

Advanced DataBase Systems- Project

Adella Neeharika(nadella@kent.edu)

GitHub repository Url :https://github.com/neehariko/ADBS_Project.git

This is the Postgres Admin application(pgadmin). Here I have used Users and Orders tables.

The screenshot shows the pgAdmin interface with the following details:

- Object Explorer:** On the left, it shows the database structure:
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - Publications
 - Schemas (1)
 - public
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Operators
 - Procedures
 - Sequences
 - Tables (3)
 - Users
 - Columns (3)
 - id
 - name
 - email
 - Constraints
 - Indexes
 - RLS Policies
 - Rules
 - Triggers
 - orders
 - Columns (4)
 - order_id
 - order_description
 - user_id
- Query Editor:** The main window contains a SQL query editor with the following code:

```
1 CREATE TABLE orders (
2     order_id INT PRIMARY KEY,
3     order_description VARCHAR(255) NOT NULL,
4     user_id INT,
5     FOREIGN KEY (user_id) REFERENCES Users(id)
6 );
7
8 Insert into orders(order_id,order_description,user_id)
9 values (1234,'Note books',2);
10
11 select * from orders;
12
13 SELECT o FROM Orders o JOIN FETCH o.users;
14
15 Select * from Orders;
```
- Data Output:** Below the query editor, it shows "Total rows: 2 of 2" and "Query complete 00:00:00.370".
- Notifications:** A table at the bottom right shows recorded events with columns: Recorded time, Event, Process ID, and Payload.

Below attached are Users tables's model class, controller class, repository class

The screenshot shows a Java-based Spring Boot application structure in an IDE. The project is named "CRUD-Demo". The file "Users.java" is open in the code editor, which contains the following code:

```
1 package com.Neeharika.CRUDDemo.Model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 @Entity
9 public class Users {
10     @Id
11     private int id;
12     private String name;
13     private String email;
14
15     public Users() {
16     }
17
18     public Users(int id, String name, String email) {
19         this.id = id;
20         this.name = name;
21         this.email = email;
22     }
23
24     public int getId() {
25         return id;
26     }
27
28     public void setId(int id) {
29         this.id = id;
30     }
31
32     public String getName() {
33         return name;
34     }
35
36     public void setName(String name) {
37         this.name = name;
38     }
39
40     public String getEmail() {
41         return email;
42     }
43
44     public void setEmail(String email) {
45         this.email = email;
46     }
47 }
```

The project structure on the left includes:

- Project (Idea)
- .mvn
- src
 - main
 - java
 - com.Neeharika.CRUDDemo
 - Controller
 - OrdersController
 - UsersController
 - Model
 - Orders
 - Users
 - Repository
 - OrdersRepo
 - UserRepo
 - Service
 - OrdersService
 - UserService
 - resources
 - static
 - templates
 - application.properties
 - test
 - target
 - gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

CRUD-Demo Version control

UserRepo.java

```
package com.Neeharika.CRUDDemo.Repository;

import com.Neeharika.CRUDDemo.Model.Users;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepo extends JpaRepository<Users, Integer> { }
```

Project

- .idea
- .mvn
- src
 - main
 - java
 - com.Neeharika.CRUDDemo.Controller
 - OrdersController
 - UsersController
 - Model
 - Orders
 - Users
 - Repository
 - OrdersRepo
 - UserRepo
 - Service
 - OrdersService
 - UserService
 - resources
 - static
 - templates
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

Run Unnamed

CRUD-Demo > src > main > java > com > Neeharika > CRUDDemo > Repository > UserRepo

7:12 LF UTF-8 4 spaces

CRUD-Demo Version control

UserController.java

```
package com.Neeharika.CRUDDemo.Controller;

import com.Neeharika.CRUDDemo.Model.Orders;
import com.Neeharika.CRUDDemo.Model.Users;
import com.Neeharika.CRUDDemo.Repository.OrdersRepo;
import com.Neeharika.CRUDDemo.Repository.UserRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
public class UsersController {
    @Autowired
    private UserRepo userRepo;
    //get

    @GetMapping("/")
    public String welcomeMessage(){
        return "Welcome";
    }

    @GetMapping("/getUsers")
    public List<Users> getUsers(){
        return this.userRepo.findAll();
    }

    //get user by id
    @GetMapping("/getUsers/{id}")
    public Optional<Users> getUserId(@PathVariable(value = "id") int id ){
        Optional<Users> user;
        return user;
    }
}
```

Project

- .idea
- .mvn
- src
 - main
 - java
 - com.Neeharika.CRUDDemo.Controller
 - OrdersController
 - UsersController
 - Model
 - Orders
 - Users
 - Repository
 - OrdersRepo
 - UserRepo
 - Service
 - OrdersService
 - UserService
 - resources
 - static
 - templates
 - test
 - target
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

Run Unnamed

CRUD-Demo > src > main > java > com > Neeharika > CRUDDemo > Service > OrdersService

40:41 LF UTF-8 4 spaces

Below attached are Orders table's model,repository and controller classes

The screenshot shows the IntelliJ IDEA interface with the file `OrdersRepo.java` open in the editor. The code defines a repository interface for managing orders. It imports various Java and Spring framework packages, including `com.Neeharika.CRUDDemo.Model.Orders`, `jakarta.persistence.criteria.Order`, `org.hibernate.query.Query`, `org.springframework.data.jpa.repository.JpaRepository`, `org.springframework.data.jpa.repository.Query`, `org.springframework.stereotype.Repository`, and `java.util.List`. The interface itself extends `JpaRepository<Orders, Integer>` and contains two methods: `findAllOrdersWithUsers()` and `findAllOrdersWithUsers()`.

```
1 package com.Neeharika.CRUDDemo.Repository;
2
3 import com.Neeharika.CRUDDemo.Model.Orders;
4 import jakarta.persistence.criteria.Order;
5 //import org.hibernate.query.Query;
6 import org.springframework.data.jpa.repository.JpaRepository;
7 //import org.springframework.data.jpa.repository.Query;
8 import org.springframework.data.jpa.repository.Query;
9 import org.springframework.stereotype.Repository;
10
11 import java.util.List;
12
13 @Repository
14 public interface OrdersRepo extends JpaRepository<Orders, Integer> {
15
16     //Query("SELECT o FROM Order o JOIN FETCH o.user")
17     @Query("SELECT o FROM Orders o JOIN FETCH o.user")
18     List<Orders> findAllOrdersWithUsers();
19 }
```

The screenshot shows the IntelliJ IDEA interface with the file `OrdersController.java` open in the editor. This controller handles requests for orders. It imports `com.Neeharika.CRUDDemo.Model.Orders`, `com.Neeharika.CRUDDemo.Model.Users`, `com.Neeharika.CRUDDemo.Repository.OrdersRepo`, `jakarta.persistence.criteria.Order`, `org.springframework.beans.factory.annotation.Autowired`, `org.springframework.http.ResponseEntity`, `org.springframework.web.bind.annotation.*`, `java.util.List`, and `java.util.Optional`. The controller is annotated with `@RestController` and contains two methods: `getOrders()` and `getOrderById()`.

```
1 package com.Neeharika.CRUDDemo.Controller;
2
3 import com.Neeharika.CRUDDemo.Model.Orders;
4 import com.Neeharika.CRUDDemo.Model.Users;
5 import com.Neeharika.CRUDDemo.Repository.OrdersRepo;
6 import jakarta.persistence.criteria.Order;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.web.bind.annotation.*;
10
11 import java.util.List;
12 import java.util.Optional;
13
14 @RestController
15 public class OrdersController {
16
17     @Autowired
18     private OrdersRepo ordersRepo;
19
20     @GetMapping("/getOrders")
21     public List<Orders> getOrders(){
22         return this.ordersRepo.findAll();
23         //return this.ordersRepo.findAllOrdersWithUsers();
24     }
25
26     @GetMapping("/getOrders/{id}")
27     public Optional<Orders> getOrderById(@PathVariable(value = "id") int id){
28         Optional<Orders> order;
29         order = ordersRepo.findById(id);
30         return (order);
31     }
32 }
```

The screenshot shows the IntelliJ IDEA interface with the project 'CRUD-Demo' open. The left sidebar displays the project structure, including the 'src' directory with 'main' and 'java' sub-directories containing 'com.Neeharika.CRUDDemo' package. Inside 'Orders.java', the code is shown:

```

15     @ManyToOne
16     @JoinColumn(name = "user_id", referencedColumnName = "id")
17     private Users user;
18
19     public int getOrder_id() {
20         return order_id;
21     }
22
23     public void setOrder_id(int order_id) {
24         this.order_id = order_id;
25     }
26
27     public String getOrder_description() {
28         return order_description;
29     }
30
31     public void setOrder_description(String order_description) {
32         this.order_description = order_description;
33     }
34
35     public Users getUser() {
36         return user;
37     }
38
39     public void setUser(Users user) {
40         this.user = user;
41     }
42
43     public Orders() {
44
45
46

```

The status bar at the bottom shows the path 'CRUD-Demo > src > main > java > com > Neeharika > CRUDDemo > Model > Orders > @getOrder_description' and the time '28:41 (22 chars) LF UTF-8 4 spaces'.

Below attached is the properties of database connection

The screenshot shows the IntelliJ IDEA interface with the project 'CRUD-Demo' open. The left sidebar displays the project structure, including the 'resources' directory with 'application.properties'. The code in 'application.properties' is:

```

1 spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
2 spring.datasource.username=vs
3 spring.datasource.password=vs
4 spring.jpa.hibernate.ddl-auto=update
5 spring.h2.console.enabled=true
6 spring.jpa.show-sql=true
7 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
8 spring.sql.init.mode=always
9 spring.sql.init.platform=postgres
10 spring.jpa.defer-datasource-initialization=true
11
12
13 spring.http.converters.preferred-json-mapper=jackson
14

```

A tooltip 'Unused property' is displayed over the line 'spring.http.converters.preferred-json-mapper=jackson'. The status bar at the bottom shows the path 'CRUD-Demo > src > main > resources > application.properties' and the time '14:1 LF ISO-8859-1 4 spaces'.

Read All Users

The screenshot shows the Postman interface with a collection named "expense_tracker". A new request is being made to the URL `http://localhost:8080/getUsers`. The method is set to GET. The response body is displayed in JSON format, showing a list of users:

```
1 [ {  
2     "id": 2,  
3     "name": "Nikku",  
4     "email": "Nikku@gmail"  
5 },  
6 {  
7     "id": 1,  
8     "name": "Nani",  
9     "email": "Nani@gmail"  
10 },  
11 {  
12     "id": 4,  
13     "name": "Neeharika",  
14     "email": "Neeharika@gmail"  
15 }  
16 ]
```

The status bar at the bottom indicates a 200 OK response with a time of 52 ms.

Read particular User

The screenshot shows the Postman interface with the same collection. A new request is being made to the URL `http://localhost:8080/getUsers/2`. The method is set to GET. The response body is displayed in JSON format, showing a single user:

```
1 [ {  
2     "id": 2,  
3     "name": "Nikku",  
4     "email": "Nikku@gmail"  
5 }]
```

The status bar at the bottom indicates a 200 OK response with a time of 124 ms.

Insert one user

The screenshot shows two separate API requests in the Postman interface.

Request 1: Insert one user

- Method: POST
- URL: `http://localhost:8080/getUsers`
- Body (JSON):

```
1 ...
2 ... "id": 4,
3 ... "name": "Neeharika",
4 ... "email": "Neeharika@gmail.com"
```

Request 2: Update user

- Method: PUT
- URL: `http://localhost:8080/getUsers/2`
- Body (JSON):

```
1 ...
2 ... "id": 2,
3 ... "name": "Nikku_updated",
4 ... "email": "Nikku-Updated@gmail.com"
```

Above is for Update user

Delete User

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Home', 'Workspaces', 'API Network', and 'Explore' tabs. A search bar says 'Search Postman'. On the right, there are buttons for 'Invite', 'Upgrade', and environment settings.

The main workspace is titled 'My Workspace'. It contains two collections: 'expense_tracker' and 'REST API basics: CRUD, test & variable'. A 'DELETE' request is selected in the list, with the URL 'http://localhost:8080/getUsers/3' displayed.

The request details panel shows the method as 'DELETE' and the URL as 'http://localhost:8080/getUsers/3'. The 'Body' tab is active, showing the following JSON payload:

```
1 {  
2   "id": 4,  
3   "name": "Neeharika",  
4   "email": "Neeharika@gmail.com"  
5 }
```

Below the body, the 'Test Results' section shows a single result: 'Deleted: chinini'. At the bottom, status information is provided: 'Status: 200 OK Time: 102 ms Size: 179 B'.

At the very bottom, there are various status icons: 'Online' (green), 'Find and replace' (blue), 'Console' (grey), 'Postbot' (grey), 'Runner' (grey), 'Start Proxy' (grey), 'Cookies' (grey), 'Trash' (grey), and a help icon (grey).

Home Workspaces API Network Explore

My Workspace New Import POST New Request GET http://localhost:8080/getUsers POST http://localhost:8080/putUsers PUT http://localhost:8080/putUsers DEL http://localhost:8080/deleteUsers + *** No Environment

Collections Environments History

HTTP http://localhost:8080/getUsers

GET http://localhost:8080/getUsers

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

Query Params

Key	Value	Description	... Bulk Edit
Key	Value	Description	...

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 2,
4     "name": "Nikku",
5     "email": "Nikku@gmail"
6   },
7   {
8     "id": 1,
9     "name": "Nani",
10    "email": "Nani@gmail"
11  },
12  {
13    "id": 3,
14    "name": "chinni",
15    "email": "chinni@gmail"
16  }
17 ]
```

Status: 200 OK Time: 346 ms Size: 303 B Save as example

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

This screenshot shows the Postman interface with a successful GET request to the '/getUsers' endpoint. The response body is a JSON array containing three user objects. Each user has an 'id', 'name', and 'email' field.

id	name	email
2	Nikku	Nikku@gmail
1	Nani	Nani@gmail
3	chinni	chinni@gmail

Delete order

Home Workspaces API Network Explore

My Workspace New Import POST New Request DEL http://localhost:8080/getOrders/1234 POST http://localhost:8080/putOrders/1234 PUT http://localhost:8080/putOrders/1234 DEL http://localhost:8080/deleteOrders/1234 + *** No Environment

Collections Environments History

HTTP http://localhost:8080/getOrders/1234

DELETE http://localhost:8080/getOrders/1234

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

Body

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

Pretty Raw Preview Visualize Text

```
1 {
2   "order_id": 12,
3   "order_description": "New Year Gifts",
4   "user": {
5     "id": 2,
6     "name": "Nikku",
7     "email": "Nikku@gmail"
8   }
9 }
```

Status: 200 OK Time: 109 ms Size: 188 B Save as example

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

This screenshot shows the Postman interface with a successful DELETE request to the '/getOrders/1234' endpoint. The response body is a simple message indicating the order was deleted.

Deleted: 1234 Note books

The screenshot shows the Postman application interface. In the top navigation bar, there are links for Home, Workspaces, API Network, and Explore. A search bar is located at the top right. The main workspace is titled "My Workspace". On the left sidebar, there are sections for Collections (with "expense_tracker" selected), Environments, and History. The main content area displays a request configuration for a PUT operation to "http://localhost:8080/getOrders/12". The "Body" tab is selected, showing a JSON payload:

```
1 {
2     "order_id": 12,
3     "order_description": "New Year Gifts",
4     "user": {
5         "id": 2,
6         "name": "Nikku",
7         "email": "Nikku@gmail"
8     }
9 }
```

Below the body, the response status is shown as 200 OK with a time of 86 ms and a size of 269 B. The response body is identical to the request body. At the bottom of the interface, there are various icons for Postbot, Runner, Start Proxy, Cookies, Trash, and Help.

Above is Update particular order

Read particular order

The screenshot shows the Postman interface with a request to read a specific order. The URL is `http://localhost:8080/getOrders/1234`. The response status is 200 OK, time 72 ms, size 267 B. The response body is:

```
1 "order_id": 1234,
2   "order_description": "Note books",
3   "user": {
4     "id": 2,
5     "name": "Nikku",
6     "email": "Nikku@gmail"
7 }
```

The screenshot shows the Postman interface with a request to read all orders. The URL is `http://localhost:8080/getOrders`. The response status is 200 OK, time 85 ms, size 383 B. The response body is:

```
1 [
2   {
3     "order_id": 1234,
4     "order_description": "Note books",
5     "user": {
6       "id": 2,
7       "name": "Nikku",
8       "email": "Nikku@gmail"
9     }
10   },
11   {
12     "order_id": 123456,
13     "order_description": "third Perfume bottle",
14     "user": {
15       "id": 1,
16       "name": "Nani",
17       "email": "Nani@gmail"
18     }
19   }
20 ]
```

Above is read all orders

- 1.I have used SpringBoot with Postgres and Postman tool
- 2.Used two tables- Users and Orders
- 3.Users(id,name,email)
Orders(order_id,order_description,User_id) user_id references to id in Users table acts as foreign key
4. This is the simple web api, we can integrate it to any service who are in need of it.(I am working on it to add Simple UI to it), I will push my remaining work to GIT.
5. The reason why I used two tables is: I don't want to make ambiguity by putting orders and users in the same table, another one is I want to create many to one relations from orders to user because users may have many orders.
The last important one is, i wanted to show relations between them maintain scalability and achieve Normalisation.
6. I have implemented controllers for CRUD operations on Users and Orders separately. You can create a user separately but when you create an Order, you have to associate its user as well(Thats is how I have implemented these operations).
- 7.The reasons why i choose this database is that PostgreSQL is reliable and powerful dbms, spring boot is known for its enterprise grade features, scalability, it provides comprehensive ecosystem that includes Spring Data JPA for easy database integrations which simplifies CRUD operations.

Below is the code

Orders.java

```
package com.Neeharika.CRUDDemo.Model;
```

```
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
```

```
@Entity
public class Orders {

    @Id
    private int order_id;
    private String order_description;
```

```

    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private Users user;

    public int getOrder_id() {
        return order_id;
    }

    public void setOrder_id(int order_id) {
        this.order_id = order_id;
    }

    public String getOrder_description() {
        return order_description;
    }

    public void setOrder_description(String order_description) {
        this.order_description = order_description;
    }

    public Users getUser() {
        return user;
    }

    public void setUser(Users user) {
        this.user = user;
    }

    public Orders() {
    }

}

```

OrdersController.java

```

package com.Neeharika.CRUDDemo.Controller;

import com.Neeharika.CRUDDemo.Model.Orders;
import com.Neeharika.CRUDDemo.Model.Users;
import com.Neeharika.CRUDDemo.Repository.OrdersRepo;
import jakarta.persistence.criteria.Order;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

```

```

import java.util.List;
import java.util.Optional;

@RestController
public class OrdersController {

    @Autowired
    private OrdersRepo ordersRepo;

    @GetMapping("/getOrders")
    public List<Orders> getOrders(){
        return this.ordersRepo.findAll();
        //return this.ordersRepo.findAllOrdersWithUsers();
    }

    @GetMapping("/getOrders/{id}")
    public Optional<Orders> getOrderById(@PathVariable(value = "id") int id ){
        Optional<Orders> order;
        order = ordersRepo.findById(id);
        return (order);
    }

    @PostMapping("/getOrders")
    public Orders createOrder(@RequestBody Orders order){
        return this.ordersRepo.save(order);
    }

    @PutMapping("/getOrders/{id}")
    public ResponseEntity<Orders> updateOrder(@PathVariable(value = "id") int id, @RequestBody
Orders newOrder ){
        Orders order;
        order = ordersRepo.findById(id).orElse(null);
        assert order != null;
        order.setOrder_description(newOrder.getOrder_description());
        //order.setName(newUser.getName());
        return ResponseEntity.ok(this.ordersRepo.save(order));
    }

    @DeleteMapping("/getOrders/{id}")
    public String deleteEmployee( @PathVariable (value ="id") int id){
        Orders order;
        order = ordersRepo.findById(id).orElse(null);
        this.ordersRepo.delete(order);
        return "Deleted: " + order.getOrder_id() + " " + order.getOrder_description();
    }
}

```

```
}
```

OrdersRepo.java

```
package com.Neeharika.CRUDDemo.Repository;

import com.Neeharika.CRUDDemo.Model.Orders;
import jakarta.persistence.criteria.Order;
//import org.hibernate.query.Order;
import org.springframework.data.jpa.repository.JpaRepository;
//import org.springframework.data.jpa.repository.Query;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface OrdersRepo extends JpaRepository<Orders, Integer> {

    // @Query("SELECT o FROM Order o JOIN FETCH o.user")
    // @Query("SELECT o FROM Orders o JOIN FETCH o.user")
    List<Orders> findAllOrdersWithUsers();
}
```

UserRepo.java

```
package com.Neeharika.CRUDDemo.Repository;

import com.Neeharika.CRUDDemo.Model.Users;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepo extends JpaRepository<Users, Integer> {
```

Users.java

```
package com.Neeharika.CRUDDemo.Model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
```

```
public class Users {  
    @Id  
    private int id;  
    private String name;  
    private String email;  
  
    public Users() {}  
  
    public Users(int id, String name, String email) {  
        this.id = id;  
        this.name = name;  
        this.email = email;  
    }  
  
    public int getId() {  
        return id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
  
    public void setEmail(String email) {  
        this.email = email;  
    }  
}
```

UsersController.java

```
package com.Neeharika.CRUDDemo.Controller;
```

```
import com.Neeharika.CRUDDemo.Model.Orders;
import com.Neeharika.CRUDDemo.Model.Users;
import com.Neeharika.CRUDDemo.Repository.OrdersRepo;
import com.Neeharika.CRUDDemo.Repository.UserRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
public class UsersController {
    @Autowired
    private UserRepo userRepo;
    //get

    @GetMapping("/")
    public String welcomeMessage(){
        return "Welcome";
    }
    @GetMapping("/getUsers")
    public List<Users> getUsers(){
        return this.userRepo.findAll();
    }

    //get user by id
    @GetMapping("/getUsers/{id}")
    public Optional<Users> getUserById(@PathVariable(value = "id") int id ){
        Optional<Users> user;
        user = userRepo.findById(id);
        return (user);
    }

    //create user
    @PostMapping("/getUsers")
    public Users createUser(@RequestBody Users user){
        return this.userRepo.save(user);
    }

    //Update User
    @PutMapping("/getUsers/{id}")
```

```
public ResponseEntity<Users> updateUser(@PathVariable(value = "id") int id,
@RequestParam Users newUser ){
    Users user;
    user = userRepo.findById(id).orElse(null);
    assert user != null;
    user.setEmail(newUser.getEmail());
    user.setName(newUser.getName());
    return ResponseEntity.ok(this.userRepo.save(user));
}

//Delete User
@DeleteMapping("/getUsers/{id}")
public String deleteEmployee( @PathVariable (value ="id") int id){
    Users user;
    user = userRepo.findById(id).orElse(null);
    this.userRepo.delete(user);
    return "Deleted: " + user.getName();
}

}
```