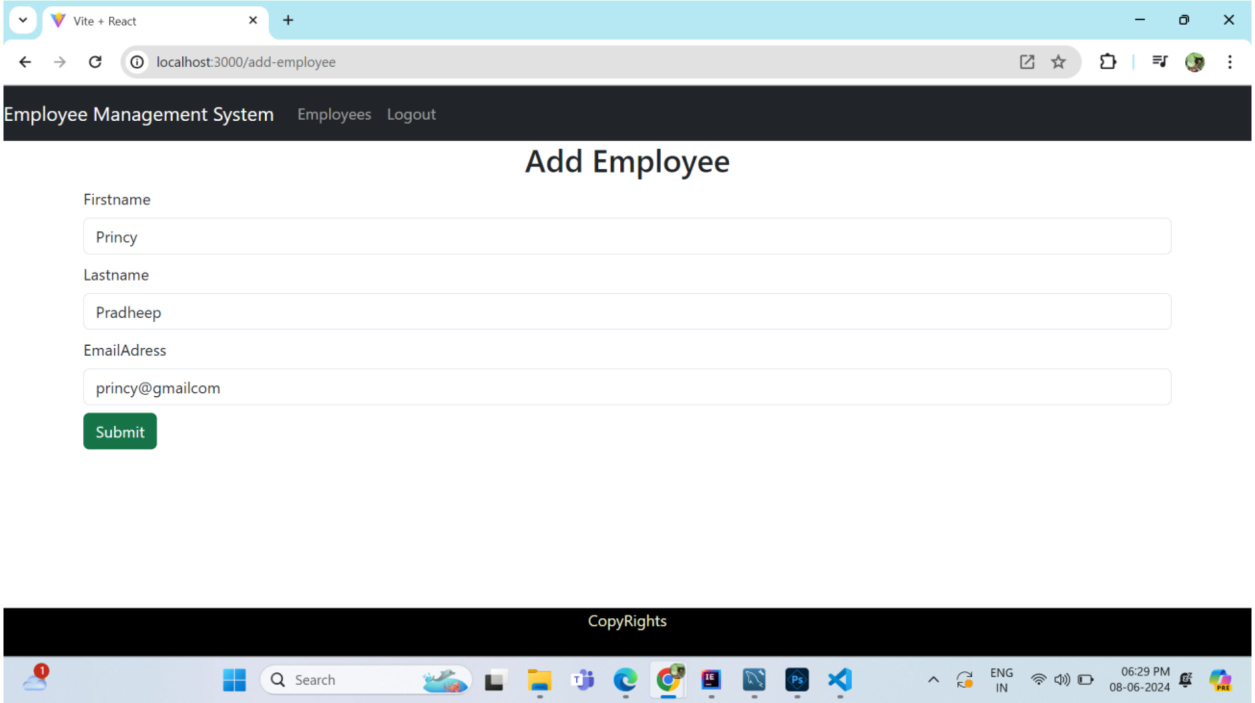
LastnameFernaesh

EmailAdressbob@gmail.com

Adding a New Employee

1. **Navigate to the Add Employee Page**: On the employee list page, there should be a button or link to add a new employee. Click this to open the employee registration form.
2. **Fill Out Employee Form**: Enter the required employee details (e.g., first name, last name, email). Click the "Submit" button to save the new employee.
3. **Confirmation**: After successfully adding the employee, you will be redirected back to the employee list page, where you can see the new employee added to the list.

Add Employee Form



Add Employee

Firstname

Princy

Lastname

Pradheep

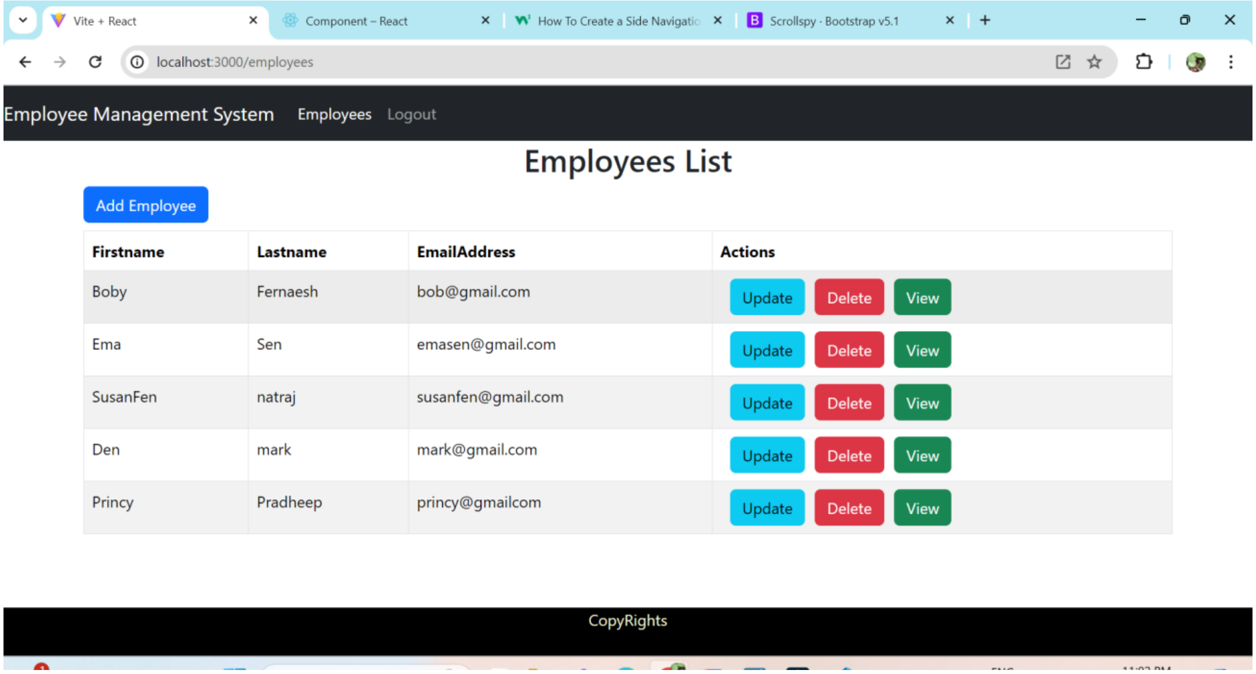
EmailAdress

princy@gmailcom

Submit

CopyRights

Home Page


Vite + React
Component – React
How To Create a Side Navigatic
Scrollspy · Bootstrap v5.1
localhost:3000/employees
Employee Management System
Employees
Logout

Employees List

Add Employee

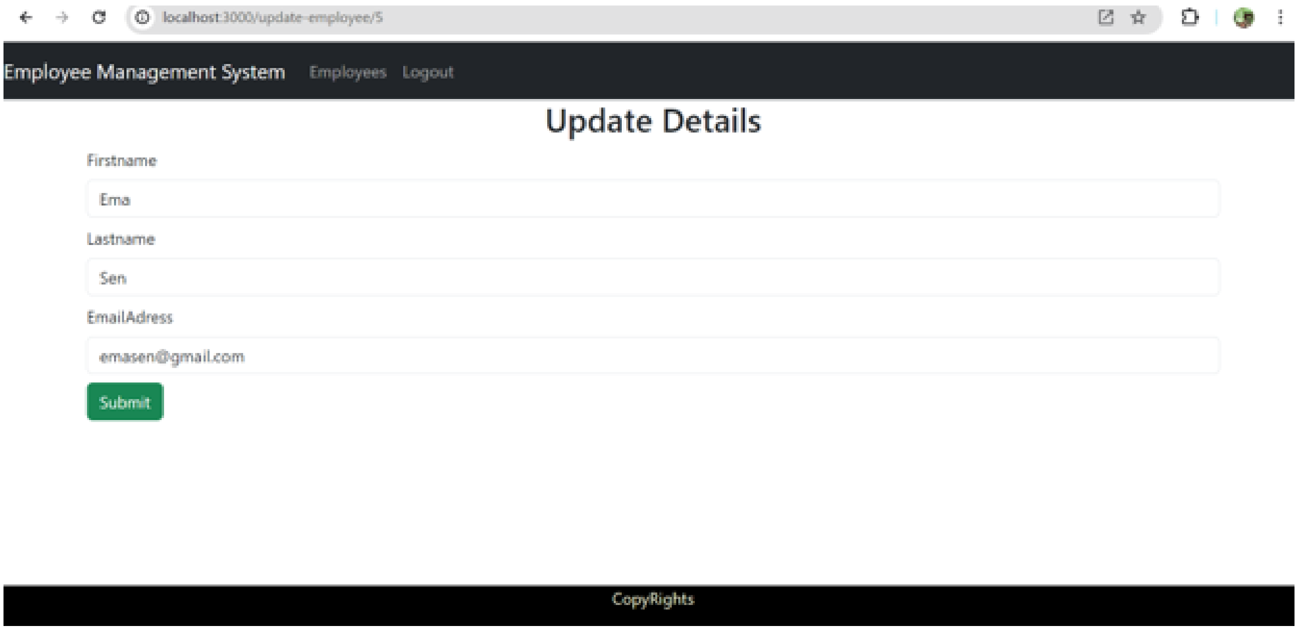
Firstname	Lastname	EmailAddress	Actions
Boby	Fernaesh	bob@gmail.com	Update Delete View
Ema	Sen	emasen@gmail.com	Update Delete View
SusanFen	natraj	susanfen@gmail.com	Update Delete View
Den	mark	mark@gmail.com	Update Delete View
Princy	Pradheep	princy@gmailcom	Update Delete View

CopyRights

Editing an Employee

1. **Access Edit Page**: On the employee list page, each employee entry should have an "Edit" button or link. Click this to open the edit form for the selected employee.
2. **Update Employee Details**: Modify the necessary information and click "Submit" to update the employee details.
3. **Confirmation**: After successfully updating, you will be redirected to the employee list page with the updated information.

Update Form



Update Details

Firstname

Ema

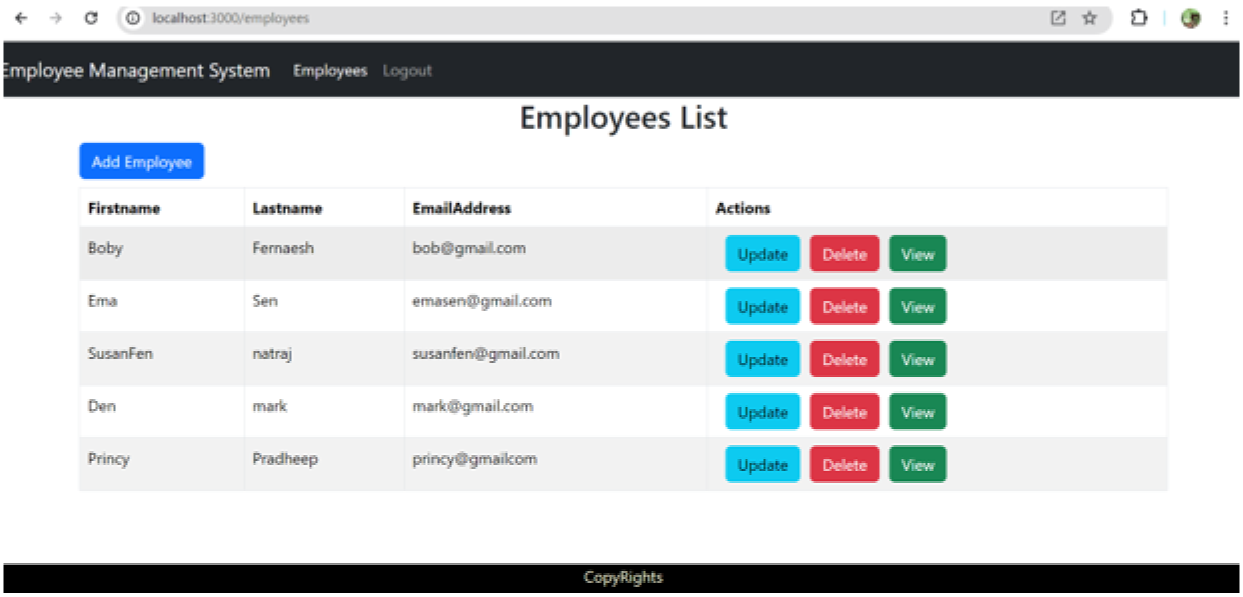
Lastname

Sen

EmailAdress

emasen@gmail.com

After Submitting the updating Form



Employees List

Firstname	Lastname	EmailAddress	Actions
Boby	Fernaesh	bob@gmail.com	Update Delete View
Ema	Sen	emasen@gmail.com	Update Delete View
SusanFen	natraj	susanfen@gmail.com	Update Delete View
Den	mark	mark@gmail.com	Update Delete View
Princy	Pradheep	princy@gmailcom	Update Delete View

API Endpoints

The backend Spring Boot application exposes several RESTful API endpoints to manage users and employees. Here is an overview of the key endpoints:

User Endpoints

- **Register User**: `POST /api/users/register`
    - Request Body: `{ "username": "ourusername", "password": "ourpassword" }`
    - Response: The newly created user object or an error message.
- **Login User**: `POST /api/users/login`
    - Request Body: `{ "username": "ourusername", "password": "ourpassword" }`
    - Response: The authenticated user object or an error message.

Employee Endpoints

- **Get All Employees**: `GET /api/employees`
    - Response: A list of all employees.
- **Create New Employee**: `POST /api/employees`
    - Request Body: `{ "firstName": "John", "lastName": "Doe", "email": "john.doe@example.com" }`
    - Response: The newly created employee object.
- **Update Employee**: `PUT /api/employees/{id}`
    - Request Body: `{ "firstName": "John", "lastName": "Doe", "email": "john.doe@example.com" }`
    - Response: The updated employee object.
- **Delete Employee**: `DELETE /api/employees/{id}`
    - Response: A success message or an error message.

Security

Password Encryption

To enhance security, it is recommended to encrypt user passwords before storing them in the database. This can be achieved using Spring Security's `BCryptPasswordEncoder` for hashing passwords.

Authentication and Authorization

- **Authentication**: Ensure that only registered users can log in.
- **Authorization**: Restrict access to certain endpoints based on user roles (e.g., only admin users can add or delete employees).

Frontend Deployment

Build the React App:

```
npm run build
```

Deploy the Build: Copy the contents of the `build` directory to your web server.

Conclusion

This Employee Management Application provides a comprehensive solution for managing employees, including features for user registration, login, and CRUD operations for employee data. By following this documentation, users should be able to set up, run, and use the application effectively.

It looks like you've uploaded a screenshot of a web page describing a Java Spring Boot, React JS, and MySQL project on an Employee Management System. I'll provide a detailed explanation based on the visible content of the screenshot.