

Contents

1. Your application should be able to add roles in the database dynamically in the db.	2
2. Your application should be able to add Users in the db which can be used for authentication purposes.....	4
3. Now Your application should be able to add employees data in the db if and	6
4. Your application should provide an endpoint to list all the employees stored in the database	7
5. Your application should provide endpoint to fetch or get an employee record specifically based on the id of that employee-	8
6. Your application should provide an endpoint to update an existing employee record with the given updated json object.....	9
7. Your application should also provide an endpoint to delete an existing employee record based on the id of the employee-	10
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-.....	11
9. Your application should be able to list all employee records sorted on their first name in either ascending order or descending order	12

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.

Request URL: localhost:8080/roles/create

Ex payload-

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

```
{  
  "name":"ADMIN"  
}
```

POST localhost:8080/roles/create Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** Beautify

```
1 {
2   "name": "ADMIN"
3 }
4
```

Body Cookies (1) Headers (11) Test Results Status: 201 Created Time: 320 ms Size: 379 B Save Response

Pretty Raw Preview Visualize **JSON** Copy Search

```
1 {
2   "id": 1,
3   "name": "ROLE_ADMIN"
4 }
```

POST Create POST Regis POST saveE GET getEmj GET getAIE PUT update DEL deletet GET localhc GET localhc + ... No Environment

Employee_assignment / CreateRole Save Edit Comment

POST localhost:8080/roles/create Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** Beautify

```
1 {
2   "name": "USER"
3 }
4
```

Body Cookies (1) Headers (11) Test Results Status: 201 Created Time: 13 ms Size: 378 B Save Response

Pretty Raw Preview Visualize **JSON** Copy Search

```
1 {
2   "id": 2,
3   "name": "ROLE_USER"
4 }
```

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

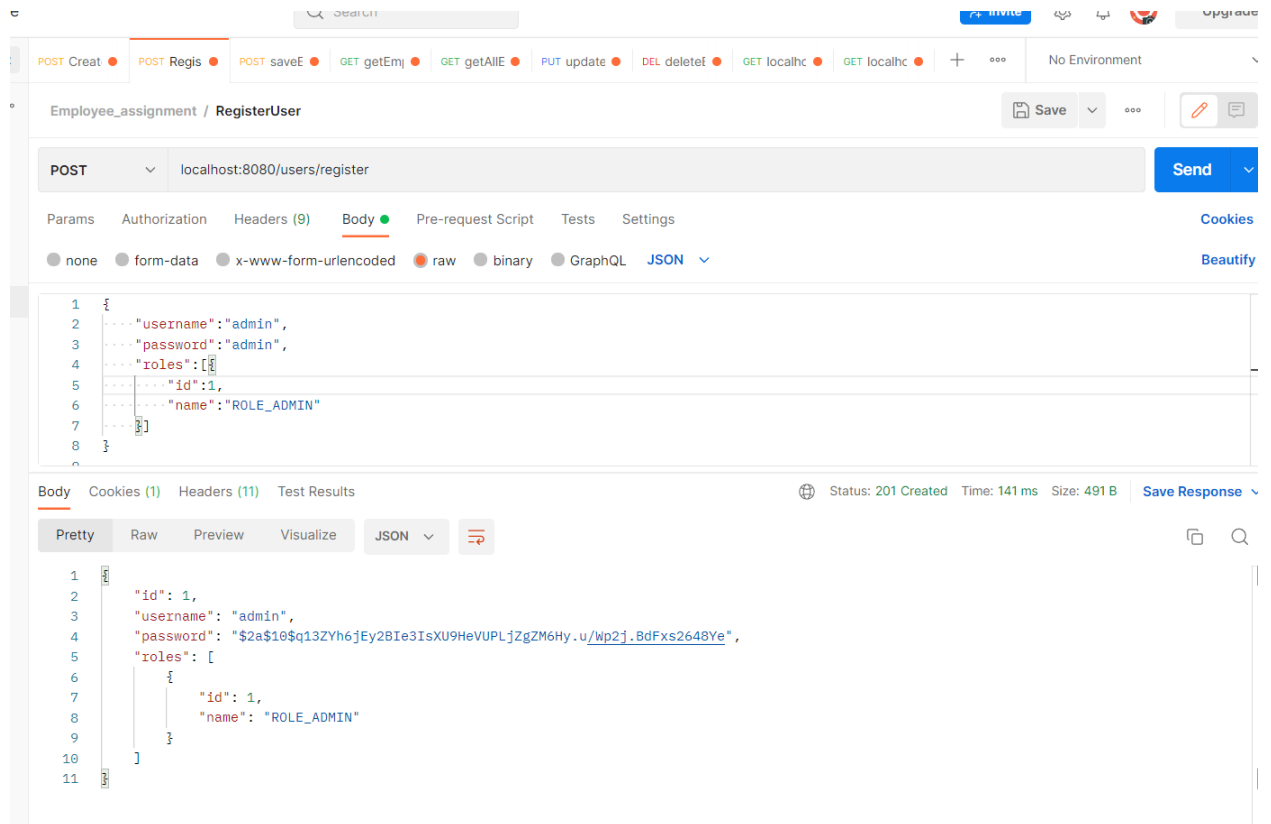
Ex-

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

Creating user with Admin Role:

localhost:8080/users/register

```
{  
  "username":"admin",  
  "password":"admin",  
  "roles":[{"  
    "id":1,  
    "name":"ROLE_ADMIN"  
  }]  
}
```



Employee_assignment / RegisterUser

POST localhost:8080/users/register

Params Authorization Headers (9) Body Pre-request Script Tests Settings

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

```

1 {
2   ...."username": "admin",
3   ...."password": "admin",
4   ...."roles": [
5     ...."id": 1,
6     ...."name": "ROLE_ADMIN"
7   ]
8 }

```

Body Cookies (1) Headers (11) Test Results

Status: 201 Created Time: 141 ms Size: 491 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 1,
3   "username": "admin",
4   "password": "$2a$10$q13ZYh6jEy28Ie3IsXU9HeVUPLjZgZM6Hy.u/Wp2j.BdFxs2648Ye",
5   "roles": [
6     {
7       "id": 1,
8       "name": "ROLE_ADMIN"
9     }
10  ]
11 }

```

Creating user with user role:

localhost:8080/users/register

```

{
  "username": "john",
  "password": "john",
  "roles": [
    {
      "id": 2,
      "name": "ROLE_USER"
    }
  ]
}

```

Employee_assignment / RegisterUser

POST localhost:8080/users/register

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

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

```

1 {
2   "username": "john",
3   "password": "john",
4   "roles": [{
5     "id": 2,
6     "name": "ROLE_USER"
7   }]
8 }

```

Body Cookies (1) Headers (11) Test Results

Status: 201 Created Time: 85 ms Size: 489 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 2,
3   "username": "john",
4   "password": "$2a$10$7v0xp5tTs/129FMax6jFJuCgJ01AtMMpSJQI3ngjUG6qDP4l7GShy",
5   "roles": [
6     {
7       "id": 2,
8       "name": "ROLE_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": "Mary",
  "lastName": "Longbottom",
  "email": "cb@gmail.com"
}

```

Saving an employee with the Admin Role

Employee_assignment / saveEmployee

POST localhost:8080/api/employees/save

Authorization Basic Auth

Username: admin

Password: admin

☒ Show Password

Status: 201 Created Time: 269 ms Size: 500 B

```

1  {
2    "id": 1,
3    "firstName": "Mary",
4    "lastName": "Longbottom",
5    "email": "cb@gmail.com"
6  }

```

Saving an employee with the User Role gives the 403 forbidden message

Employee_assignment / saveEmployee

POST localhost:8080/api/employees/save

Authorization Basic Auth

Username: john

Password: john

☒ Show Password

Status: 403 Forbidden Time: 104 ms Size: 482 B

```

1  {
2    "timestamp": "2022-07-19T04:38:08.994+00:00",
3    "status": 403,
4    "error": "Forbidden",
5    "message": "Forbidden",
6    "path": "/api/employees/save"
7  }

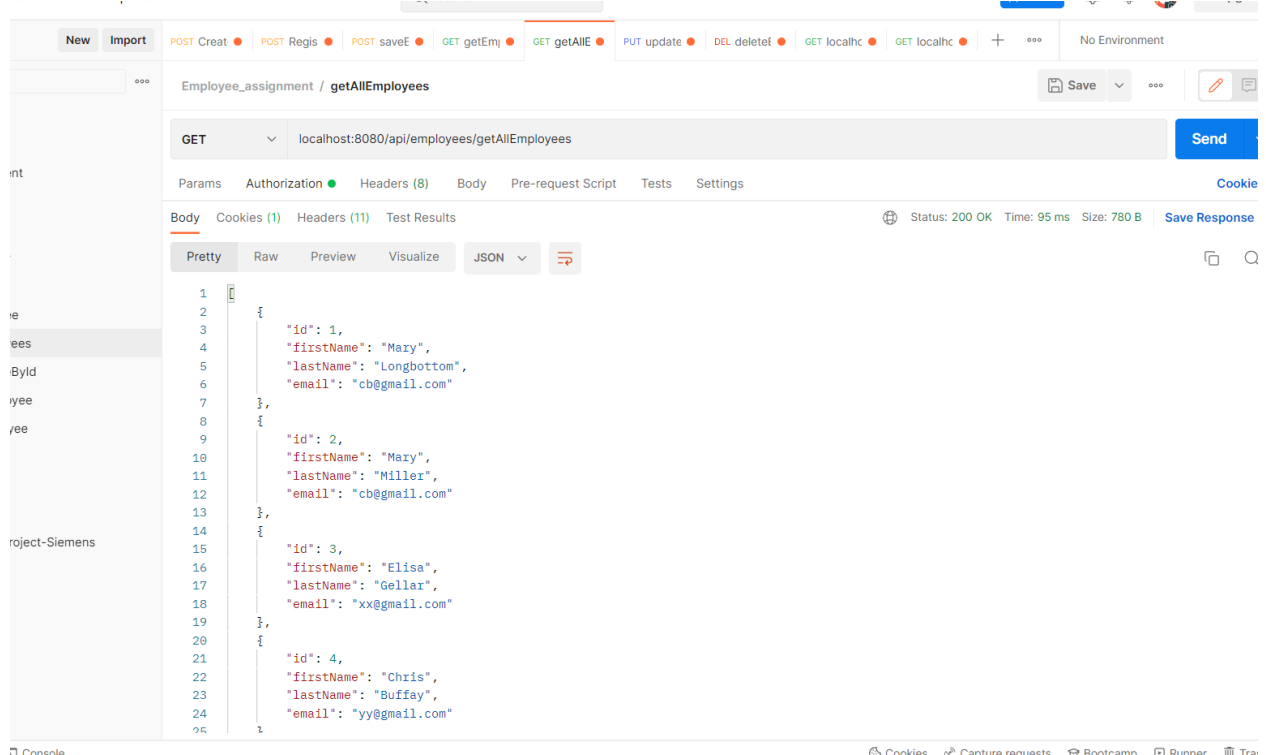
```

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

the database.

Ex-

localhost:8080/api/employees/getAllEmployees



The screenshot shows a REST client interface with a GET request to `localhost:8080/api/employees/getAllEmployees`. The response status is 200 OK, with a time of 95 ms and a size of 780 B. The response body is a JSON array of 4 employee objects, displayed in a pretty-printed format.

```
1 {
2   {
3     "id": 1,
4     "firstName": "Mary",
5     "lastName": "Longbottom",
6     "email": "cb@gmail.com"
7   },
8   {
9     "id": 2,
10    "firstName": "Mary",
11    "lastName": "Miller",
12    "email": "cb@gmail.com"
13  },
14  {
15    "id": 3,
16    "firstName": "Elisa",
17    "lastName": "Gellar",
18    "email": "xx@gmail.com"
19  },
20  {
21    "id": 4,
22    "firstName": "Chris",
23    "lastName": "Buffay",
24    "email": "yy@gmail.com"
25  }
26 }
```

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

Ex- Url- localhost:8080/api/employees/getEmployeeById/1

Employee_assignment / getEmployeeById Save

GET localhost:8080/api/employees/getEmployeeById/1

Params Authorization Headers (8) 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 recommend using variables. [Learn more about variables](#)

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

Username john

Password john

☒ Show Password

Body Cookies (1) Headers (11) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": 1,
3    "firstName": "Mary",
4    "lastName": "Longbottom",
5    "email": "cb@gmail.com"
6  }

```

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

Ex-

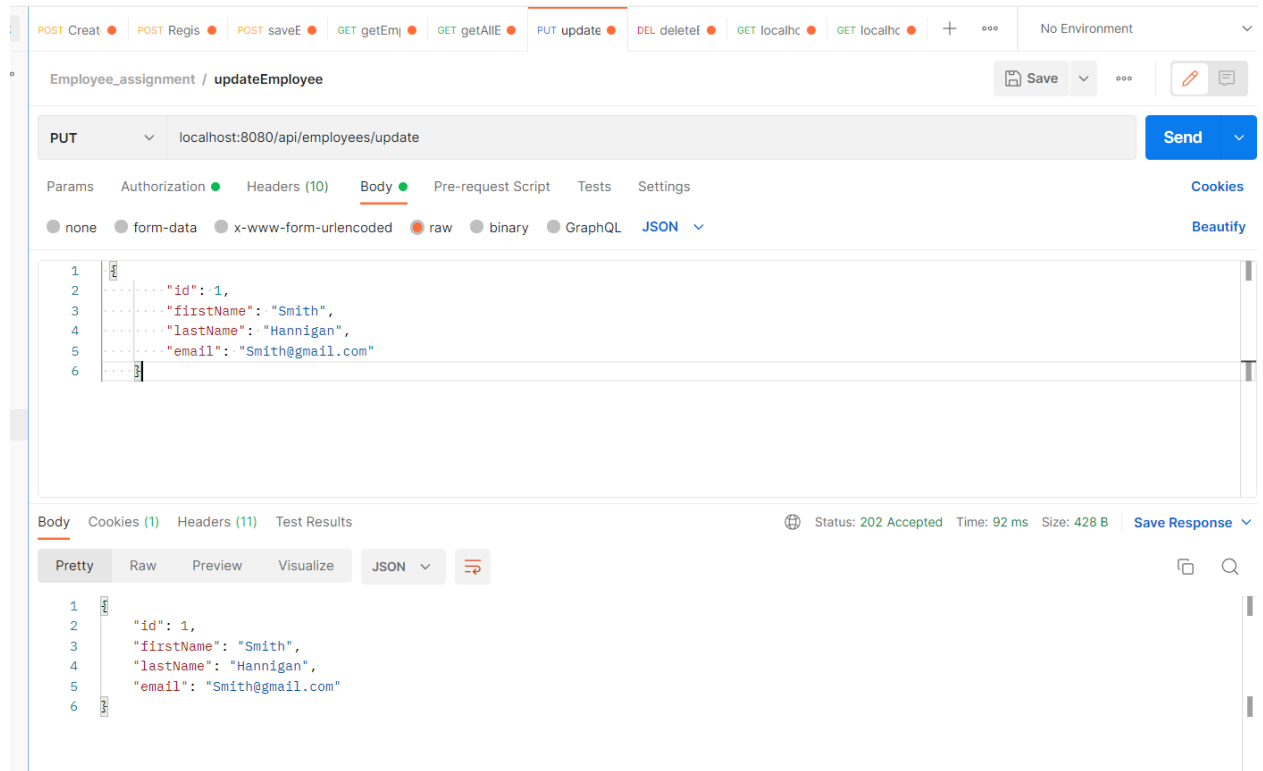
localhost:8080/api/employees/update

Request Body after updation-

```

{
  "id": 1,
  "firstName": "Smith",
  "lastName": "Hannigan",
  "email": "Smith@gmail.com"
}

```

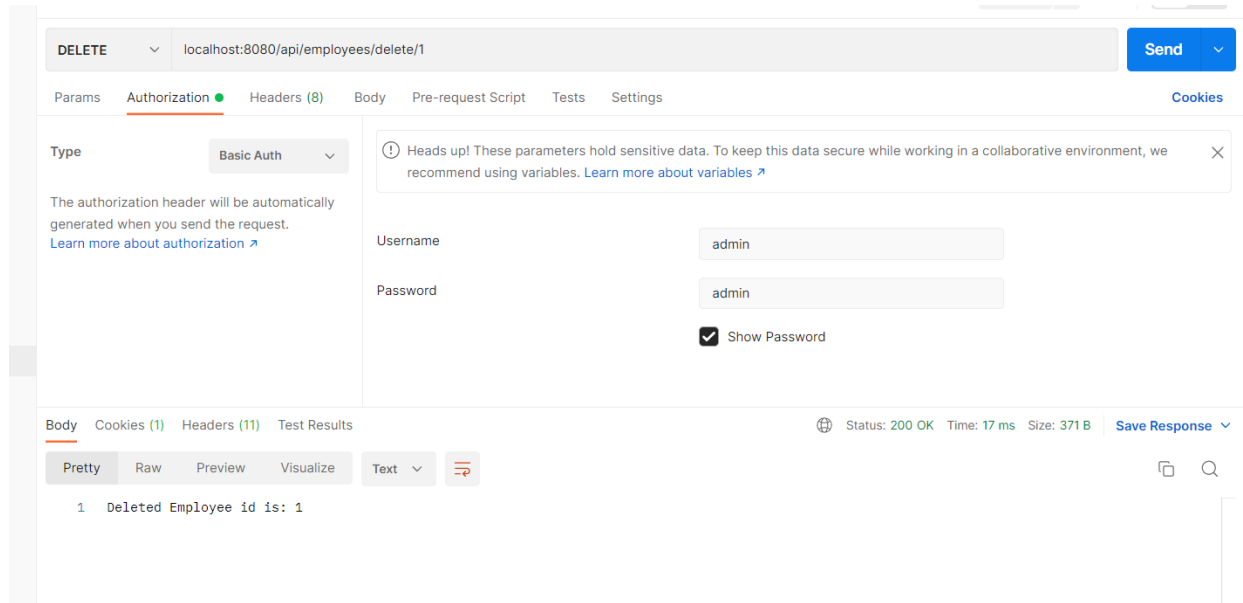


The screenshot shows a REST client interface with a list of requests at the top: POST Creat, POST Regis, POST saveE, GET getEmj, GET getAlIE, PUT update (selected), DEL deletel, GET localhc, and GET localhc. The selected request is for the endpoint `localhost:8080/api/employees/update` using the `PUT` method. The request body is a JSON object: `{ "id": 1, "firstName": "Smith", "lastName": "Hannigan", "email": "Smith@gmail.com" }`. The response status is `202 Accepted` with a time of `92 ms` and a size of `428 B`. The response body is displayed in a pretty-printed JSON format: `{ "id": 1, "firstName": "Smith", "lastName": "Hannigan", "email": "Smith@gmail.com" }`.

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

Ex-

Url- `localhost:8080/api/employees/delete/1`



The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** localhost:8080/api/employees/delete/1
- Send Button:** A blue button labeled "Send".
- Tabs:** Params, Authorization (selected), Headers (8), Body, Pre-request Script, Tests, Settings. A "Cookies" link is also present.
- Authorization:** Type is set to "Basic Auth". A warning message states: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)".
- Username:** admin
- Password:** admin
- Show Password:** Checked (indicated by a checked checkbox).
- Body:** The response is displayed in the "Body" tab, showing "1 Deleted Employee id is: 1".
- Response Details:** Status: 200 OK, Time: 17 ms, Size: 371 B. A "Save Response" button is available.
- Response Format:** Pretty, Raw, Preview, Visualize. A "Text" dropdown and a "Copy" icon are also visible.

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-

localhost:8080/api/employees/search/mary

Explore

Import POST Creat POST Regis POST saveE GET getEm GET getAlIE PUT update DEL delete GET Search GET localhc + ... No Environment Upgrade

Employee_assignment / SearchByFirstName Save ...

GET localhost:8080/api/employees/search/mary Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit

Body Cookies (1) Headers (11) Test Results Status: 200 OK Time: 11 ms Size: 490 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": 2,
3    "firstName": "Mary",
4    "lastName": "Miller",
5    "email": "cb@gmail.com"
6  },
7  {
8    "id": 7,
9    "firstName": "Mary",
10   "lastName": "Goldman",
11   "email": "aa@gmail.com"
12 }
13
14

```

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- localhost:8080/api/employees/sort?firstName=desc

OR

Url- localhost:8080/api/employees/sort?firstName=asc

rt

POST Creat ● POST Regis ● POST saveE ● GET getEmj ● GET getAIIIE ● PUT update ● DEL delete ● GET Search ● GET sorting

+

...

No Environment

Employee_assignment / sorting

Save

GET localhost:8080/api/employees/sort?firstName=desc

Send

Params ● Authorization ● Headers (8) Body Pre-request Script Tests Settings

Body Cookies (1) Headers (11) Test Results

Status: 200 OK Time: 18 ms Size: 777 B Save Res

Pretty Raw Preview Visualize JSON

```
8      {
9        "id": 7,
10       "firstName": "Mary",
11       "lastName": "Goldman",
12       "email": "aa@gmail.com"
13     },
14     {
15       "id": 6,
16       "firstName": "Jack",
17       "lastName": "Hannigan",
18       "email": "aa@gmail.com"
19     },
20     {
21       "id": 3,
22       "firstName": "Elisa",
23       "lastName": "Gellar",
24       "email": "xx@gmail.com"
25     },
26     {
27       "id": 4,
28       "firstName": "Chris",
29       "lastName": "Buffay",
30       "email": "yy@gmail.com"
31     },
32   ],
33 }
```

⌵ Cookies ⌵ Capture requests ⌵ Postman ⌵ Docker

Explore

Import POST Creat POST Regis POST saveE GET getEm GET getAIE PUT update DEL deletet GET Search GET sorting + ... No Environment Upgrade

Employee_assignment / sorting Save ...

GET localhost:8080/api/employees/sort?firstName=asc Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Body Cookies (1) Headers (11) Test Results Status: 200 OK Time: 10 ms Size: 777 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    {
3      "id": 5,
4      "firstName": "Adam",
5      "lastName": "Green",
6      "email": "aa@gmail.com"
7    },
8    {
9      "id": 4,
10     "firstName": "Chris",
11     "lastName": "Buffay",
12     "email": "yy@gmail.com"
13   },
14   {
15     "id": 3,
16     "firstName": "Elisa",
17     "lastName": "Gellar",
18     "email": "xx@gmail.com"
19   },
20   {
21     "id": 6,
22     "firstName": "Jack",
23     "lastName": "Hannigan",
24     "email": "aa@gmail.com"
  
```

Important instructions

- i) You should use the H2 In Memory database 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)

