



Spring Boot CRUD Application: A Deep Dive

This presentation explores a Spring Boot CRUD app using PostgreSQL and JPA. The application demonstrates Create, Read, Update, Delete operations with REST APIs. Code is hosted on GitHub for full access.

 by girish kudachi

Project Setup and Dependencies

Development Environment

Using IntelliJ IDEA for efficient Java and Spring Boot coding.

Core Dependencies

- spring-boot-starter-data-jpa for JPA integration
- postgresql driver for database connection
- lombok for automatic code generation
- spring-boot-starter-web for REST API functions
- spring-boot-devtools for hot reload

```
ryController.java  pom.xml (journalApp)  JournalEntryRepository.java
object xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3
<dependencies>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
  </dependency>

  <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.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.postgresql</groupId>
```

entryController.java pom.xml (JournalApp) application.properties

Root configuration files are supported by IntelliJ IDEA Ultimate

```
spring.datasource.url=jdbc:postgresql://localhost:5433/Journal_App
spring.datasource.username=postgres
spring.datasource.password=Admin09
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Database Configuration with PostgreSQL



Database Setup

Configured PostgreSQL locally and optionally cloud-based instances.



Key Configuration

Datasource URL, username, password set in properties file.



Schema Management

Used `spring.jpa.hibernate.ddl-auto` for automatic schema updates.

Entity Definition with JPA

Entity Setup

Defined entities with `@Entity` and mapped to database tables.

Example: User entity involves id, name, email fields.

Annotations

- `@Id` and `@GeneratedValue` for primary keys
- `@Column` for column mappings
- Lombok `@Data` and constructors for boilerplate code

Repository Layer with Spring Data JPA



Repository Interface

Extended JpaRepository to enable CRUD without extra code.



Built-in Methods

- save(), findById(), findAll(), deleteById()



Custom Queries

Derived query methods by method naming like findByEmail().

```
application.properties  JournalEntryRepository.java  JournalAp
t.journal.journalApp.repository;

journal.journalApp.entity.journalEntry;
springframework.data.jpa.repository.JpaRepository;

interface JournalEntryRepository extends JpaRepository<journalEntry, Long>
```

Service Layer Implementation

Business Logic

Services handle data manipulation and business rules.

Example methods: createUser, getUserById, updateUser, deleteUser.

Transaction Management

@Transactional ensures data integrity during operations.

```
package net.journal.journalApp.controller;

import net.journal.journalApp.entity.journalEntry;
import net.journal.journalApp.service.JournalEntryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDateTime;
import java.util.*;

@RestController
@RequestMapping("/journal")
public class JournalEntryController
{
    @Autowired
    private JournalEntryService JournalEntryService;

    @GetMapping
    public List<journalEntry> getAll()
    {
        return JournalEntryService.getAll();
    }

    @PostMapping
    public boolean createEntry(@RequestBody journalEntry myEntry)
    {
        JournalEntryService.saveEntry(myEntry);
        return true;
    }

    @GetMapping("id/{myId}")
    public journalEntry getJournalEntryById(@PathVariable Long myId)
    {

```

REST API Controller Design

1

Controller Annotations

@RestController and @RequestMapping organize API endpoints.

2

HTTP Methods

- @PostMapping for create
- @GetMapping for read
- @PutMapping for update
- @DeleteMapping for delete

3

Parameter Binding

@PathVariable and @RequestBody simplify data handling.

```
Oll Vih Vis Q Postman
Histnasteprrify postmans x +
11 RESTAPI request: {
11 Postman Carpte: leburatorred ragdied)
22 Papfiet: 0 (uuces 120:00(011.1512)012.1684
33 raupetite Feareag tol(
46 sourt (ascer(PEEPRE(20T,74COL5)
34 sort: (.2018)
25 zuustion (1.221)
66 censuset:(Sooe.=.2000)
47 consprolesdert 30 Bech postapt (facar) cill,we)
68 { =- ion resquncer
64 suporte ful {
45 connectte(43 (dpirt na.rocel)
40 rouncate (11C
89 rouncate (11
49 fanprancte: Sa45,145([16,457].011,468.15-466))
60 frair cour (Jase")
87 ittles: I
88 fillers)
66 >>
95 { rhp "1.cdpuce"
19 varrecl "5.202328,752001.com")
17 kurrit 85:10 {
19 cequest: "1.9615 {
19 Daupie: 0 9,1115
19 racuige: rocar: bastem: Ctod27434.18,245)
16 condbestion: =1,7086
17 coort culletent: 0 (1822,,4600")
16 }
27 souueste: 9.2291)
14 saunesst: 9.2125
11 Soucat: (110)
12 Soucce: "1112)
23 spinner: Baupteripel/ 3,253")
14 Soucie: (809)
22 Soucee: (340")
23 rocess: 4.2751)
22 cAudanberccs.=.12288
38 superlines (/hpkc/) adiresteptonf 0472-0[4.30749011.488328)
17 saungest: =1,2.8)
28 Spcat: (8801)
14 Spcat: (a80")
24 Spnations: 27)
20 sppert Carpte Wil Contrerition UrC FactConneyor(Presalted
20 Spour spate: Irevateinn" ltes day (he.ton)
21 contastel vilu6 {
27 Succet Cantur-lte
27 rannest: Daskevovs/Capplen ([E1T81.15)
19 Spcast (ab0")
26 tapuie: =.2577)
```

Testing the APIs with Postman

1

Create User

Sent POST request with JSON payload to /api/users.

2

Retrieve User

Tested GET endpoint using user ID parameter.

3

Update and Delete

Confirmed PUT and DELETE operations via API calls.

Code Structure and Best Practices

Project Layout

- src/main/java structured by layers
- resources hold config files

Standards

Consistent naming and error handling improve maintainability.

Conclusion and Future Enhancements

Summary

CRUD app leverages Spring Boot with PostgreSQL efficiently.

Next Steps

- Add pagination and sorting support
- Implement request validation
- Enable authentication and authorization
- Deploy to cloud platforms like AWS or Heroku

Discussion

Open floor for questions and future collaboration ideas.

