Setup Spring Boot Project

Create the project:

Go to Spring Initializr.
Fill in the project details:
Project: Maven
Language: Java
Spring Boot version: (Latest stable version, e.g., 3.1.x)
Dependencies:
Spring Web
Spring Data JPA
PostgreSQL Driver
Click **Generate** to download the project.

Unzip and Import:

Unzip the project.
Import it into your IDE (e.g., IntelliJ IDEA or Eclipse) as a Maven project.

2. Configure `application.properties`

Configure the connection to the PostgreSQL database in `src/main/resources/application.properties` (or `application.yml` if using YAML):

```
properties
Copy code
spring.datasource.url=jdbc:postgresql://localhost:5432/your_database
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
```

3. Create PostgreSQL Database

Before running the application, make sure PostgreSQL is installed and running. Then:

Create a new database
Update the username and password in `application.properties` to match your PostgreSQL setup.

4. Define an Entity

Create a simple entity class in the `src/main/java/com/example/demo` package (or appropriate package).

```
java
Copy code
package com.example.demo.entity;
import jakarta.persistence.Entity;import
jakarta.persistence.GeneratedValue;import
jakarta.persistence.GenerationType;import jakarta.persistence.Id;
@Entitypublic class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    // Getters and Setters
    public Long getId() {
        return id;
    }
    public void setId(Long 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;
    }
}
```

5. Create a Repository

Create a JPA repository interface for CRUD operations.

```
java
Copy code
package com.example.demo.repository;
import com.example.demo.entity.User;import
org.springframework.data.jpa.repository.JpaRepository;
public interface UserRepository extends JpaRepository<User, Long> {
}
```

6. Create a REST Controller

Create a REST controller to expose CRUD operations.

```
java
Copy code
package com.example.demo.controller;
import com.example.demo.entity.User;import
com.example.demo.repository.UserRepository;import
org.springframework.beans.factory.annotation.Autowired;import
org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController@RequestMapping("/users")public class UserController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userRepository.save(user);
    }
}
```

7. Run the Application

Run the application:In the terminal:

```
bash
Copy code
```

```
mvn spring-boot:run
```

Or in your IDE by running the `main` method in the `DemoApplication` class.

8. Test the Application

Use a tool like Postman or `curl` to test the endpoints.

GET all users:

```
bash
Copy code
curl -X GET http://localhost:8080/users
```

POST a new user:

```
bash
Copy code
curl -X POST -H "Content-Type: application/json" -d
'{"name":"John Doe", "email":"john.doe@example.com"}'
http://localhost:8080/users
```