How to use Spring JPA with PostgreSQL | Spring Boot

In tradition approach, implementing Data Access Layer makes lots of boilerplate code. Spring JPA is a part of Spring Data, helps us improve our codes and reduce efforts for development and maintenance. Spring JPA supports us the ways to write interface for repositories and custom finder methods, the implementation will be done automatically by Spring Framework.

The tutorial shows you how to use Spring JPA with PostgreSQL using Spring Boot.

Related Posts:
– How to use Spring JPA MySQL | Spring Boot
– Spring JPA + PostgreSQL + AngularJS example | Spring Boot

– Angular 4 + Spring JPA + PostgreSQL example | Angular 4 Http Client – Spring Boot RestApi Server
– React Redux + Spring Boot + PostgreSQL CRUD example
– Spring Boot + Angular 6 example | Spring Data JPA + REST + PostgreSQL CRUD example

How to do this with **Kotlin**: Spring JPA + Postgresql | Spring Boot Example

Contents [hide]

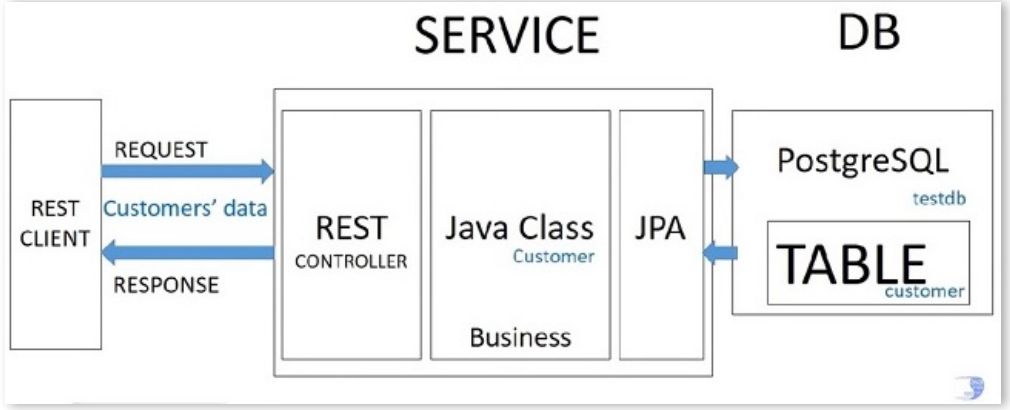
I. Technology
II. Overview
1. Goal
2. Project Structure
3. Step to do
4. Demo Video
III. Practice
1. Create Spring Boot project & add Dependencies
2. Configure Spring JPA
3. Create DataModel Class
4. Create Spring JPA Repository Interface
5. Create Web Controller
6. Using CommandLineRunner to clear old data
7. Create PostGreSQL table
8. Run Spring Boot Application & Enjoy Result
IV. Source Code

I. Technology

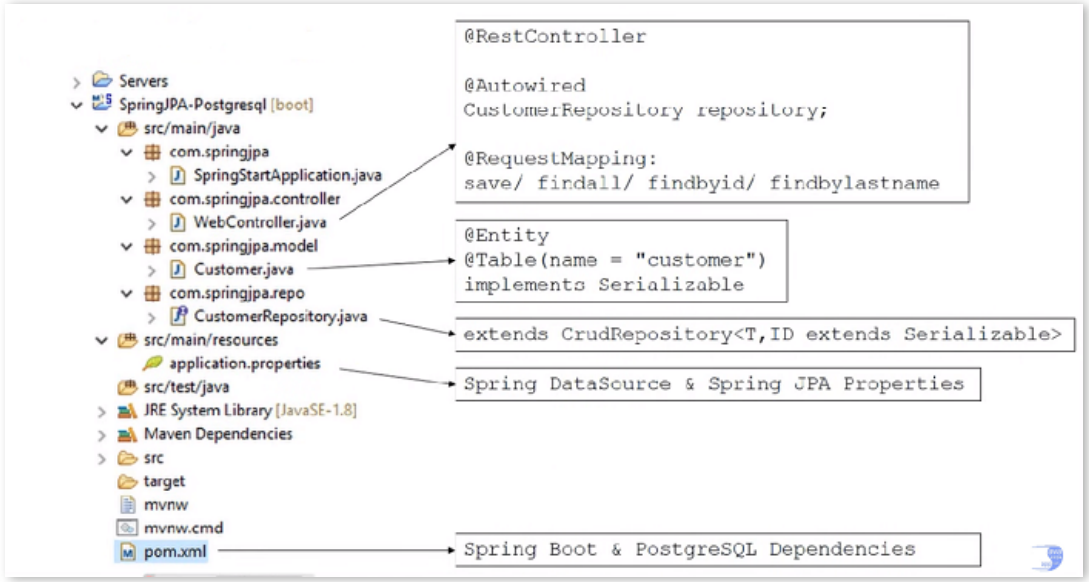
– Java 1.8
– Maven 3.3.9
– Spring Tool Suite – Version 3.8.1.RELEASE
– Spring Boot: RELEASE

II. Overview

1. Goal



2. Project Structure

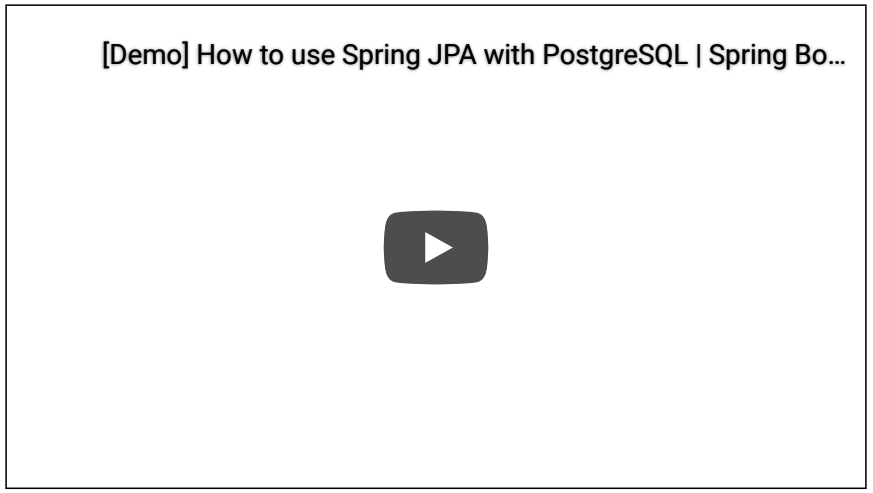


– Class **Customer** corresponds to entity and table **customer**, it should be implemented **Serializable**.
– **CustomerRepository** is an interface extends **CrudRepository**, will be autowired in **WebController** for implementing repository methods and custom finder methods.
– **WebController** is a REST Controller which has request mapping methods for RESTful requests such as: save. findall, findbyid, findbylastname.
– Configuration for Spring Datasource and Spring JPA properties in **application.properties**
– **Dependencies** for **Spring Boot** and **PostgreSQL** in **pom.xml**

3. Step to do

– Create Spring Boot project & add Dependencies
– Configure Spring JPA
– Create DataModel Class
– Create Spring JPA Repository Interface
– Create Web Controller
– Using CommandLineRunner to clear old data
– Create PostGreSQL table
– Run Spring Boot Application & Enjoy Result

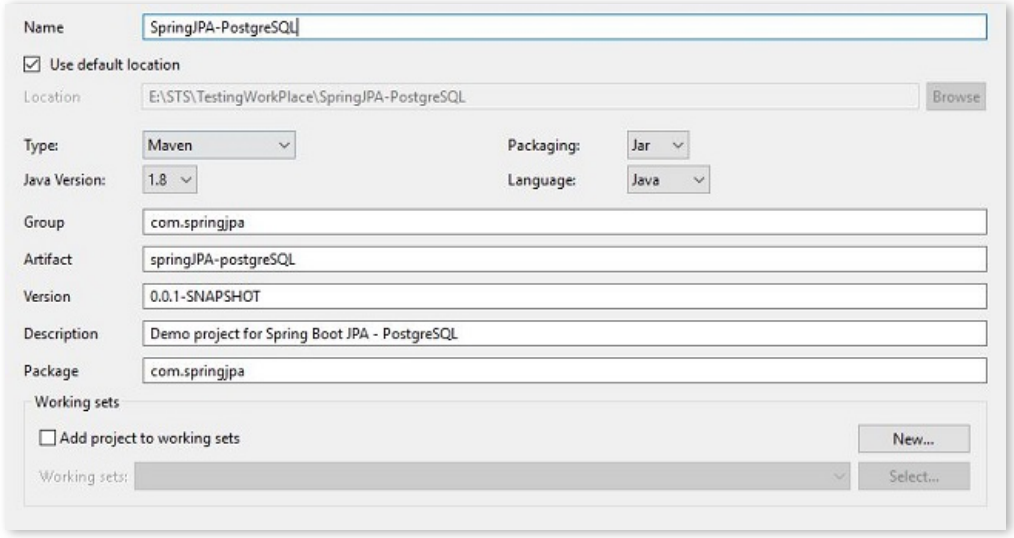
4. Demo Video



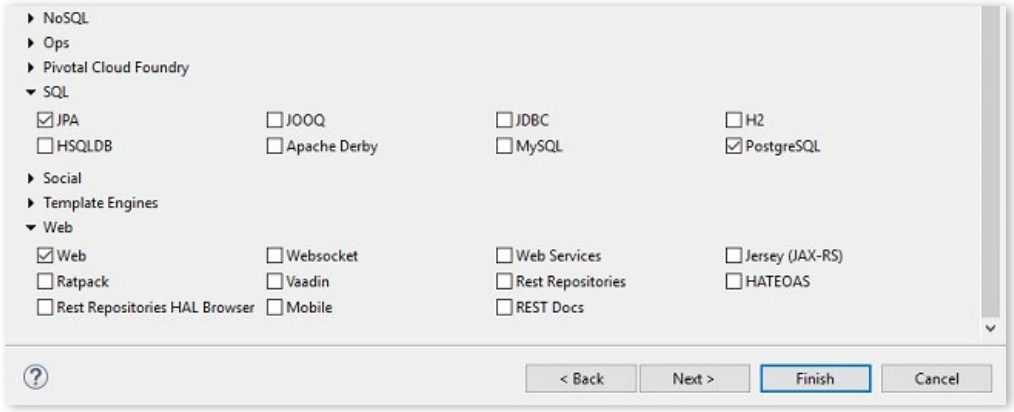
III. Practice

1. Create Spring Boot project & add Dependencies

Open **Spring Tool Suite**, on **Menu**, choose **File** -> **New** -> Spring Starter Project, then fill each fields:



Click **Next**, in **SQL**: choose **JPA** and **PostgreSQL**, in **Web**: choose **Web**.



Click **Finish**, then our project will be created successfully.

Open **pom.xml** and check 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.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

These dependencies were auto-generated by the configuration we have done before.

2. Configure Spring JPA

Open **application.properties**

```
spring.datasource.url=jdbc:postgresql://localhost/testdb
spring.datasource.username=postgres
spring.datasource.password=123
spring.jpa.generate-ddl=true
```

3. Create DataModel Class

Under package **model**, create class **Customer**.

Content of **Customer.java**:

```
@Entity
@Table(name = "customer")
public class Customer implements Serializable {

  private static final long serialVersionUID = -3009157732242241606L;
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long id;

  @Column(name = "firstname")
  private String firstName;

  @Column(name = "lastname")
  private String lastName;

  protected Customer() {
  }

  public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  @Override
  public String toString() {
    return String.format("Customer[id=%d, firstName='%s', lastName='%s']", id, firstName, lastName);
  }
}
```

Annotation **@Entity** indicates that **Customer** is an **Entity** and **@Table** specifies the primary table (name **customer**) for the annotated **@Entity**.

@ID specifies the primary key and **@GeneratedValue** indicates generation strategy for value of primary key.

@Column: mapped column (in the table) for persistent fields (in Java class).

We have 2 constructor methods:
– **protected** constructor will be used by Spring JPA.
– **public** constructor is for creating instances.

4. Create Spring JPA Repository Interface

This interface helps us do all **CRUD functions** for class **Customer**.

```
public interface CustomerRepository extends CrudRepository<Customer, Long>{
  List<Customer> findByLastName(String lastName);
}
```

5. Create Web Controller

The controller receives requests from client, using repository to update/get data and return results.

Content of **WebController.java**

```
package com.springjpa.controller;

import java.util.Arrays;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```
import com.springjpa.model.Customer;
import com.springjpa.repo.CustomerRepository;

@RestController
public class WebController {
  @Autowired
  CustomerRepository repository;

  @RequestMapping("/save")
  public String process(){
    // save a single Customer
    repository.save(new Customer("Jack", "Smith"));

    // save a list of Customers
    repository.save(Arrays.asList(new Customer("Adam", "Johnson"), new Customer("Kim", "Smith"),
                    new Customer("David", "Williams"), new Customer("Peter", "Davis")));

    return "Done";
  }


  @RequestMapping("/findall")
  public String findAll(){
    String result = "";

    for(Customer cust : repository.findAll()){
      result += cust.toString() + "<br>";
    }

    return result;
  }

  @RequestMapping("/findbyid")
  public String findById(@RequestParam("id") long id){
    String result = "";
    result = repository.findOne(id).toString();
    return result;
  }

  @RequestMapping("/findbylastname")
  public String fetchDataByLastName(@RequestParam("lastname") String lastName){
    String result = "";

    for(Customer cust: repository.findByLastName(lastName)){
      result += cust.toString() + "<br>";
```

```
    }

    return result;
  }
}
```

In the web controller methods which are annotated by @RequestMapping, we have used some methods of autowired **repository** which are implemented interface **CrudRepository**:

```
<S extends T> S save(S entity); //for @RequestMapping("/save")
T findOne(ID id); //for @RequestMapping("/findbyid")
Iterable<T> findAll(); //for @RequestMapping("/findall")
```

and the method **findByLastName** that we create in our interface **CustomerRepository**.

```
List<Customer> findByLastName(String lastName);
```

6. Using CommandLineRunner to clear old data

In main class, implement `CommandLineRunner` to clear old data if existed:

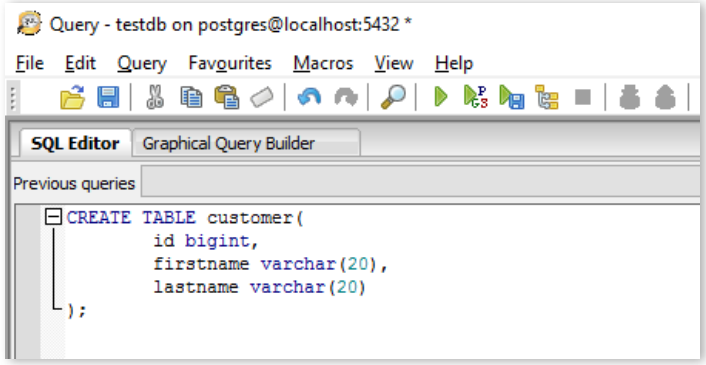
```
package com.springjpa;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.springjpa.repo.CustomerRepository;

@SpringBootApplication
public class SpringJpaPostgreSqlApplication implements CommandLineRunner{

  @Autowired
  CustomerRepository repository;

  public static void main(String[] args){
    SpringApplication.run(SpringJpaPostgreSqlApplication.class, args);
  }

  @Override
  public void run(String... arg0) throws Exception {
    // clear all record if existed before do the tutorial with new data
    repository.deleteAll();
```

```
  }
}
```

7. Create PostGreSQL table

Open **pdAdmin III**, use **SQL Editor** and make a query to create customer table:

```
CREATE TABLE customer(
    id BIGINT PRIMARY KEY     NOT NULL,
    firstname VARCHAR(20),
    lastname VARCHAR(20)
);
```

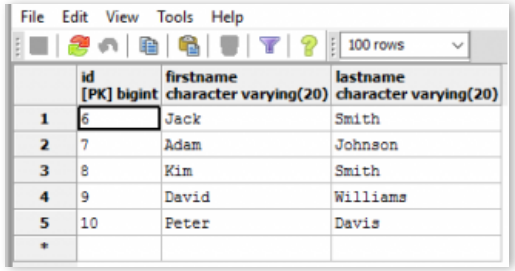


8. Run Spring Boot Application & Enjoy Result

– Config maven build:
`clean install`
– Run project with mode **Spring Boot App**
– Check results:

Request 1
`http://localhost:8080/save`

The browser returns `Done` and if checking database **testdb** with table **customer**, we can see some data rows has been added:

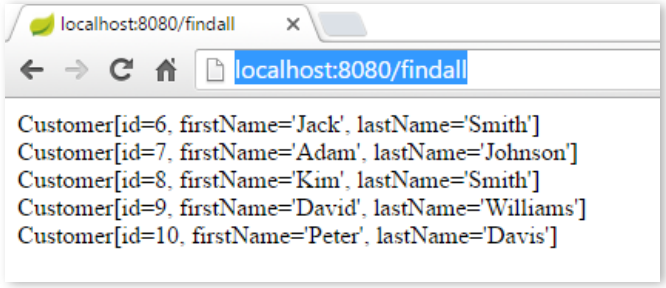

File Edit View Tools Help

100 rows

	id [PK] bigint	firstname character varying(20)	lastname character varying(20)
1	6	Jack	Smith
2	7	Adam	Johnson
3	8	Kim	Smith
4	9	David	Williams
5	10	Peter	Davis
*			

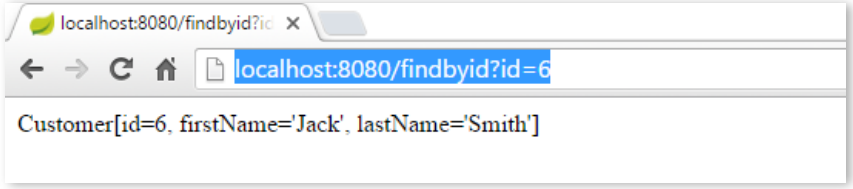
Request 2

`http://localhost:8080/findall`



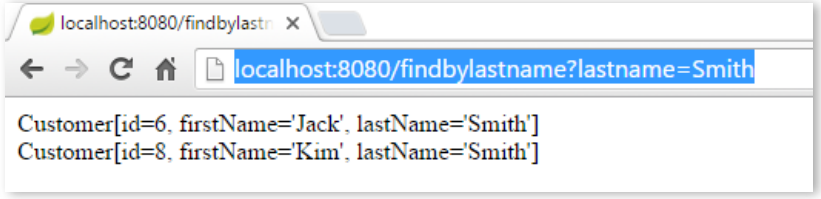
Request 3

`http://localhost:8080/findbyid?id=6`



Request 4

`http://localhost:8080/findbylastname?lastname=Smith`



IV. Source Code

SpringJPA-PostgreSQL

By grokonez | September 2, 2016.

Last updated on **March 15, 2019**.

Related Posts

- Spring Hibernate JPA – Upload/Download File/Image to PostgreSQL with @Lob Example
- Spring Boot + React Redux + PostgreSQL CRUD example
- SpringBoot Hazelcast cache with PostgreSQL backend
- SpringBoot Caffeine cache with PostgreSQL backend
- Angular 4 + Spring JPA + PostgreSQL example | Angular 4 Http Client – Spring Boot RestApi Server
- Spring JPA + PostgreSQL + AngularJS example | Spring Boot
- How to create paging and sorting results with Spring JPA and PostgreSQL | Spring Boot
- How to execute asynchronous query with Spring JPA and PostgreSQL | Spring Boot
- How to get streaming results with Spring JPA, Java 8 Stream and PostgreSQL | Spring Boot
- How to configure Spring Batch Step for restart

Post Tags

postgresql | spring boot | spring data | spring jpa

24 thoughts on “How to use Spring JPA with PostgreSQL | Spring Boot”

katrina

April 5, 2017 at 3:33 am

Im getting this error for /findbylastname and /findbyid

There was an unexpected error (type=Bad Request, status=400).
Required long parameter ‘id’ is not present

Any thoughts why I’m getting this error?

thank you!

javasampleapproachco

April 12, 2017 at 2:24 pm

Hi Katrina,

You should provide parameters in the Urls: findbyid**?id=6** and findbylastname**?lastname=Smith**.
`/findbylastname` or `/findbyid` is not enough.

Best Regards.

Nayantara Jeyaraj

July 14, 2017 at 6:42 am

Hi,
You’re getting this error since there is no record by the id 6. There are only 5 records with ids upto 5. Hence the query is run on a record that doesn’t exist.

Try changing the ID localhost url to query any record with an ID from 1-5 and you'll get the output as expected.

Example:
http://localhost:8080/findbyid?id=5
or
http://localhost:8080/findbyid?id=4
or
http://localhost:8080/findbyid?id=3
…
You get the idea.

Hope this was helpful!

---

jack

August 4, 2017 at 3:01 pm

Any way to change the packaging as war and deploy on a server? If i do and deploy it deploys but keep getting 404s

---

JavaSampleApproach

August 5, 2017 at 5:31 am

Hello,

Below post will help you to deploy war file for SpringBoot app:

https://grokonez.com/spring-framework/spring-boot/deploy-spring-boot-web-app-war-file-tomcat-server-maven-build

Regards,
JSA

Vasanth

May 7, 2017 at 3:17 am

@Autowired CustomerRepository not working.
Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.springboot.repo.CustomerRepository]

---

javasampleapproachco

May 9, 2017 at 11:42 am

Maybe the Maven repository is out of date, So I had changed the version
of SpringBoot from of 1.4 to RELEASE for supporting your case. You can download sourcecode & check again!

Best Regards!

Mohamed Adel

January 23, 2018 at 10:11 pm

make sure that springStart in the level before your entity in packages beacuse it scann all the inner packages for entities

---

Juliana

June 17, 2017 at 3:06 pm

Hi. I can’t make it’s work.
when I tried to build, I received an error:
Tests in error:
SpringJpaPostgreSqlApplicationTests.contextLoads » IllegalState Failed to load…

JavaSampleApproach

June 18, 2017 at 9:44 am

Hi Juliana,

Please check if you have add these lines of code to **application.properties** and have relative **PostgreSQL** database:

```
spring.datasource.url=jdbc:postgresql://localhost/testdb
```

```
spring.datasource.username=postgres
spring.datasource.password=123
spring.jpa.generate-ddl=true
```

Best Regards,

Phil

June 20, 2017 at 6:26 pm

Hi!
Great tutorial! Unfortunately I can't seem to get it to work. I get this error:
2017-06-20 11:16:52.465 WARN 39054 — [restartedMain] ationConfigEmbeddedWebApplicationContext : Exception encountered during context initialization – cancelling refresh attempt: org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration': Unsatisfied dependency expressed through constructor parameter 0; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'javax.sql.DataSource' available: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {}

JavaSampleApproach

June 24, 2017 at 1:56 am

Hi,

From the exception, I see you have wrong configuration with DataSource.
Please follow step by step guide in video to resolve it!

Regards,

Paul

July 1, 2017 at 2:13 am

Very cool, thank you very much!

---

alvin

November 6, 2017 at 4:08 am

hi, i got some error

"Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Nov 06 11:06:07 WIB 2017
There was an unexpected error (type=Not Found, status=404).
No message available"

whats wrong sir?
thanks

JavaSampleApproach

November 6, 2017 at 7:26 am

Hello Alvin,

We see that your problem comes from Interal Server Exception. Please double-check your implementation again with our article.
We have double-checked our sourcecode and it works well!
And also update some new code for cleaner and clearer view!

Please check out and try it!

Regards,
JSA

Vasilii

November 11, 2017 at 5:32 pm

Thank you!

---

burak

November 18, 2017 at 7:49 pm

Thanks for this great tutorial!

---

sid

December 13, 2017 at 10:00 pm

Downloaded your code and tried to run it. It wont run.
I am getting following error:

```
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'springJpaPostgreSqlApplication': Post-processing of merged bean definition failed;
  at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:524) ~[spring-beans-4.3.4.RELEASE.jc
  at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:482) ~[spring-beans-4.3.4.RELEASE.jar:
  at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:306) ~[spring-beans-4.3.4.RELEASE.jar:4.3.4.RELEASE]
  at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:230) ~[spring-beans-4.3.4.RELEASE.jar:4.3.4.RELE
  at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:302) ~[spring-beans-4.3.4.RELEASE.jar:4.3.4.RELEASE]
  at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:197) ~[spring-beans-4.3.4.RELEASE.jar:4.3.4.RELEASE]
  at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:754) ~[spring-beans-4.3.4.RELEASE.jar:4.
  at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:866) ~[spring-context-4.3.4.RELEASE.jar
  at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:542) ~[spring-context-4.3.4.RELEASE.jar:4.3.4.RELEASE]
  at org.springframework.boot.context.embedded.EmbeddedWebApplicationContext.refresh(EmbeddedWebApplicationContext.java:122) ~[spring-boot-1.4.2.RELEASE.jar:1.4.2.RELEASE]
  at org.springframework.boot.SpringApplication.refresh(SpringApplication.java:761) [spring-boot-1.4.2.RELEASE.jar:1.4.2.RELEASE]
  at org.springframework.boot.SpringApplication.refreshContext(SpringApplication.java:371) [spring-boot-1.4.2.RELEASE.jar:1.4.2.RELEASE]
  at org.springframework.boot.SpringApplication.run(SpringApplication.java:315) [spring-boot-1.4.2.RELEASE.jar:1.4.2.RELEASE]
  at org.springframework.boot.SpringApplication.run(SpringApplication.java:1186) [spring-boot-1.4.2.RELEASE.jar:1.4.2.RELEASE]
  at org.springframework.boot.SpringApplication.run(SpringApplication.java:1175) [spring-boot-1.4.2.RELEASE.jar:1.4.2.RELEASE]
  at com.springjpa.SpringJpaPostgreSqlApplication.main(SpringJpaPostgreSqlApplication.java:17) [classes/:na]
Caused by: java.lang.IllegalStateException: Failed to introspect bean class [com.springjpa.SpringJpaPostgreSqlApplication$$EnhancerBySpringCGLIB$$56221f41] for persistence
  at org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor.findPersistenceMetadata(PersistenceAnnotationBeanPostProcessor.java:401) ~[spring-orm-4.3.4
  at org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor.postProcessMergedBeanDefinition(PersistenceAnnotationBeanPostProcessor.java:333) ~[spring-c
  at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.applyMergedBeanDefinitionPostProcessors(AbstractAutowireCapableBeanFactory.java:947) ~[sp
  at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:521) ~[spring-beans-4.3.4.RELEASE.jc
  ... 15 common frames omitted
Caused by: java.lang.NoClassDefFoundError: org/springframework/data/repository/CrudRepository
  at java.lang.ClassLoader.defineClass1(Native Method) ~[na:1.8.0_151]
  at java.lang.ClassLoader.defineClass(ClassLoader.java:763) ~[na:1.8.0_151]
  at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142) ~[na:1.8.0_151]
  at java.net.URLClassLoader.defineClass(URLClassLoader.java:467) ~[na:1.8.0_151]
  at java.net.URLClassLoader.access$100(URLClassLoader.java:73) ~[na:1.8.0_151]
  at java.net.URLClassLoader$1.run(URLClassLoader.java:368) ~[na:1.8.0_151]
  at java.net.URLClassLoader$1.run(URLClassLoader.java:362) ~[na:1.8.0_151]
  at java.security.AccessController.doPrivileged(Native Method) ~[na:1.8.0_151]
```

```
  at java.net.URLClassLoader.findClass(URLClassLoader.java:361) ~[na:1.8.0_151]
  at java.lang.ClassLoader.loadClass(ClassLoader.java:424) ~[na:1.8.0_151]
  at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:335) ~[na:1.8.0_151]
  at java.lang.ClassLoader.loadClass(ClassLoader.java:357) ~[na:1.8.0_151]
  at java.lang.Class.getDeclaredFields0(Native Method) ~[na:1.8.0_151]
  at java.lang.Class.privateGetDeclaredFields(Class.java:2583) ~[na:1.8.0_151]
  at java.lang.Class.getDeclaredFields(Class.java:1916) ~[na:1.8.0_151]
  at org.springframework.util.ReflectionUtils.getDeclaredFields(ReflectionUtils.java:715) ~[spring-core-4.3.4.RELEASE.jar:4.3.4.RELEASE]
  at org.springframework.util.ReflectionUtils.doWithLocalFields(ReflectionUtils.java:656) ~[spring-core-4.3.4.RELEASE.jar:4.3.4.RELEASE]
  at org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor.buildPersistenceMetadata(PersistenceAnnotationBeanPostProcessor.java:418) ~[spring-orm-4.3.
  at org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor.findPersistenceMetadata(PersistenceAnnotationBeanPostProcessor.java:397) ~[spring-orm-4.3.4
  ... 18 common frames omitted
Caused by: java.lang.ClassNotFoundException: org.springframework.data.repository.CrudRepository
  at java.net.URLClassLoader.findClass(URLClassLoader.java:381) ~[na:1.8.0_151]
  at java.lang.ClassLoader.loadClass(ClassLoader.java:424) ~[na:1.8.0_151]
  at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:335) ~[na:1.8.0_151]
  at java.lang.ClassLoader.loadClass(ClassLoader.java:357) ~[na:1.8.0_151]
  ... 37 common frames omitted
```

JavaSampleApproach

December 14, 2017 at 4:14 am

Hello,

Caused by:

```
Caused by: java.lang.ClassNotFoundException: org.springframework.data.repository.CrudRepository
  at java.net.URLClassLoader.findClass(URLClassLoader.java:381) ~[na:1.8.0_151]
  at java.lang.ClassLoader.loadClass(ClassLoader.java:424) ~[na:1.8.0_151]
  at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:335) ~[na:1.8.0_151]
  at java.lang.ClassLoader.loadClass(ClassLoader.java:357) ~[na:1.8.0_151]
```

Please double check Spring JPA in your maven repository.
Refer video guide for setup the project at: https://youtu.be/ZURZMrfGsOA

Regards,
JSA

Ben

March 4, 2018 at 9:35 pm

Hi,
Thanks for the tutorial.

Shouldn’t repository.save(List..) be repository.saveAll(List…) ?

best regards.
Benjamin

constrict0r

April 10, 2018 at 3:51 pm

Hello, nice tutorial.

A correction: on the controller code to save the list of customers you should use repository.saveAll instead of repository.save

```
repository.saveAll(Arrays.asList(new Customer("Adam", "Johnson"), new Customer("Kim", "Smith")));
```

SHWETA SINHA

August 6, 2018 at 12:02 pm

How to ignore ERROR: duplicate key value violates unique constraint “customer_pkey” in this Application and proceed.
I added one more save statement with same values and update the table to have first name and last name as primary key.
repository.save(new Customer(“Jack”, “Smith”));
repository.save(new Customer(“Jack”, “Smith”));

My own application has lot of duplicate data, I want to ignore all that and proceed with other data.
Any idea how to handle the same in this tutorial

apostrophe

November 22, 2018 at 5:53 am

Thanks for the tutorial!

I created a version of this application, added another related entity (a “Department”), and included a bunch of psql commands (postgresql’s powerful interactive utility) in the readme.txt.

https://github.com/apostrophe/simple-postgresql-app

Megha Krishnappa

February 7, 2019 at 6:30 am

great tutorial!!