write curd using java 8 & spring boot

**ChatGPT said:**

ChatGPT

Here’s an example of a Spring Boot application that implements CRUD (Create, Read, Update, Delete) operations using Java 8 with Spring Data JPA. This example demonstrates how to manage users in a users table.

**Step 1: Setup**

**Add Dependencies**

Add the following dependencies to your pom.xml file for Spring Boot and JPA:

xml

Copy code

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

</dependencies>

**Step 2: Database Configuration**

Add your database configuration to application.properties:

properties

Copy code

spring.datasource.url=jdbc:mysql://localhost:3306/your\_database

spring.datasource.username=your\_username

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**Step 3: Create the User Entity**

java

Copy code

import jakarta.persistence.\*;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Size;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotBlank

@Size(max = 100)

private String name;

@NotBlank

@Email

private String email;

private int age;

// 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;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

**Step 4: Create the Repository**

java

Copy code

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**Step 5: Create the Service**

java

Copy code

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

// Create or Update a user

public User saveUser(User user) {

return userRepository.save(user);

}

// Read all users

public List<User> getAllUsers() {

return userRepository.findAll();

}

// Read a user by ID

public Optional<User> getUserById(Long id) {

return userRepository.findById(id);

}

// Delete a user

public void deleteUser(Long id) {

userRepository.deleteById(id);

}

}

**Step 6: Create the Controller**

java

Copy code

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/users")

public class UserController {

@Autowired

private UserService userService;

// Create or Update

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) {

return new ResponseEntity<>(userService.saveUser(user), HttpStatus.CREATED);

}

// Read All

@GetMapping

public List<User> getAllUsers() {

return userService.getAllUsers();

}

// Read By ID

@GetMapping("/{id}")

public ResponseEntity<User> getUserById(@PathVariable Long id) {

return userService.getUserById(id)

.map(user -> new ResponseEntity<>(user, HttpStatus.OK))

.orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

}

// Update

@PutMapping("/{id}")

public ResponseEntity<User> updateUser(@PathVariable Long id, @RequestBody User userDetails) {

return userService.getUserById(id)

.map(user -> {

user.setName(userDetails.getName());

user.setEmail(userDetails.getEmail());

user.setAge(userDetails.getAge());

return new ResponseEntity<>(userService.saveUser(user), HttpStatus.OK);

})

.orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

}

// Delete

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteUser(@PathVariable Long id) {

if (userService.getUserById(id).isPresent()) {

userService.deleteUser(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

} else {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

}

}

**Step 7: Run the Application**

Start the Spring Boot application. Use tools like **Postman** or **cURL** to test the API:

1. **Create User**:  
   POST /api/users  
   Body:

json

Copy code

{

"name": "Alice",

"email": "alice@example.com",

"age": 25

}

1. **Get All Users**:  
   GET /api/users
2. **Get User by ID**:  
   GET /api/users/{id}
3. **Update User**:  
   PUT /api/users/{id}  
   Body:

json

Copy code

{

"name": "Alice Smith",

"email": "alice.smith@example.com",

"age": 26

}

1. **Delete User**:  
   DELETE /api/users/{id}

Let me know if you need further assistance!

**You said:**

userService.getUserById(id).isPresent()

**ChatGPT said:**

ChatGPT

In Java 8, the Optional class provides a convenient way to handle null values. The isPresent() method of Optional is used to check whether the Optional contains a non-null value. However, in some scenarios, we might want to use other functional programming features like ifPresent() or orElse().

Here’s an example explaining the usage of userService.getUserById(id).isPresent() in the context of the deleteUser method:

**Using isPresent()**

java

Copy code

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteUser(@PathVariable Long id) {

if (userService.getUserById(id).isPresent()) {

userService.deleteUser(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

} else {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

}

**Explanation**

1. **userService.getUserById(id)**: Returns an Optional<User> from the database.
2. **isPresent()**: Checks if the Optional contains a User object.
3. If a user exists, the deleteUser method is called, and 204 No Content is returned.
4. If no user exists, 404 Not Found is returned.

**Alternative: Using orElseThrow()**

You can rewrite the method to use orElseThrow() for better readability:

java

Copy code

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteUser(@PathVariable Long id) {

userService.getUserById(id)

.orElseThrow(() -> new RuntimeException("User not found"));

userService.deleteUser(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

Here, if the user is not found, an exception is thrown.

**Functional Approach: Using ifPresent()**

Alternatively, you can use ifPresent():

java

Copy code

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteUser(@PathVariable Long id) {

Optional<User> user = userService.getUserById(id);

user.ifPresent(u -> userService.deleteUser(id));

return user.isPresent() ? new ResponseEntity<>(HttpStatus.NO\_CONTENT)

: new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

**Best Practices**

* Use isPresent() when you need a simple check.
* Use orElseThrow() when you want to throw a custom exception if the value is not present.
* Use ifPresent() for concise functional-style handling.