

## Question-

You are required to create a Employee Management Rest Api based Web application, where you will be developing CRUD(Create,Read,Update and Delete) functionality along with Sorting and some concepts of security.

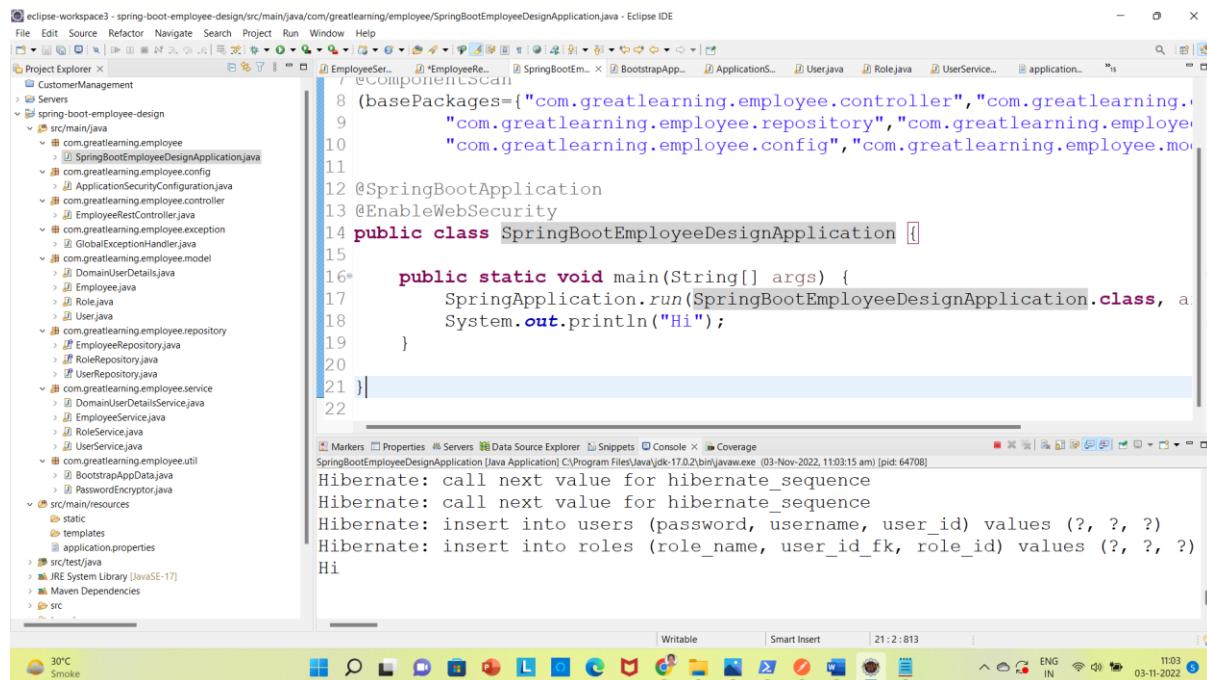
Your Rest Api should be secure. And should have different endpoints for different operations-

1. Your application should be able to add roles in the database dynamically in the db.

Ex-

```
{  
    "name": "USER"  
}
```

Where name specifies a role which can be assigned to a user that will be used for authentication purposes while interacting with the api.



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure under "spring-boot-employee-design".
- Code Editor:** Displays the `SpringBootEmployeeDesignApplication.java` file content:

```
@ComponentScan  
@BasePackages = {"com.greatlearning.employee.controller", "com.greatlearning.employee.repository", "com.greatlearning.employee.exception", "com.greatlearning.employee.config", "com.greatlearning.employee.model"}  
@SpringBootApplication  
@EnableWebSecurity  
public class SpringBootEmployeeDesignApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(SpringBootEmployeeDesignApplication.class, args);  
        System.out.println("Hi");  
    }  
}
```
- Console:** Shows Hibernate logs indicating sequence generation and user/role insertions.
- Bottom Status Bar:** Shows system information like temperature (30°C), battery level (Smoke), and system date/time (03-Nov-2022, 11:03:15 am).

2. Your application should be able to add Users in the db which can be used for authentication purposes.

Ex-

```
{
```

```

    "username": "temp",
    "password": "12345",
    "roles": [ {
        "id": 2,
        "name": "USER"
    } ]
}

```

The screenshot shows the Postman interface with a POST request to `http://localhost:8223/api/employees/setuser`. The 'Authorization' tab is selected, showing 'Basic Auth' with 'Username' set to 'Admin' and 'Password' set to 'admin'. The response body is displayed in a JSONpretty-printed format:

```

1
2   "id": 9,
3   "username": "Ramesh",
4   "password": "12345"
5   "roles": [
6     {
7       "roleId": 10,
8       "roleName": "USER"
9     }
10  ]
11

```

USERNAME AS ADMIN DOES NOT HAVE THE AUTHORITY TO SET USERS.

The screenshot shows the Postman interface with a failed API request. The URL is `http://localhost:8223/api/employees/setuser/`. The request method is POST. The status bar at the bottom indicates a 403 Forbidden error with a time of 805 ms and a size of 289 B. A red circle highlights the status code and message.

The screenshot shows the Postman interface with a successful API request after adding basic authentication. The URL is `http://localhost:8223/api/employees/setuser/`. The request method is POST. The status bar at the bottom indicates a 403 Forbidden error with a time of 805 ms and a size of 289 B. A red circle highlights the status code and message. The basic auth section shows 'superAdmin' as the username and 'super' as the password.

USER AS SUPERADMIN HAS THE AUTHORISATION TO ADD  
USERS .

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8223/api/employees/setuser/`. The response body is:

```

1 {
2   "id": 9,
3   "username": "Ramesh",
4   "password": "12345",
5   "roles": [
6     {
7       "roleId": 10,
8       "roleName": "USER"
9     }
10 ]
11

```

3. Now Your application should be able to add employees data in the db if and only if the authenticated user is **ADMIN**-

Ex-

```
{
  "firstName": "gl",
  "lastName": "postman",
  "email": "postman@gmail.com"
}
```

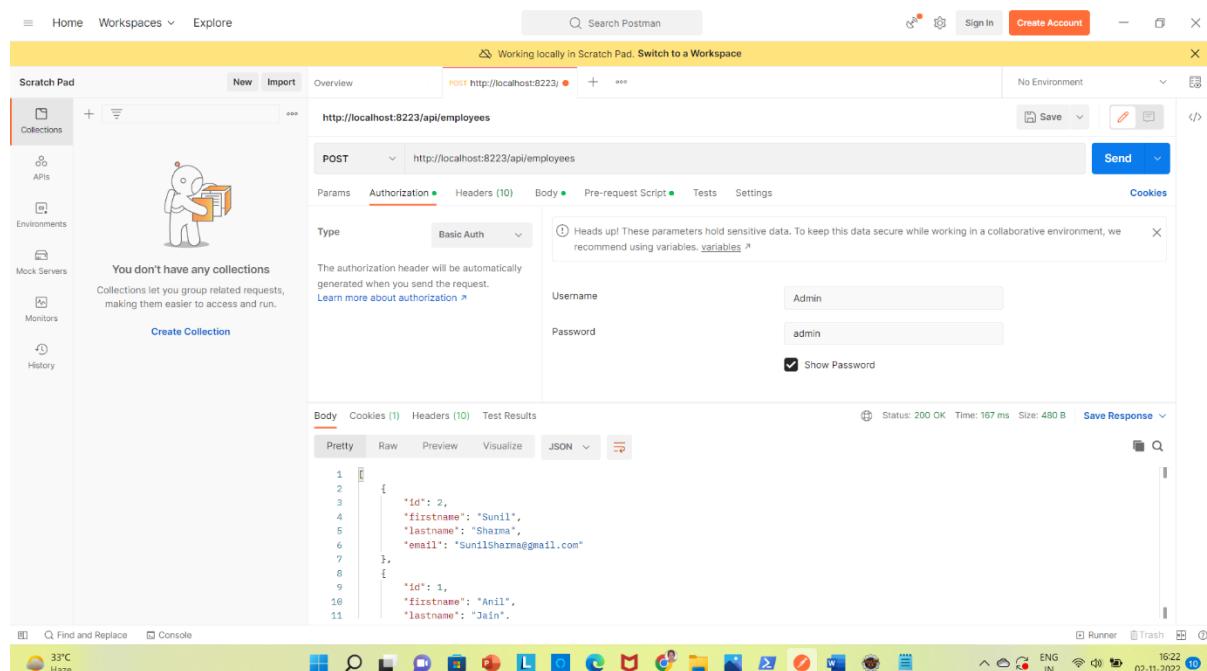
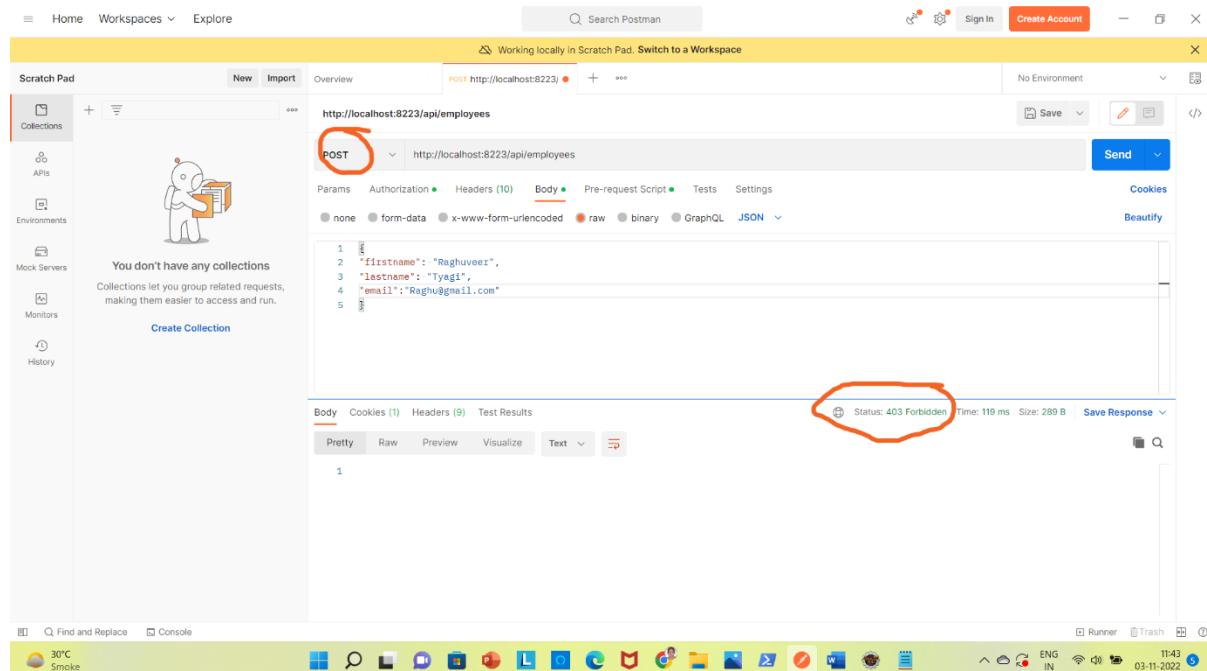
The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8223/api/employees`. The response body is identical to the previous one:

```

1 {
2   "id": 9,
3   "username": "Ramesh",
4   "password": "12345",
5   "roles": [
6     {
7       "roleId": 10,
8       "roleName": "USER"
9     }
10 ]
11

```

USER LOGGED IN AS A USER DOES NOT HAVE THE PERMISSION TO ADD EMPLOYEES.



## USER LOGGED IN AS AN ADMIN CAN ADD EMPLOYEES

The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Home, Workspaces, Explore, Scratch Pad, APIs, Environments, Mock Servers, Monitors, and History. The main area has a yellow header bar with the text "Working locally in Scratch Pad. Switch to a Workspace". Below it, a search bar says "Search Postman". The main workspace shows a POST request to "http://localhost:8223/api/employees". The "Body" tab is selected, showing JSON input:

```
1 "id": null,  
2 "firstname": "Anshu",  
3 "lastname": "Garg",  
4 "email": "AnshuGarg@gmail.com"
```

Below the input, the response status is shown as "Status: 200 OK Time: 260 ms Size: 399 B". The response body is also displayed in JSON format:

```
1 {  
2   "id": 7,  
3   "firstname": "Anshu",  
4   "lastname": "Garg",  
5   "email": "AnshuGarg@gmail.com"  
6 }
```

The bottom of the screen shows a taskbar with various icons and system status information like battery level (33°C Haze), network, and date/time (16:22 02-11-2022).

4. Your application should provide an endpoint to list all the employees stored in the database.

Ex-

Response Body-

```
[  
{  
  "id": 1,  
  "firstName": "Ujjawal",  
  "lastName": "Sharma",  
  "email": "fdfdfdf@gmail.com"  
},  
{  
  "id": 2,  
  "firstName": "temp",  
  "lastName": "kaushik",  
  "email": "jdfdkfdjj@gmail.com"  
},  
{
```

```

        "id": 3,
        "firstName": "postman",
        "lastName": "postman",
        "email": "postman@gamil.com"
    }
]

```

USER LOGGED IN AS USER CAN GET THE EMPLOYEES FOR THE DATABASE.

USER LOGGED IN AS USER CANNOT ADD THE EMPLOYEES TO THE DATABASE.

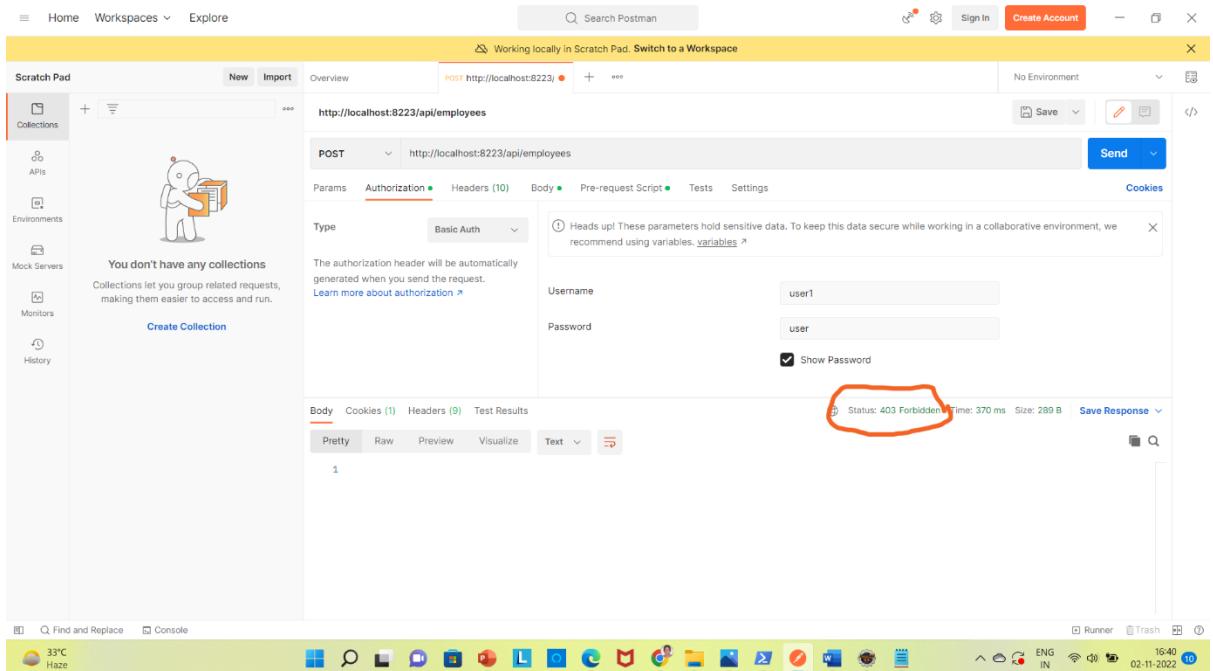
The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Scratch Pad' selected, showing options for 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', and 'History'. A central panel displays a request configuration for a 'GET' method to 'http://localhost:8223/api/employees'. The 'Authorization' tab is active, set to 'Basic Auth' with fields for 'Username' (user1) and 'Password' (user). Below the request, the 'Body' tab is selected, showing a JSON response with two employees:

```

2  [
3   {
4     "id": 2,
5     "firstname": "Sunil",
6     "lastname": "Sharma",
7     "email": "SunilSharma@gmail.com"
8   },
9   {
10    "id": 1,
11    "firstname": "Anil",
12    "lastname": "Jain",
13    "email": "AnilJain@gmail.com"
14  ]

```

The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 574 ms'. The system tray at the bottom right shows the date and time as '02-11-2022 16:38'.



5. Your application should provide endpoint to fetch or get an employee record specifically based on the id of that employee-

Ex- Url- <http://localhost:8080/api/employees/3>

Response Body-

```
{
  "id": 3,
  "firstName": "postman",
  "lastName": "postman",
  "email": "postman@gamil.com"
}
```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8223/api/employees`. The response body is:

```

2 {
  "id": 2,
  "firstname": "Sunil",
  "lastname": "Sharma",
  "email": "SunilSharma@gmail.com"
},
{
  "id": 1,
  "firstname": "Anil",
  "lastname": "Jain",
  "email": "AnilJain@gmail.com"
}

```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8223/api/employees/2`. The response body is:

```

{
  "id": 2,
  "firstname": "Sunil",
  "lastname": "Sharma",
  "email": "SunilSharma@gmail.com"
}

```

6. Your application should provide an endpoint to update an existing employee record with the given updated json object.

Ex-

Object to be updated(raw->Json)-

{

```

    "id":1,
    "firstName":"postman",
    "lastName":"postman",
    "email":"postman@gamil.com"
}

```

## Response Body after updation-

```
{
    "id": 1,
    "firstName": "postman",
    "lastName": "postman",
    "email": "postman@gamil.com"
}
```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:8223/api/employees/2`. The response body is:

```

1
2   "id": 2,
3   "firstname": "Sunil",
4   "lastname": "Sharma",
5   "email": "SunilSharma@gmail.com"
6

```

The 'user1' and 'user' fields in the Basic Auth section are circled in red.

Home Workspaces Explore

Search Postman

Working locally in Scratch Pad. Switch to a Workspace

Scratch Pad New Import Overview PUT http://localhost:8223/s + \*\*\*

http://localhost:8223/api/employees/2

Params Authorization Headers (10) Body Pre-request Script Tests Settings

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

```
1 "id": 2,
2 "firstname": "Sunila",
3 "lastname": "Gupta",
4 "email": "Suni@gmail.com"
```

Send Cookies (1) Headers (9) Test Results

Pretty Raw Preview Visualize Text

Status: 403 Forbidden Time: 113 ms Size: 289 B Save Response

You don't have any collections

Collections let you group related requests, making them easier to access and run.

Create Collection

Find and Replace Console

33°C Sunny

Runner Trash

16:49 ENG IN 02-11-2022 10

The screenshot shows the Postman interface with a PUT request to http://localhost:8223/api/employees/2. The body contains JSON data: { "id": 2, "firstname": "Sunila", "lastname": "Gupta", "email": "Suni@gmail.com" }. The response status is 403 Forbidden. A red circle highlights the status code and message in the response header. The Postman toolbar at the bottom shows various icons for different tools like Figma, GitHub, and Slack.

Home Workspaces Explore

Search Postman

Working locally in Scratch Pad. Switch to a Workspace

Scratch Pad New Import Overview PUT http://localhost:8223/s + \*\*\*

http://localhost:8223/api/employees/2

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Type Basic Auth

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [variables](#)

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Username Admin

Password admin

Show Password

Body Cookies (1) Headers (10) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 478 ms Size: 396 B Save Response

You don't have any collections

Collections let you group related requests, making them easier to access and run.

Create Collection

Find and Replace Console

33°C Sunny

Runner Trash

16:53 ENG IN 02-11-2022 10

The screenshot shows the Postman interface with the same PUT request to http://localhost:8223/api/employees/2. This time, the 'Authorization' tab is selected in the Params section, and 'Basic Auth' is chosen under 'Type'. The 'Username' field is filled with 'Admin' and the 'Password' field with 'admin'. The response status is now 200 OK. A red circle highlights the status code and message in the response header. The Postman toolbar at the bottom shows various icons for different tools like Figma, GitHub, and Slack.

The screenshot shows the Postman interface with a PUT request to `http://localhost:8223/api/employees/2`. The request body is set to `JSON` and contains the following data:

```

1
2 "id": 2,
3 "firstname": "Sunila",
4 "lastname": "Gupta",
5 "email": "Suni@gmail.com"
6

```

The response status is `200 OK` with a time of `478 ms` and a size of `396 B`.

7. Your application should also provide an endpoint to delete an existing employee record based on the id of the employee-

Ex-

Url- <http://localhost:8080/api/employees/4>

Response Body-

"Deleted employee id - 4"

The screenshot shows the Postman interface with a DELETE request to `http://localhost:8223/api/employees`. The request type is `Basic Auth` with `Username: user` and `Password: user1`. The response status is `401 Unauthorized`.

The screenshot shows the Postman interface with a successful DELETE request to `http://localhost:8223/api/employees/2`. The response status is 200 OK, and the response body is empty.

The screenshot shows the Postman interface with a successful GET request to `http://localhost:8223/api/employees`. The response status is 200 OK, and the response body is a JSON array containing one employee record:

```
[{"id": 1, "firstname": "Anil", "lastname": "Jain", "email": "AnilJain@gmail.com"}]
```

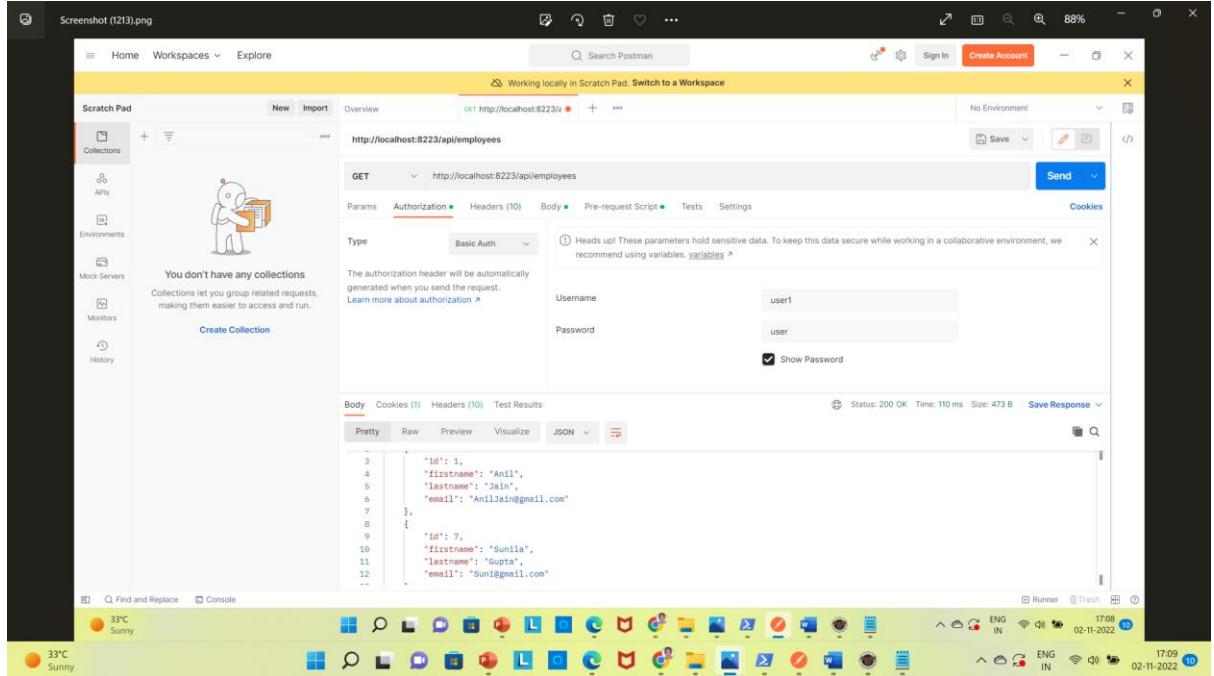
8. Your application should provide an endpoint to fetch an employee by his/her first name and if found more than one record then list them all-

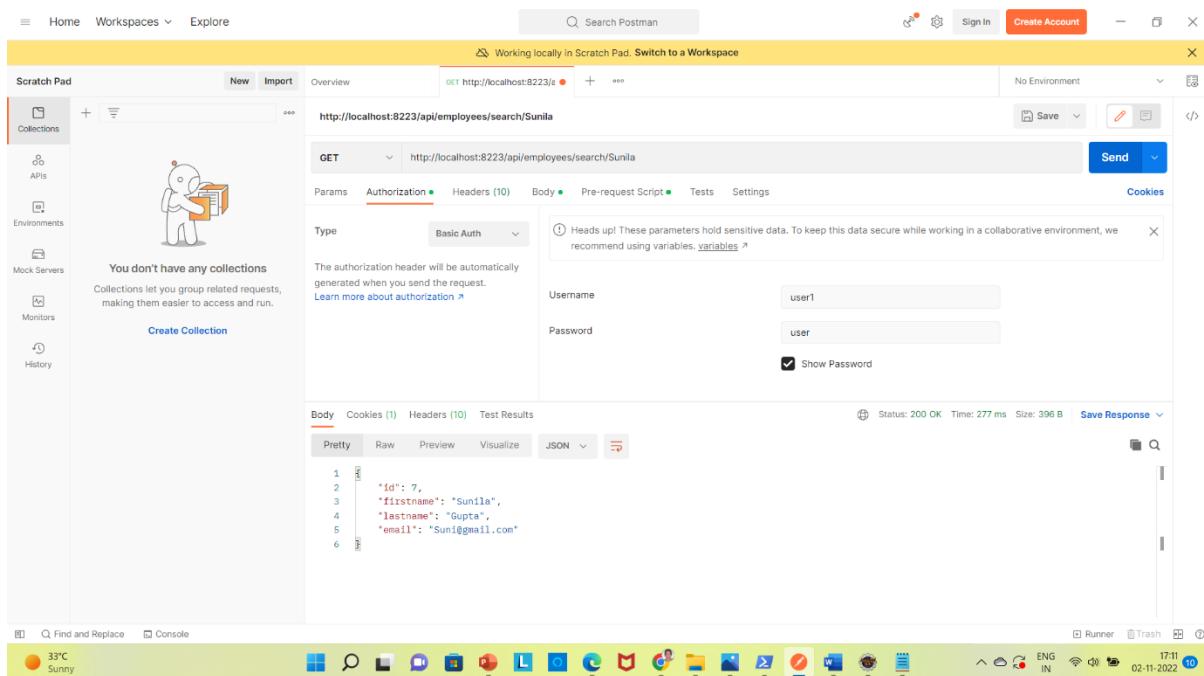
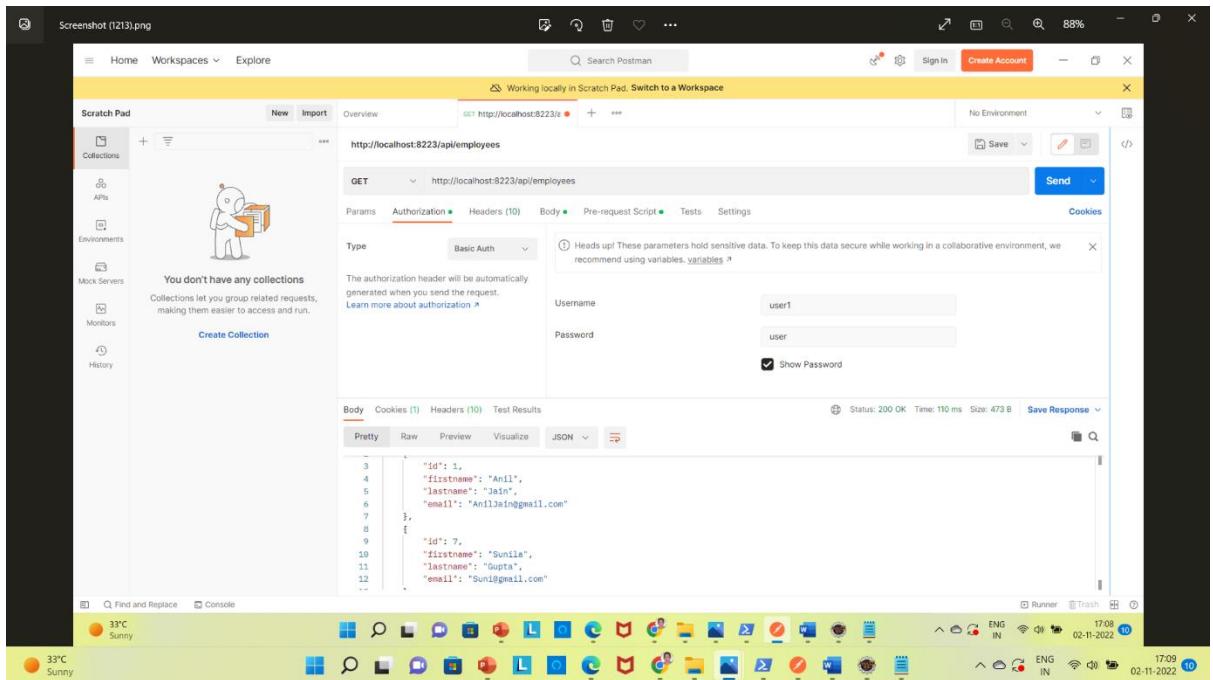
Ex-

Url- `http://localhost:8080/api/employees/search/gl`

## Response Body-

```
[  
 {  
   "id": 11,  
   "firstName": "gl",  
   "lastName": "postman",  
   "email": "postman@gamil.com"  
 }  
 ]
```





9. Your application should be able to list all employee records sorted on their first name in either ascending order or descending order .

Ex-

Url- <http://localhost:8080/api/employees/sort?order=asc>

OR

Url- <http://localhost:8080/api/employees/sort?order=desc>

## LIST OF EMPLOYEES –

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, Explore, Scratch Pad, Collections, APIs, Environments, Mock Servers, Monitors, and History. The main area displays a request configuration for a GET method to the URL `http://localhost:8223/api/employees/`. The Body tab is selected, showing a JSON payload:

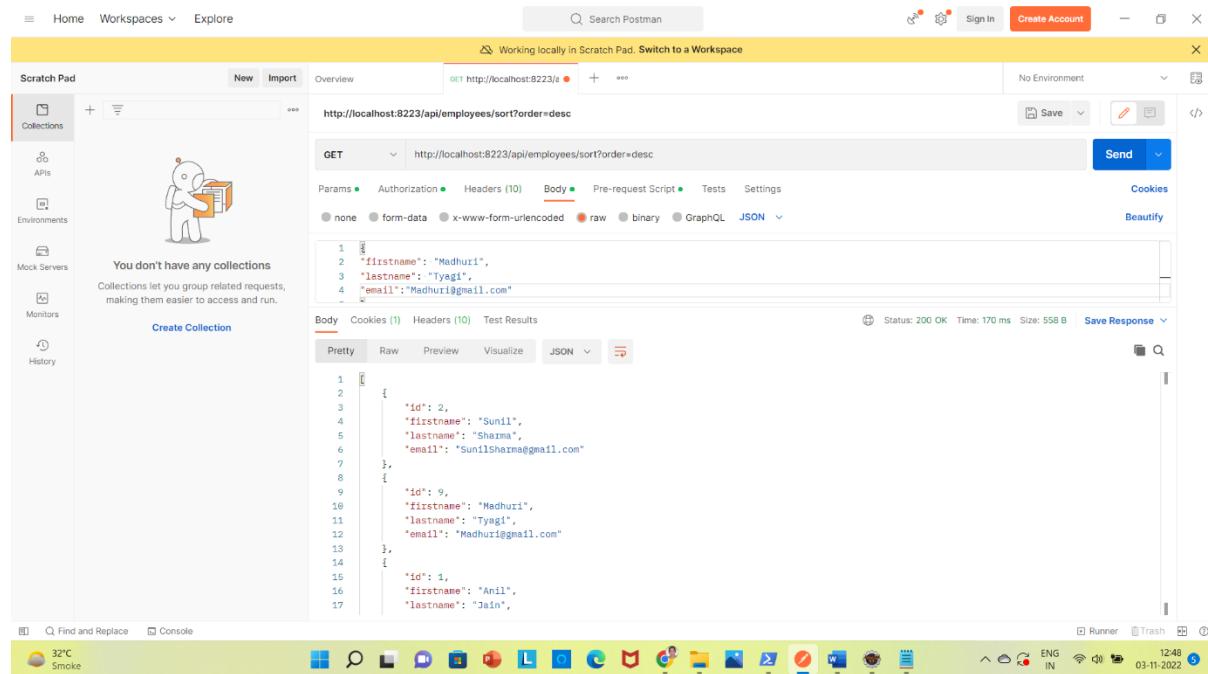
```
1 {
2   "id": 9,
3   "firstname": "Madhuri",
4   "lastname": "Tyagi",
5   "email": "Madhuri@gmail.com"
6 },
7 {
8   "id": 1,
9   "firstname": "Anil",
10  "lastname": "Jain",
11  "email": "AnilJain@gmail.com"
12 },
13 {
14   "id": 2,
15   "firstname": "Sunil",
16   "lastname": "Sharma",
17 }
```

The response status is 200 OK, with a time of 150 ms and a size of 558 B. Below the interface, the Windows taskbar shows various pinned icons and the system tray indicates it's 12:48 on 03-11-2022.

Url- <http://localhost:8080/api/employees/sort?order=asc>

This screenshot is identical to the one above, showing the Postman interface with a successful GET request to `http://localhost:8223/api/employees/sort?order=asc`. The response body is the same as the first screenshot, listing three employees. The interface and system status at the bottom are also identical.

Url- <http://localhost:8080/api/employees/sort?order=desc>



The screenshot shows the Postman interface with a GET request to `http://localhost:8223/api/employees/sort?order=desc`. The response body is a JSON array of employees:

```
1
2   {
3     "id": 2,
4     "firstname": "Sunil",
5     "lastname": "Sharma",
6     "email": "sunilSharma@gmail.com"
7   },
8   {
9     "id": 9,
10    "firstname": "Madhuri",
11    "lastname": "Tyagi",
12    "email": "Madhuri@gmail.com"
13  },
14  {
15    "id": 1,
16    "firstname": "Anil",
17    "lastname": "Jain",
```

## Important instructions

- i) You should use the H2 In Memory database/MySQL for the whole project along with Spring JPA and Spring Security.
- ii) Provide Screenshots of the operations(PostMan/Browser) along with code submission. (note → Screenshots will one of the criterias while grading)
- iii) You can also record your screen while demonstrating CRUD operation, upload on the drive and share the drive link along with code.
- iv) Spring Boot Application must follow the standard project structure .
- v) Code should follow naming conventions along with proper indentations.
- vi) You are free to choose any Rest client to interact with api while implementation.(Prefer PostMan)