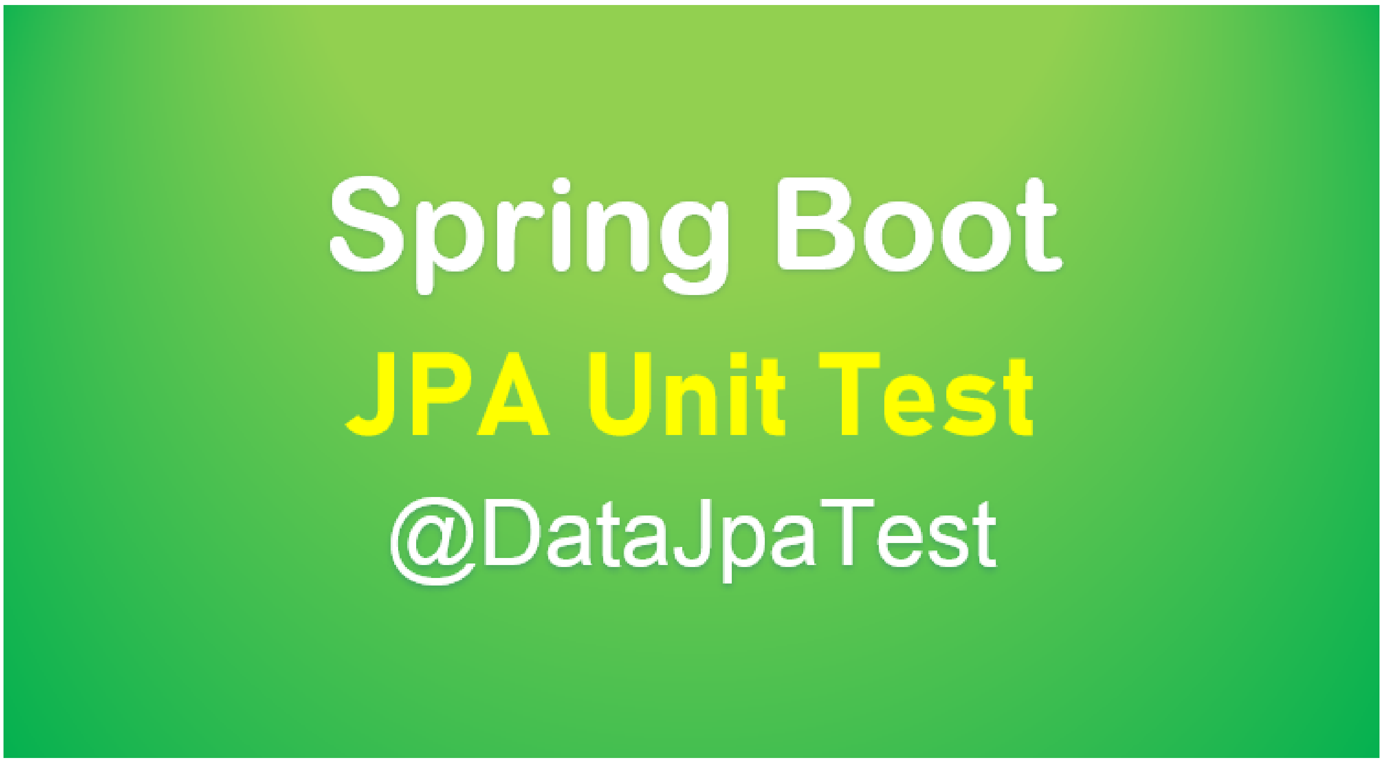


@DataJpaTest example for Spring Data Repositiory Unit Test

Last modified: August 3, 2020 (https://bezkoder.com/spring-boot-unit-test-jpa-repo-datajpatest/) bezkoder (https://bezkoder.com/author/bezkoder/) Spring (https://bezkoder.com/category/spring/), Testing (https://bezkoder.com/category/testing/)

Nowadays Unit Test is so important in Software Development, and Spring Framework also provides `@DataJpaTest` annotation to make writing test for JPA Repository more simpler. In this tutorial, we're gonna look at how to apply `@DataJpaTest` in our Spring Boot Project with `TestEntityManager` .

This tutorial gives you an additional unit test for the Post:
– Spring Boot, Spring Data JPA – Rest CRUD API example (https://bezkoder.com/spring-boot-jpa-crud-rest-api/)

More Practice:
– Spring Boot Token based Authentication with Spring Security example (https://bezkoder.com/spring-boot-jwt-authentication/)
– Spring Boot @ControllerAdvice & @ExceptionHandler example (https://bezkoder.com/spring-boot-controlleradvice-exceptionhandler/)

Contents [hide]

Spring Boot @DataJpaTest example Overview

We have *Tutorial* model with some fields: id, title, description, published.
There is a repository to interact with Tutorials from the database called `TutorialRepository` interface that extends `JpaRepository` :

```
public interface TutorialRepository extends JpaRepository<Tutorial, Long> {
  List<Tutorial> findByPublished(boolean published);
  List<Tutorial> findByTitleContaining(String title);
}
```

`JpaRepository` also supports following methods: `save()` , `findOne()` , `findById()` , `findAll()` , `count()` , `delete()` , `deleteById()` .

So how to write unit test for all of them?
=> We're gonna use `@DataJpaTest` with `TestEntityManager` .

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class JPAUnitTest {

  @Autowired
  private TestEntityManager entityManager;

  @Autowired
  TutorialRepository repository;

  @Test
  public void should_find_no_tutorials_if_repository_is_empty() { }

  @Test
  public void should_store_a_tutorial() { }

  @Test
  public void should_find_all_tutorials() { }

  @Test
  public void should_find_tutorial_by_id() { }

  @Test
  public void should_find_published_tutorials() { }

  @Test
  public void should_find_tutorials_by_title_containing_string() { }

  @Test
  public void should_update_tutorial_by_id() { }

  @Test
  public void should_delete_tutorial_by_id() { }

  @Test
  public void should_delete_all_tutorials() { }
}
```

For testing, we'll work with **H2** in-memory database. It eliminates the need for configuring and starting an actual database.

@DataJpaTest annotation for testing JPA Repository

`@DataJpaTest` is the annotation that Spring supports for a JPA test that focuses only on JPA components.

It will disable full auto-configuration and then, apply only enable configuration relevant to JPA tests. The list of the auto-configuration settings that are enabled can be found here (https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-test-auto-configuration.html#test-auto-configuration).

By default, tests annotated with `@DataJpaTest` are transactional and roll back at the end of each test. If you don't want it, you can disable transaction management for a test or for the whole class using `@Transactional` annotation:

```
@DataJpaTest
@Transactional(propagation = Propagation.NOT_SUPPORTED)
class YourNonTransactionalTests {

}
```

In-memory embedded database (like H2 database in this example) generally works well for tests, it is fast and does not require any installation. However, we can configure for a real database with `@AutoConfigureTestDatabase` annotation:

```
@DataJpaTest
@AutoConfigureTestDatabase(replace=Replace.NONE)
class YourRepositoryTests {

}
```

If you are using JUnit 4, you need to add `@RunWith(SpringRunner.class)` to the test:

```
@RunWith(SpringRunner.class)
@DataJpaTest
class YourRepositoryTests {

}
```

TestEntityManager

The purpose of the EntityManager is to interact with the persistence context. Spring Data JPA abstractes you from the EntityManager through Repository interfaces. And `TestEntityManager` allows us to use EntityManager in tests.

We can inject a `TestEntityManager` bean in Data JPA tests. If you want to use `TestEntityManager` outside of `@DataJpaTest` instances, just add `@AutoConfigureTestEntityManager` annotation.

The following example shows the `@DataJpaTest` annotation with `TestEntityManager` :

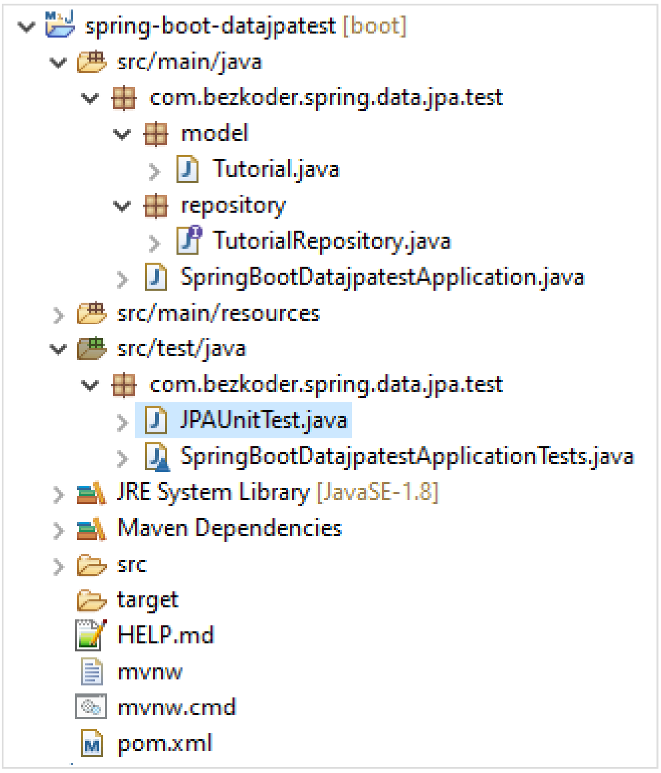
```
@DataJpaTest
class YourRepositoryTests {

  @Autowired
  private TestEntityManager entityManager;

  @Test
  void testExample() throws Exception {
    this.entityManager.persist(new Tutorial("Tut#1", "Desc#1", true));
    ...
  }
}
```

Project Structure

Let’s look at our Spring Boot project:



```
spring-boot-datajpatest [boot]
  src/main/java
    com.bezkoder.spring.data.jpa.test
      model
        Tutorial.java
      repository
        TutorialRepository.java
      SpringBootDatajpatestApplication.java
  src/main/resources
  src/test/java
    com.bezkoder.spring.data.jpa.test
      JPAUnitTest.java
      SpringBootDatajpatestApplicationTests.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

– `Tutorial` data model class corresponds to entity and table tutorials.

– `TutorialRepository` is an interface that extends `JpaRepository` for CRUD methods and custom finder methods. It will be autowired in `JPAUnitTest` .

– `JPAUnitTest` is the main Test Class used for testing JPA and annotated with `@DataJpaTest` .

– pom.xml contains dependencies for Spring Boot, JPA, H2 database.

Setup Spring Boot @DataJpaTest Project

Use Spring web tool (http://start.spring.io/) or your development tool (Spring Tool Suite (https://spring.io/tools), Eclipse, Intellij (https://www.jetbrains.com/idea/download/)) to create a Spring Boot project.

Then open **pom.xml** and add these 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-test</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>
```

Define Data Model

In **model** package, we define `Tutorial` class.
Our Data model (entity) contains 4 fields: id, title, description, published.

model/Tutorial.java

```
package com.bezkoder.spring.data.jpa.test.model;

import javax.persistence.*;

@Entity
@Table(name = "tutorials")
public class Tutorial {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "title")
    private String title;

    @Column(name = "description")
    private String description;

    @Column(name = "published")
    private boolean published;

    public Tutorial() {

    }

    public Tutorial(String title, String description, boolean published) {
        this.title = title;
        this.description = description;
        this.published = published;
    }

    public long getId() {
        return id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean isPublished() {
        return published;
    }

    public void setPublished(boolean isPublished) {
        this.published = isPublished;
    }

    @Override
    public String toString() {
        return "Tutorial [id=" + id + ", title=" + title + ", desc=" + description + ", p
    }
}
```

– `@Entity` annotation indicates that the class is a persistent Java class.
– `@Table` annotation provides the table that maps this entity.
– `@Id` annotation is for the primary key.
– `@GeneratedValue` annotation is used to define generation strategy for the primary key. `GenerationType.AUTO` means Auto Increment field.
– `@Column` annotation is used to define the column in database that maps annotated field.

Create JPA Repository

In **repository** package, create `TutorialRepository` interface that extends `JpaRepository` . This repository will interact with Tutorials from the database.

repository/TutorialRepository.java

```
package com.bezkoder.spring.data.jpa.test.repository;

import java.util.List;

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

import com.bezkoder.spring.data.jpa.test.model.Tutorial;

public interface TutorialRepository extends JpaRepository<Tutorial, Long> {
  List<Tutorial> findByPublished(boolean published);

  List<Tutorial> findByTitleContaining(String title);
}
```

Now we can use JpaRepository's methods: `save()` , `findOne()` , `findById()` , `findAll()` , `count()` , `delete()` , `deleteById()` … without implementing these methods.

We also define custom finder methods:
– `findByPublished()` : returns all Tutorials with `published` having value as input `published` .
– `findByTitleContaining()` : returns all Tutorials which title contains input `title` .

The implementation is plugged in by Spring Data JPA (https://docs.spring.io/spring-data/jpa/docs/current/reference/html/) automatically.

You can also modify this Repository to work with Pagination, the instruction can be found at:
Spring Boot Pagination & Filter example | Spring JPA, Pageable (https://bezkoder.com/spring-boot-pagination-filter-jpa-pageable/)

Write Unit Test With @DataJpaTest

Under **src**/**test**/**java**, create a class named `JPAUnitTest` that extends. We're gonna test many cases (CRUD operations, finder methods) inside this class.

```
package com.bezkoder.spring.data.jpa.test;

import static org.assertj.core.api.Assertions.assertThat;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.boot.test.autoconfigure.orm.jpa.TestEntityManager;
import org.springframework.test.context.junit4.SpringRunner;

import com.bezkoder.spring.data.jpa.test.model.Tutorial;
import com.bezkoder.spring.data.jpa.test.repository.TutorialRepository;

@RunWith(SpringRunner.class)
@DataJpaTest
public class JPAUnitTest {

  @Autowired
  private TestEntityManager entityManager;

  @Autowired
  TutorialRepository repository;

  @Test
  public void should_find_no_tutorials_if_repository_is_empty() {
    Iterable<Tutorial> tutorials = repository.findAll();

    assertThat(tutorials).isEmpty();
  }

  @Test
  public void should_store_a_tutorial() {
    Tutorial tutorial = repository.save(new Tutorial("Tut title", "Tut desc", true));

    assertThat(tutorial).hasFieldOrPropertyWithValue("title", "Tut title");
    assertThat(tutorial).hasFieldOrPropertyWithValue("description", "Tut desc");
    assertThat(tutorial).hasFieldOrPropertyWithValue("published", true);
  }

  @Test
  public void should_find_all_tutorials() {
    Tutorial tut1 = new Tutorial("Tut#1", "Desc#1", true);
    entityManager.persist(tut1);

    Tutorial tut2 = new Tutorial("Tut#2", "Desc#2", false);
    entityManager.persist(tut2);

    Tutorial tut3 = new Tutorial("Tut#3", "Desc#3", true);
    entityManager.persist(tut3);

    Iterable<Tutorial> tutorials = repository.findAll();

    assertThat(tutorials).hasSize(3).contains(tut1, tut2, tut3);
  }

  @Test
  public void should_find_tutorial_by_id() {
    Tutorial tut1 = new Tutorial("Tut#1", "Desc#1", true);
    entityManager.persist(tut1);

    Tutorial tut2 = new Tutorial("Tut#2", "Desc#2", false);
    entityManager.persist(tut2);

    Tutorial foundTutorial = repository.findById(tut2.getId()).get();

    assertThat(foundTutorial).isEqualTo(tut2);
  }

  @Test
  public void should_find_published_tutorials() {
    Tutorial tut1 = new Tutorial("Tut#1", "Desc#1", true);
    entityManager.persist(tut1);

    Tutorial tut2 = new Tutorial("Tut#2", "Desc#2", false);
    entityManager.persist(tut2);

    Tutorial tut3 = new Tutorial("Tut#3", "Desc#3", true);
    entityManager.persist(tut3);

    Iterable<Tutorial> tutorials = repository.findByPublished(true);

    assertThat(tutorials).hasSize(2).contains(tut1, tut3);
  }

  @Test
  public void should_find_tutorials_by_title_containing_string() {
    Tutorial tut1 = new Tutorial("Spring Boot Tut#1", "Desc#1", true);
    entityManager.persist(tut1);

    Tutorial tut2 = new Tutorial("Java Tut#2", "Desc#2", false);
    entityManager.persist(tut2);

    Tutorial tut3 = new Tutorial("Spring Data JPA Tut#3", "Desc#3", true);
    entityManager.persist(tut3);

    Iterable<Tutorial> tutorials = repository.findByTitleContaining("ring");
    assertThat(tutorials).hasSize(2).contains(tut1, tut3);
  }

  @Test
  public void should_update_tutorial_by_id() {
    Tutorial tut1 = new Tutorial("Tut#1", "Desc#1", true);
    entityManager.persist(tut1);

    Tutorial tut2 = new Tutorial("Tut#2", "Desc#2", false);
    entityManager.persist(tut2);

    Tutorial updatedTut = new Tutorial("updated Tut#2", "updated Desc#2", true);

    Tutorial tut = repository.findById(tut2.getId()).get();
    tut.setTitle(updatedTut.getTitle());
    tut.setDescription(updatedTut.getDescription());
    tut.setPublished(updatedTut.isPublished());
    repository.save(tut);

    Tutorial checkTut = repository.findById(tut2.getId()).get();

    assertThat(checkTut.getId()).isEqualTo(tut2.getId());
    assertThat(checkTut.getTitle()).isEqualTo(updatedTut.getTitle());
    assertThat(checkTut.getDescription()).isEqualTo(updatedTut.getDescription());
    assertThat(checkTut.isPublished()).isEqualTo(updatedTut.isPublished());
  }

  @Test
  public void should_delete_tutorial_by_id() {
    Tutorial tut1 = new Tutorial("Tut#1", "Desc#1", true);
    entityManager.persist(tut1);

    Tutorial tut2 = new Tutorial("Tut#2", "Desc#2", false);
    entityManager.persist(tut2);

    Tutorial tut3 = new Tutorial("Tut#3", "Desc#3", true);
    entityManager.persist(tut3);

    repository.deleteById(tut2.getId());

    Iterable<Tutorial> tutorials = repository.findAll();

    assertThat(tutorials).hasSize(2).contains(tut1, tut3);
  }

  @Test
  public void should_delete_all_tutorials() {
    entityManager.persist(new Tutorial("Tut#1", "Desc#1", true));
    entityManager.persist(new Tutorial("Tut#2", "Desc#2", false));
```

```
        repository.deleteAll();

        assertThat(repository.findAll()).isEmpty();
      }
    }
```

Run Unit Test

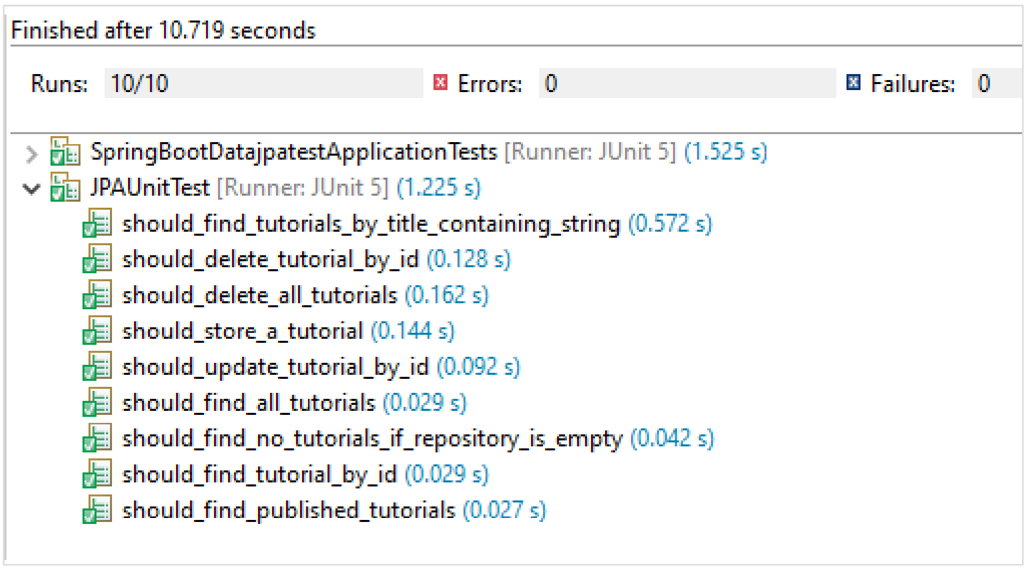
– First run command: `mvn clean install`.

– Then run Test: `mvn test`

```
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 7.901 s - in com.b
ing.data.jpa.test.JPAUnitTest
...
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  32.594 s
[INFO] Finished at: 2020-05-07T20:48:47+07:00
[INFO] ------------------------------------------------------------------------
```

– Or you can also run Spring Boot project with mode **JUnit Test**.

– Now see the Junit result as following:



```
Finished after 10.719 seconds
Runs: 10/10    Errors: 0    Failures: 0

> SpringBootDatajpatestApplicationTests [Runner: JUnit 5] (1.525 s)
v JPAUnitTest [Runner: JUnit 5] (1.225 s)
    should_find_tutorials_by_title_containing_string (0.572 s)
    should_delete_tutorial_by_id (0.128 s)
    should_delete_all_tutorials (0.162 s)
    should_store_a_tutorial (0.144 s)
    should_update_tutorial_by_id (0.092 s)
    should_find_all_tutorials (0.029 s)
    should_find_no_tutorials_if_repository_is_empty (0.042 s)
    should_find_tutorial_by_id (0.029 s)
    should_find_published_tutorials (0.027 s)
```

Conclusion

Today we've create Spring Boot Test for JPA Repository with H2 database using `@DataJPATest` and `TestEntityManager` with H2 Database. We also run unit test for many CRUD operations and custom finder methods.

You may need to handle Exception with:
Spring Boot @ControllerAdvice & @ExceptionHandler example (https://bezkoder.com/spring-boot-controlleradvice-exceptionhandler/)

Happy learning! See you again.

Further Reading

- @DataJpaTest (https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/test/autoconfigure/orm/jpa/DataJpaTest.html)
- Auto-configured Data JPA Tests (https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-testing-spring-boot-applications-testing-autoconfigured-jpa-test)

Source Code

You can find the complete source code for this tutorial on Github (https://github.com/bezkoder/spring-boot-datajpatest).

datajpatest (https://bezkoder.com/tag/datajpatest/) h2 database (https://bezkoder.com/tag/h2-database/)
junit (https://bezkoder.com/tag/junit/) spring boot (https://bezkoder.com/tag/spring-boot/)
spring data jpa (https://bezkoder.com/tag/spring-data-jpa/) unit test (https://bezkoder.com/tag/unit-test/)

4 thoughts to "@DataJpaTest example for Spring Data Repositiory Unit Test"

Francois Loriot
June 25, 2020 at 8:38 pm (https://bezkoder.com/spring-boot-unit-test-jpa-repo-datajpatest/#comment-3454)

Hi,
You are using @Autowired on fields, but I thought field injection was not recommended.

Any comments ?

REPLY

Oleksandr Pastukhov
September 13, 2020 at 5:03 pm (https://bezkoder.com/spring-boot-unit-test-jpa-repo-datajpatest/#comment-4698)

for tests it's ok

REPLY

Oleksandr Pastukhov
September 13, 2020 at 5:05 pm (https://bezkoder.com/spring-boot-unit-test-jpa-repo-datajpatest/#comment-4699)

Why do we have repository and a `TestEntityManager` in the test?
I see we use both to save the same entity, what the difference?

REPLY

bezkoder
September 14, 2020 at 2:35 am (https://bezkoder.com/spring-boot-unit-test-jpa-repo-datajpatest/#comment-4704)

Hi,

1. When you want to check Repository's `save()` method, you must use it.
2. When you want to check other methods of Repository (not including `save()`), you must have data in database table, and you are NOT sure that Repository's `save()` method is good or not, then you use `TestEntityManager` for persisting data.

REPLY

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

☐

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

❮ Dart/Flutter – Convert XML to JSON using xml2json (https://bezkoder.com/dart-flutter-xml-to-json-xml2json/)

Dart/Flutter Constructors tutorial with examples ❯ (https://bezkoder.com/dart-flutter-constructors/)

Search...

FOLLOW US

(https://facebook.com/bezkoder)

(https://www.youtube.com/channel/UCp0mx9RH0JxaFsmMvKXA86Q)

(https://github.com/bezkoder)

TOOLS

Json Formatter (https://bezkoder.com/json-formatter/)

DMCA PROTECTED (https://www.dmca.com/Protection/Status.aspx?ID=3f543dd5-c6d8-4208-9a6b-0e92057fd597&refurl=https://bezkoder.com/spring-boot-unit-test-jpa-repo-datajpatest/)