

# Spring Data JPA

## 1. CRUD Data User

### a. Create, Read, Update, Delete Code

#### i. Model

```
User.java
package com.ecommerce.ecommerce.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;
import java.io.Serializable;

10 usages
@Entity(name = "User")
@Table(name = "users")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class User implements Serializable {

    private static final long serialVersionUID = -449241275888190961L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false, insertable = false, unique = true)
    private Long id;

    @Column(name = "username", nullable = false, length = 255)
    private String username;

    @Column(name = "phone", nullable = false, length = 12)
    private String phone;

    @Column(name = "address", nullable = false, columnDefinition = "TEXT")
    private String address;
}
```

#### ii. Repository

```
User.java x UserRepository.java x
package com.ecommerce.ecommerce.repository;

import com.ecommerce.ecommerce.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

2 usages
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

#### iii. DTO

```
User.java x UserRepository.java x UserService.java
package com.ecommerce.ecommerce.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

4 usages
@Data
@AllArgsConstructor
@NoArgsConstructor
public class CreateUserDto {

    private String username;
    private String phone;
    private String address;
}
```

```

1 package com.ecommerce.ecommerce.dto;
2
3 import ...
4
5 4 usages
6 @Data
7 @AllArgsConstructor
8 @NoArgsConstructor
9 public class UpdateUserDto {
10
11     private Long id;
12     private String username;
13     private String phone;
14     private String address;
15 }

```

```

1 package com.ecommerce.ecommerce.dto;
2
3 import ...
4
5 5 usages
6 @Data
7 @AllArgsConstructor
8 @NoArgsConstructor
9 public class DeleteUserDto {
10
11     private Long id;
12 }

```

#### iv. Service

```

23 @Service
24 @Transactional(rollbackOn = Exception.class)
25 public class UserService {
26
27     6 usages
28     @Autowired
29     private UserRepository userRepository;
30
31     1 usage
32     @SneakyThrows(Exception.class)
33     public ResponseEntity<Object> create(CreateUserDto dto){
34
35         HttpHeaders headers = new HttpHeaders();
36         headers.setContentType(MediaType.APPLICATION_JSON);
37
38         Map<String, Object> res = new HashMap<>();
39
40         User user = new User();
41         user.setUsername(dto.getUsername().trim());
42         user.setPhone(dto.getPhone().trim());
43         user.setAddress(dto.getAddress().trim());
44
45         userRepository.save(user);
46
47         res.put("code", 201);
48         res.put("message", "success");
49         res.put("data", user);
50
51         return ResponseEntity.status(HttpStatus.CREATED).headers(headers).body(res);
52     }

```

```

52     @SneakyThrows(Exception.class)
53     @PostMapping
54     public ResponseEntity<Object> update(UpdateUserDto dto){
55
56         HttpHeaders headers = new HttpHeaders();
57         headers.setContentType(MediaType.APPLICATION_JSON);
58
59         Map<String, Object> res = new HashMap<>();
60
61         User user = userRepository.findById(dto.getId()).orElse( other: null);
62
63         if(Optional.ofNullable(user).isPresent()){
64             user.setUsername(dto.getUsername().trim());
65             user.setPhone(dto.getPhone().trim());
66             user.setAddress(dto.getAddress().trim());
67         }else{
68             ErrorResponse errorResponse = new ErrorResponse( errorCode: "9999",  errorResponse: "data not found");
69             return ResponseEntity.status(HttpStatus.OK).headers(headers).body(errorResponse);
70         }
71
72         userRepository.save(user);
73         res.put("code", 200);
74         res.put("message", "success");
75         res.put("data", user);
76
77         return ResponseEntity.status(HttpStatus.CREATED).headers(headers).body(res);
78     }
79
80     @SneakyThrows(Exception.class)
81     @GetMapping
82     public ResponseEntity<Object> get(){
83
84         HttpHeaders headers = new HttpHeaders();
85         headers.setContentType(MediaType.APPLICATION_JSON);
86
87         ArrayList<User> user = new ArrayList<>(userRepository.findAll());
88
89         if (user == null){
90             ErrorResponse errorResponse = new ErrorResponse( errorCode: "9999",  errorResponse: "data not found");
91             return ResponseEntity.status(HttpStatus.OK).headers(headers).body(errorResponse);
92         }else {
93             Map<String, Object> res = new HashMap<>();
94             res.put("code", 200);
95             res.put("message", "success");
96             res.put("data", user);
97
98             return ResponseEntity.status(HttpStatus.OK).headers(headers).body(res);
99         }
100     }
101
102     @SneakyThrows(Exception.class)
103     @DeleteMapping
104     public ResponseEntity<Object> delete(DeleteUserDto dto){
105
106         HttpHeaders headers = new HttpHeaders();
107         headers.setContentType(MediaType.APPLICATION_JSON);
108
109         Map<String, Object> res = new HashMap<>();
110
111         User user = userRepository.findById(dto.getId()).orElse( other: null);
112
113         if(Optional.ofNullable(user).isPresent()){
114             userRepository.deleteById(dto.getId());
115         }else{
116             ErrorResponse errorResponse = new ErrorResponse( errorCode: "9999",  errorResponse: "data not found");
117             return ResponseEntity.status(HttpStatus.OK).headers(headers).body(errorResponse);
118         }
119
120         res.put("code", 200);
121         res.put("message", "success deleted user");
122
123         return ResponseEntity.status(HttpStatus.OK).headers(headers).body(res);
124     }

```

## v. Controller

```
UserController.java
14 @RestController
15 @RequestMapping("/api/v1")
16 @Slf4j
17 public class UserController {
18
19     4 usages
20     @Autowired
21     private UserService userService;
22
23     @SneakyThrows(Exception.class)
24     @PostMapping(path = "/user", produces = MediaType.APPLICATION_JSON_VALUE)
25     public ResponseEntity<Object> create(@RequestBody CreateUserDto dto){
26         log.info("api/v1/user for POST is executed...");
27
28         return userService.create(dto);
29     }
30
31     @SneakyThrows(Exception.class)
32     @PutMapping(path = "/user", produces = MediaType.APPLICATION_JSON_VALUE)
33     public ResponseEntity<Object> update(@RequestBody UpdateUserDto dto){
34         log.info("api/v1/user for PUT is executed...");
35
36         return userService.update(dto);
37     }
38
39     @SneakyThrows
40     @GetMapping("/user")
41     public ResponseEntity<Object> get(){
42         log.info("api/v1/user for GET is executed...");
43
44         return userService.get();
45     }
46
47     @SneakyThrows
48     @DeleteMapping("/user")
49     public ResponseEntity<Object> delete(@RequestBody DeleteUserDto dto){
50         log.info("api/v1/user for DELETE is executed...");
51
52         return userService.delete(dto);
53     }
54 }
```

## b. Test API CRUD

### i. Create

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

- Request:** Method **POST**, URL **localhost:8080/api/v1/user**. The request body is a JSON object: 

```
{
  "username": "Octavia Pratama Putra",
  "phone": "085645890621",
  "address": "Malang-Ngawi"
}
```
- Response:** Status **201 Created**, time **883 ms**, size **300 B**. The response body is a JSON object: 

```
{
  "code": 201,
  "data": {
    "id": 3,
    "username": "Octavia Pratama Putra",
    "phone": "085645890621",
    "address": "Malang-Ngawi"
  },
  "message": "success"
}
```

## ii. Read

SpringBoot API / User / **Get All User**

GET localhost:8080/api/v1/user **Send**

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

| KEY      | VALUE       | DESCRIPTION |
|----------|-------------|-------------|
| foodName | Nasi Goreng |             |

Body Cookies Headers (5) Test Results 200 OK 42 ms 469 B **Save Response**

Pretty Raw Preview Visualize JSON

```
1  {
2    "code": 200,
3    "data": [
4      {
5        "id": 1,
6        "username": "Yesi Inka",
7        "phone": "085645890999",
8        "address": "Malang Regency Klayatan"
9      },
10     {
11       "id": 2,
12       "username": "Andika OPP",
13       "phone": "085645890621",
14       "address": "Malang-Ngawi"
15     },
16     {
17       "id": 3,
18       "username": "Octavia Pratama Putra",
19       "phone": "085645890621",
20       "address": "Malang-Ngawi"
21     }
22   ],
23   "message": "success"
}
```

## iii. Update

PUT Edit User By Id

SpringBoot API / User / **Edit User By Id**

PUT localhost:8080/api/v1/user **Send**

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

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

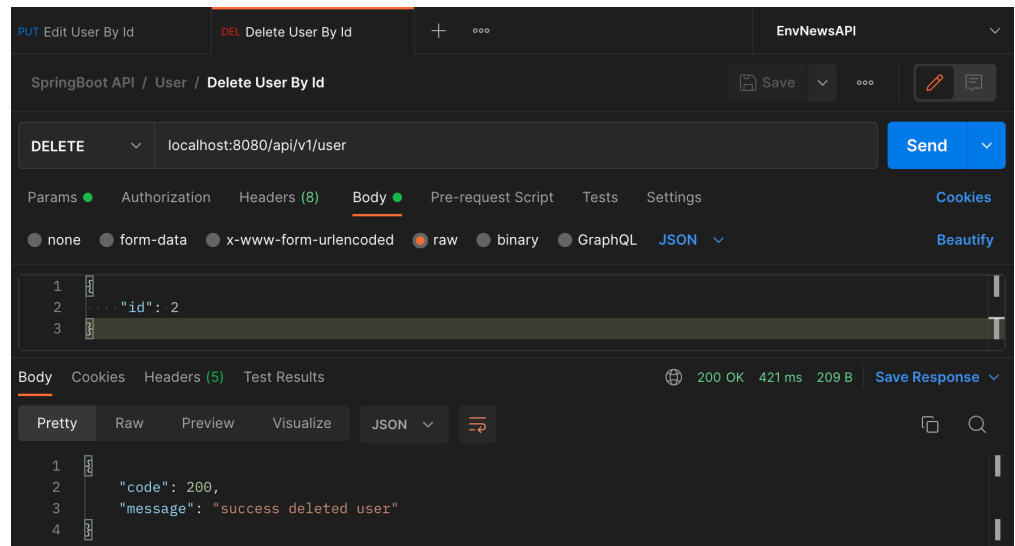
```
1  {
2    "id": 3,
3    "username": "Octavia Baru",
4    "phone": "085645890111",
5    "address": "Malang Regency Timur"
6  }
```

Body Cookies Headers (5) Test Results 201 Created 186 ms 299 B **Save Response**

Pretty Raw Preview Visualize JSON

```
1  {
2    "code": 200,
3    "data": {
4      "id": 3,
5      "username": "Octavia Baru",
6      "phone": "085645890111",
7      "address": "Malang Regency Timur"
8    },
9    "message": "success"
10 }
```

#### iv. Delete



## 2. CRUD Data Product

### a. Create, Read, Update, Delete Code

#### i. Model

```
Product.java
1 package com.ecommerce.ecommerce.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Getter;
5 import lombok.NoArgsConstructor;
6 import lombok.Setter;
7
8 import javax.persistence.*;
9 import java.io.Serializable;
10
11 @Entity(name = "Product")
12 @Table(name = "products")
13 @Setter
14 @Getter
15 @AllArgsConstructor
16 @NoArgsConstructor
17 public class Product implements Serializable {
18
19     private static final long serialVersionUID = -449241275888190961L;
20
21     @Id
22     @GeneratedValue(strategy = GenerationType.IDENTITY)
23     @Column(name = "id", nullable = false, insertable = false, unique = true)
24     private Long id;
25
26     @Column(name = "name", nullable = false, length = 255)
27     private String name;
28
29     @Column(name = "price", nullable = false)
30     private Long price;
31
32     @Column(name = "qty", nullable = false)
33     private Integer qty;
34 }
```

## ii. Repository

```
ProductRepository.java x
1 package com.ecommerce.ecommerce.repository;
2
3 import com.ecommerce.ecommerce.model.Product;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 2 usages
8 @Repository
9 public interface ProductRepository extends JpaRepository<Product, Long> {
10 }
```

## iii. DTO

```
CreateProductDto.java x
1 package com.ecommerce.ecommerce.dto;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 4 usages
8 @Data
9 @AllArgsConstructor
10 @NoArgsConstructor
11 public class CreateProductDto {
12     private String name;
13     private Long price;
14     private Integer qty;
15 }
```

```
UpdateProductDto.java x
1 package com.ecommerce.ecommerce.dto;
2
3 import ...
4
5 4 usages
6 @Data
7 @AllArgsConstructor
8 @NoArgsConstructor
9 public class UpdateProductDto {
10     private Long id;
11     private String name;
12     private Long price;
13     private Integer qty;
14 }
```

```
UpdateProductDto.java x DeleteProductDto.java x
1 package com.ecommerce.ecommerce.dto;
2
3 import ...
4
5 4 usages
6 @Data
7 @AllArgsConstructor
8 @NoArgsConstructor
9 public class DeleteProductDto {
10     private Long id;
11 }
```

#### iv. Service

```
ProductService.java
26 @Service
27 @Transactional(rollbackOn = Exception.class)
28 public class ProductService {
29
30     5 usages
31     @Autowired
32     private ProductRepository productRepository;
33
34     1 usage
35     @SneakyThrows
36     public ResponseEntity<Object> create(CreateProductDto dto){
37
38         HttpHeaders headers = new HttpHeaders();
39         headers.setContentType(MediaType.APPLICATION_JSON);
40
41         Product product = new Product();
42         product.setName(dto.getName());
43         product.setPrice(dto.getPrice());
44         product.setQty(dto.getQty());
45         productRepository.save(product);
46
47         Map<String, Object> res = new HashMap<>();
48         res.put("code", 201);
49         res.put("message", "success");
50         res.put("data", product);
51
52         return ResponseEntity.status(HttpStatus.OK).headers(headers).body(res);
53     }
54
55     1 usage
56     @SneakyThrows
57     public ResponseEntity<Object> update(UpdateProductDto dto){
58
59         HttpHeaders headers = new HttpHeaders();
60         headers.setContentType(MediaType.APPLICATION_JSON);
61
62         Product product = productRepository.findById(dto.getId()).orElse( other: null);
63         if (Optional.ofNullable(product).isPresent()){
64             product.setName(dto.getName().trim());
65             product.setPrice(dto.getPrice());
66             product.setQty(dto.getQty());
67         }else {
68             ErrorResponse errorResponse = new ErrorResponse( errorCode: "9999", errorResponse: "data not found");
69         }
70
71         Map<String, Object> res = new HashMap<>();
72         res.put("code", 200);
73         res.put("message", "success");
74         res.put("data", product);
75
76         return ResponseEntity.status(HttpStatus.CREATED).headers(headers).body(res);
77     }
78
79     1 usage
80     @SneakyThrows
81     public ResponseEntity<Object> get(){
82
83         HttpHeaders headers = new HttpHeaders();
84         headers.setContentType(MediaType.APPLICATION_JSON);
85
86         ArrayList<Product> products = new ArrayList<>(productRepository.findAll());
87
88         if (products == null){
89             ErrorResponse errorResponse = new ErrorResponse( errorCode: "9999", errorResponse: "data not found");
90             return ResponseEntity.status(HttpStatus.OK).headers(headers).body(errorResponse);
91         }else {
92             Map<String, Object> res = new HashMap<>();
93             res.put("code", 200);
94             res.put("message", "success");
95             res.put("data", products);
96
97             return ResponseEntity.status(HttpStatus.OK).headers(headers).body(res);
98         }
99     }
100 }
```



```

1 usage
@SneakyThrows(Exception.class)
95 public ResponseEntity<Object> delete(DeleteProductDto dto){
96
97     HttpHeaders headers = new HttpHeaders();
98     headers.setContentType(MediaType.APPLICATION_JSON);
99
100     Map<String, Object> res = new HashMap<>();
101
102     Product product = productRepository.findById(dto.getId()).orElse( other: null);
103
104     if(Optional.ofNullable(product).isPresent()){
105         productRepository.deleteById(dto.getId());
106     }else{
107         ErrorResponse errorResponse = new ErrorResponse( errorCode: "9999", errorMessage: "data not found");
108         return ResponseEntity.status(HttpStatus.OK).headers(headers).body(errorResponse);
109     }
110
111     res.put("code", 200);
112     res.put("message", "success deleted product");
113
114     return ResponseEntity.status(HttpStatus.OK).headers(headers).body(res);
115 }
116
117 }

```

## v. Controller

```

ProductController.java
14 @RestController
15 @RequestMapping("${api/v1}")
16 @Slf4j
17 public class ProductController {
18
19     4 usages
20     @Autowired
21     private ProductService productService;
22
23     @SneakyThrows
24     @PostMapping(path = "${"/product}", produces = MediaType.APPLICATION_JSON_VALUE)
25     public ResponseEntity<Object> create(@RequestBody CreateProductDto dto){
26         log.info("api/v1/product for POST is executed....");
27
28         return productService.create(dto);
29     }
30
31     @SneakyThrows
32     @PutMapping(path = "${"/product}", produces = MediaType.APPLICATION_JSON_VALUE)
33     public ResponseEntity<Object> update(@RequestBody UpdateProductDto dto){
34         log.info("api/v1/product for PUT is executed....");
35
36         return productService.update(dto);
37     }
38
39     @SneakyThrows
40     @GetMapping(path = "${"/product}")
41     public ResponseEntity<Object> get(){
42         log.info("api/v1/product for GET is executed....");
43
44         return productService.get();
45     }
46
47     @SneakyThrows
48     @DeleteMapping(path = "${"/product}", produces = MediaType.APPLICATION_JSON_VALUE)
49     public ResponseEntity<Object> delete(@RequestBody DeleteProductDto dto){
50         log.info("api/v1/product for DELETE is executed....");
51
52         return productService.delete(dto);
53     }
54 }

```

b. Test API CRUD  
i. Create

The screenshot shows the Postman interface for a POST request named "Add Product" to the endpoint `localhost:8080/api/v1/product`. The request body is a JSON object: `{ "name": "Samsung Galaxy 21", "price": "12000000", "qty": "9" }`. The response status is 200 OK, and the response body is a JSON object: `{ "code": 201, "data": { "id": 4, "name": "Samsung Galaxy 21", "price": 12000000, "qty": 9 }, "message": "success" }`.

```
1 {
2   "name": "Samsung Galaxy 21",
3   "price": "12000000",
4   "qty": "9"
5 }
```

```
1 {
2   "code": 201,
3   "data": {
4     "id": 4,
5     "name": "Samsung Galaxy 21",
6     "price": 12000000,
7     "qty": 9
8   },
9   "message": "success"
10 }
```

ii. Read

The screenshot shows the Postman interface for a GET request named "Get All Product" to the endpoint `localhost:8080/api/v1/product`. The response status is 200 OK, and the response body is a JSON object: `{ "code": 200, "data": [ { "id": 1, "name": "Mac Book Pro 15", "price": 30000000, "qty": 12 }, { "id": 2, "name": "Iphone 13 Pro", "price": 28000000, "qty": 15 }, { "id": 4, "name": "Samsung Galaxy 21", "price": 12000000, "qty": 9 } ], "message": "success" }`.

```
1 {
2   "code": 200,
3   "data": [
4     {
5       "id": 1,
6       "name": "Mac Book Pro 15",
7       "price": 30000000,
8       "qty": 12
9     },
10    {
11      "id": 2,
12      "name": "Iphone 13 Pro",
13      "price": 28000000,
14      "qty": 15
15    },
16    {
17      "id": 4,
18      "name": "Samsung Galaxy 21",
19      "price": 12000000,
20      "qty": 9
21    }
22  ],
23  "message": "success"
24 }
```

### iii. Update

The screenshot shows a REST client interface for a SpringBoot API. The request is a PUT to `localhost:8080/api/v1/product`. The body is a JSON object representing a product update.

```
1 {
2   "id": 4,
3   "name": "Samsung Galaxy Ultra",
4   "price": "10000000",
5   "qty": "10"
6 }
```

The response is shown in the bottom panel, indicating a successful update with a 200 status code.

```
1 {
2   "code": 200,
3   "data": {
4     "id": 4,
5     "name": "Samsung Galaxy Ultra",
6     "price": 10000000,
7     "qty": 10
8   },
9   "message": "success"
10 }
```

### iv. Delete

The screenshot shows a REST client interface for a SpringBoot API. The request is a DELETE to `localhost:8080/api/v1/product`. The body is a JSON object containing the product ID to be deleted.

```
1 {
2   "id": 4
3 }
```

The response is shown in the bottom panel, indicating a successful deletion with a 200 status code.

```
1 {
2   "code": 200,
3   "message": "success deleted product"
4 }
```