

# Employee Management System

<b>Name</b>	<b>Shreyash Hake</b>
<b>Emp id</b>	<b>70983700</b>
<b>Mail</b>	<b>shreyash.hake@hitachivantara.com</b>

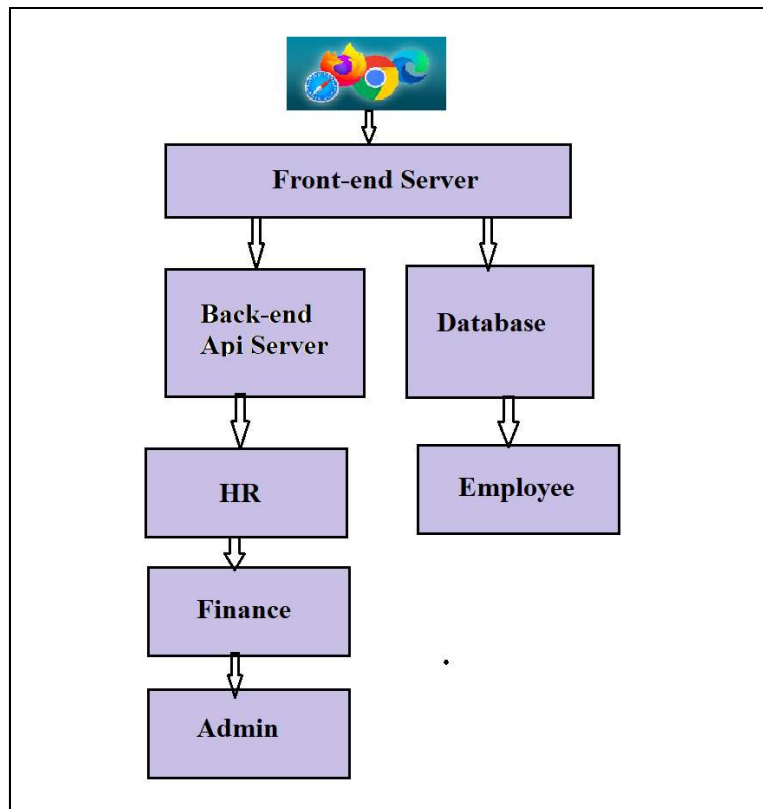
**Use case:** Develop an Application for Employee Management, Add Different roles like Employee, HR, Finance, Admin Each one will be able to different action based on role.

## Tech stacks used:

- **Frontend:** Angular Material, HTML, SCCS, Typescript.
- **Backend:** Spring-boot to create REST API.
- **Database:** Postgress sql

**Introduction:** Employee management application's main goal is to streamline employee data administration and simplify HR (Human Resources) activities. The application aims to address common HR issues like the storing, retrieving, and updating of employee data. Additionally, it strives to give each role within the company such as employee, HR, finance, and admin specific capabilities to carry out their individual tasks. Organizations can increase personnel data management accuracy, save manual labour, and improve operational efficiency by deploying this application.

## Architecture:



The above architecture shows that, the user interacts with the application through a web browser, which communicates with the front-end server. The front-end server serves the user interface and handles user input, which is then passed to the back-end API server via HTTP requests. The back-end API server is responsible for processing requests, performing business logic, and interacting with the database to retrieve or update data. The database is where all the employee data is stored and managed. It consists of tables that store information about employees, such as their personal details, employment history, and performance metrics.

## **Functionality:**

### 1. Employee Role:

- View personal information: The employee can view their personal information, such as name, contact details, address, etc.
- Update personal information: The employee can update their personal information, such as phone number or address, in case of any changes.

### 2. HR Role:

- View employee information: The HR can view all employee information, including personal, employment, and performance details.
- Add new employee: The HR can add new employee details and assign roles and permissions.

### 3. Finance Role:

- View payroll information: The finance role can view all payroll information, such as salary details, tax information, etc.
- Manage payroll: The finance role can manage the payroll process, including calculating and processing employee salaries, bonuses, taxes, and deductions.

### 4. Admin Role:

- Manage employee roles and permissions: The admin role can manage employee roles and permissions, including adding or removing roles and updating role permissions.
- Manage user accounts: The admin role can create or remove user accounts, reset passwords, and update user information.
- View all information: The admin role can view all information, including employee, attendance, and payroll details.

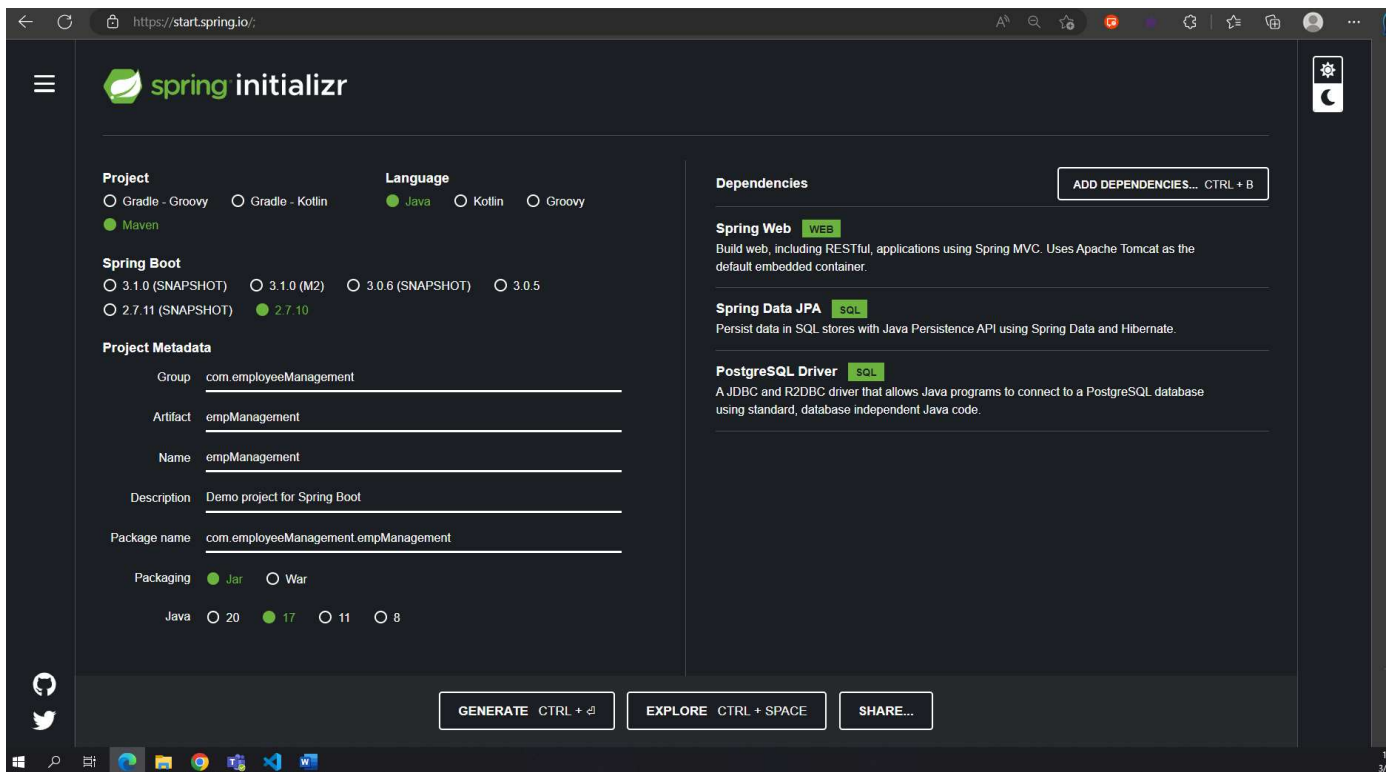
## **Implementation:**

### **Backend Part:**

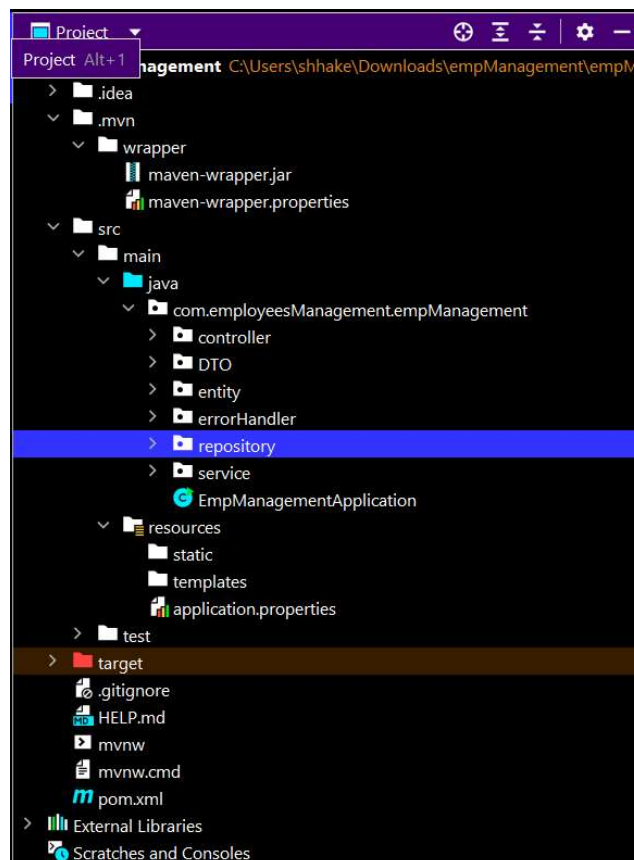
Initially I started working on backend to create tables from where I can fetch data on frontend, for this I have used spring initializer as initial setup, I have added following dependencies:

- Spring web: It simplifies the development of web-based applications as it provides set of pre-built components and modules that can be easily integrated into a project.
- Spring data JPA: It helps in performing database operation from Java code.
- Postgress Driver: Links the Java code to progress database.

Spring initializer provides a zip file and I have added my code in that.



I have created following file structure of my REST API which contains respective classes as per their functions.



interacting with the database, we first need to connect to it, considering the 'Postgress Driver' that we added as dependencies so using it we can connect to the database from below code.

Before that we need to create a database 'employeeManagement' this is the name of my database where tables will be created.

```
application.properties X
1  spring.application.name=EmployeeManagement
2  server.port=8085
3
4  spring.datasource.url=jdbc:postgresql://localhost:5432/employeeManagement
5  spring.datasource.username=postgres
6  spring.datasource.password=postgres
7
8  spring.datasource.driver-class-name=org.postgresql.Driver
9  spring.jpa.show-sql=true
10 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
11 spring.jpa.hibernate.ddl-auto=update
```

This code will make our tables available at port **8085** form where we can perform crud operations.

Below is the controller code of employee table which helps to control the database like adding data, updating data, deleting the data from the database tables.

```
@RestController
@CrossOrigin(origins = "http://localhost:4200")
@RequestMapping("/api/v1/employee")
public class EmployeeController {
    5 usages
    @Autowired
    private EmployeeService employeeService;
    18

    no usages
    @PostMapping(path = "/save")
    public String saveEmployee(@RequestBody EmployeeSaveDTO employeeSaveDTO) {
    21 |     return employeeService.addEmployee(employeeSaveDTO);
    22 | }
    23

    no usages
    @GetMapping(path = "/get")
    public List<EmployeeDTO> getAllEmployee() {
    26 |     List<EmployeeDTO> allCustomers = employeeService.getAllEmployee();
    27 |     return allCustomers;
    28 | }
    29

    no usages
    @GetMapping(path = "/get/{id}")
    public EmployeeDTO getEmployeeById(@PathVariable int id) {
    32 |     EmployeeDTO employee = employeeService.getEmployeeById(id);
    33 |     return employee;
    34 | }
    35

    no usages
    @PutMapping(path = "/update/{employeeId}")
    public String updateEmployee(@PathVariable int employeeId, @RequestBody EmployeeUpdateDTO employeeUpdateDTO) {
    39 |     return employeeService.updateEmployee(employeeId, employeeUpdateDTO);
    40 | }
    41

    no usages
    @DeleteMapping(path = "/delete/{id}")
    public String deleteCustomer(@PathVariable(value = "id") int id) {
    44 |     boolean deleteCustomer = employeeService.deleteEmployee(id);
    45 |     return "deleted";
    46 | }
    47 }
```

And employee table is created using below code which includes various columns to hold employees data.

```

5  @Entity
6  @Table(name = "employee")
7  public class Employee {
8      4 usages
9      @Id
10     @Column(name = "employee_id", length = 50)
11     @GeneratedValue(strategy = GenerationType.AUTO)
12     private int employeeId;
13
14     5 usages
15     @Column(name = "employee_name", length = 50)
16     private String employeeName;
17
18     4 usages
19     @Column(name = "employee_email", length = 50)
20     private String employeeEmail;
21
22     5 usages
23     @Column(name = "employee_address", length = 50)
24     private String employeeAddress;
25
26     5 usages
27     @Column(name = "employee_phone", length = 12)
28     private String employeePhone;
29
30     5 usages
31     @Column(name = "employee_salary", length = 15)
32     private long employeeSalary;
33
34     5 usages
35     @Column(name = "employee_hr", length = 50)
36     private String employeeHr;
37
38 }

```

This is how table gets created in PgAdmin:

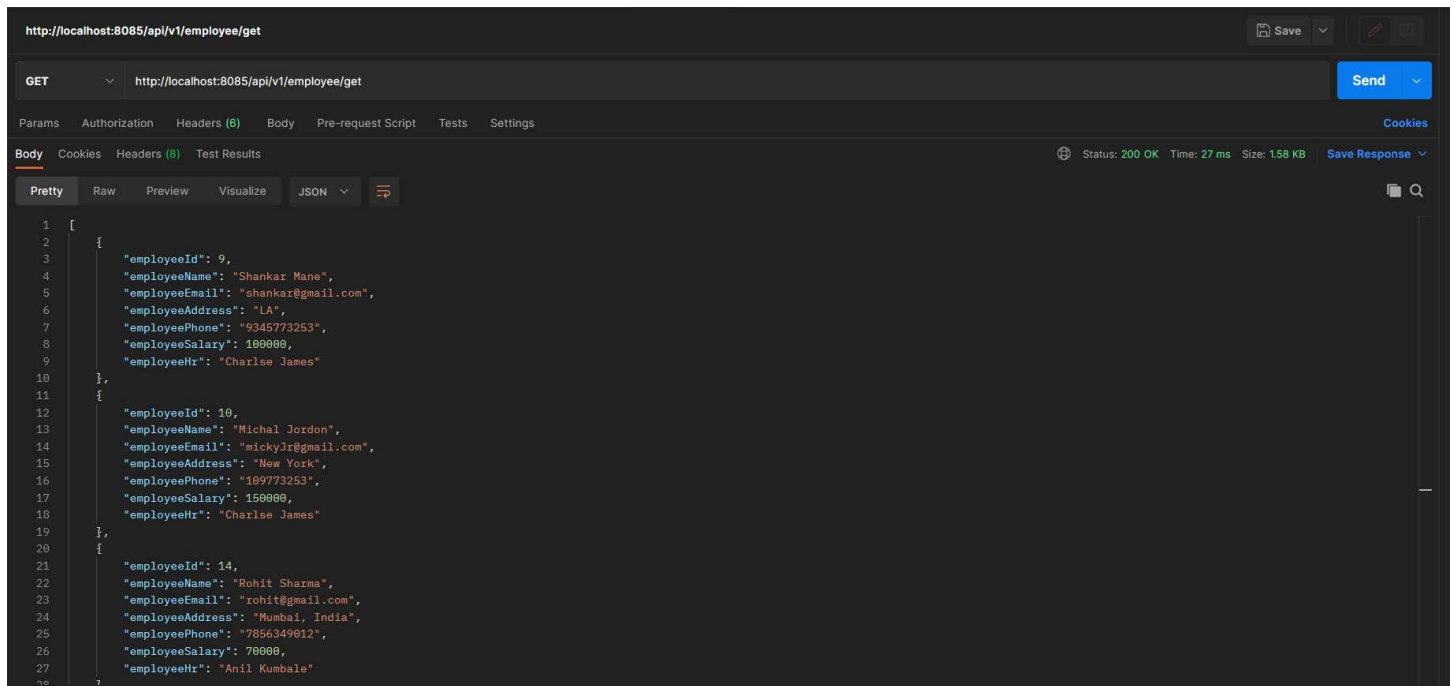
Data Output		Explain	Messages	Notifications			
	employee_id [PK] integer	 employee_address character varying (50)	 employee_email character varying (50)	 employee_hr character varying (50)	 employee_name character varying (50)	 employee_phone character varying (12)	 employee_salary bigint
1	9	LA	shankar@gmail.com	Charlse James	Shankar Mane	9345773253	100000
2	10	New York	mickyJr@gmail.com	Charlse James	Michal Jordon	109773253	150000
3	13	Goa, India	ramesh@gmail.com	Dipali Sharma	Ramesh Hake	8080776453	45000
4	14	Mumbai, India	rohit@gmail.com	Anil Kumbale	Rohit Sharma	7856349012	70000
5	16	LA, USA	shankar@gmail.com	Rashmi Singh	Shankar Mane	09345773253	34567
6	17	Pune	shankar@gmail.com	Rasmi Singgh	Shankar Mane	09345773253	45678
7	22	Pimpri Chinchwad, Pune	omkar@gmail.com	Kiran Arsh	Omkar Gaware	89765321134	21500

Similarly, various required table can be created like this Java code.

For testing of the database, I have used **Postman** from where we can get, add, update, delete the data, below are some outputs of postman.

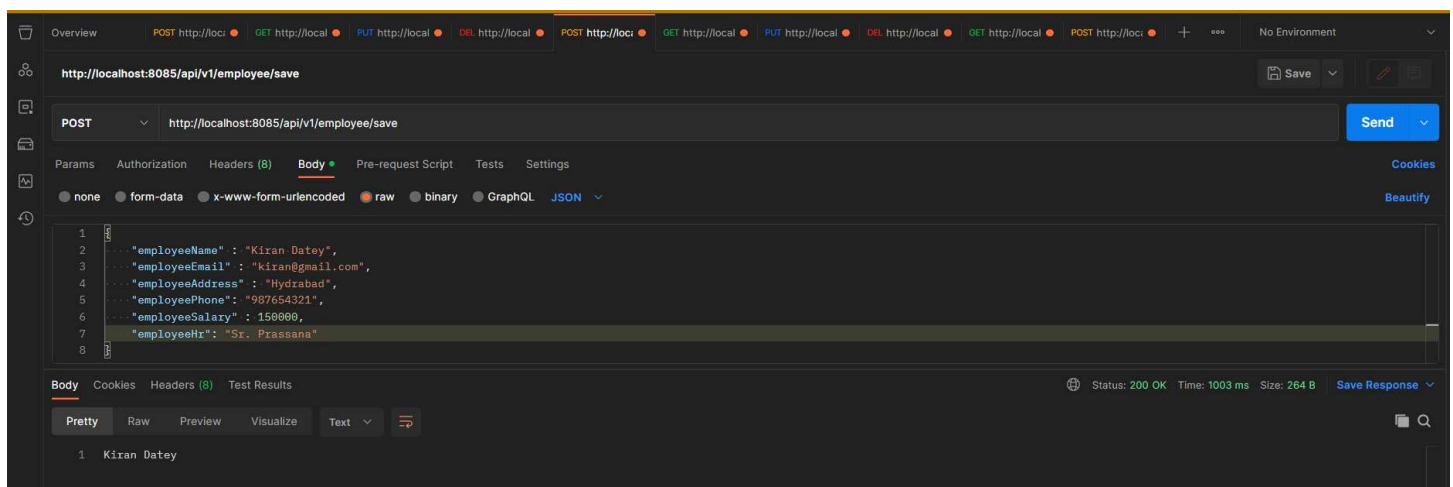
1. Getting data from database using postman:

<http://localhost:8085/api/v1/employee/get> using this link we can get data of all employees, which I have used in my frontend.



2. Adding data to database using postman:

<http://localhost:8085/api/v1/employee/save> using this link we can add data to database.



3. Updating data for employee in database using postman:

<http://localhost:8085/api/v1/employee/update/13> using this link we can update employee base on its id

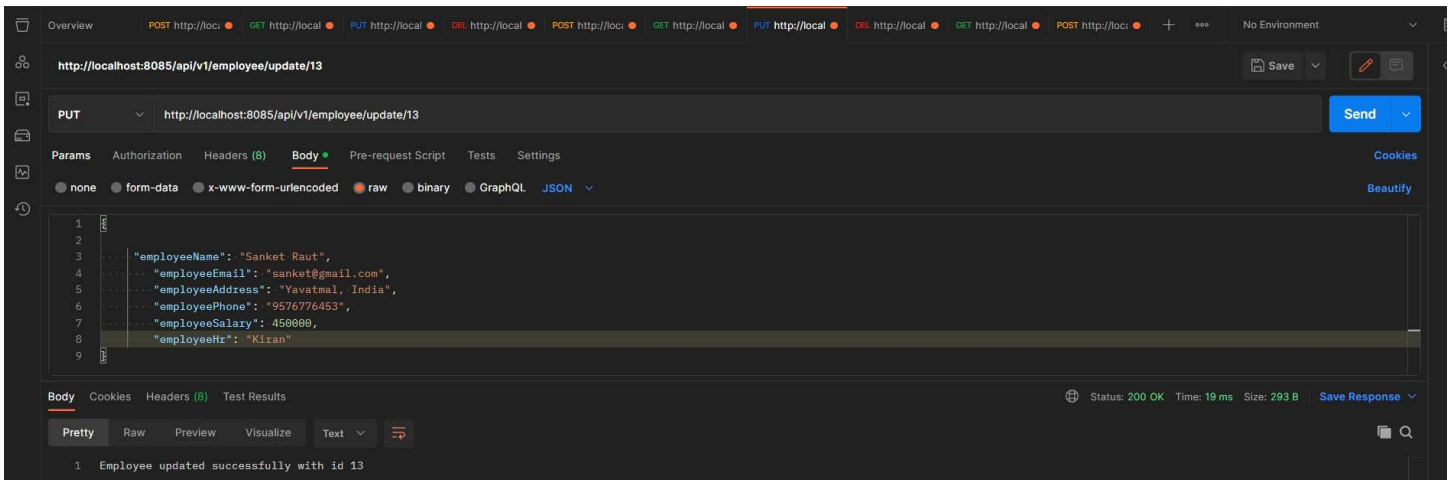
Before updating:

3	13	Goa, India	ramesh@gmail.com	Dipali Sharma	Ramesh Hake	8080776453	45000
---	----	------------	------------------	---------------	-------------	------------	-------

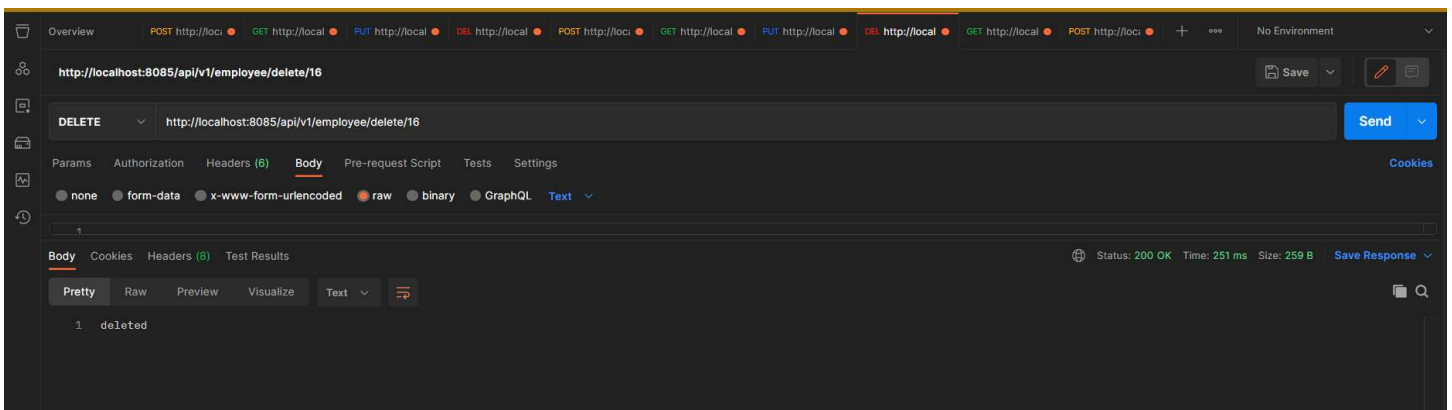
After updating:

3	13	Yavatmal, India	sanket@gmail.com	Kiran	Sanket Raut	9576776453	450000
---	----	-----------------	------------------	-------	-------------	------------	--------



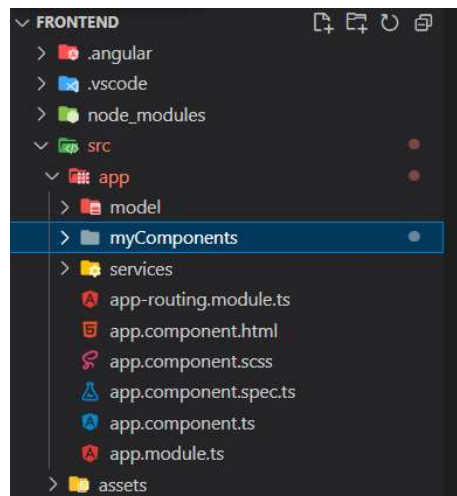


4. Deleting data from database using postman:  
<http://localhost:8085/api/v1/employee/delete/15> this link will delete employee with id 15



## Frontend:

For frontend I have used Angular 15, where I have created various components, used Angular material for development help



Project Structure of Frontend

Below is the service class of one of the component, which use the REST API to perform CRUD operation on the database

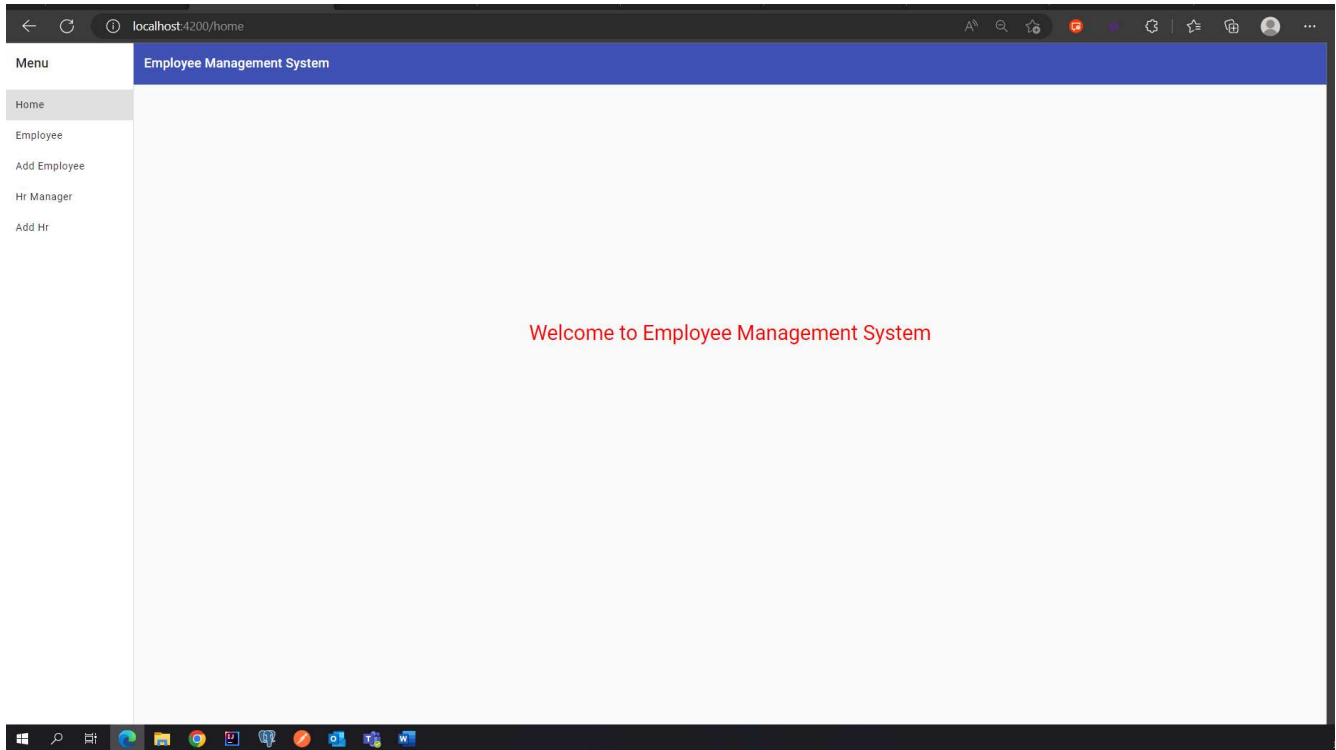
```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { map } from 'rxjs';
4 import { EmployeeModel } from '../model/employee.model';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class EmployeeService {
10
11   employeeGet = "http://localhost:8085/api/v1/employee/get";
12   employeeDelete = "http://localhost:8085/api/v1/employee/delete";
13   employeePost = "http://localhost:8085/api/v1/employee/save";
14   employeePut = "http://localhost:8085/api/v1/employee/save";
15
16   constructor(private http: HttpClient) { }
17
18   addUser = (user: any) => {
19     if (user.employeeId == 0)
20       return this.http.post(this.employeePost, user);
21     else
22       return this.http.put(this.employeePut, user);
23   }
24
25   getUsers = () =>
26     this.http.get(this.employeeGet, { observe: 'response' })
27       .pipe(
28         map(response => {
29           const users = response.body as EmployeeModel[];
30           return { users }
31         })
32       )
33
34   getById = (id: number) => this.http.get<EmployeeModel>(this.employeeGet + `/${id}`)
35
36   delete = (id: number) => this.http.delete<any>(this.employeeDelete + `/${id}`)
37
38   update = (id: number, user: EmployeeModel) => this.http.put(`${this.employeePut}/${id}`, user);
39
40 }
41
```



















## User Interface:

Below are few Interfaces from application:

1. After Login the interface for Admin, will look like:



2. Admin Can View all employees, along with this admin can edit and delete employee:

Employee Management System						
All Employees						
Id	Name	Email	Id	Name	Email	Action
8	Asmita Godse	amita@gmail.com	Ajani	8080773253	50000	 
9	Shankar Mane	shankar@gmail.com	LA	9345773253	100000	 
10	Michal Jordon	mickyjr@gmail.com	New York	109773253	150000	 
14	Rohit Sharma	rohit@gmail.com	Mumbai, India	7856349012	70000	 
13	Ramesh Hake	ramesh@gmail.com	Goa, India	8080776453	45000	 
16	Shankar Mane	shankar@gmail.com	LA, USA	09345773253	34567	 
17	Shankar Mane	shankar@gmail.com	Pune	09345773253	45678	 
22	Omkar Gaware	omkar@gmail.com	Pimpri Chinchwad, Pune	89765321134	21500	 

### 3. Admin Can also Add employee in employee table:

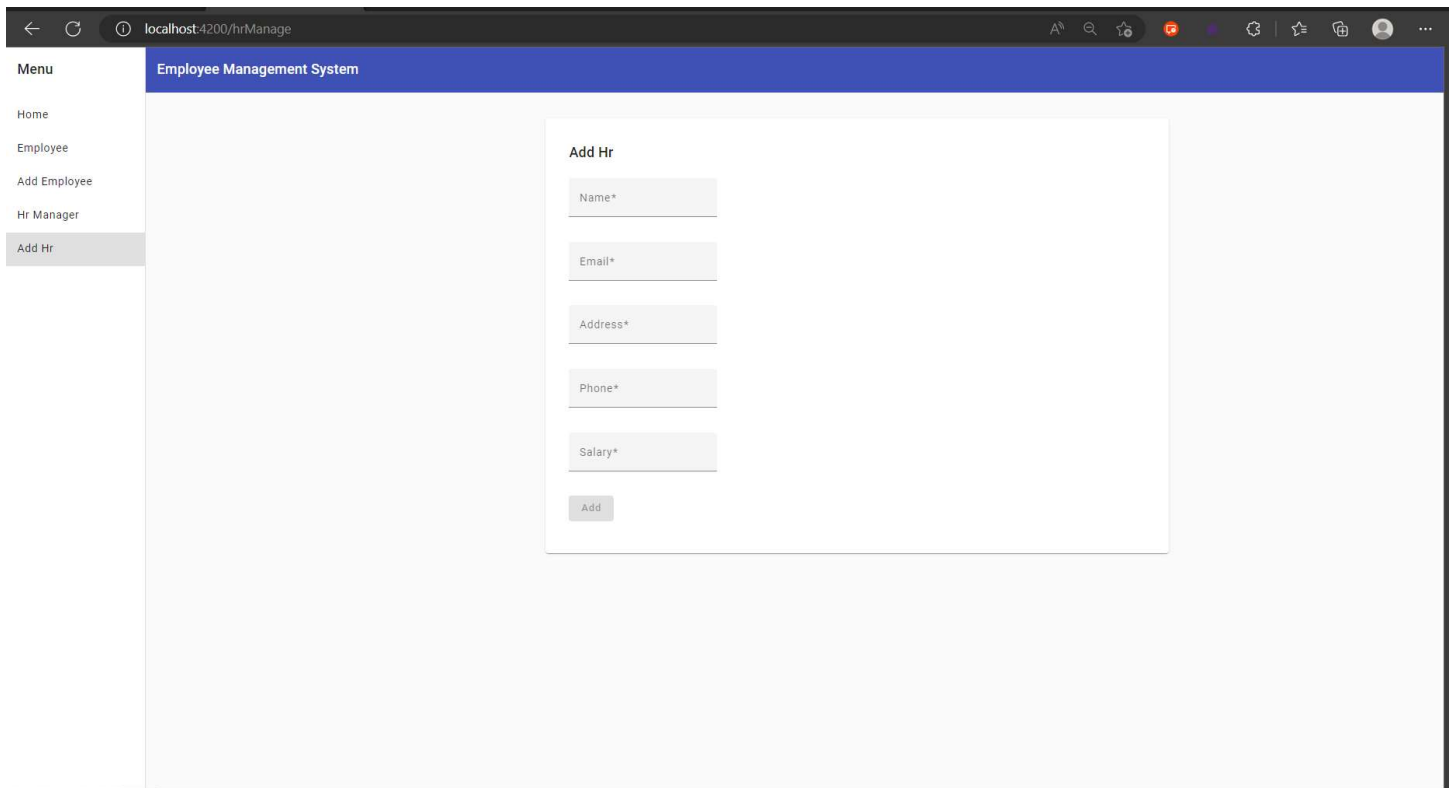
The screenshot shows a web browser at localhost:4200/empManage. The page has a blue header 'Employee Management System' and a left sidebar menu with 'Add Employee' highlighted. The main content area contains a white box titled 'Add employee' with the following fields: Name\*, Email\*, Address\*, Phone\*, Salary\*, and Hr Name\*. An 'Add' button is at the bottom of the box.

### 4. Admin can monitor Hr also:

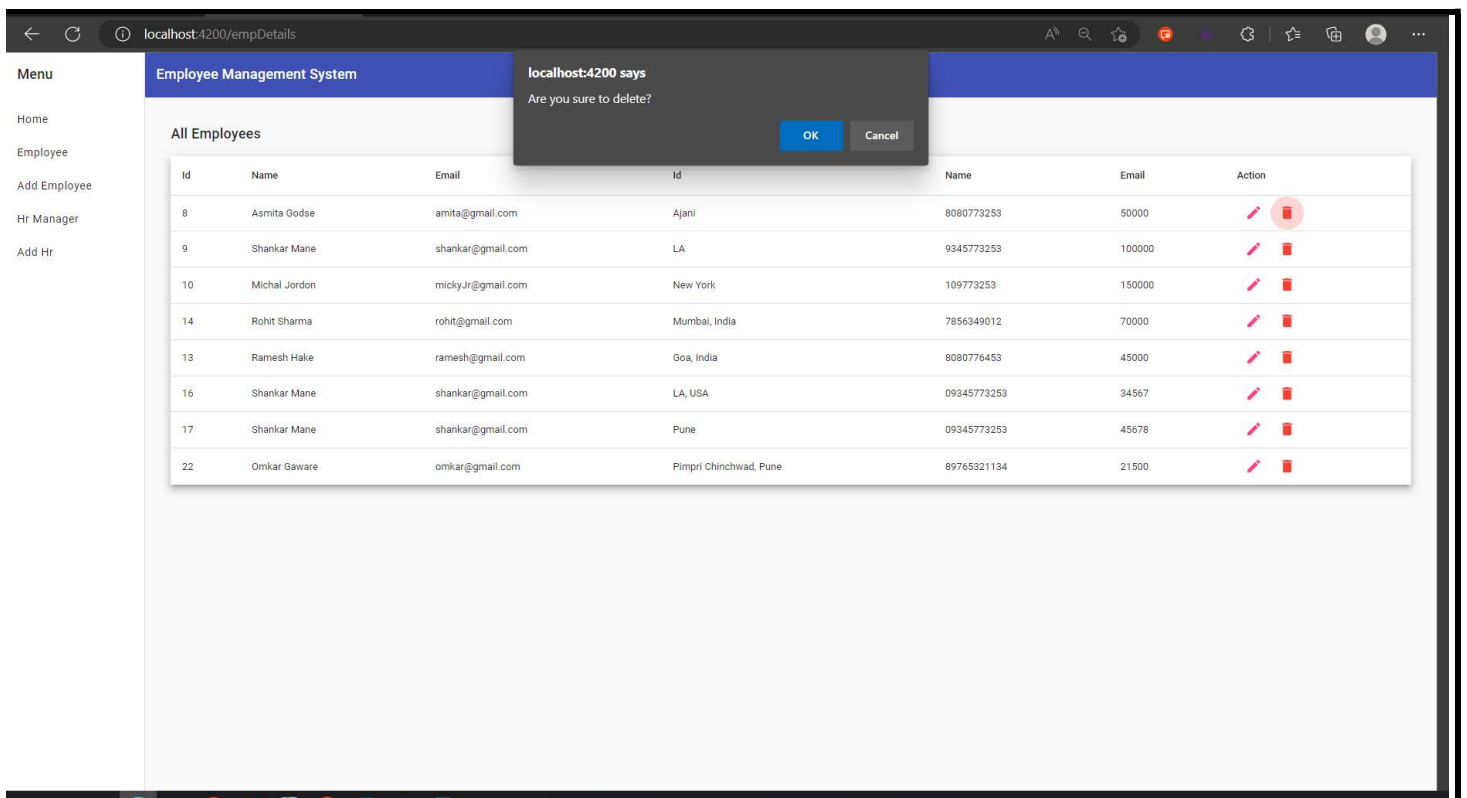
The screenshot shows a web browser at localhost:4200/hrDetails. The page has a blue header 'Employee Management System' and a left sidebar menu with 'Hr Manager' highlighted. The main content area displays a table titled 'All Employees' with 7 columns: Id, Name, Email, Id, Name, Email, and Action. The table contains 4 rows of employee data, each with edit and delete icons in the Action column.

Id	Name	Email	Id	Name	Email	Action
11	Dipali Sharma	dipali@gmail.com	Mumbai	72356478	700000	
12	Charlse James	charles@gmail.com	New York	897356478	1000000	
20	Mehul Ghyar	mehul@gmail.com	Nagpur, Maharashtra	8965432123	67000	
21	Nikita	nikita@gmail.com	Pune,Mh	21234	399999	

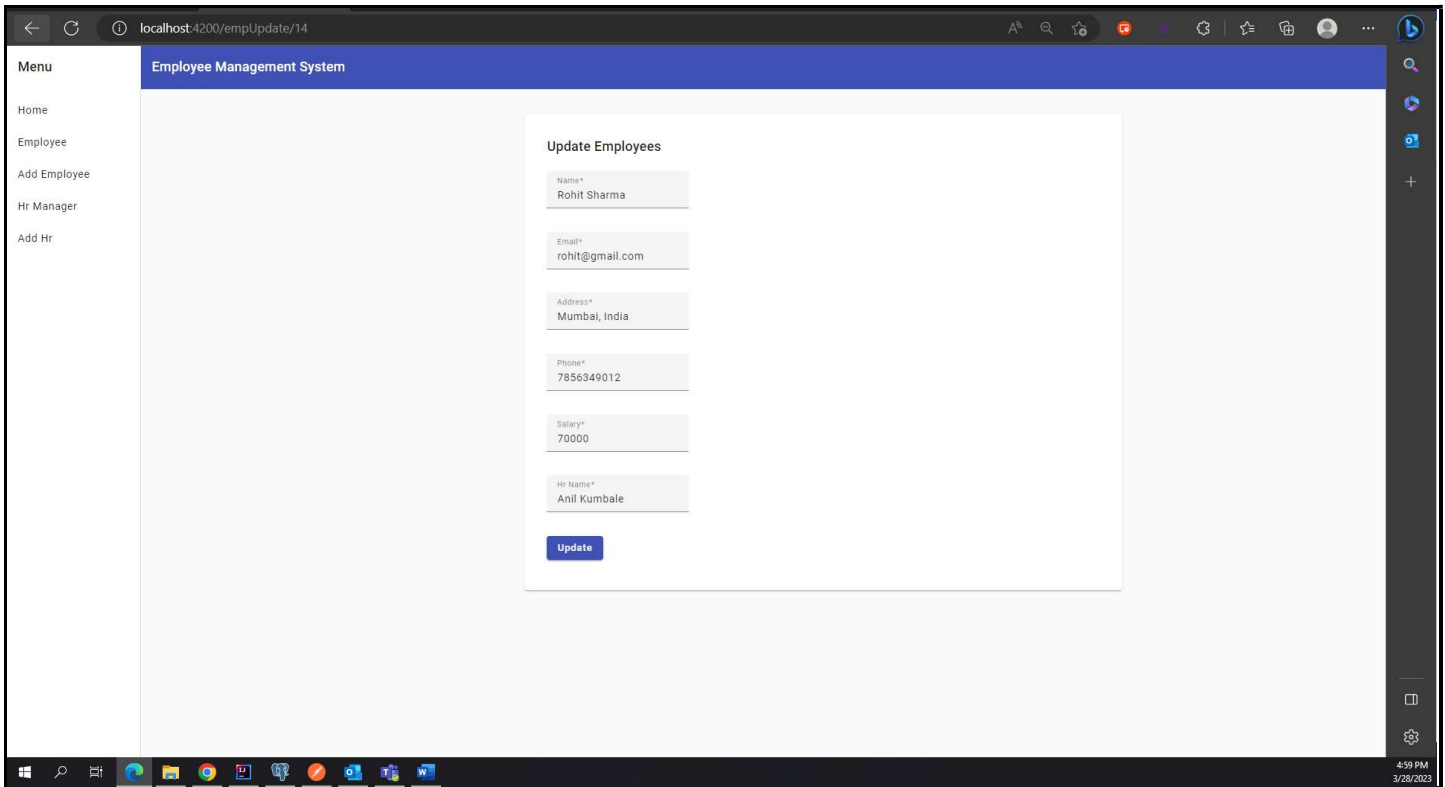
### 5. Also, Admin can add Hr to Hr table:



## 6. This is how we can delete the employee and hr from this tables:



## 7. This is how we can update employee similarly we can update Hr.



The screenshot displays a web browser window at the URL `localhost:4200/empUpdate/14`. The application has a dark blue header with the text "Employee Management System". On the left, there is a sidebar menu with the following items: "Menu", "Home", "Employee", "Add Employee", "Hr Manager", and "Add Hr". The main content area features a form titled "Update Employees". The form contains the following fields: "Name\*" with the value "Rohit Sharma", "Email\*" with the value "rohit@gmail.com", "Address\*" with the value "Mumbai, India", "Phone\*" with the value "7856349012", "Salary\*" with the value "70000", and "Hr Name\*" with the value "Anil Kumbale". At the bottom of the form is a blue "Update" button. The browser's taskbar at the bottom shows various application icons and the system clock indicating 4:59 PM on 3/28/2023.

### Security:

1. Authentication and Authorization: Implementing a strong authentication and authorization system is essential to ensure only authorized users can access the application. This can be achieved by implementing a login system that requires users to enter their username and password or by using multi-factor authentication.
2. Authentication and Authorization: Implementing a strong authentication and authorization system is essential to ensure only authorized users can access the application. This can be achieved by implementing a login system that requires users to enter their username and password or by using multi-factor authentication.

### Database Structure:

1. Users Table: This table stores information about the users of the application, including their username, email address, password, and role.
2. Employees Table: This table stores information about the employees of the company, including their name, employee ID, contact information.
3. Hr Table; This table store information about the HR including their salary.

## Deployment Approach:

To deploy the employee management application created using Angular and spring-boot, we can use a cloud-based infrastructure that offers scalability, high availability, and security. Here is a plan for deploying the application:

1. Amazon EC2: Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. we can deploy our Angular Spring Boot application on Amazon EC2 by creating an instance and installing the necessary software.
2. AWS Lambda: AWS Lambda is a serverless computing service that allows you to run code without provisioning or managing servers. we can deploy our Angular Spring Boot application on AWS Lambda by creating a Lambda function and configuring it to trigger when certain events occur.
3. Amazon ECS: Amazon Elastic Container Service (Amazon ECS) is a fully managed container orchestration service that supports Docker containers. We can deploy our Angular Spring Boot application on Amazon ECS by creating a task definition and a service.