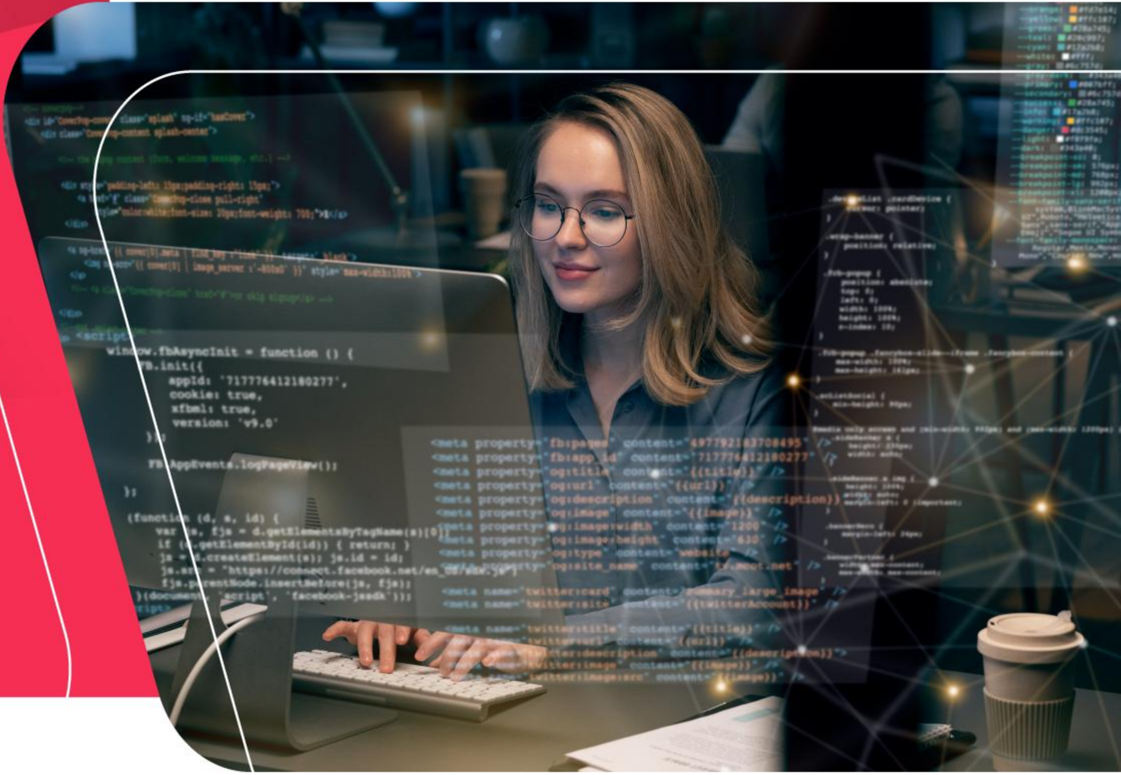
TIA-Academy

Java Bootcamp

Day 34



Technologies will be Use

- JDK 8/**11**/15
- JRE 8/**11**/15
- **Intellij IDEA Community Edition**
- JAVA 3rd Party Library (Network, DB, etc)
- JAVA Framework (Spring Boot & Spring JDBC)
- **MySQL Server Community**

SPRING BOOT

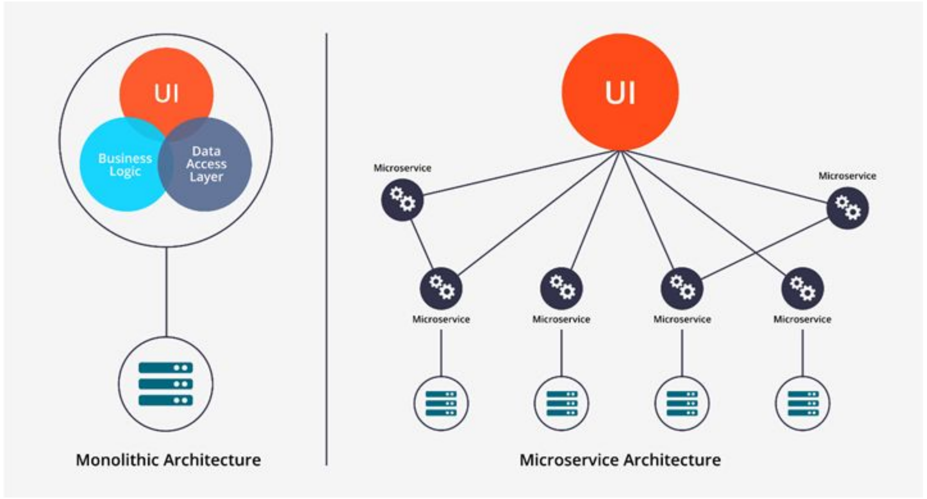
Introduction

- Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.
- This chapter will give you an introduction to Spring Boot and familiarizes you with its basic concepts.

TIA-Academy

What is Micro Service?

- Micro Service is an architecture that allows the developers to develop and deploy services independently. Each service running has its own process and this achieves the lightweight model to support business applications.



TIA-Academy

Advantages

- Micro services offers the following advantages to its developers –
  - Easy deployment
  - Simple scalability
  - Compatible with Containers
  - Minimum configuration
  - Lesser production time

TIA-Academy

What is Spring Boot?

- Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can **just run**.
- You can get started with minimum configurations without the need for an entire Spring configuration setup.
- Spring Boot offers the following advantages to its developers –
  - Easy to understand and develop spring applications
  - Increases productivity
  - Reduces the development time

What is Spring Boot?

- Spring Boot is designed with the following goals –
  - To avoid complex XML configuration in Spring
  - To develop a production ready Spring applications in an easier way
  - To reduce the development time and run the application independently
  - Offer an easier way of getting started with the application

Why Spring Boot?

- You can choose Spring Boot because of the features and benefits it offers as given here –
  - It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
  - It provides a powerful batch processing and manages REST endpoints.
  - In Spring Boot, everything is auto configured; no manual configurations are needed.
  - It offers annotation-based spring application
  - Eases dependency management
  - It includes Embedded Servlet Container

TIA-Academy

How does it work?

- Spring Boot automatically configures your application based on the dependencies you have added to the project by using **@EnableAutoConfiguration** annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.
- The entry point of the spring boot application is the class contains **@SpringBootApplication** annotation and the main method.
- Spring Boot automatically scans all the components included in the project by using **@ComponentScan** annotation.

TIA-Academy

Spring Boot Starters

- Handling dependency management is a difficult task for big projects. Spring Boot resolves this problem by providing a set of dependencies for developers convenience.
- For example, if you want to use Spring and JPA for database access, it is sufficient if you include **spring-boot-starter-data-jpa** dependency in your project.
- Note that all Spring Boot starters follow the same naming pattern **spring-boot-starter-** *, where * indicates that it is a type of the application.

TIA-Academy

Examples

- **Spring Boot Starter Actuator dependency** is used to monitor and manage your application. Its code is shown below –

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

- **Spring Boot Starter Security dependency** is used for Spring Security. Its code is shown below –

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

TIA-Academy

Examples

- **Spring Boot Starter web dependency** is used to write a Rest Endpoints. Its code is shown below –

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

- **Spring Boot Starter Thyme Leaf dependency** is used to create a web application. Its code is shown below –

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

TIA-Academy

Examples

- **Spring Boot Starter Test dependency** is used for writing Test cases. Its code is shown below –

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

Auto Configuration

- Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project. For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.

- For this purpose, you need to add **@EnableAutoConfiguration** annotation or **@SpringBootApplication** annotation to your main class file. Then, your Spring Boot application will be automatically configured.

Auto Configuration

- Observe the following code for a better understanding –

```
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@EnableAutoConfiguration
public class DemoApplication {
   public static void main(String[] args) {
      System.setProperty("spring.profiles.default", "dev");
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

Spring Boot Application

- The entry point of the Spring Boot Application is the class contains **@SpringBootApplication** annotation. This class should have the main method to run the Spring Boot application. **@SpringBootApplication** annotation includes Auto-Configuration, Component Scan, and Spring Boot Configuration.

- If you added **@SpringBootApplication** annotation to the class, you do not need to add the **@EnableAutoConfiguration, @ComponentScan** and **@SpringBootConfiguration** annotation. The **@SpringBootApplication** annotation includes all other annotations.

TIA-Academy

Spring Boot Application

- Observe the following code for a better understanding –

```
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

TIA-Academy

Component Scan

- Spring Boot application scans all the beans and package declarations when the application initializes. You need to add the **@ComponentScan** annotation for your class file to scan your components added in your project.
- Observe the following code for a better understanding –

```
import org.springframework.boot.SpringApplication;
import org.springframework.context.annotation.ComponentScan;

@ComponentScan
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

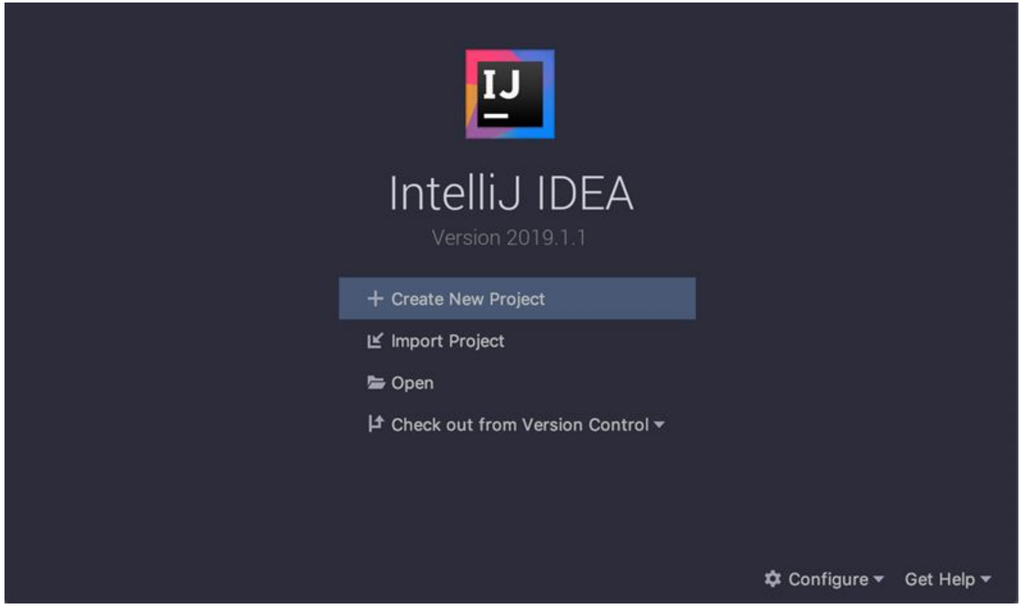
Requirements

- Java JDK 8 or higher
- Good Internet Connection for downloading Depedencies

TIA-Academy

Creating the project

- As soon as it starts, you'll see the following screen:



TIA-Academy

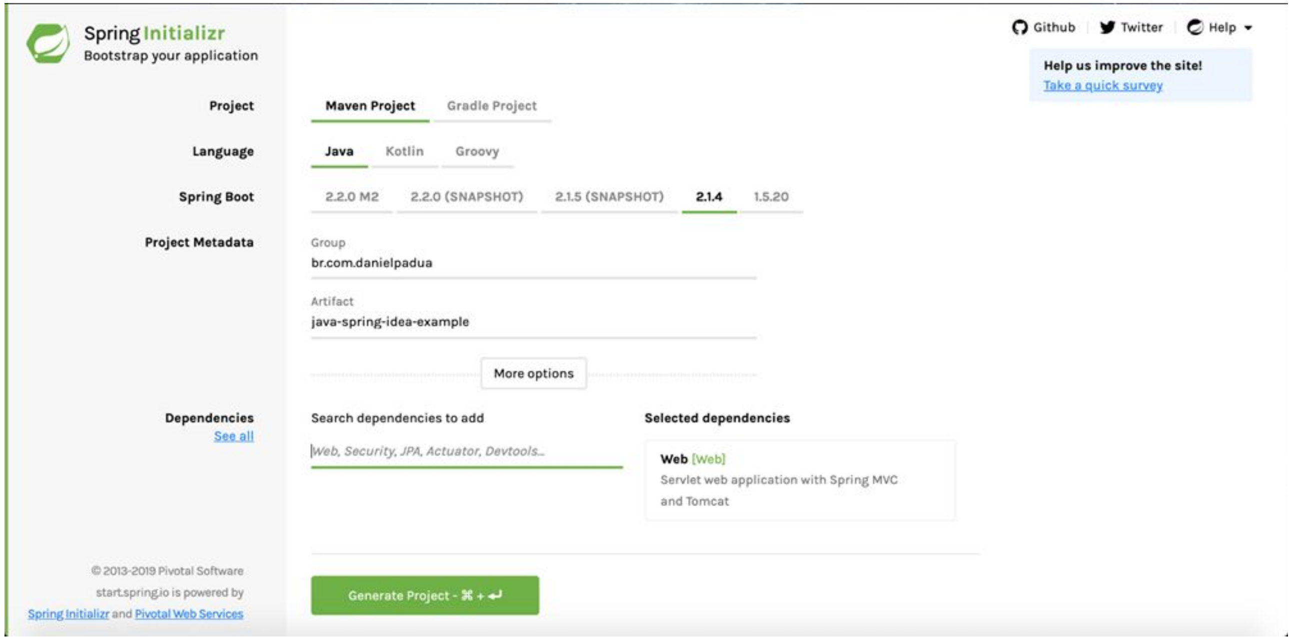
Creating the project

- For this example, we'll be using Maven as *build-tool*.

- Unfortunately using IntelliJ IDEA Community, according to the documentation, there's no support to create Spring Boot projects using Spring Initializr through the IDE in Community version, only in the Ultimate Edition. So, we have two choices that we can explore:

1st Way : Use Spring Initializr Web

- Access: https://start.spring.io, and fill the fields like below:


Spring Initializr
Bootstrap your application

Github | Twitter | Help

Help us improve the site!
Take a quick survey

Project: [✓ Maven Project] Gradle Project

Language: [✓ Java] Kotlin Groovy

Spring Boot: 2.2.0 M2 2.2.0 (SNAPSHOT) 2.1.5 (SNAPSHOT) [✓ 2.1.4] 1.5.20

Project Metadata
Group: br.com.danielpadua
Artifact: java-spring-idea-example

More options

Dependencies
See all

Search dependencies to add
Web, Security, JPA, Actuator, Devtools...

Selected dependencies
Web [Web]
Servlet web application with Spring MVC and Tomcat

Generate Project - ⌘ + ↵

© 2013-2019 Pivotal Software
start.spring.io is powered by
Spring Initializr and Pivotal Web Services

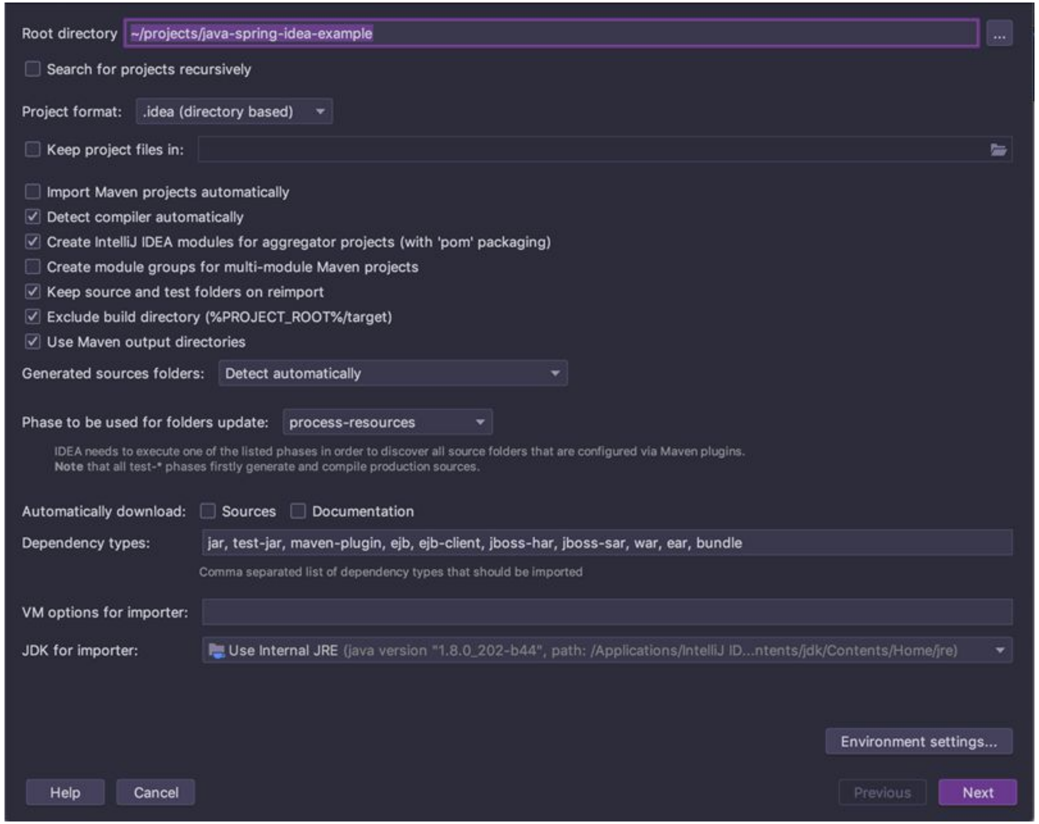
TIA-Academy

1st Way : Use Spring Initializr Web

- Don’t forget to select *Web* as dependency. Click *Generate Project* to download the project *zip* file.
- Extract it to a directory of your choice, go back to IntelliJ IDEA and select *Import Project*.
- Navigate to project’s directory and select the *pom.xml* file.
- You’ll see a window that is responsible for importing the Maven project, leave the defaults configs:

TIA-Academy

1st Way : Use Spring Initializr Web



Root directory ~/projects/java-spring-idea-example

☐ Search for projects recursively

Project format: .idea (directory based)

☐ Keep project files in:

☐ Import Maven projects automatically
☑ Detect compiler automatically
☑ Create IntelliJ IDEA modules for aggregator projects (with 'pom' packaging)
☐ Create module groups for multi-module Maven projects
☑ Keep source and test folders on reimport
☑ Exclude build directory (%PROJECT_ROOT%/target)
☑ Use Maven output directories

Generated sources folders: Detect automatically

Phase to be used for folders update: process-resources

IDEA needs to execute one of the listed phases in order to discover all source folders that are configured via Maven plugins.
Note that all test-* phases firstly generate and compile production sources.

Automatically download: ☐ Sources ☐ Documentation

Dependency types: jar, test-jar, maven-plugin, ejb, ejb-client, jboss-har, jboss-sar, war, ear, bundle

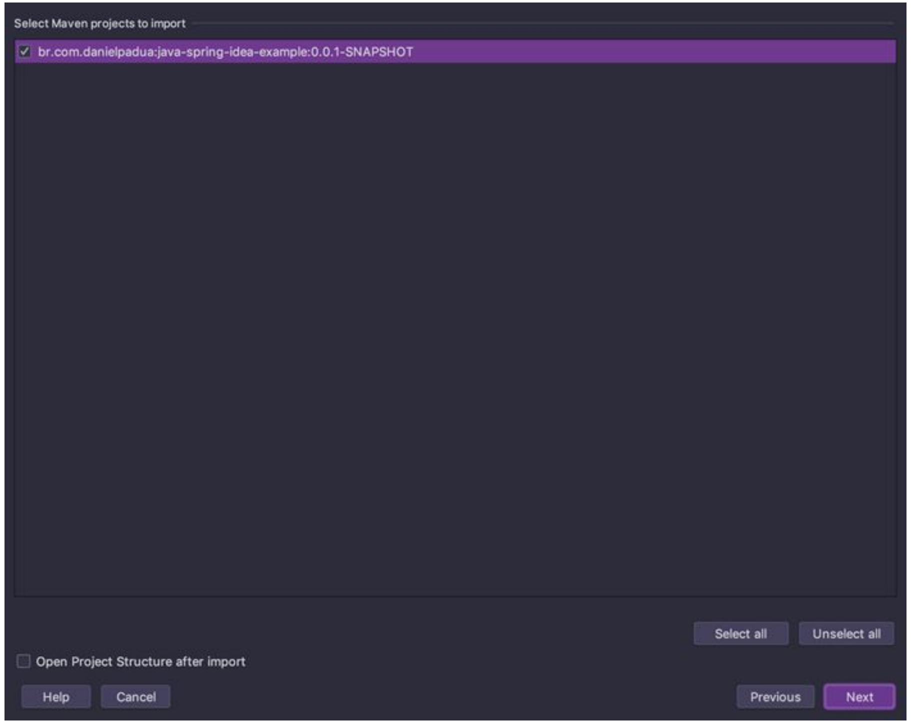
Comma separated list of dependency types that should be imported

VM options for importer:

JDK for importer: Use Internal JRE (java version "1.8.0_202-b44", path: /Applications/IntelliJ ID...ntents/jdk/Contents/Home/jre)

1st Way : Use Spring Initializr Web

- Select the project to import and click *next*:



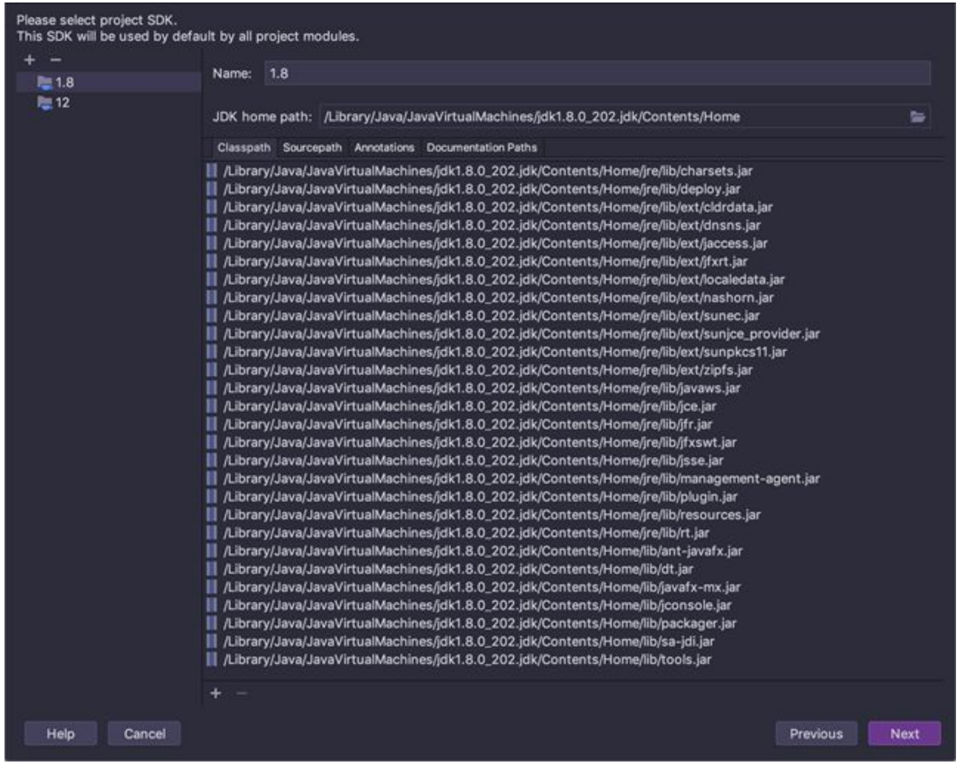
Select Maven projects to import

☑ br.com.danielpadua:java-spring-idea-example:0.0.1-SNAPSHOT

TIA-Academy

1st Way : Use Spring Initializr Web

- In the next screen, set the JDK version that you installed on the *Right*:
- In the next screen confirm the project name and click *finish*.

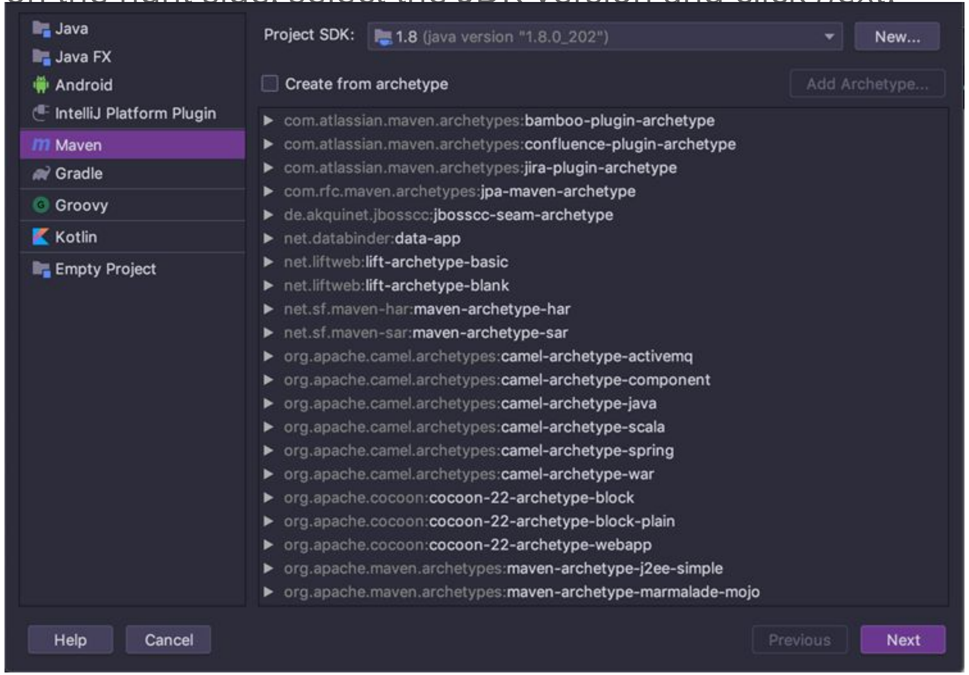


```
Please select project SDK.
This SDK will be used by default by all project modules.

1.8
12

Name: 1.8
JDK home path: /Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home

Classpath  Sourcepath  Annotations  Documentation Paths

/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/charsets.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/deploy.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/cldrdata.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/dnsns.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/jaccess.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/jfxrt.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/localedata.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/nashorn.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/sunec.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/sunjce_provider.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/sunpkcs11.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/ext/zipfs.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/javaws.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/jce.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/jfr.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/jfxswt.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/jsse.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/management-agent.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/plugin.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/resources.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/jre/lib/rt.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/lib/ant-javafx.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/lib/dt.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/lib/javafx-mx.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/lib/jconsole.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/lib/packager.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/lib/sa-jdi.jar
/Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/lib/tools.jar
```

TIA-Academy

2nd Way : Creating a Maven project and add Spring manually

- In IntelliJ IDEA's initial screen, select *Create New Project*, located on the left side tab and select *Maven*, on the right side, select the JDK version and click *next*:



Project SDK: 1.8 (java version "1.8.0_202")

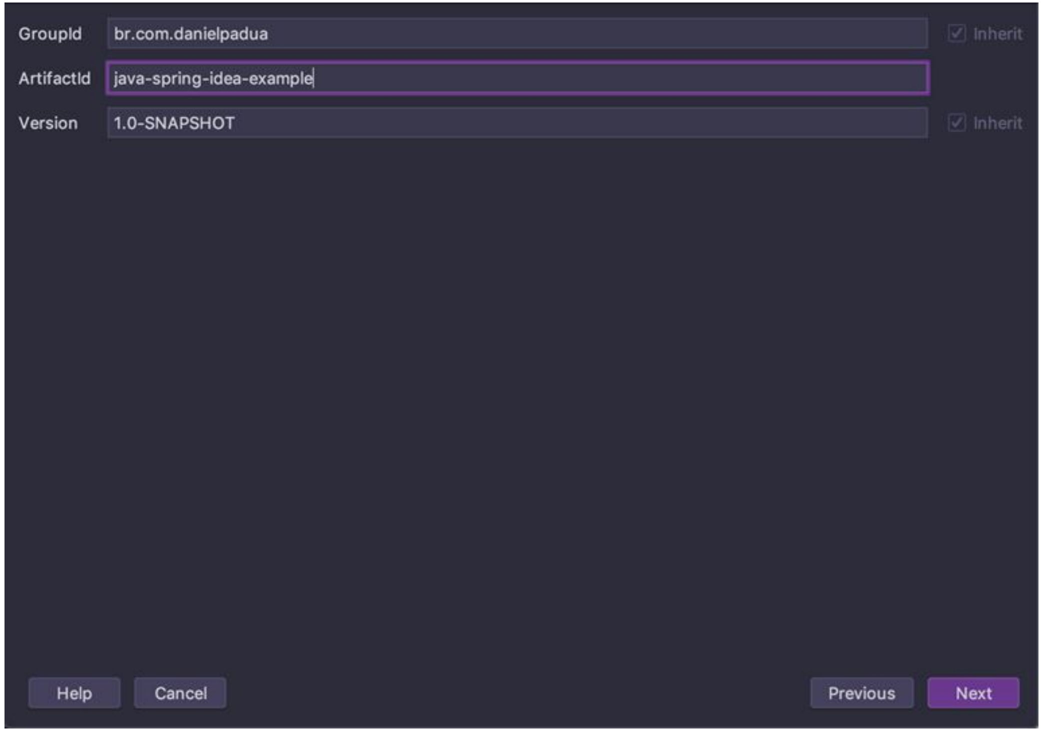
☐ Create from archetype

```
▶ com.atlassian.maven.archetypes:bamboo-plugin-archetype
▶ com.atlassian.maven.archetypes:confluence-plugin-archetype
▶ com.atlassian.maven.archetypes:jira-plugin-archetype
▶ com.rfc.maven.archetypes:jpa-maven-archetype
▶ de.akquinet.jbosscc:jbosscc-seam-archetype
▶ net.databinder:data-app
▶ net.liftweb:lift-archetype-basic
▶ net.liftweb:lift-archetype-blank
▶ net.sf.maven-har:maven-archetype-har
▶ net.sf.maven-sar:maven-archetype-sar
▶ org.apache.camel.archetypes:camel-archetype-activemq
▶ org.apache.camel.archetypes:camel-archetype-component
▶ org.apache.camel.archetypes:camel-archetype-java
▶ org.apache.camel.archetypes:camel-archetype-scala
▶ org.apache.camel.archetypes:camel-archetype-spring
▶ org.apache.camel.archetypes:camel-archetype-war
▶ org.apache.cocoon:cocoon-22-archetype-block
▶ org.apache.cocoon:cocoon-22-archetype-block-plain
▶ org.apache.cocoon:cocoon-22-archetype-webapp
▶ org.apache.maven.archetypes:maven-archetype-j2ee-simple
▶ org.apache.maven.archetypes:maven-archetype-marmalade-mojo
```

When selecting the archetype, IntelliJ IDEA will assume that you will use Quickstart archetype, which is ok for our goal.

2nd Way : Creating a Maven project and add Spring manually

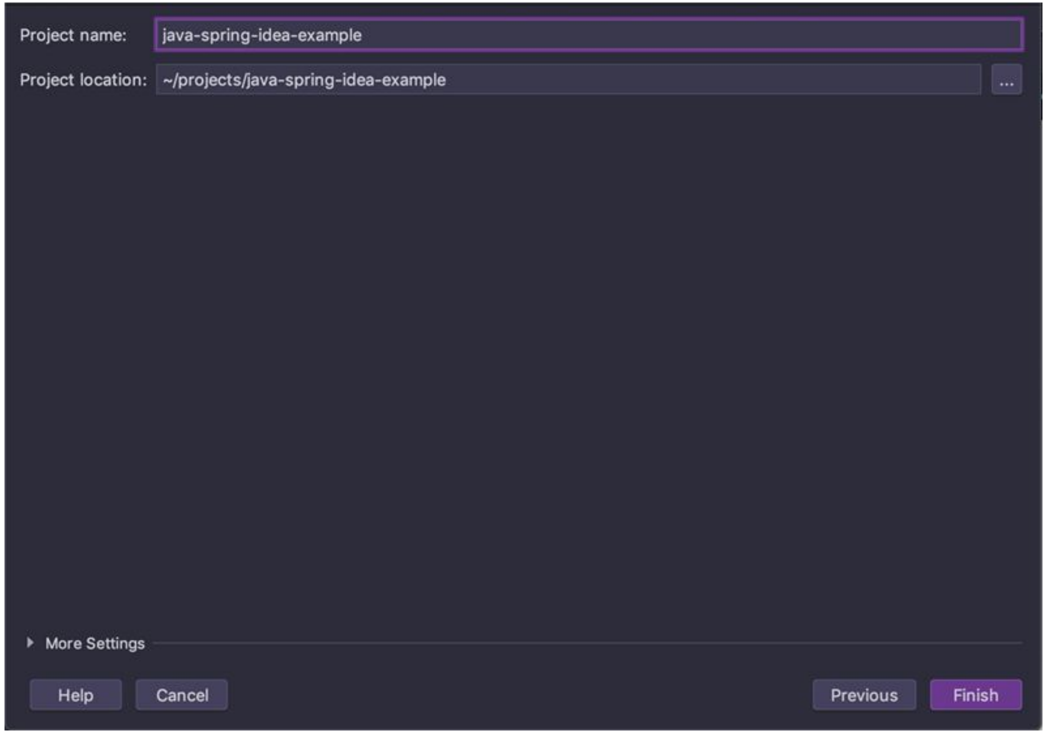
- In the next screen specify the GroupId, ArtifactId and the Version and click *next*:



GroupId	br.com.danielpadua	✓ Inherit
ArtifactId	java-spring-idea-example	
Version	1.0-SNAPSHOT	✓ Inherit

2nd Way : Creating a Maven project and add Spring manually

- After you just have to name your project and click finish:



Project name: java-spring-idea-example

Project location: ~/projects/java-spring-idea-example

TIA-Academy

Next Step

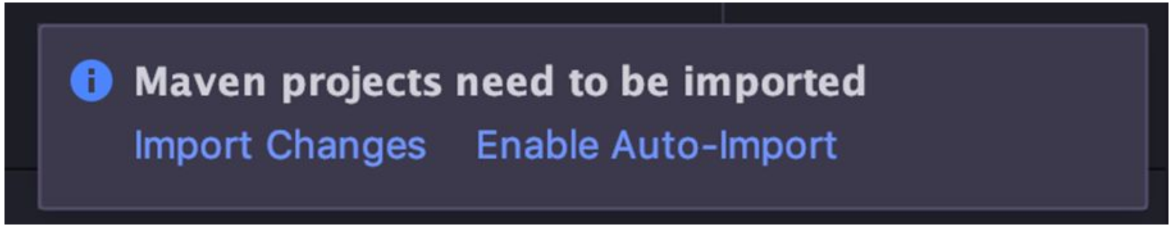
- With the project created, configure *pom.xml* according to the following file:

pom.xml

TIA-Academy

Next Step

- After you update *pom.xml* a notification will pop-up at the inferior right side of the screen:


Maven projects need to be imported
Import Changes Enable Auto-Import

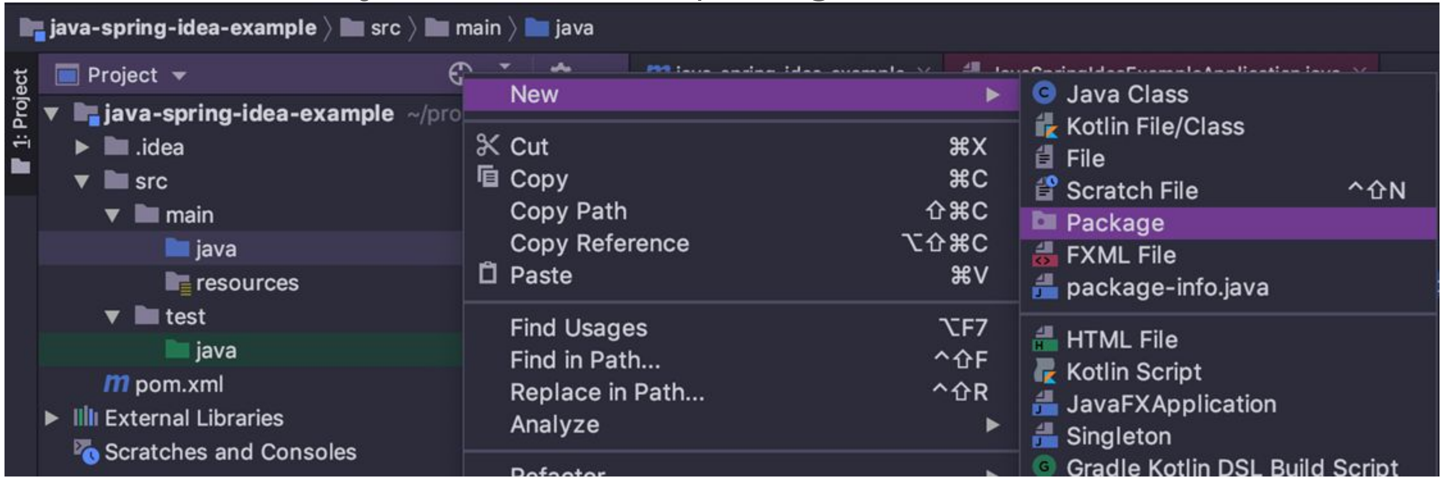
- Click *Import Changes* for *Maven* refresh all project dependencies.

TIA-Academy

Next Step

- Now we'll create a class that will contain the main function of the project.

- Remember that creating a class in default package is not a good java practice, so, click in source folder main/java and create a package:



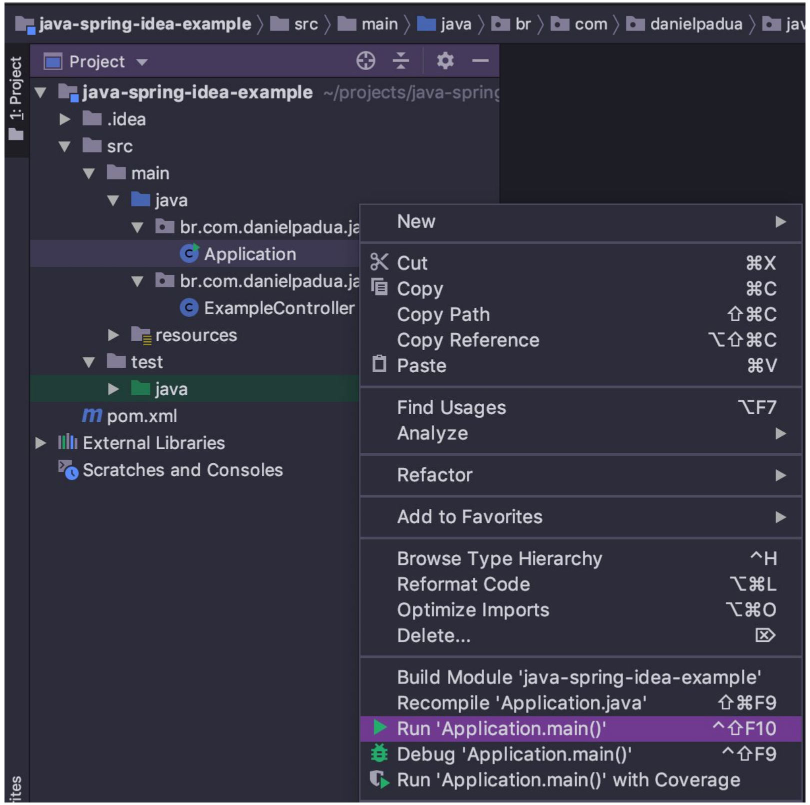
Next Step

- Write: *controllers* and confirm. Inside the generated package, create a class named: *ExampleController.java* and write the code:

```
package br.com.danielpadua.java_spring_idea_example.controllers;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/example")
public class ExampleController {
    @GetMapping("/hello-world")
    public ResponseEntity<String> get() {
        return ResponseEntity.ok("Hello World!");
    }
}
```

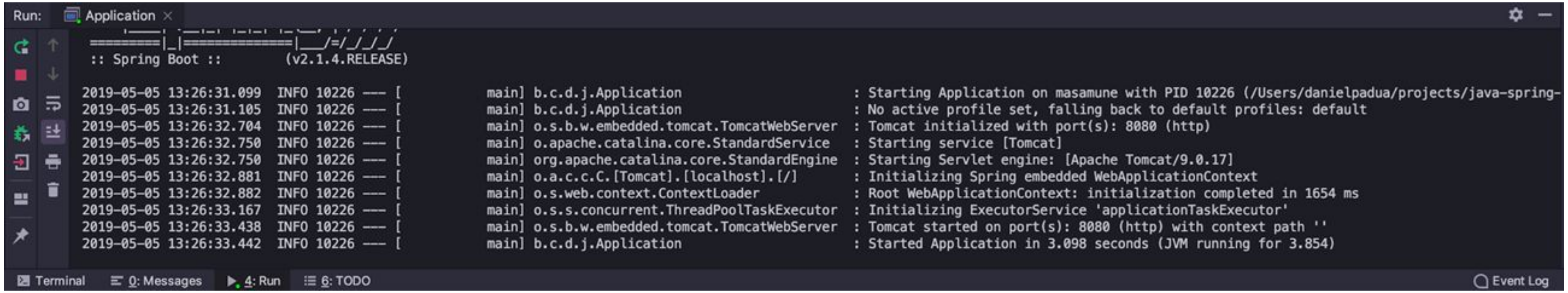
Completing the project

- Run the project by right clicking over the main class *ExampleApplication.java* and select the option *Run 'Application.main()'*:



Completing the project

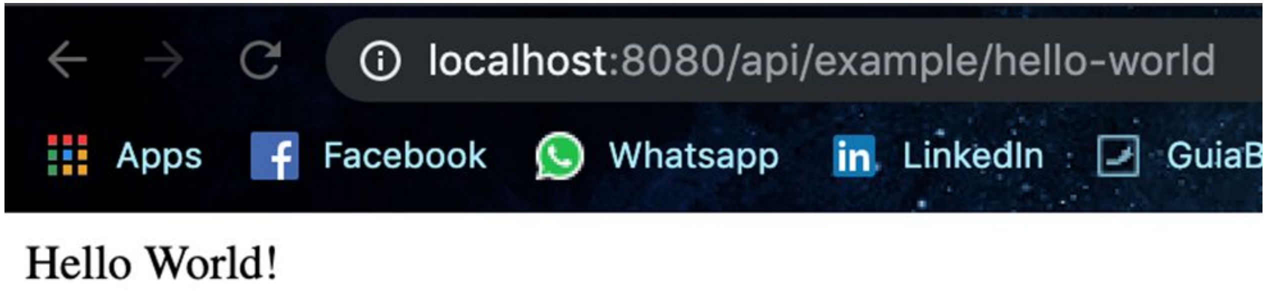
- After clicking run, you should see the output in the *Run* tab located at screen's bottom:



```
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.4.RELEASE)

2019-05-05 13:26:31.099  INFO 10226 --- [           main] b.c.d.j.Application                      : Starting Application on masamune with PID 10226 (/Users/danielpadua/projects/java-spring-
2019-05-05 13:26:31.105  INFO 10226 --- [           main] b.c.d.j.Application                      : No active profile set, falling back to default profiles: default
2019-05-05 13:26:32.704  INFO 10226 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2019-05-05 13:26:32.750  INFO 10226 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2019-05-05 13:26:32.750  INFO 10226 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.17]
2019-05-05 13:26:32.881  INFO 10226 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2019-05-05 13:26:32.882  INFO 10226 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1654 ms
2019-05-05 13:26:33.167  INFO 10226 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2019-05-05 13:26:33.438  INFO 10226 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2019-05-05 13:26:33.442  INFO 10226 --- [           main] b.c.d.j.Application                      : Started Application in 3.098 seconds (JVM running for 3.854)
```

Completing the project

- To test the app, you only have to open your favorite browser and access: http://localhost:8080/**api/example/hello-world**, and you should see the *Hello World* message:



Hello World!

TIA-Academy

Spring Boot REST API

TIA-Academy

Introduction

- Spring Boot has REST support by providing default dependencies right out of the box.
- Spring Boot provides `RestTemplateBuilder` that can be used to customize the `RestTemplate` before calling the REST endpoints.
- In this tutoriall we will show how to create and run a simple CRUD REST API project using Spring Boot.

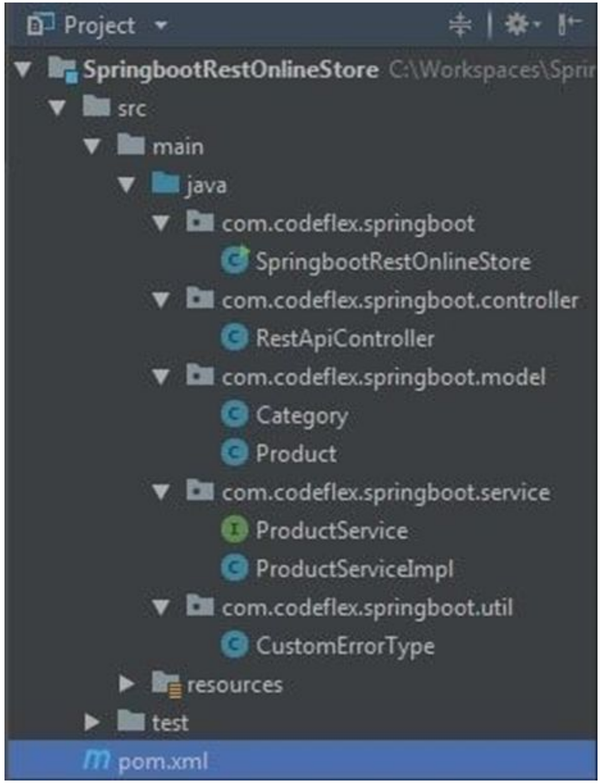
TIA-Academy

Prerequisites

- IntelliJ IDEA (Community edition)
- Spring Boot 1.4.3.RELEASE
- Spring 4.3.5.RELEASE
- Maven 3.1
- JDK 1.8 or More

TIA-Academy

Project Structure


```
Project
▼ SpringbootRestOnlineStore C:\Workspaces\Sprir
  ▼ src
    ▼ main
      ▼ java
        ▼ com.codeflex.springboot
            SpringbootRestOnlineStore
        ▼ com.codeflex.springboot.controller
            RestApiController
        ▼ com.codeflex.springboot.model
            Category
            Product
        ▼ com.codeflex.springboot.service
            ProductService
            ProductServiceImpl
        ▼ com.codeflex.springboot.util
            CustomErrorType
      ▶ resources
    ▶ test
  pom.xml
```

Project Structure

- Our project represents a simple online store that contain products. We want our application to expose REST API's CRUD principle – create, read, update, and delete.
  - To Create a product : HTTP POST should be used
  - To Retrieve a product : HTTP GET should be used
  - To Update a product : HTTP PUT should be used
  - To Delete a product : HTTP DELETE should be used

Project Structure

- Usually REST Web services return JSON or XML as response, although it is not limited to these types only.
- Clients can specify (using HTTP Accept header) the resource type they are interested in, and server will return the resource.
- Any Spring `@RestController` in a Spring Boot application will render JSON response by default as long as Jackson2 [jackson-databind] is on the class-path.

Maven Dependency Management (pom.xml)

pom.xml

Main Class

```
package com.codeflex.springboot;
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication(scanBasePackages={"com.codeflex.springboot"}
)
public class SpringbootRestOnlineStore {


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

TIA-Academy

REST Controller

- These are our REST APIs:
  - GET request to /api/product/ returns a list of all products
  - GET request to /api/product/3 returns the product with ID 3
  - POST request to /api/product/ with a JSON product object in the request's body will create a new product
  - PUT request to /api/product/5 with a JSON product object in the request's body will update the object with ID 5
  - DELETE request to /api/product/7 deletes the product with ID 7
  - DELETE request to /api/product/ deletes all the products

RestApiController.java

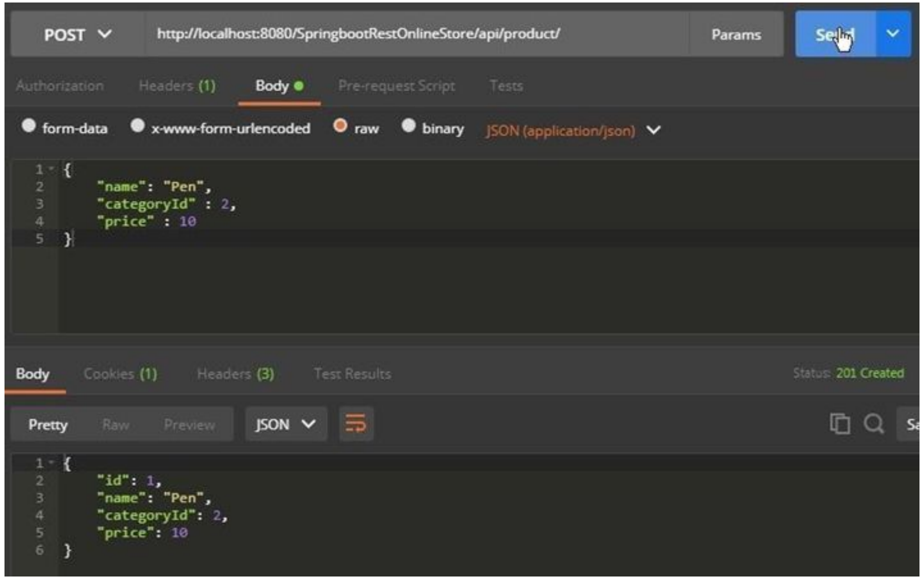
REST Controller

- **@RestController:** Spring 4′s new @RestController annotation annotation eliminates the need of annotating each method with @ResponseBody.
- **@RequestBody:** Indicates that Spring will bind the incoming HTTP request body to that parameter. While doing that, Spring will use HTTP Message converters to convert the HTTP request body into domain object, based on ACCEPT or Content-Type header present in request.
- **@ResponseBody:** Indictaes that Spring will bind the return value to outgoing HTTP response body. While doing that, Spring will use HTTP Message converters to convert the return value to HTTP response body, based on Content-Type present in request HTTP header. As already mentioned, in Spring 4, you may stop using this annotation.

  ResponseEntity: Represents the entire HTTP response.
- **@PathVariable:** This annotation indicates that a method parameter should be bound to a URI template variable.

Testing

- Let's run our project and test it via Postman.
- Create product (POST)
- As you can see we filled our request body with JSON object and after clicking on "Send" we received 201 Status code and the response body with the same object including the ID.



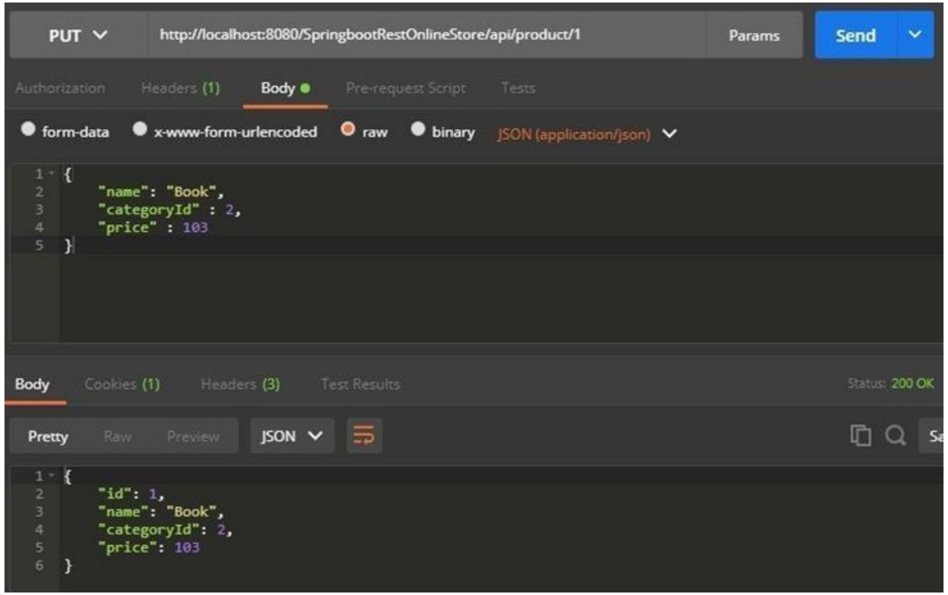
```
{
    "name": "Pen",
    "categoryId" : 2,
    "price" : 10
}
```

```
{
    "id": 1,
    "name": "Pen",
    "categoryId": 2,
    "price": 10
}
```

TIA-Academy

Testing

- Update product (PUT)
- We updated our pen that we created earlier with another information. It becomes a book now.



http://localhost:8080/SpringbootRestOnlineStore/api/product/1

JSON (application/json)

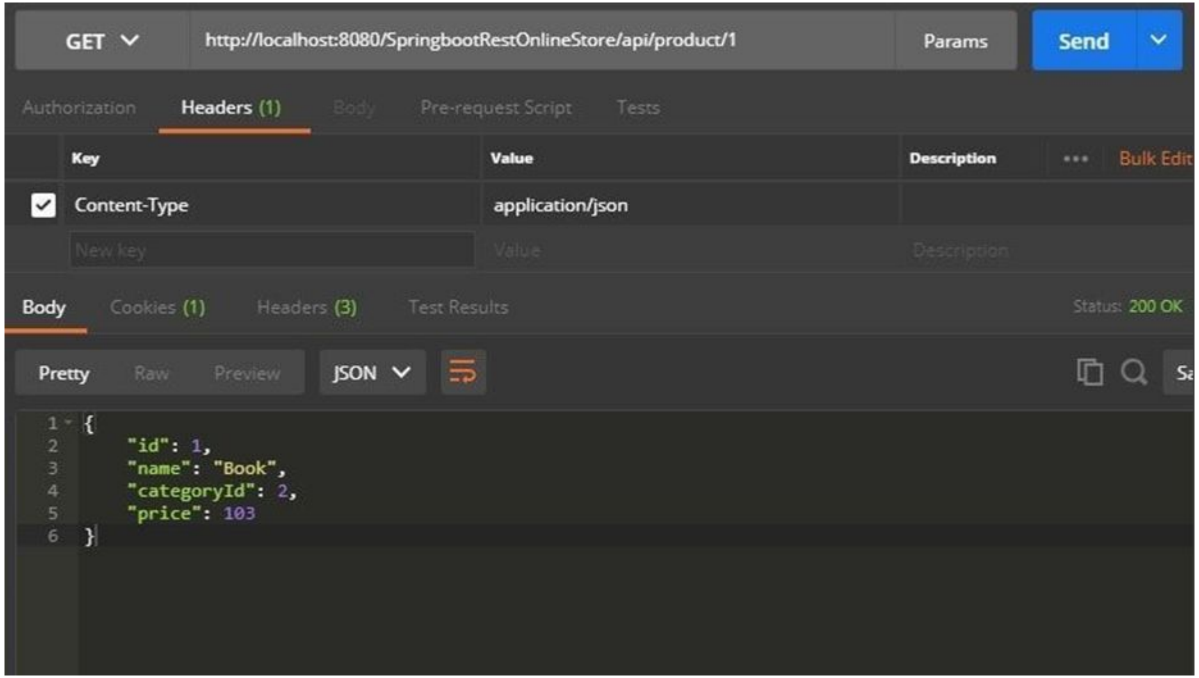
```
{
    "name": "Book",
    "categoryId" : 2,
    "price" : 103
}
```

Status: 200 OK

```
{
    "id": 1,
    "name": "Book",
    "categoryId": 2,
    "price": 103
}
```

Testing

- Get product (GET)



```
http://localhost:8080/SpringbootRestOnlineStore/api/product/1
```

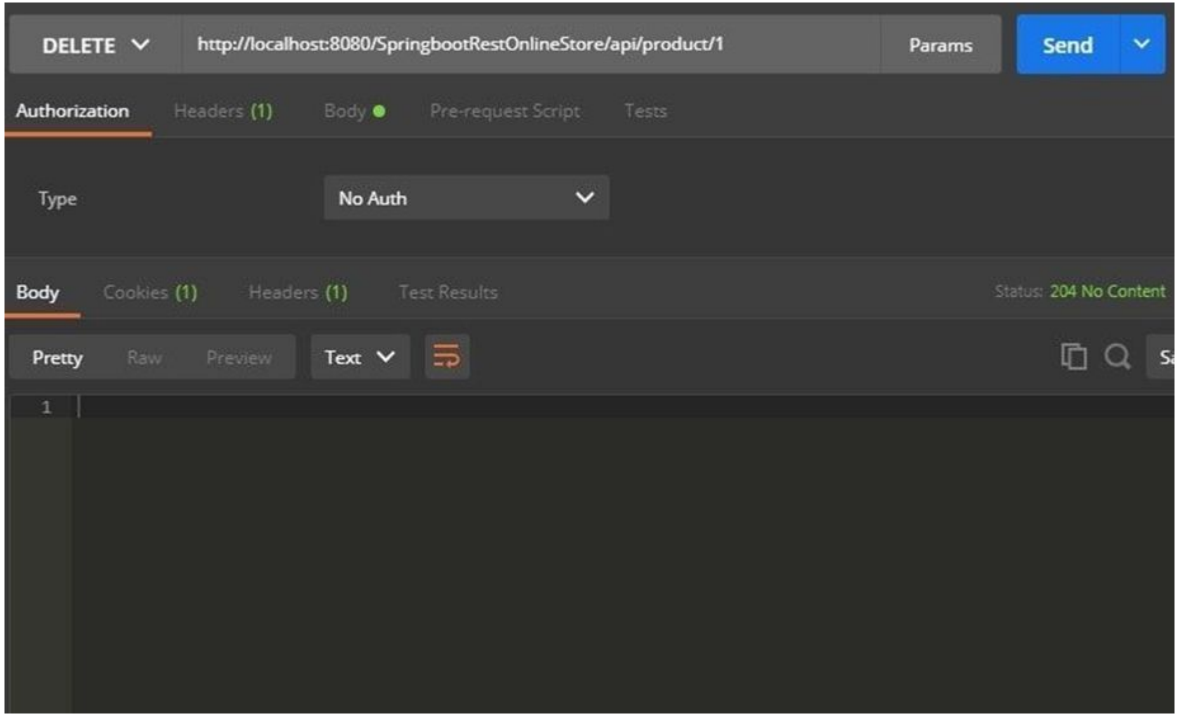
Key	Value	Description
Content-Type	application/json	

Status: 200 OK

```
{
    "id": 1,
    "name": "Book",
    "categoryId": 2,
    "price": 103
}
```

Testing

- Delete product (DELETE)



http://localhost:8080/SpringbootRestOnlineStore/api/product/1

Type: No Auth

Status: 204 No Content

Springboot JDBC

Spring Boot JDBC

- **Spring Boot JDBC** provides starter and libraries for connecting an application with JDBC.
- In Spring Boot JDBC, the database related beans such as **DataSource, JdbcTemplate,** and **NamedParameterJdbcTemplate** auto-configures and created during the startup.
- We can autowire these classes if we want to use it. For example:

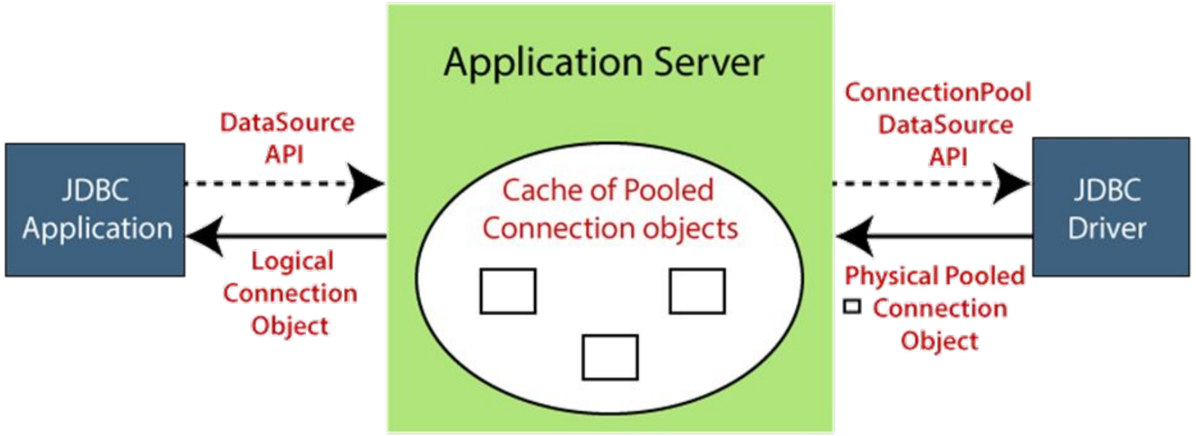
```
@Autowired
JdbcTemplate jdbcTemplate;
@Autowired
private NamedParameterJdbcTemplate jdbcTemplate;
```

- In **application.properties** file, we configure **DataSource** and **connection pooling**. Spring Boot chooses **tomcat** pooling by default.

TIA-Academy

JDBC Connection Pooling

- **JDBC connection pooling** is a mechanism that manages **multiple** database connection requests. In other words, it facilitates connection reuse, a memory cache of database connections, called a **connection pool.** A connection pooling module maintains it as a layer on top of any standard JDBC driver product.

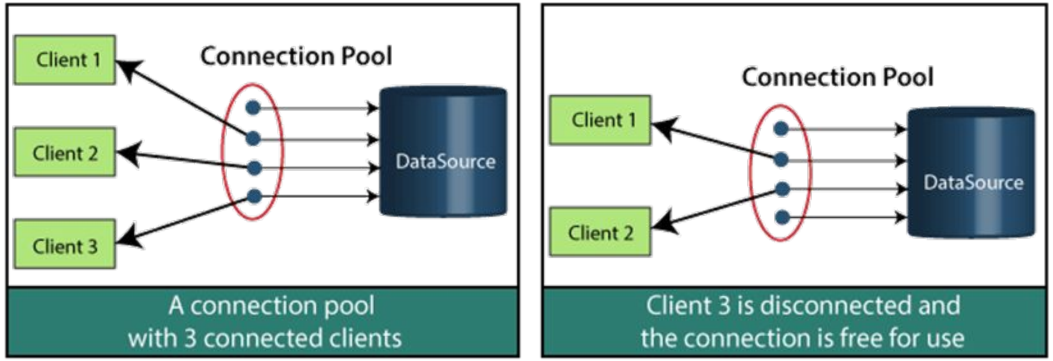

Application Server
JDBC Application
DataSource API
Logical Connection Object
Cache of Pooled Connection objects
ConnectionPool DataSource API
Physical Pooled Connection Object
JDBC Driver

JDBC Connection Pooling

- It increases the speed of data access and reduces the number of database connections for an application.
- It also improves the performance of an application. Connection pool performs the following tasks:
  - Manage available connection
  - Allocate new connection
  - Close connection

TIA-Academy

JDBC Connection Pooling



Client 1
Client 2
Client 3
Connection Pool
DataSource
A connection pool with 3 connected clients
Client 1
Client 2
Connection Pool
DataSource
Client 3 is disconnected and the connection is free for use

- In the above figure, there are **clients, a connection pool** (that has four available connections), and **a DataSource**.
- In the first figure, there are three clients connected with different connections, and a connection is available. In the second figure, Client 3 has disconnected, and that connection is available.
- When a client completes his work, it releases the connection, and that connection is available for other clients.

TIA-Academy

Spring JDBC

- If we want to connect to MySQL database, we need to include the JDBC driver in the application's classpath:

```
<!-- MySQL JDBC driver -->
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
</dependency>
```

- After that, define the **datasoure** properties in **application.properties** file.

TIA-Academy

Spring JDBC

- Use the following properties if you are using **MySQL** database:

```
spring.datasource.url=jdbc:mysql://192.168.1.4:3306/test
spring.datasource.username=javatpoint
spring.datasource.password=password
```

- Use the following properties if you are using **Oracle** database:

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:orcl
spring.datasource.username=system
spring.datasource.password=Password123
```

Spring Boot JDBC Examples

- Technologies used :
  - Spring Boot 2.1.2.RELEASE
  - Spring JDBC 5.1.4.RELEASE
  - HikariCP 3.2.0
  - H2 in-memory database 1.4.197
  - Maven 3
  - Java 8

TIA-Academy

Spring Boot JDBC Examples

- In Spring Boot JDBC, the database related beans like DataSource, JdbcTemplate and NamedParameterJdbcTemplate will be configured and created during the startup, to use it, just **@Autowired** the bean you want, for examples:

```
@Autowired
   JdbcTemplate jdbcTemplate;

@Autowired
   private NamedParameterJdbcTemplate jdbcTemplate;
```

TIA-Academy

Spring Boot JDBC Examples

- To connect to a database (e.g MySQL), include the JDBC driver in the project classpath

  <pom.xml> :

```
<!-- MySQL JDBC driver -->
<dependency>
     <groupId>mysql</groupId>
     <artifactId>mysql-connector-java</artifactId>
</dependency>
```

TIA-Academy

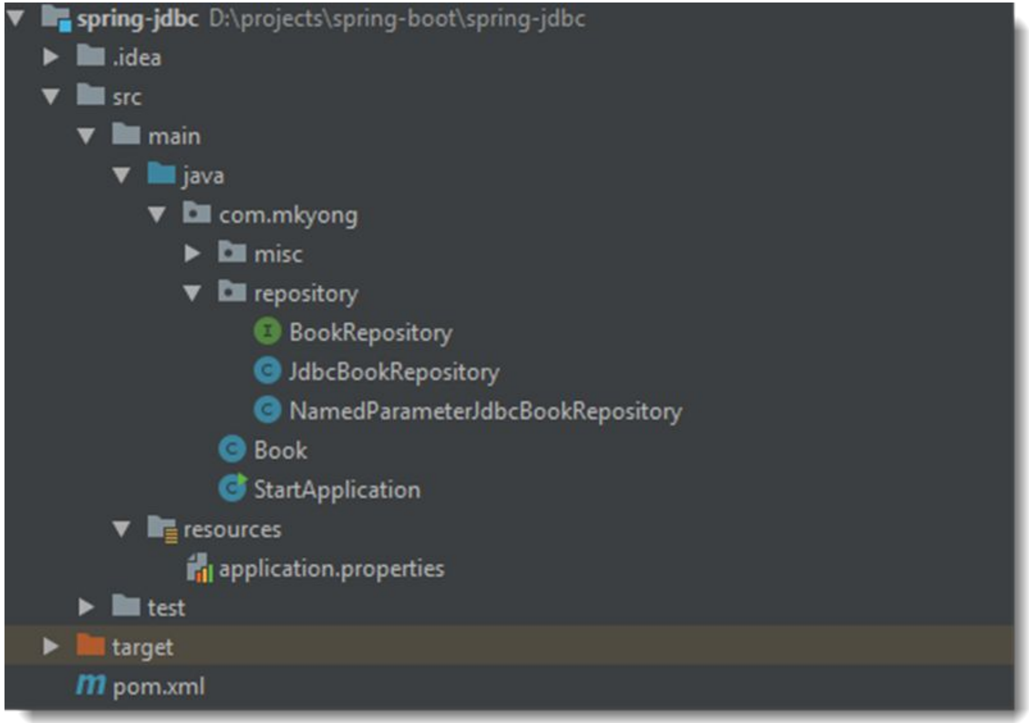
Spring Boot JDBC Examples

- And define the datasoure properties in application.properties

```
## MySQL
#spring.datasource.url=jdbc:mysql://192.168.1.4:3306/test
#spring.datasource.username=mkyong
#spring.datasource.password=password

# Oracle
#spring.datasource.url=jdbc:oracle:thin:@localhost:1521:orcl
#spring.datasource.username=system
#spring.datasource.password=Password123
```

TIA-Academy

Project Directory



```
▼ spring-jdbc D:\projects\spring-boot\spring-jdbc
  ▶ .idea
  ▼ src
    ▼ main
      ▼ java
        ▼ com.mkyong
          ▶ misc
          ▼ repository
              BookRepository
              JdbcBookRepository
              NamedParameterJdbcBookRepository
            Book
            StartApplication
      ▼ resources
          application.properties
    ▶ test
  ▶ target
    pom.xml
```

Maven

- spring-boot-starter-jdbc is what we need.

pom_jdbx.xml

Maven

- Display the project dependencies.

```
$ mvn dependency:tree
[INFO] org.springframework.boot:spring-jdbc:jar:1.0
[INFO] +- org.springframework.boot:spring-boot-starter-jdbc:jar:2.1.2.RELEASE:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter:jar:2.1.2.RELEASE:compile
[INFO] |  |  +- org.springframework.boot:spring-boot:jar:2.1.2.RELEASE:compile
[INFO] |  |  |  \- org.springframework:spring-context:jar:5.1.4.RELEASE:compile
[INFO] |  |  |     +- org.springframework:spring-aop:jar:5.1.4.RELEASE:compile
[INFO] |  |  |     \- org.springframework:spring-expression:jar:5.1.4.RELEASE:compile
[INFO] |  |  +- org.springframework.boot:spring-boot-autoconfigure:jar:2.1.2.RELEASE:compile
[INFO] |  |  +- org.springframework.boot:spring-boot-starter-logging:jar:2.1.2.RELEASE:compile
[INFO] |  |  |  +- ch.qos.logback:logback-classic:jar:1.2.3:compile
[INFO] |  |  |  |  \- ch.qos.logback:logback-core:jar:1.2.3:compile
[INFO] |  |  |  +- org.apache.logging.log4j:log4j-to-slf4j:jar:2.11.1:compile
[INFO] |  |  |  |  \- org.apache.logging.log4j:log4j-api:jar:2.11.1:compile
[INFO] |  |  |  \- org.slf4j:jul-to-slf4j:jar:1.7.25:compile
[INFO] |  |  +- javax.annotation:javax.annotation-api:jar:1.3.2:compile
[INFO] |  |  +- org.springframework:spring-core:jar:5.1.4.RELEASE:compile
[INFO] |  |  |  \- org.springframework:spring-jcl:jar:5.1.4.RELEASE:compile
[INFO] |  |  \- org.yaml:snakeyaml:jar:1.23:runtime
[INFO] |  +- com.zaxxer:HikariCP:jar:3.2.0:compile
[INFO] |  |  \- org.slf4j:slf4j-api:jar:1.7.25:compile
[INFO] |  \- org.springframework:spring-jdbc:jar:5.1.4.RELEASE:compile
[INFO] |     +- org.springframework:spring-beans:jar:5.1.4.RELEASE:compile
[INFO] |     \- org.springframework:spring-tx:jar:5.1.4.RELEASE:compile
[INFO] +- com.h2database:h2:jar:1.4.197:compile
```

TIA-Academy

BookRepository

- A pure Java interface for the repository, later implement with JdbcTemplate and NamedParameterJdbcTemplate (BookRepository.java)

```
package com.mkyong.repository;
import com.mkyong.Book;
import java.math.BigDecimal;
import java.util.List;
import java.util.Optional;

public interface BookRepository {
    int count();
    int save(Book book);
    int update(Book book);
    int deleteById(Long id);
    List<Book> findAll();

    List<Book> findByNameAndPrice(String name, BigDecimal price);
    Optional<Book> findById(Long id);
    String getNameById(Long id);
}
```

BookRepository

- (Book.java)

```
package com.mkyong;

import java.math.BigDecimal;

public class Book {

    private Long id;
    private String name;
    private BigDecimal price;

    //... setters getters constructors...
}
```

JdbcTemplate

- Crud Example

JdbcBookRepository.java

TIA-Academy

NamedParameterJdbcTemplate

- The NamedParameterJdbcTemplate adds support for named parameters in steads of classic placeholder ? argument.

NamedParameterJdbcBookRepository.java

TIA-Academy

application.properties

- For in-memory database, nothing to configure, if we want to connect to a real database, define a datasource.url property: (application.properties)

```
logging.level.org.springframework=info
#logging.level.org.springframework.jdbc=DEBUG
logging.level.com.mkyong=INFO
logging.level.com.zaxxer=DEBUG
logging.level.root=ERROR
spring.datasource.hikari.connectionTimeout=20000
spring.datasource.hikari.maximumPoolSize=5
logging.pattern.console=%-5level %logger{36} - %msg%n

## MySQL
spring.datasource.url=jdbc:mysql://192.168.1.4:3306/test
spring.datasource.username=mkyong
spring.datasource.password=password

# Oracle
#spring.datasource.url=jdbc:oracle:thin:@localhost:1521:orcl
#spring.datasource.username=system
#spring.datasource.password=Password123
```

Start Spring Boot

- Start Spring Boot application, test CRUD. (StartApplication.java)

NamedParameterJdbcBookRepository.java

TIA-Academy

Demo

```
$ mvn spring-boot:run

INFO  com.mkyong.StartApplication - Started StartApplication in 1.051 seconds (JVM running for 1.3)
INFO  com.mkyong.StartApplication - StartApplication...
INFO  com.mkyong.StartApplication - Creating tables for testing...
DEBUG com.zaxxer.hikari.HikariConfig - HikariPool-1 - configuration:
DEBUG com.zaxxer.hikari.HikariConfig - allowPoolSuspension.............false
DEBUG com.zaxxer.hikari.HikariConfig - autoCommit......................true
DEBUG com.zaxxer.hikari.HikariConfig - catalog.........................none
…
…
…
…
…
…
…
INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...
DEBUG com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - Before shutdown stats (total=1, active=0, idle=1, waiting=0)
DEBUG com.zaxxer.hikari.pool.PoolBase - HikariPool-1 - Closing connection conn0: url=jdbc:h2:mem:testdb user=SA: (connection evicted)
DEBUG com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - After shutdown stats (total=0, active=0, idle=0, waiting=0)
INFO  com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown completed.
```

TIA-Academy

ASSIGNMENT 00 (HOME ASSIGNMENT)

TIA-Academy

Sources

- https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm
- https://www.javatpoint.com/spring-boot-jdbc

TIA-Academy

Thank You