



## SPRING BOOT TUTORIALS



## About the Tutorial

---

Spring Boot is an open source Java-based framework used to create a Micro Service. It is developed by Pivotal Team. It is easy to create a stand-alone and production ready spring applications using Spring Boot. Spring Boot contains a comprehensive infrastructure support for developing a microservice and enables you to develop enterprise-ready applications that you can “**just run**”.

## Audience

---

This tutorial is designed for Java developers to understand and develop production-ready spring applications with minimum configurations. It explores major features of Spring Boot such as Starters, Auto-configuration, Beans, Actuator and more.

By the end of this tutorial, you will gain an intermediate level of expertise in Spring Boot.

## Prerequisites

---

This tutorial is written for readers who have a prior experience of Java, Spring and Maven. You can easily understand the concepts of Spring Boot if you have knowledge on these concepts. It would be an additional advantage if you have an idea about writing a RESTful Web Service. If you are a beginner, we suggest you to go through tutorials related to these concepts before you start with Spring Boot.

# 1. Spring Boot - Introduction

## 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.

### Advantages

Spring Boot offers the following advantages to its developers:

- Easy to understand and develop spring applications
- Increases productivity
- Reduces the development time

### Goals

Spring Boot is designed with the following goals:

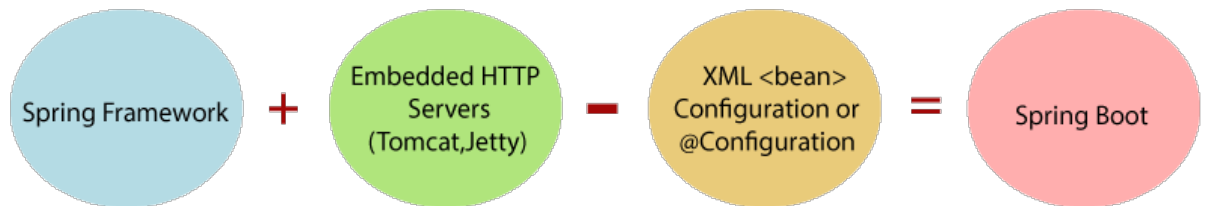
- To avoid complex XML configuration in Spring
- To develop a production ready Spring applications in an easier way
- To reduce the development time and run the application independently
- Offer an easier way of getting started with the application

## 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
- It is a Spring module that provides the **RAD (Rapid Application Development)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.



### 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.

### 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.

### Examples

Look at the following Spring Boot starters explained below for a better understanding:

**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>
```

**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>
```

**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 autoconfigures 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.

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) {
        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 **AutoConfiguration**, **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.

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);
    }
}
```

## 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;

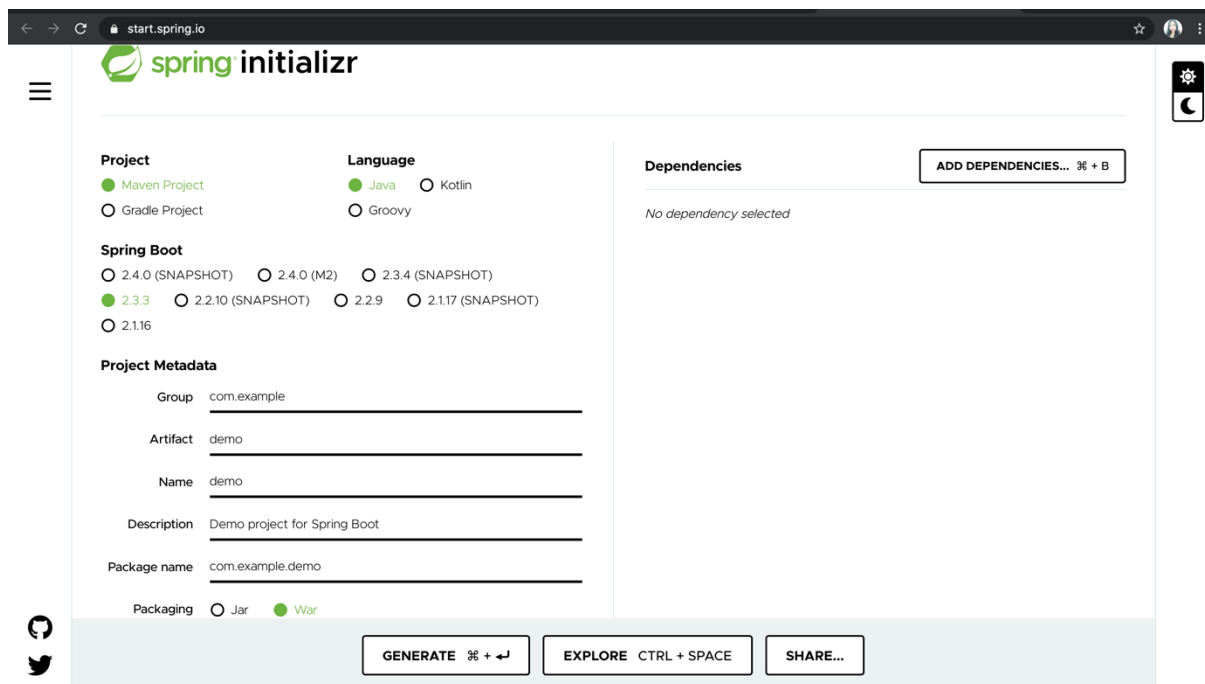
@ComponentScan
public class DemoApplication {

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



## 2. Spring Initializr

One of the ways to Bootstrapping a Spring Boot application is by using Spring Initializer. To do this, you will have to visit the Spring Initializer web page <http://start.spring.io/> and choose your Build, Spring Boot Version and platform. Also, you need to provide a Group, Artifact and required dependencies to run the application.



The screenshot shows the Spring Initializr web application interface. The browser address bar displays 'start.spring.io'. The page features a sidebar with a hamburger menu and a settings icon. The main content area is divided into three sections: 'Project', 'Language', and 'Dependencies'. The 'Project' section has radio buttons for 'Maven Project' (selected) and 'Gradle Project'. The 'Language' section has radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'. The 'Spring Boot' section has radio buttons for various versions, with '2.3.3' selected. The 'Project Metadata' section contains input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo). The 'Dependencies' section has a button 'ADD DEPENDENCIES... ⌘ + B' and the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE ⌘ + ↵', 'EXPLORE CTRL + SPACE', and 'SHARE...'. Social media icons for GitHub and Twitter are visible on the left side.

### 1. Generating a Project

Before creating a project, we must be friendly with UI. Spring Initializr UI has the following labels:

- **Project:** It defines the **kind** of project. We can create either **Maven Project** or **Gradle Project**. We will create a **Maven Project** throughout the tutorial.
- **Language:** Spring Initializr provides the choice among three languages **Java**, **Kotlin**, and **Groovy**. Java is by default selected.
- **Spring Boot:** We can select the Spring Boot **version**.
- **Project Metadata:** It contains information related to the project, such as **Group**, **Artifact**, etc. **Group** denotes the **package** name; **Artifact** denotes the **Application** name. The

default Group name is **com.example**, and the default Artifact name is **demo**.

- **Dependencies:** Dependencies are the collection of artifacts that we can add to our project.

There is another **Options** section that contains the following fields:

- **Name:** It is the same as **Artifact**.
- **Description:** In the description field, we can write a **description** of the project.
- **Package Name:** It is also similar to the **Group** name.
- **Packaging:** We can select the **packing** of the project. We can choose either **Jar** or **War**.
- **Java:** We can select the **JVM** version which we want to use. We will use **Java 8** version throughout the tutorial.

There is a **Generate** button. When we click on the button, it starts packing the project and downloads the **Jar** or **War** file, which you have selected.

## 2. How to Import zip file to Eclipse

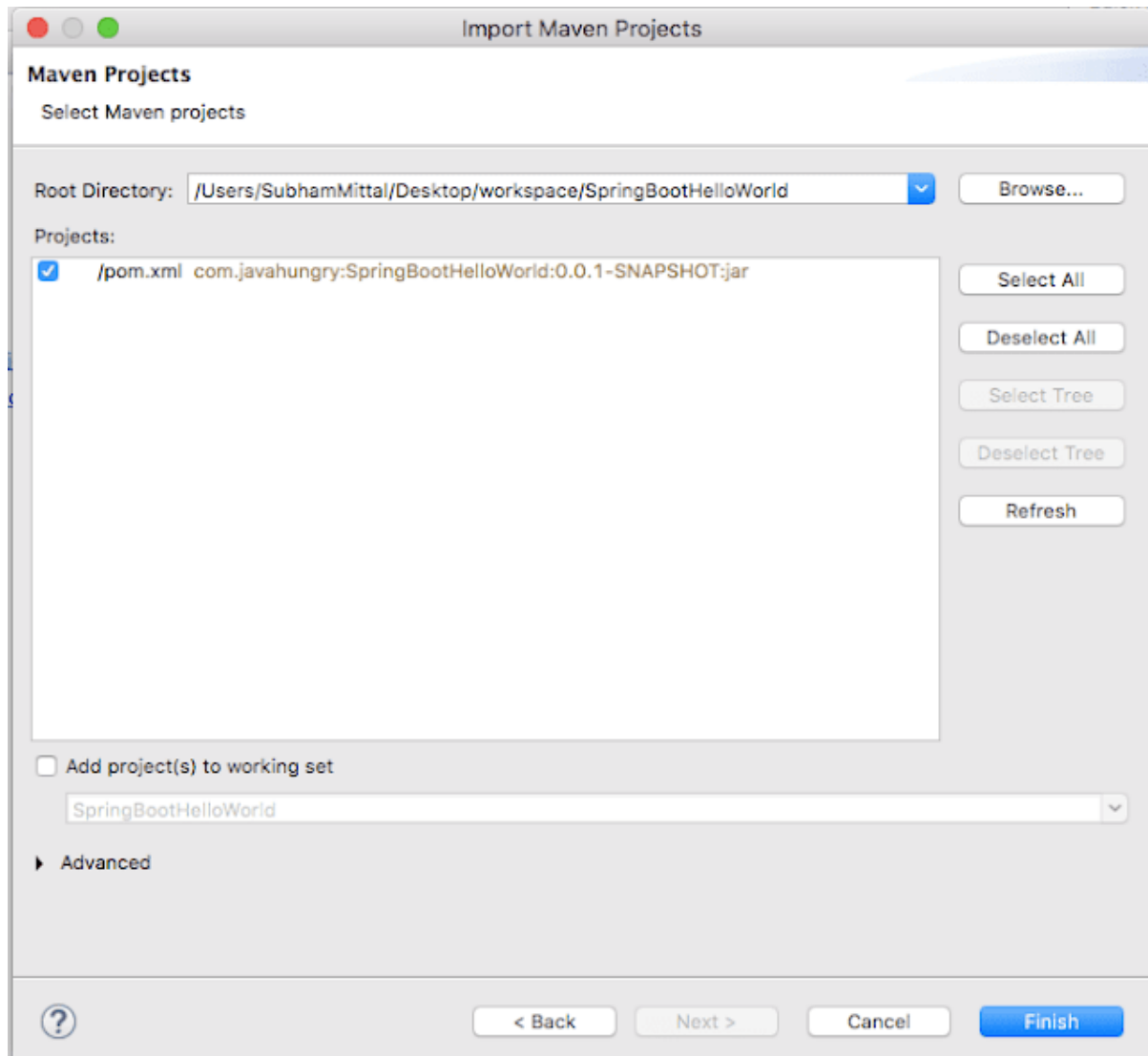
Import the Spring Boot project as an existing maven project.

Go to **File > Import > Maven**. Select **Existing Maven Projects** as shown below.

Click Next.

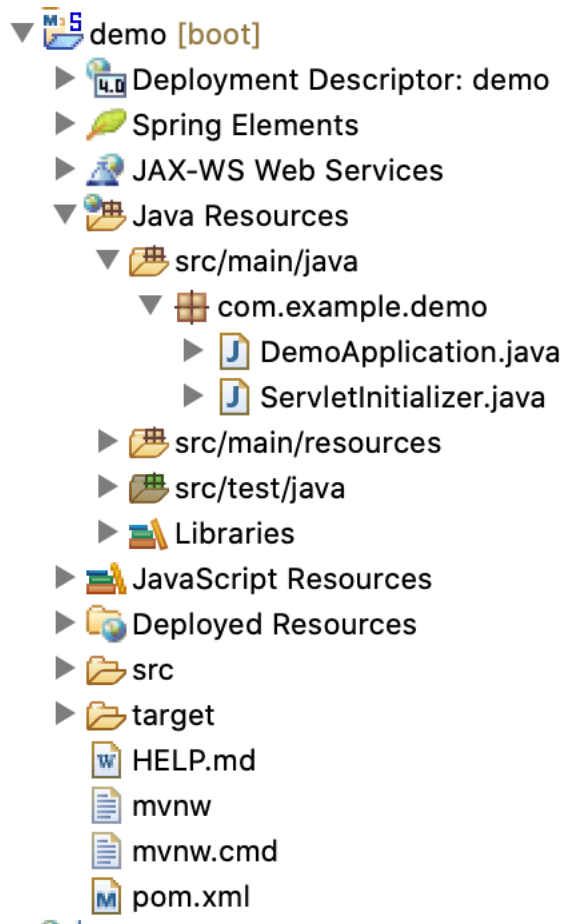


Select or Check the **pom.xml** file to import it. Click **Finish**.



**demo** project will be imported. Dependencies mentioned in the **pom.xml** will be automatically downloaded and added into the classpath.

**NOTE :** Folder structure after importing the project shown in the below image :



Congratulations! You have successfully imported the Spring Boot application into your IDE. Let's understand what it has already configured for you.

### 3. Pom.xml

You will find below code in the pom.xml

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.3.3.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Spring Boot provides various starters for creating Spring Boot based java application. The **spring-boot-starter-parent** is a special type of starter. It is used as a parent in the **pom.xml** file of any kind of Spring Boot application.

The **spring-boot-starter-parent** provides the common configurations such as plugin configuration, default java compiler level, dependency management, UTF-8 source encoding, etc.

There are other starters as well like **spring-boot-starter-web** for creating Spring web applications, **spring-boot-starter-web-services** is used for creating Spring web-services application.

#### 4. Spring Boot Application Class

##### DemoApplication.java

```
package com.example.demo;

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);
    }

}
```

**@SpringBootApplication** is equivalent to **@Configuration**, **@ComponentScan** and **@EnableAutoConfiguration** combined.

In other words :

**@SpringBootApplication = @Configuration + @ComponentScan + @EnableAutoConfiguration**

Let's understand about each annotation

**@EnableAutoConfiguration** : It tells Spring Boot to "guess" how you want to configure Spring, based on the jar dependencies that you have added.

For example, **spring-boot-starter-web** added Spring MVC and Tomcat, the auto-configuration assumes that you are developing a web application and sets up Spring accordingly.

**@Configuration** : It indicates that the class has @Bean definition methods by defining methods with the @Bean annotation. As a result, Spring container can process the class and produce Spring Beans to be used in the application

**@ComponentScan** : **@ComponentScan** without arguments indicates Spring to scan the current package and all of its sub-packages.

## 5. Create the controller

Create a new Controller java class named **HelloWorldController.java** under package **com.example.demo.controller**. Add the following code into it.

### HelloWorldController.java

```
package com.example.deom.controller;

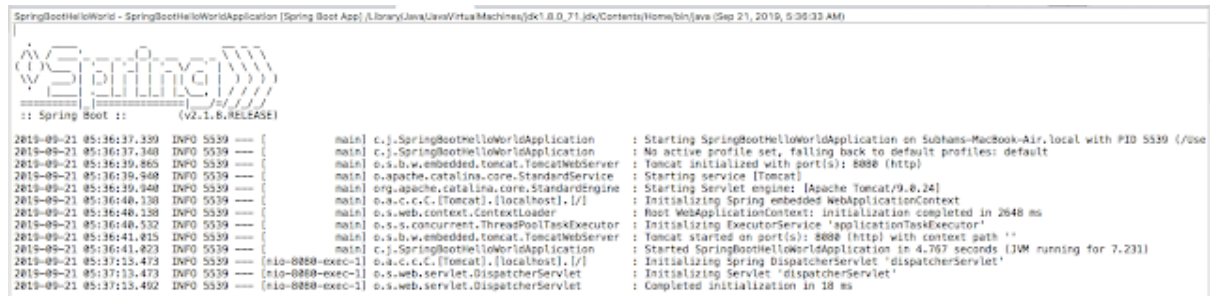
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorldController {

    @GetMapping(path="/")
    public String sayHello() {
        return "Hello World ";
    }
}
```

## 6. Run Application

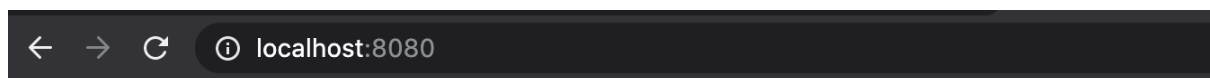
Run the **DemoApplication.java** by Run As Java Application. After executing the main() method, your console will look like as shown below. Application by default will start the embedded tomcat server on port 8080.



```
SpringBootHelloWorld - SpringBootHelloWorldApplication [Spring Boot App] [Library\Java\VirtualMachines\jdk1.8.0_71_jdk\Contents\Home\bin\java (Sep 21, 2019, 5:36:32 AM)]
:: Spring Boot ::
(v2.1.6.RELEASE)

2019-09-21 05:36:37.339 INFO 5539 --- [main] c.j.SpringBootHelloWorldApplication : Starting SpringBootHelloWorldApplication on Subhans-MacBook-Air.local with PID 5539 (/usr
2019-09-21 05:36:37.348 INFO 5539 --- [main] c.j.SpringBootHelloWorldApplication : No active profile set, falling back to default profiles: default
2019-09-21 05:36:39.865 INFO 5539 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2019-09-21 05:36:39.940 INFO 5539 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2019-09-21 05:36:39.940 INFO 5539 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.24]
2019-09-21 05:36:40.138 INFO 5539 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2019-09-21 05:36:40.138 INFO 5539 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2648 ms
2019-09-21 05:36:40.532 INFO 5539 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2019-09-21 05:36:41.815 INFO 5539 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2019-09-21 05:36:41.823 INFO 5539 --- [main] c.j.SpringBootHelloWorldApplication : Started SpringBootHelloWorldApplication in 4.767 seconds (JVM running for 7.233)
2019-09-21 05:37:12.473 INFO 5539 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2019-09-21 05:37:12.492 INFO 5539 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2019-09-21 05:37:12.492 INFO 5539 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 18 ms
```

Now enter **http://localhost:8080** in your browser and see the output.



Hello World

**Congratulations!** You just run your first Spring Boot Application.

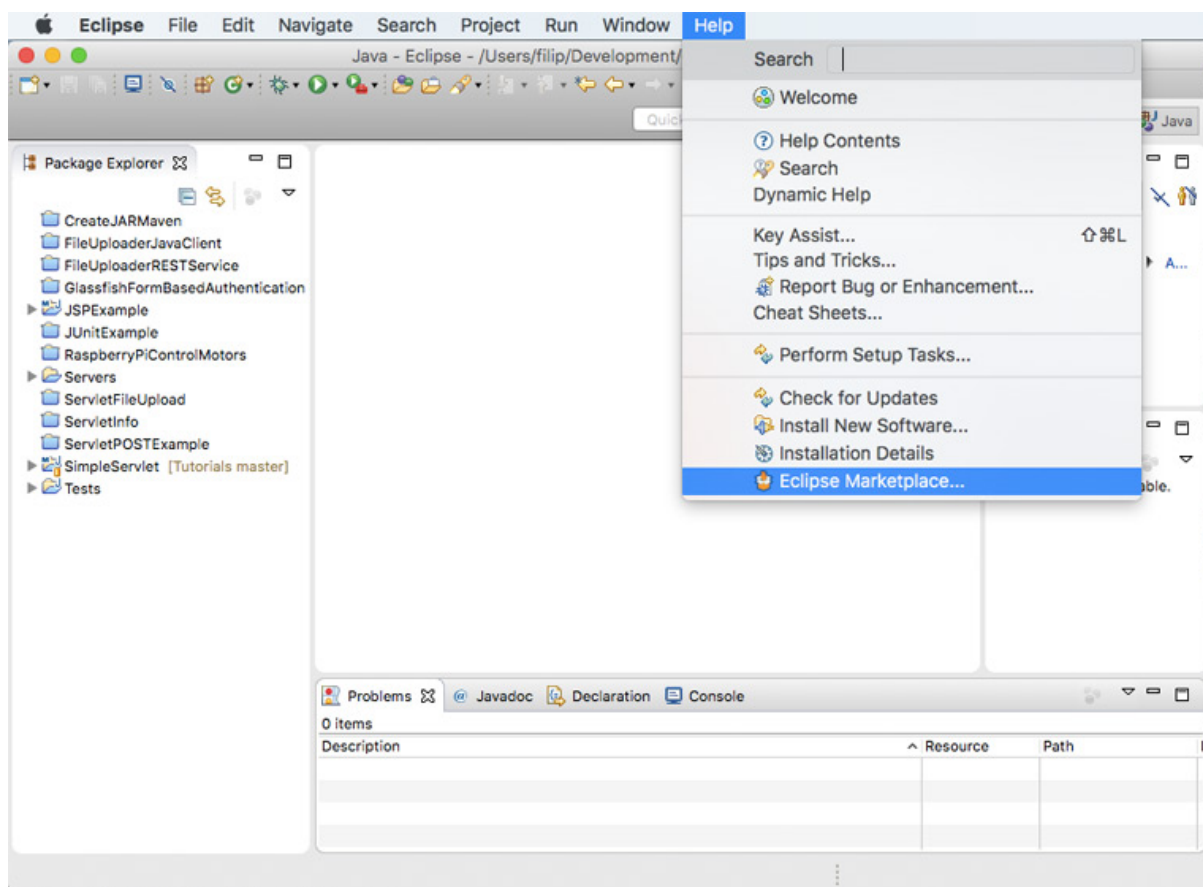


### 3. Spring Tool Suite

STS stands for **Spring Tool Suite**. It provides ready-to-use environment to implement, run, debug and deploy Spring application with Eclipse [IDE](#). It is a powerful environment which will help you make Spring development faster and easier.

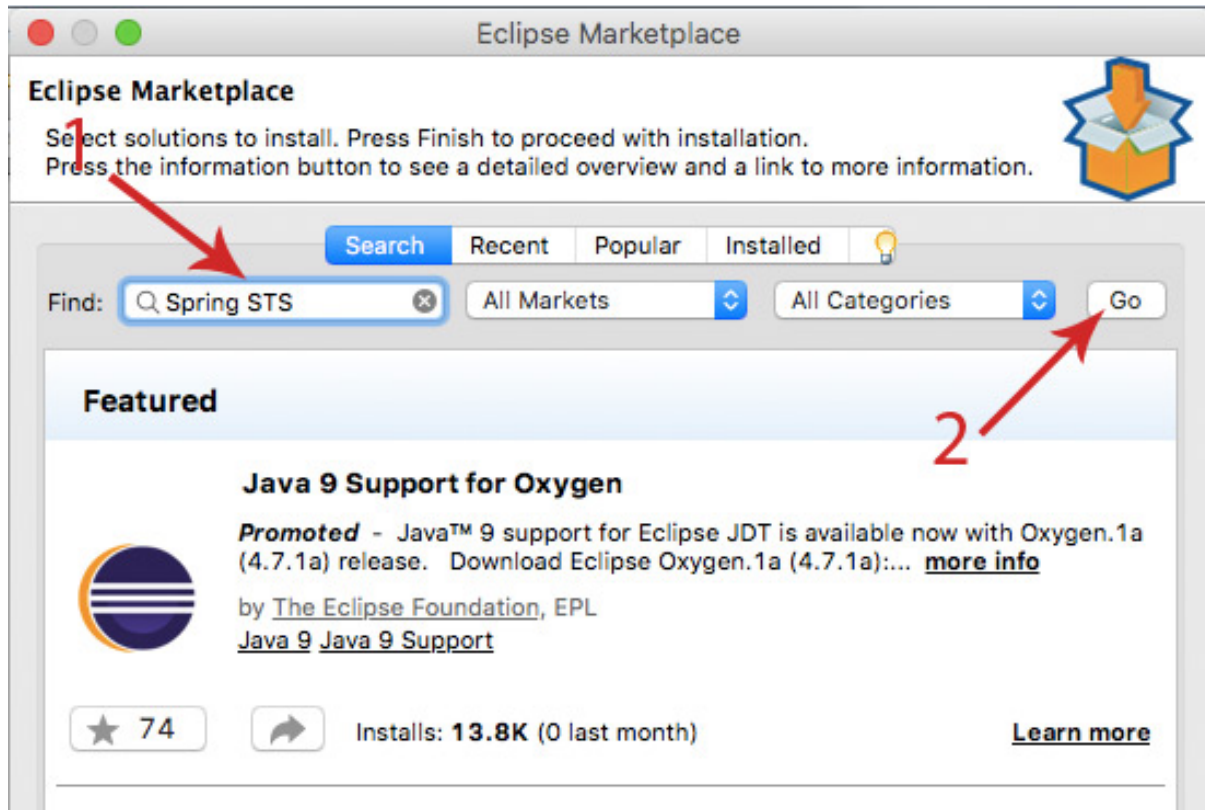
#### How to Install Spring Tool Suite in Eclipse

1. Open your Eclipse IDE
2. Go to "Help" and select "Eclipse Marketplace..."



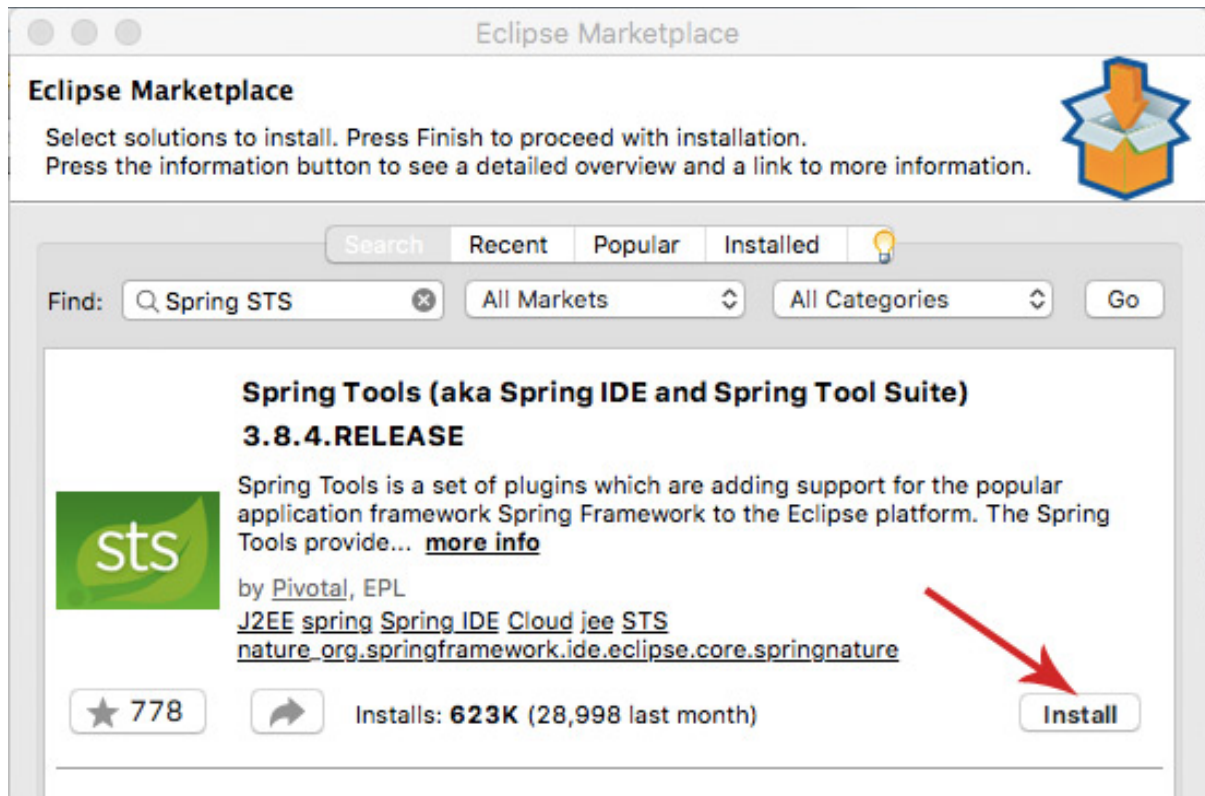
Eclipse open marketplace

3. In Eclipse Marketplace window type "Spring STS" in "Find:" field and click the "Go" button



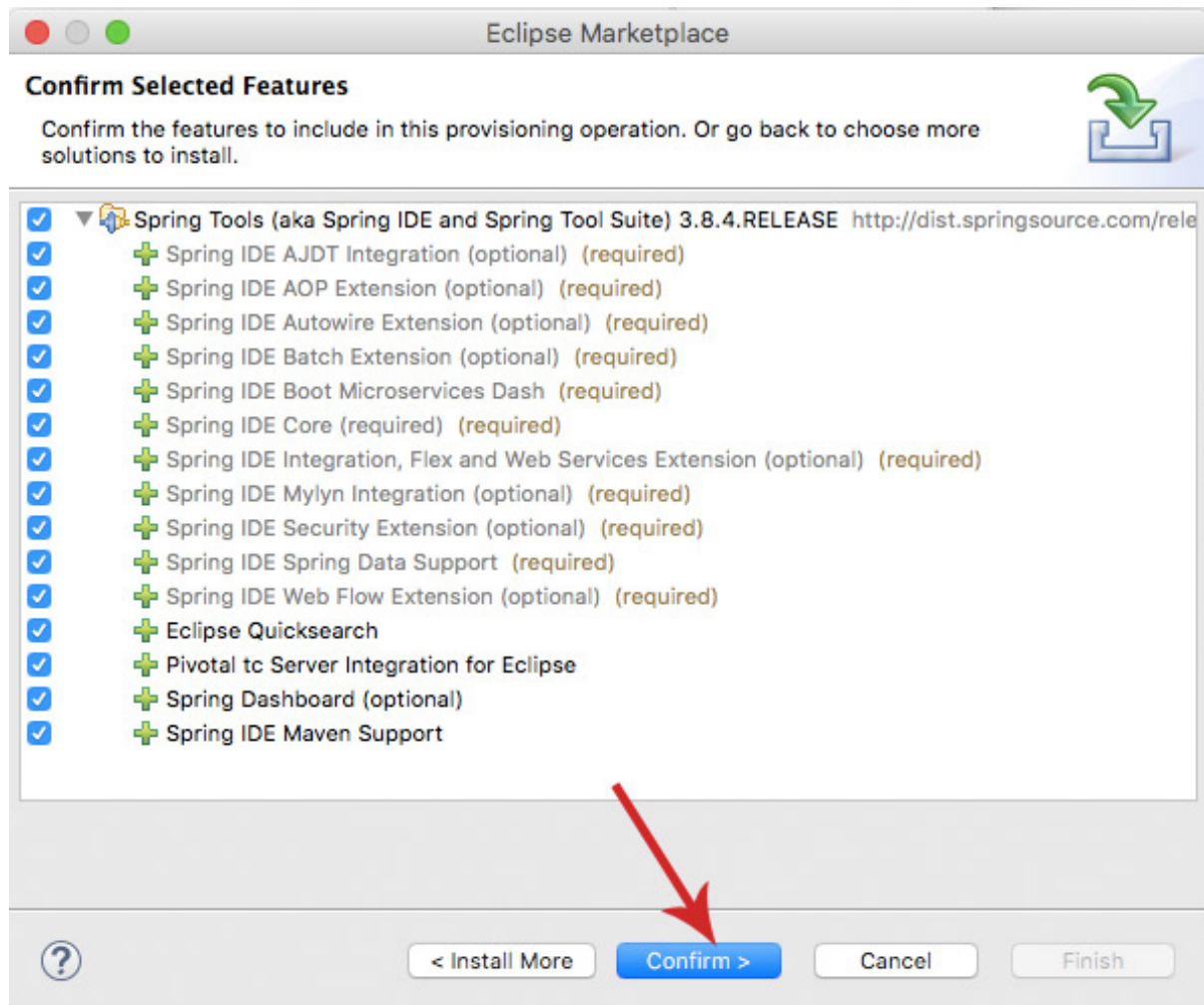
Search for Spring STS

4. In the list find "Spring Tool Suite" and click the "Install" button. The most current version in time of writing this guide is 3.8.4.RELEASE



Install Spring Tool Suite

5. On the next screen select desired features and press the “Confirm” button



STS features selection

6. Accept terms and license agreements

7. Wait for the software to be installed

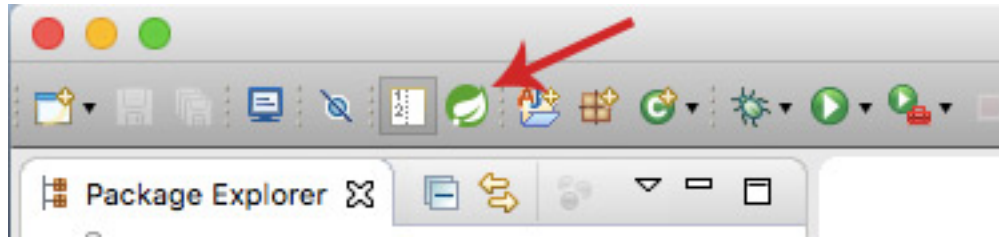
8. After installation is done you will be prompted to restart Eclipse for the changes to take effect

### Confirm Successful Installation

If the installation goes well and you have selected “Dashboard” in the features list (see step 5) you will see a new icon in your Eclipse tool-bar

You can also may want to create a simple App to test if everything is configured fine.

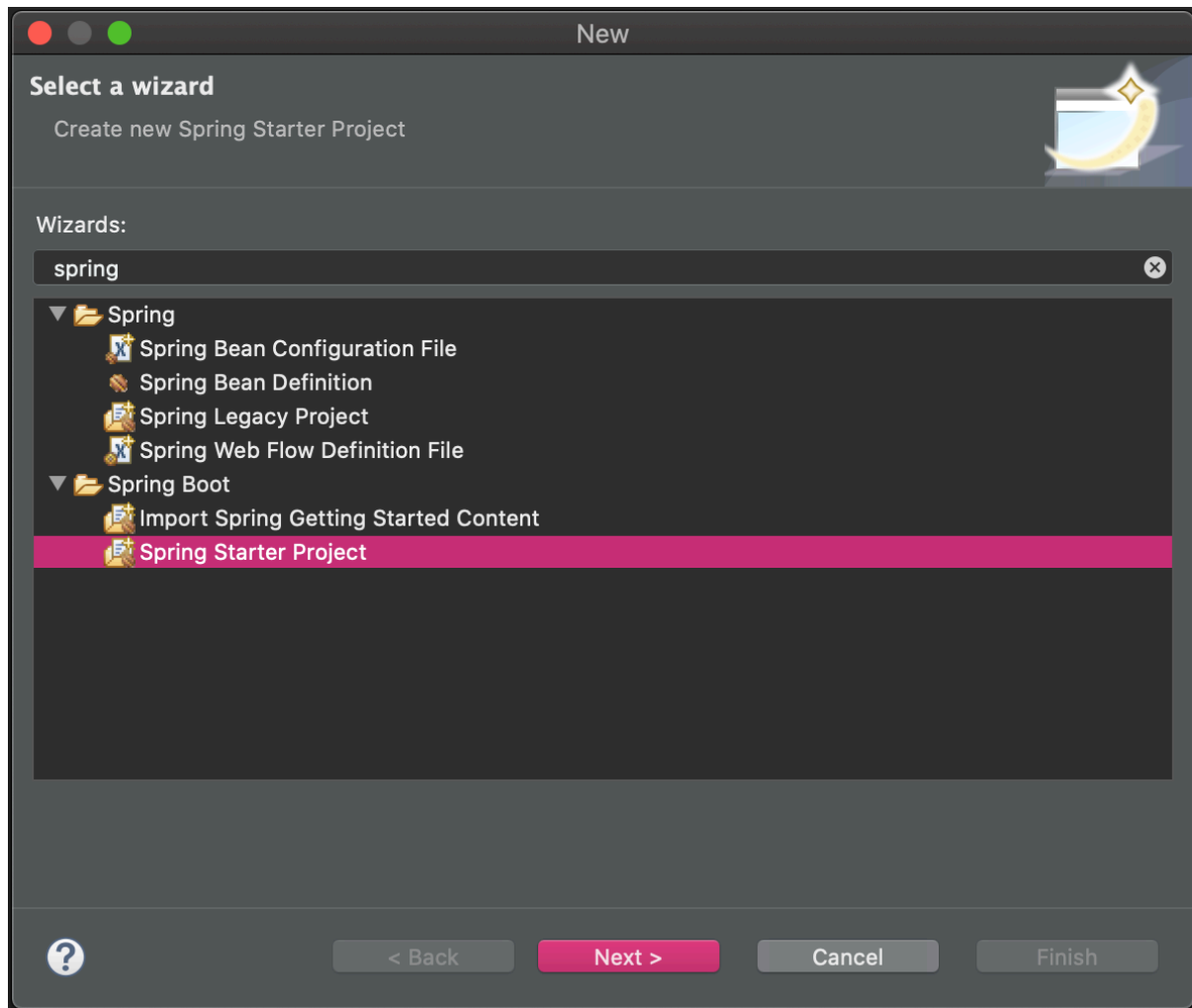
# Spring Boot Tutorial



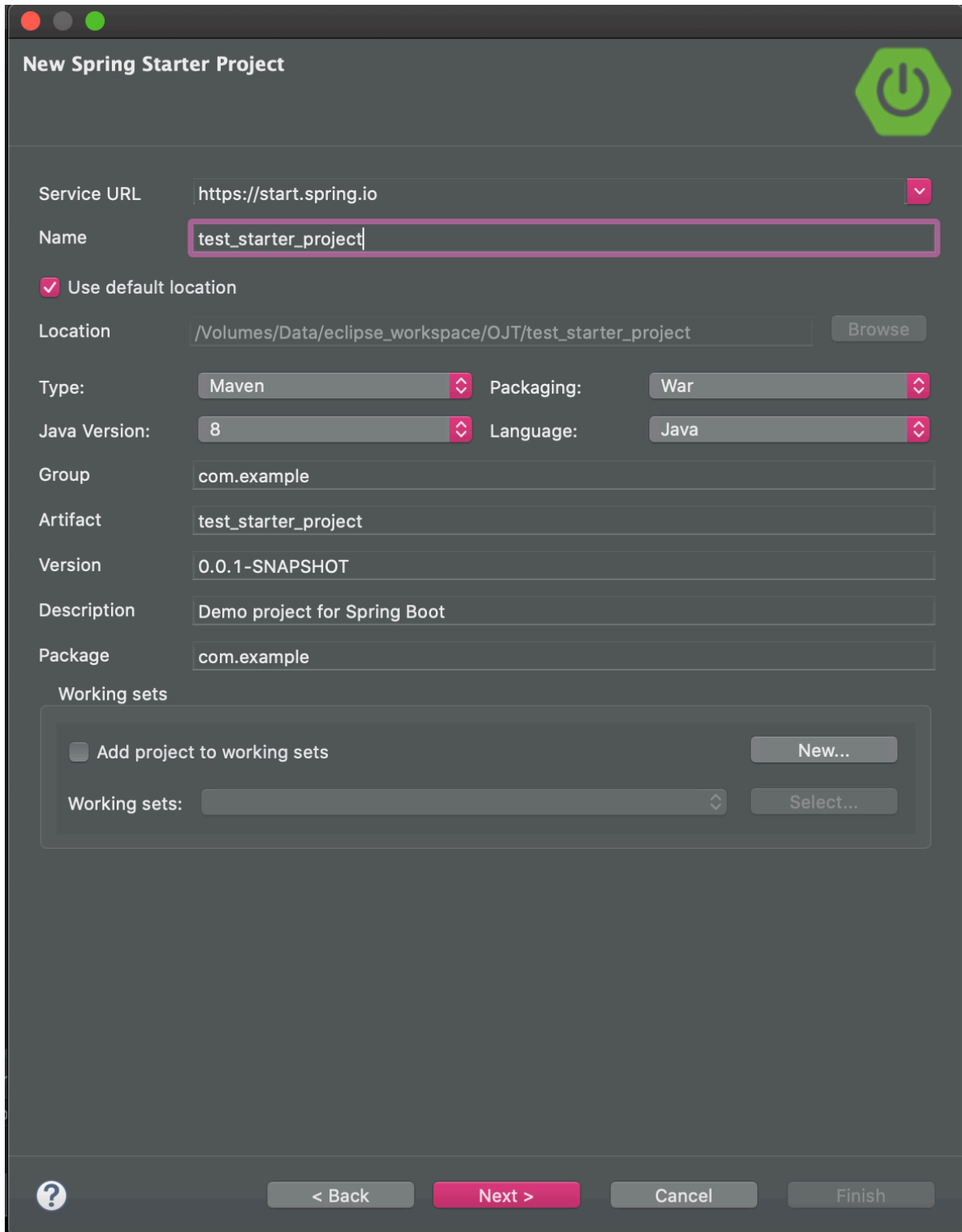
## 4. Create Spring Starter Project in Eclipse

Go to **File > New > Other**. Type **Spring** as shown below. And then choose **Spring Starter Project**.

Click Next.



Fill the following picture. Click Next.



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

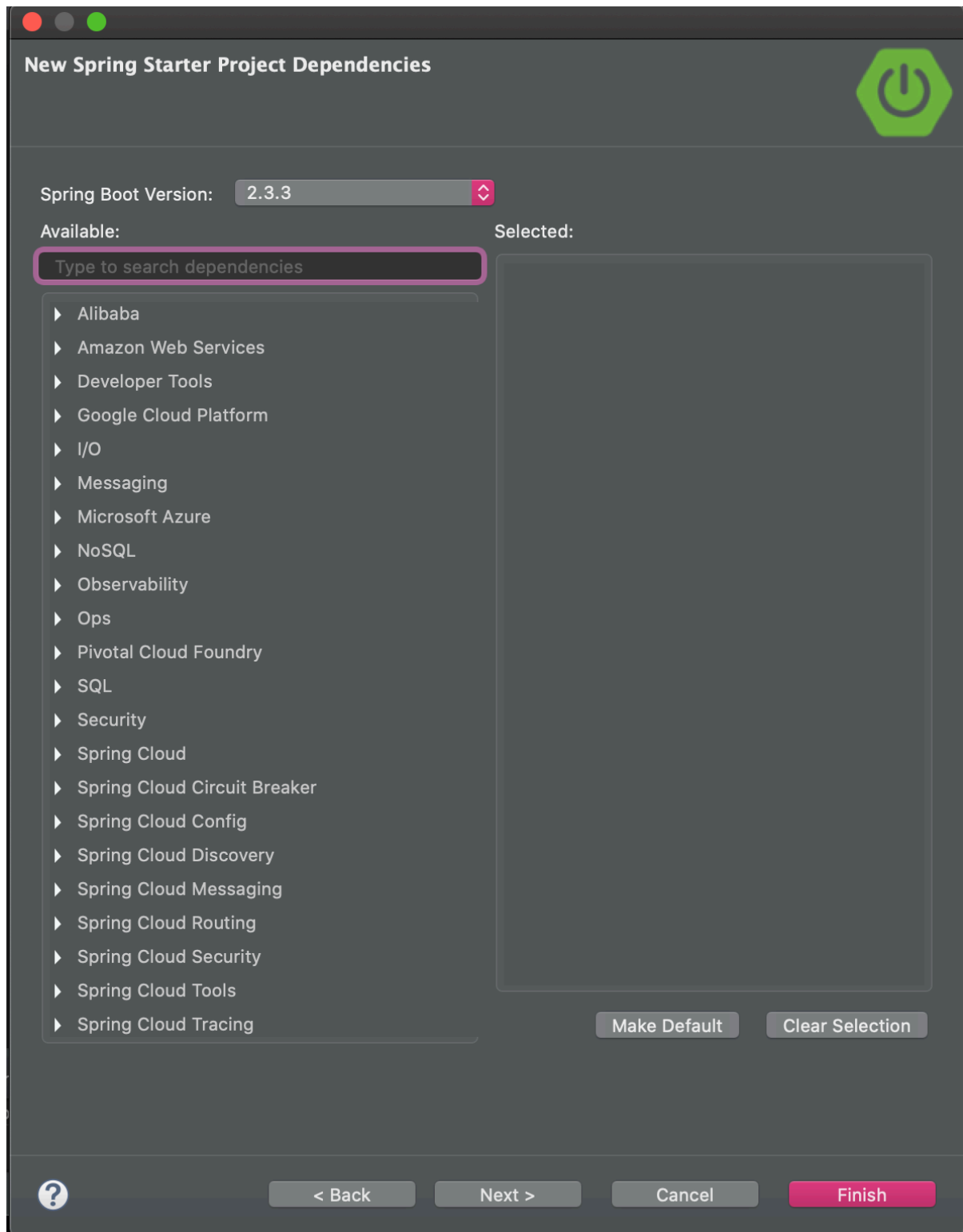
☐ Add project to working sets

Working sets:

? < Back Next > Cancel Finish

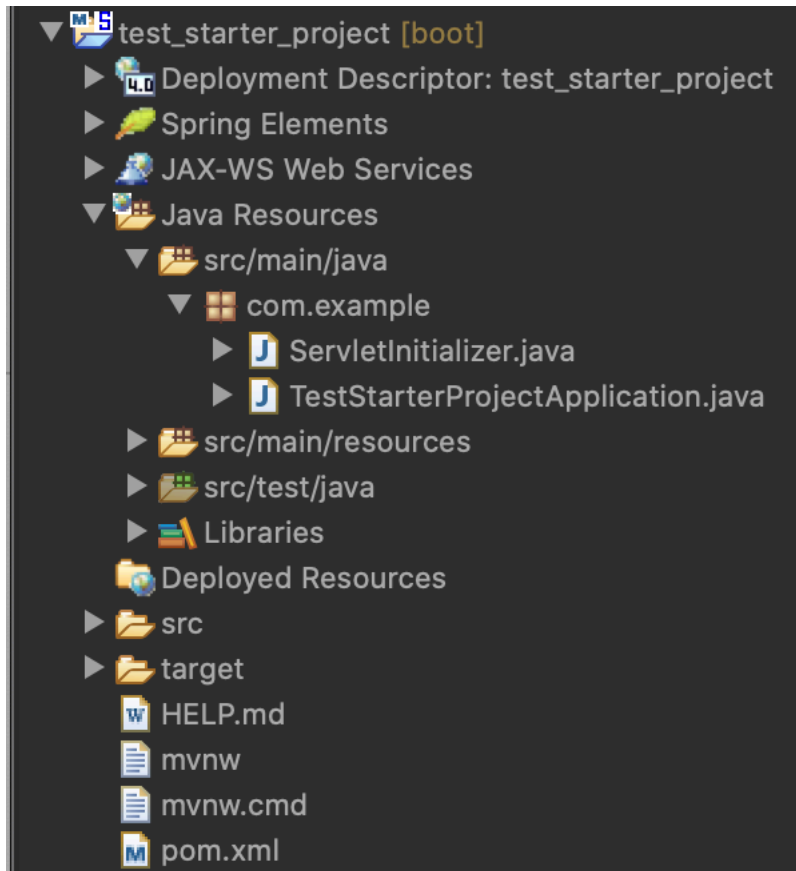
If you want to add other Dependencies eg. (Spring Data JPA, Thymeleaf, JDBC API), you can choose.

Click Finish.





Congratulations! You have successfully created the Spring Boot application in your IDE. Let's understand what it has already configured for you.



### TestStarterProjectApplication.java

```
package com.example;

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

@SpringBootApplication
public class TestStarterProjectApplication {

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

}
```

Create a new Controller java class named **HelloWorldController.java** under package **com.example.controller**. Add the following code into it.

### HelloWorldController.java


```
package com.example.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorldController {

    @GetMapping(path="/")
    public String sayHello() {
        return "Hello World ";
    }
}
```

Run the **TestStarterProjectApplication.java** by Run As Java Application. After executing the main() method, your console will look like as shown below. Application by default will start the embedded tomcat server on port 8080.



```
SpringBootHelloWorld - SpringBootHelloWorldApplication [Spring Boot App] (/Library/Java/JavaVirtualMachines/jdk1.8.0_71_jdk/Contents/Home/bin/java) (Sep 21, 2016, 5:35:33 AM)
:: Spring Boot :: (v2.1.6.RELEASE)

2016-09-21 05:35:37.359 INFO 5539 --- [main] c.j.SpringBootHelloWorldApplication : Starting SpringBootHelloWorldApplication on Subhans-MacBook-Air.local with PID 5539 (/usr/bin/java)
2016-09-21 05:35:37.360 INFO 5539 --- [main] c.j.SpringBootHelloWorldApplication : No active profile set, falling back to default profiles: default
2016-09-21 05:35:39.865 INFO 5539 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2016-09-21 05:35:39.948 INFO 5539 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2016-09-21 05:35:39.948 INFO 5539 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.24]
2016-09-21 05:35:40.130 INFO 5539 --- [main] o.a.c.c.C.[Tomcat].[/] : Initializing Spring embedded WebApplicationContext
2016-09-21 05:35:40.138 INFO 5539 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2648 ms
2016-09-21 05:35:40.532 INFO 5539 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2016-09-21 05:35:41.015 INFO 5539 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2016-09-21 05:35:41.823 INFO 5539 --- [main] c.j.SpringBootHelloWorldApplication : Started SpringBootHelloWorldApplication in 4.767 seconds (JVM running for 7.231)
2016-09-21 05:37:13.473 INFO 5539 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2016-09-21 05:37:13.473 INFO 5539 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2016-09-21 05:37:13.492 INFO 5539 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 18 ms
```

Now enter **http://localhost:8080** in your browser and see the output.



**Congratulations!** You just run your second Spring Boot Application.

## 5. Spring Boot - Annotations

**1. @Required:** It applies to the **bean** setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception **BeanInitializationException**.

### Example

```
public class Machine
{
    private Integer cost;
    @Required
    public void setCost(Integer cost)
    {
        this.cost = cost;
    }
    public Integer getCost()
    {
        return cost;
    }
}
```

**2.@Autowired:** Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

### Example

```
@Component
public class Customer
{
    private Person person;
    @Autowired
    public Customer(Person person)
    {
        this.person=person;
    }
}
```

**3.@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

### Example

```
@Configuration
public class Vehicle
{
    @BeanVehicle engine()
    {
        return new Vehicle();
    }
}
```

**4.@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

### Example

```
@ComponentScan(basePackages = "com.javatpoint")
@Configuration
public class ScanComponent
{
    // ...
}
```

**5.@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

### Example

```
@Bean
public BeanExample beanExample()
{
    return new BeanExample ();
}
```

**6.@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.

### Example

```
@Component
public class Student
{
.....
}
```

**7.@Controller:** The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.

**8.@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces,** and **value**. We use it with the class as well as the method.

### Example

```
@Controller
@RequestMapping("books")
public class BooksController
{
@RequestMapping(value =("/{name})", method = RequestMethod.GET)
public Employee getBooksByName()
{
return booksTemplate;
}
}
```

**9.@Service:** It is also used at class level. It tells the Spring that class contains the **business logic**.

### Example

```
package com.javatpoint;
@Service
public class TestService
{
    public void service1()
    {
        //business code
    }
}
```

**10.@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

```
package com.javatpoint;
@Repository
public class TestRepository
{
    public void delete()
    {
        //persistence code
    }
}
```

**11.@EnableAutoConfiguration:** It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. **@SpringBootApplication**

**12. @SpringBootApplication:** It is a combination of three annotations **@EnableAutoConfiguration**, **@ComponentScan**, and **@Configuration**.

## 6. Spring Boot – Thymeleaf – JDBC API

### What is Thymeleaf?

The **Thymeleaf** is an open-source Java library that is licensed under the **Apache License 2.0**. It is a **HTML5/XHTML/XML** template engine. It is a **server-side Java template** engine for both web (servlet-based) and non-web (offline) environments. It is perfect for modern-day HTML5 JVM web development. It provides full integration with Spring Framework. It applies a set of transformations to template files in order to display data or text produced by the application. It is appropriate for serving XHTML/HTML5 in web applications.

The goal of Thymeleaf is to provide a **stylish** and **well-formed** way of creating templates. It is based on XML tags and attributes. These XML tags define the execution of predefined logic on the DOM (Document Object Model) instead of explicitly writing that logic as code inside the template. It is a substitute for **JSP**.

The architecture of Thymeleaf allows the **fast processing** of templates that depends on the caching of parsed files. It uses the least possible amount of I/O operations during execution.

### Why we use Thymeleaf?

JSP is more or less similar to HTML. But it is not completely compatible with HTML like Thymeleaf. We can open and display a Thymeleaf template file normally in the browser while the JSP file does not.

Thymeleaf supports variable expressions (`${...}`) like Spring EL and executes on model attributes, asterisk expressions (`*{...}`) execute on the form backing bean, hash expressions (`#{...}`) are for internationalization, and link expressions (`@{...}`) rewrite URLs.

Like JSP, Thymeleaf works well for Rich HTML emails.

### Thymeleaf Implementation

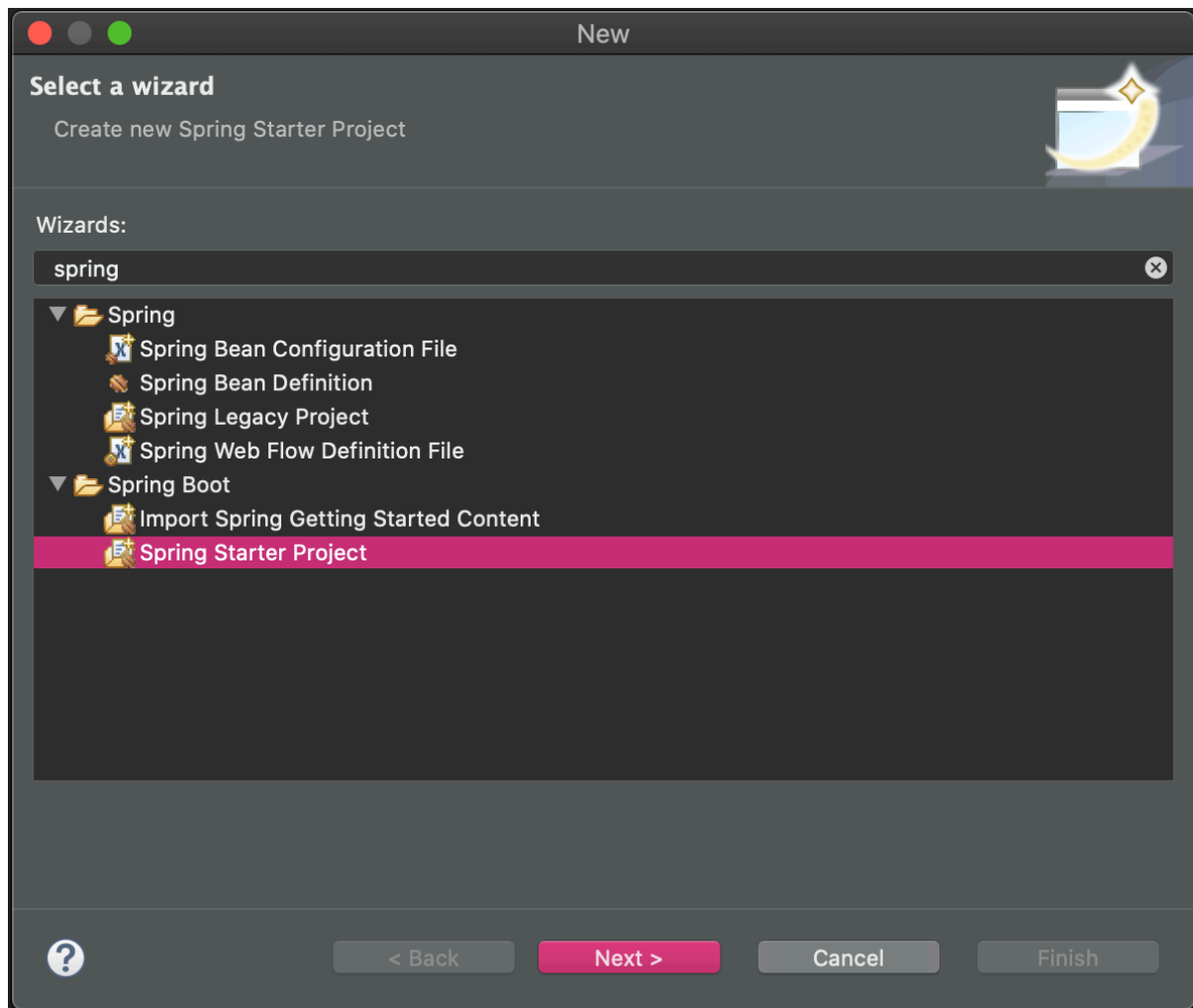
We can implement Thymeleaf template engine by adding **spring-boot-starter-thymeleaf** dependency in our application's pom.xml file. Spring Boot configures template engine to read template file from **/resource/templates**.

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



Go to **File > New > Other**. Type **Spring** as shown below. And then choose **Spring Starter Project**.

Click Next.



Fill the following picture. Click Next.

**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

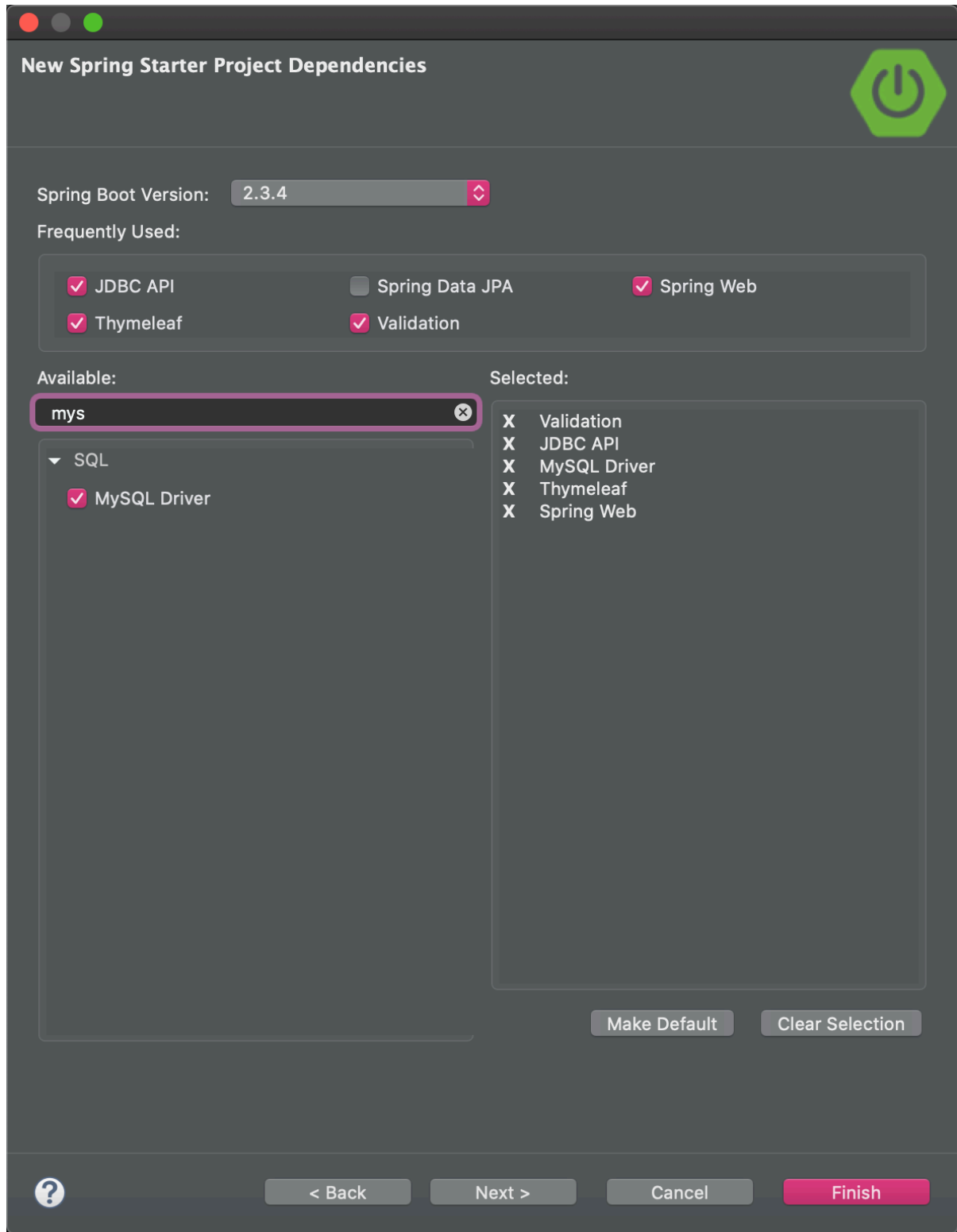
Package:

Working sets

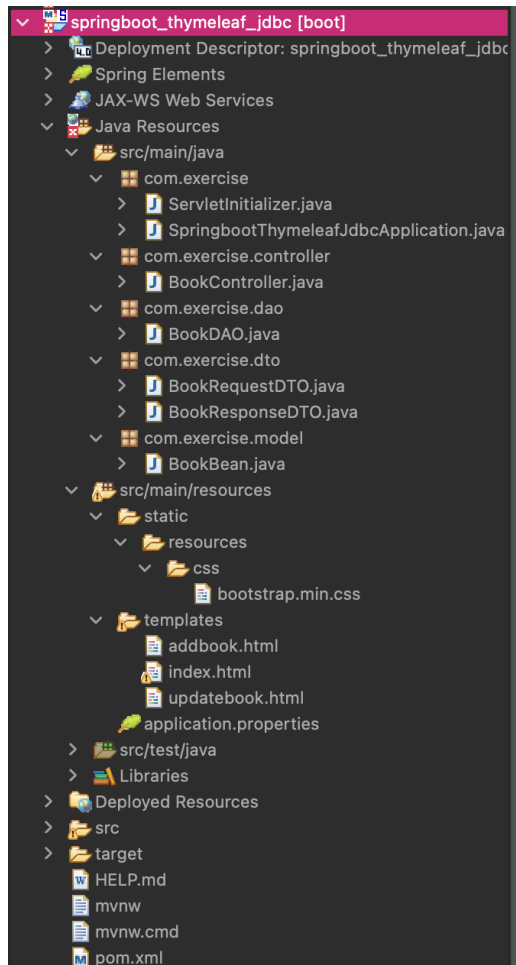
☐ Add project to working sets

Working sets:

Check **Spring Web, JDBC API, Thymeleaf, MySQL Driver, Validation** CheckBox.



Congratulations! You have successfully created the Spring Boot application in your IDE. Let's understand what it has already configured for you.



### SpringbootThymeleafJdbcApplication.java

```
package com.example;

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

@SpringBootApplication
public class SpringbootThymeleafJdbcApplication {

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

}
```

**BookBean.java**

```
package com.exercise.model;

import javax.validation.constraints.NotEmpty;

public class BookBean {
    @NotEmpty
    private String bookCode;
    @NotEmpty
    private String bookTitle;
    @NotEmpty
    private String bookAuthor;
    @NotEmpty
    private String bookPrice;
    public String getBookCode() {
        return bookCode;
    }
    public void setBookCode(String bookCode) {
        this.bookCode = bookCode;
    }
    public String getBookTitle() {
        return bookTitle;
    }
    public void setBookTitle(String bookTitle) {
        this.bookTitle = bookTitle;
    }
    public String getBookAuthor() {
        return bookAuthor;
    }
    public void setBookAuthor(String bookAuthor) {
        this.bookAuthor = bookAuthor;
    }
    public String getBookPrice() {
        return bookPrice;
    }
    public void setBookPrice(String bookPrice) {
        this.bookPrice = bookPrice;
    }
}
```

**BookRequestDTO.java**

```
package com.exercise.dto;

public class BookRequestDTO {
    private String bookCode;
    private String bookTitle;
    private String bookAuthor;
    private String bookPrice;
    public String getBookCode() {
        return bookCode;
    }
    public void setBookCode(String bookCode) {
        this.bookCode = bookCode;
    }
    public String getBookTitle() {
        return bookTitle;
    }
    public void setBookTitle(String bookTitle) {
        this.bookTitle = bookTitle;
    }
    public String getBookAuthor() {
        return bookAuthor;
    }
    public void setBookAuthor(String bookAuthor) {
        this.bookAuthor = bookAuthor;
    }
    public String getBookPrice() {
        return bookPrice;
    }
    public void setBookPrice(String bookPrice) {
        this.bookPrice = bookPrice;
    }
}
```

**BookResponseDTO.java**

```
package com.exercise.dto;

public class BookResponseDTO {
    private String bookCode;
    private String bookTitle;
    private String bookAuthor;
    private String bookPrice;
    public String getBookCode() {
        return bookCode;
    }
    public void setBookCode(String bookCode) {
        this.bookCode = bookCode;
    }
    public String getBookTitle() {
        return bookTitle;
    }
    public void setBookTitle(String bookTitle) {
        this.bookTitle = bookTitle;
    }
    public String getBookAuthor() {
        return bookAuthor;
    }
    public void setBookAuthor(String bookAuthor) {
        this.bookAuthor = bookAuthor;
    }
    public String getBookPrice() {
        return bookPrice;
    }
    public void setBookPrice(String bookPrice) {
        this.bookPrice = bookPrice;
    }
}
```

**BookDAO.java**

```

package com.exercise.dao;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import com.exercise.dto.BookRequestDTO;
import com.exercise.dto.BookResponseDTO;

@Repository
public class BookDAO {
    @Autowired
    JdbcTemplate jdbcTemplate;

    public int insertData(BookRequestDTO dto) {
        int result=0;
        String sql="insert into
book(book_code,book_title,book_author,book_price) values
(?,?,?,?)";
        result=jdbcTemplate.update(sql,
dto.getBookCode(),dto.getBookTitle(),dto.getBookAuthor(),dto.g
etBookPrice());
        return result;
    }

    public int updateData(BookRequestDTO dto) {
        int result=0;
        String sql="update book set
book_title=?,book_author=?,book_price=?"
+ "where book_code=?";
        result=jdbcTemplate.update(sql,
dto.getBookTitle(),dto.getBookAuthor(),dto.getBookPrice(),dto.
getBookCode());
        return result;
    }

    public int deleteData(BookRequestDTO dto) {
        int result=0;
        String sql="delete from book where book_code=?";
        result=jdbcTemplate.update(sql, dto.getBookCode());
        return result;
    }

    public BookResponseDTO selectOne(BookRequestDTO dto) {
        String sql="select * from book where book_code=?";
        return jdbcTemplate.queryForObject(
            sql,
            (rs, rowNum) ->

```



```

        new BookResponseDTO(
            rs.getString("book_code"),
rs.getString("book_title"),
rs.getString("book_author"),
rs.getDouble("book_price")),
        dto.getBookCode()
    );

    }

    public List<BookResponseDTO>selectAll(){

        String sql="select * from book";
        return jdbcTemplate.query(
            sql,
            (rs, rowNum) ->
                new BookResponseDTO(
                    rs.getString("book_code"),
                    rs.getString("book_title"),
                    rs.getString("book_author"),
                    rs.getDouble("book_price")
                )
            );
    }

}

```

### BookController.java

```

package com.exercise.controller;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

```

```

import com.exercise.model.BookBean;
import com.exercise.dao.BookDAO;
import com.exercise.dto.BookRequestDTO;
import com.exercise.dto.BookResponseDTO;

@Controller
public class BookController {
    @Autowired
    private BookDAO bookDao;

    @ModelAttribute("bookBean")
    public BookBean getBookBean() {
        return new BookBean();
    }

    @RequestMapping(value="/", method=RequestMethod.GET)
    public String displayView(ModelMap model) {
        List<BookResponseDTO> list=bookDao.selectAll();
        model.addAttribute("list", list);
        return "index";
    }

    @RequestMapping(value="/setupaddbook",
method=RequestMethod.GET)
    public ModelAndView setupaddbook() {
        return new ModelAndView("addbook","bookBean",new
BookBean());
    }

    @RequestMapping(value="/addbook",
method=RequestMethod.POST)
    public String addbook(@ModelAttribute("bookBean")
@Validated BookBean bean,
        BindingResult bs, ModelMap model) {
        if(bs.hasErrors()) {
            return "addbook";
        }
        BookRequestDTO dto=new BookRequestDTO();
        dto.setBookCode(bean.getBookCode());
        dto.setBookTitle(bean.getBookTitle());
        dto.setBookAuthor(bean.getBookAuthor());

        dto.setBookPrice(Double.valueOf(bean.getBookPrice()));
        int rs=bookDao.insertData(dto);
        if(rs==0) {
            model.addAttribute("error","Insert Failed");
            return "addbook";
        }
        return "redirect:/";
    }
}

```

```

    }

    @RequestMapping(value="/setupUpdateBook",
method=RequestMethod.GET)
    public ModelAndView setupUpdatebook(@RequestParam
("code") String bookCode) {
        BookRequestDTO dto=new BookRequestDTO();
        dto.setBookCode(bookCode);
        return new
ModelAndView("updatebook","bookBean",bookDao.selectOne(dto));
    }

    @RequestMapping(value="/updatebook",
method=RequestMethod.POST)
    public String updatebook(@ModelAttribute("bookBean")
@Validated BookBean bean,
        BindingResult bs, ModelMap model) {
        if(bs.hasErrors()) {
            return "updatebook";
        }
        BookRequestDTO dto=new BookRequestDTO();
        dto.setBookCode(bean.getBookCode());
        dto.setBookTitle(bean.getBookTitle());
        dto.setBookAuthor(bean.getBookAuthor());

        dto.setBookPrice(Double.valueOf(bean.getBookPrice()));
        int rs=bookDao.updateData(dto);
        if(rs==0) {
            model.addAttribute("error","Update Failed");
            return "updatebook";
        }
        return "redirect:/";
    }

    @RequestMapping(value="/deleteBook",
method=RequestMethod.GET)
    public String deleteBook(@RequestParam ("code") String
bookCode,ModelMap model) {
        BookRequestDTO dto=new BookRequestDTO();
        dto.setBookCode(bookCode);
        int res=bookDao.deleteData(dto);
        if(res==0) {
            model.addAttribute("error","Delete Failed");
        }
        return "redirect:/";
    }
}

```

**index.html**

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>List All Book</title>
    <link href="/resources/css/bootstrap.min.css"
rel="stylesheet"/>
    <style type="text/css">
        table {
            border-collapse: collapse;
            width: 100%;
        }
        th, td {
            text-align: left;
            padding: 8px;
        }

        tr:nth-child(even){background-color: #f2f2f2}
        th {
            background-color: #4CAF50;
            color: white;
        }
    </style>
</head>
<body>

        <center>
            <h2>List All Book</h2>
        </center>
        <table class="table table-striped table-hover
table-bordered">
            <tr>
                <th>Book Code</th>
                <th>Book Title</th>
                <th>Book Author</th>
                <th>Book Price</th>
                <th>Action</th>
            <tr th:each="book: ${list}">
                <td th:text="${book.bookCode}" />
                <td th:text="${book.bookTitle}" />
                <td th:text="${book.bookAuthor}" />
                <td align="right" th:text="${book.bookPrice}" />
                <td>
                    <a
th:href="@{/setupUpdateBook(code=${book.bookCode})}">Update</a>
                    <a
th:href="@{/deleteBook(code=${book.bookCode})}">Delete</a>

```

```

        </td>

</tr>
</table>
        <center><a href="setupaddbook"><input
type="submit" value="Create New Book" class="btn btn-
success"></a> </center>

</body>
</html>

```

## addbook.html

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
    <title>Create New Book Form</title>

<link th:href="@{/resources/css/bootstrap.min.css}"
rel="stylesheet" />

    </head>
    <body>
        <form id="form" class="form-horizontal"
action="addbook" method="post" th:object="${bookBean}">
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">
                    <h2 style="text-align: center">Create New
Book</h2>
                </div>
            </div>
            <hr/>
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">
                    <label for="user-name">Book Code</label>
                    <input type="text" class="form-control"
th:field="*{bookCode}"></input>
                    <label
th:if="${#fields.hasErrors('bookCode')}"
th:errors="*{bookCode}" style="color:red;">Error</label>
                </div>
            </div>
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">

```

```

        <label for="user-name">Book Title</label>
        <input type="text" class="form-
control" th:field="*{bookTitle}"></input>
        <label
th:if="${#fields.hasErrors('bookTitle')}}"
th:errors="*{bookTitle}" style="color:red;">Error</label>
    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <label for="user-name">Book
Author</label>
        <input type="text" class="form-
control" th:field="*{bookAuthor}"></input>
        <label
th:if="${#fields.hasErrors('bookAuthor')}}"
th:errors="*{bookAuthor}" style="color:red;">Error</label>
    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <label for="user-name">Book Price</label>
        <input type="text" class="form-
control" th:field="*{bookPrice}"></input>
        <label
th:if="${#fields.hasErrors('bookPrice')}}"
th:errors="*{bookPrice}" style="color:red;">Error</label>
    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <div class="col-sm-2">
            <input type="submit" value="Save"
class="btn btn-success" style="width: 80px;">
        </div>
        <div class="col-sm-1">
        </div>
        <div class="col-sm-2">
            <a href="/" class="btn btn-
primary">List All Book</a>
        </div>
    </div>
</div>
</div>
</form>
</body>
</html>

```

**updatebook.html**

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
    <title>Create New Book Form</title>

    <link th:href="@{/resources/css/bootstrap.min.css}"
    rel="stylesheet" />

    </head>
    <body>
        <form id="form" class="form-horizontal"
action="updatebook" method="post" th:object="${bookBean}">
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">
                    <h2 style="text-align: center">Update
Book</h2>
                </div>
            </div>
            <hr/>
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">
                    <label for="user-name">Book Code</label>
                    <input type="text" class="form-control"
th:field="*{bookCode}"></input>
                    <label
th:if="${#fields.hasErrors('bookCode')}"
th:errors="*{bookCode}" style="color:red;">Error</label>
                </div>
            </div>
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">
                    <label for="user-name">Book Title</label>
                    <input type="text" class="form-
control" th:field="*{bookTitle}"></input>
                    <label
th:if="${#fields.hasErrors('bookTitle')}"
th:errors="*{bookTitle}" style="color:red;">Error</label>
                </div>
            </div>
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">

```

```

        <label for="user-name">Book
Author</label>
        <input type="text" class="form-
control" th:field="*{bookAuthor}"></input>
        <label
th:if="${#fields.hasErrors('bookAuthor')}}"
th:errors="*{bookAuthor}" style="color:red;">Error</label>
    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <label for="user-name">Book Price</label>
        <input type="text" class="form-
control" th:field="*{bookPrice}"></input>
        <label
th:if="${#fields.hasErrors('bookPrice')}}"
th:errors="*{bookPrice}" style="color:red;">Error</label>
    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <div class="col-sm-2">
            <input type="submit" value="Update"
class="btn btn-success" style="width: 80px;">
        </div>
        <div class="col-sm-1">
        </div>
        <div class="col-sm-2">
            <a href="/" class="btn btn-
primary">List All Book</a>
        </div>
    </div>
</div>
</form>
</body>
</html>

```



## **application.properties**

```
# datasource
spring.datasource.url=jdbc:mysql://localhost:3306/test
spring.datasource.username=root
spring.datasource.password=root

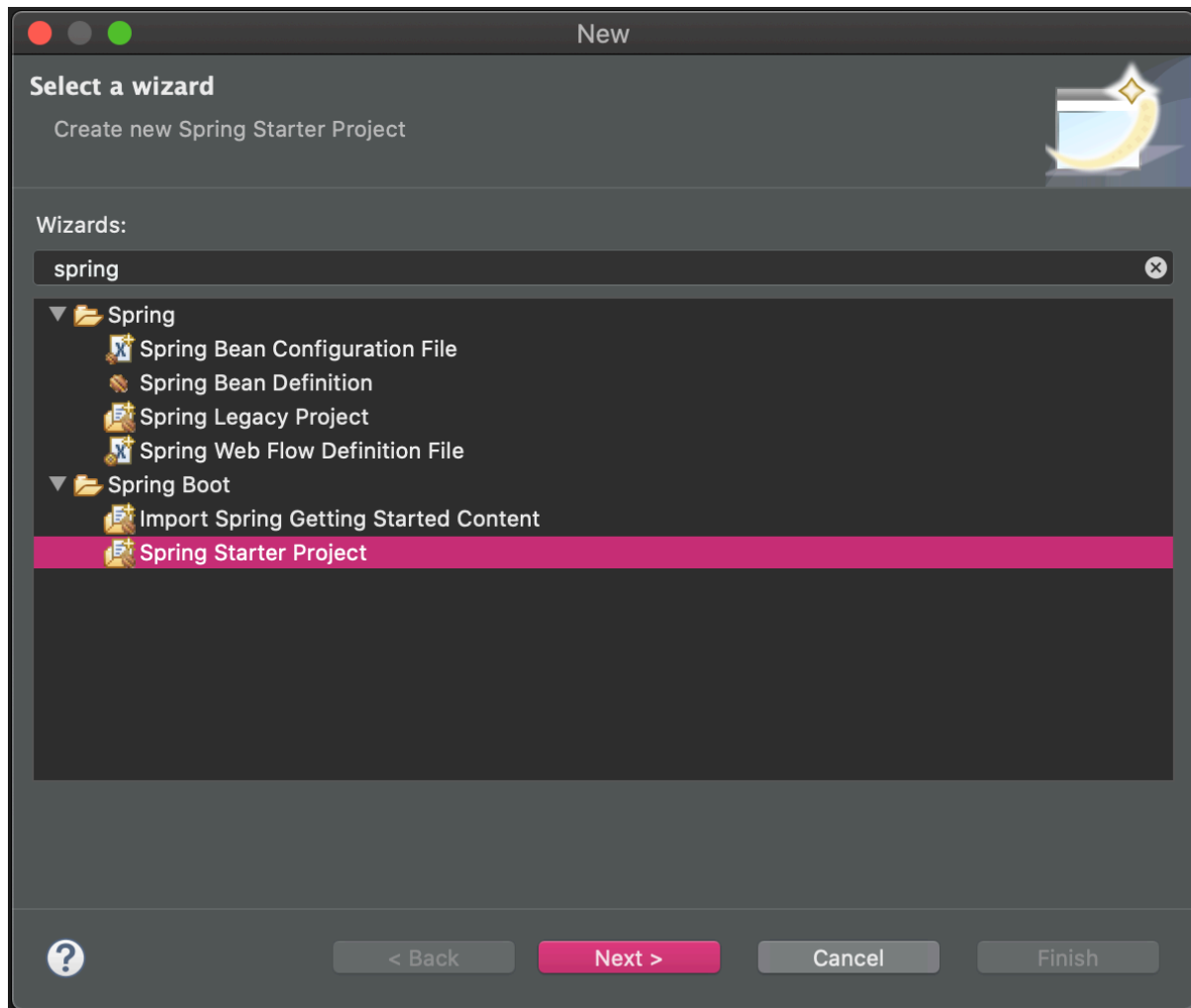
#view
spring.thymeleaf.suffix:.html

#validation
NotEmpty.bookBean.bookCode=Book Code must not be empty.
NotEmpty.bookBean.bookTitle=Book Title must not be empty.
NotEmpty.bookBean.bookAuthor=Book Author must not be empty.
NotEmpty.bookBean.bookPrice=Book Price must not be empty.
```

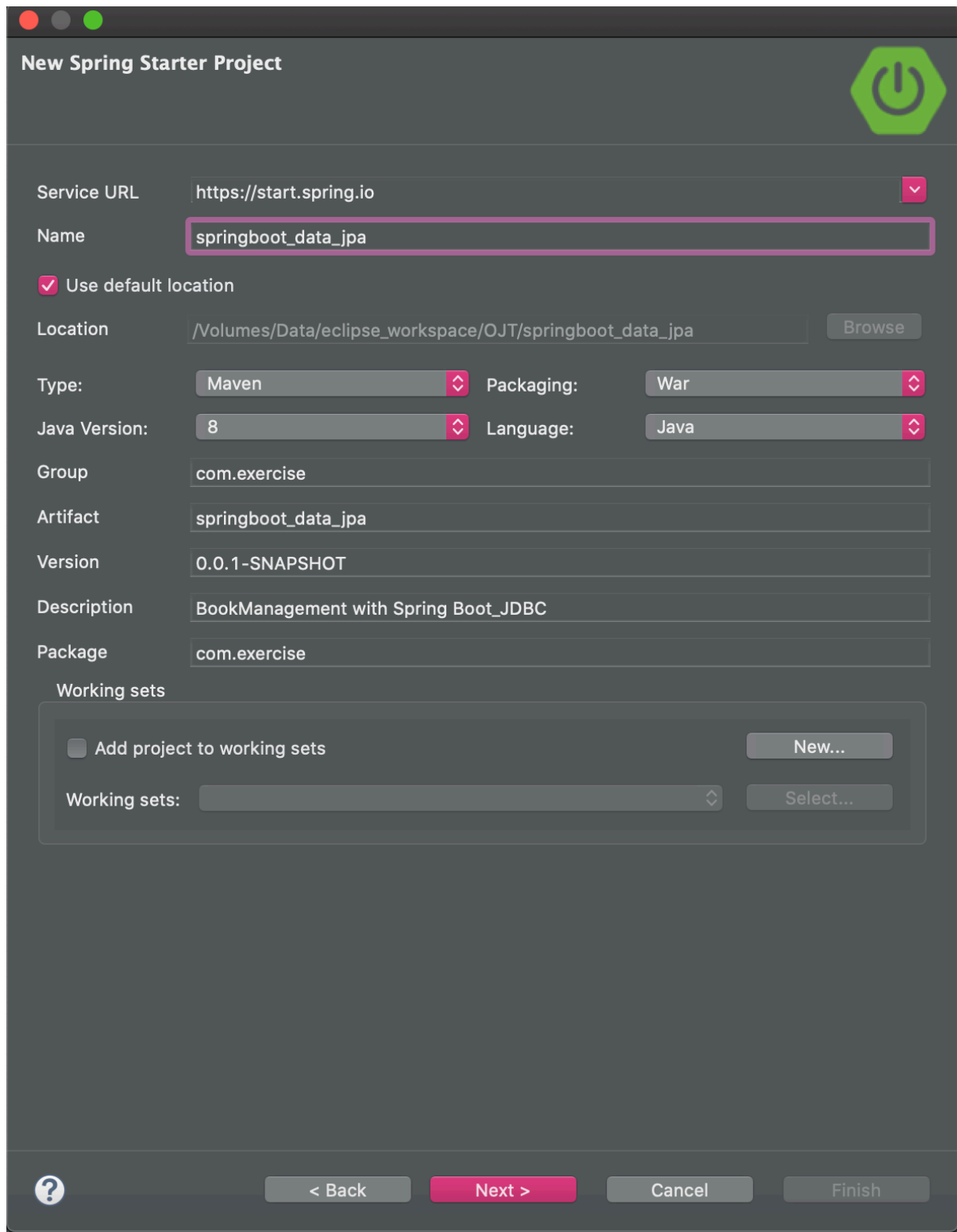
## 7. Spring Boot – Thymeleaf - JPA

Go to **File > New > Other**. Type **Spring** as shown below. And then choose **Spring Starter Project**.

Click Next.



Fill the following picture. Click Next.



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

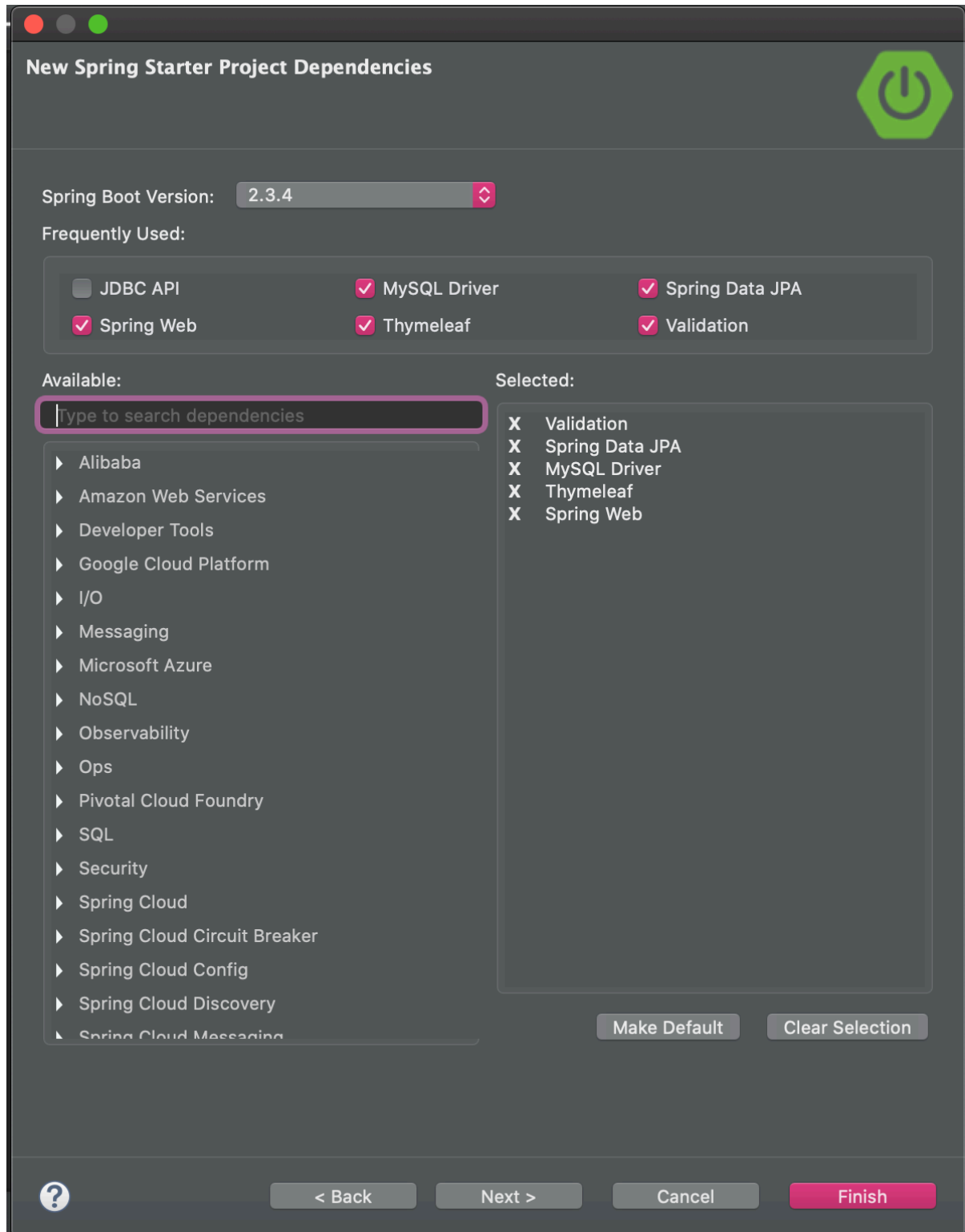
Package:

Working sets

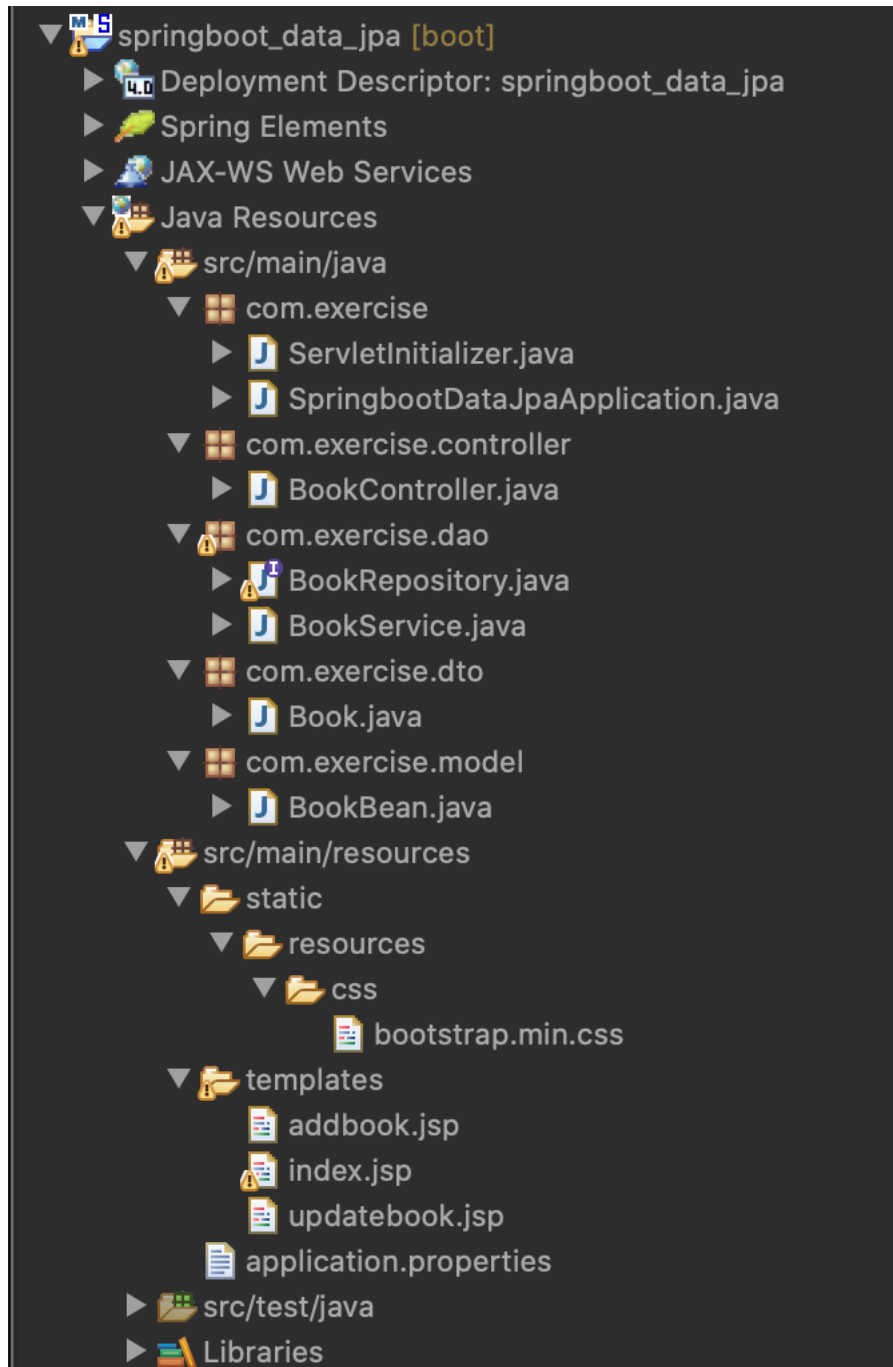
☐ Add project to working sets

Working sets:

Check **Spring Web, Spring Data JPA, Thymeleaf, MySQL Driver, Validation** CheckBox.



Congratulations! You have successfully created the Spring Boot application in your IDE. Let's understand what it has already configured for you.



**BookBean.java**

```
package com.exercise.model;

import javax.validation.constraints.NotEmpty;

public class BookBean {
    @NotEmpty
    private String bookCode;
    @NotEmpty
    private String bookTitle;
    @NotEmpty
    private String bookAuthor;
    @NotEmpty
    private String bookPrice;
    public String getBookCode() {
        return bookCode;
    }
    public void setBookCode(String bookCode) {
        this.bookCode = bookCode;
    }
    public String getBookTitle() {
        return bookTitle;
    }
    public void setBookTitle(String bookTitle) {
        this.bookTitle = bookTitle;
    }
    public String getBookAuthor() {
        return bookAuthor;
    }
    public void setBookAuthor(String bookAuthor) {
        this.bookAuthor = bookAuthor;
    }
    public String getBookPrice() {
        return bookPrice;
    }
    public void setBookPrice(String bookPrice) {
        this.bookPrice = bookPrice;
    }
}
```

**BookRepository.java**

```
package com.exercise.dao;

import org.springframework.data.repository.CrudRepository;
import com.exercise.dto.Book;

//repository that extends CrudRepository
public interface BookRepository extends CrudRepository<Book,
String>
{

}
}
```

**BookService.java**

```
package com.exercise.dao;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.exercise.dto.Book;

//defining the business logic
@Service
public class BookService
{
    @Autowired
    BookRepository booksRepository;
    //getting all books record by using the method findaAll() of
    CrudRepository
    public List<Book> getAllBooks()
    {
        List<Book> list = (List<Book>) booksRepository.findAll();
        return list;
    }
    //getting a specific record by using the method findById() of
    CrudRepository
    public Optional<Book> getBooksByBookCode(String code) {
        return booksRepository.findById(code);
    }
    //saving a specific record by using the method save() of
    CrudRepository
    public void save(Book books)
    {

```

```

booksRepository.save(books);
}
//deleting a specific record by using the method deleteById()
of CrudRepository
public void delete(String code)
{
booksRepository.deleteById(code);
}
//updating a record
public void update(Book books,String bookCode)
{
booksRepository.save(books);
}
}

```

### BookController.java

```

package com.exercise.controller;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import com.exercise.model.BookBean;
import com.exercise.dao.BookService;
import com.exercise.dto.Book;

@Controller
public class BookController {
    @Autowired
    private BookService bookService;

    @ModelAttribute("bookBean")
    public BookBean getBookBean() {
        return new BookBean();
    }

    @RequestMapping(value="/", method=RequestMethod.GET)
    public String displayView(ModelMap model) {
        List<Book> list=bookService.getAllBooks();
        model.addAttribute("list", list);
    }
}

```



```

        return "index";
    }

    @RequestMapping(value="/setupaddbook",
method=RequestMethod.GET)
    public ModelAndView setupaddbook() {
        return new ModelAndView("addbook","bookBean",new
BookBean());
    }

    @RequestMapping(value="/addbook",
method=RequestMethod.POST)
    public String addbook(@ModelAttribute("bookBean")
@Validated BookBean bean,
        BindingResult bs, ModelMap model) {
        if(bs.hasErrors()) {
            return "addbook";
        }
        Book dto=new Book();
        dto.setBookCode(bean.getBookCode());
        dto.setBookTitle(bean.getBookTitle());
        dto.setBookAuthor(bean.getBookAuthor());

        dto.setBookPrice(Double.valueOf(bean.getBookPrice()));
        bookService.save(dto);
        return "redirect:/";
    }

    @RequestMapping(value="/setupUpdateBook",
method=RequestMethod.GET)
    public ModelAndView setupUpdatebook(@RequestParam
("code") String bookCode) {
        return new
ModelAndView("updatebook","bookBean",bookService.getBooksByBoo
kCode(bookCode));
    }

    @RequestMapping(value="/updatebook",
method=RequestMethod.POST)
    public String updatebook(@ModelAttribute("bookBean")
@Validated BookBean bean,
        BindingResult bs, ModelMap model) {
        if(bs.hasErrors()) {
            return "updatebook";
        }
        Book dto=new Book();
        dto.setBookCode(bean.getBookCode());
        dto.setBookTitle(bean.getBookTitle());
        dto.setBookAuthor(bean.getBookAuthor());
    }

```

```

        dto.setBookPrice(Double.valueOf(bean.getBookPrice()));
        bookService.update(dto,dto.getBookCode());

        return "redirect:/";
    }

    @RequestMapping(value="/deleteBook",
method=RequestMethod.GET)
    public String deleteBook(@RequestParam ("code") String
bookCode,ModelMap model) {
        bookService.delete(bookCode);
        return "redirect:/";
    }
}

```

## index.jsp

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>List All Book</title>
    <link href="/resources/css/bootstrap.min.css"
rel="stylesheet"/>
    <style type="text/css">
        table {
            border-collapse: collapse;
            width: 100%;
        }
        th, td {
            text-align: left;
            padding: 8px;
        }

        tr:nth-child(even){background-color: #f2f2f2}
        th {
            background-color: #4CAF50;
            color: white;
        }
    </style>
</head>
<body>

    <center>
        <h2>List All Book</h2>
    </center>

```

```

        <table class="table table-striped table-hover
table-bordered">
        <tr>
            <th>Book Code</th>
            <th>Book Title</th>
            <th>Book Author</th>
            <th>Book Price</th>
            <th>Action</th>
        <tr th:each="book: ${list}">
            <td th:text="${book.bookCode}" />
            <td th:text="${book.bookTitle}" />
            <td th:text="${book.bookAuthor}" />
            <td align="right" th:text="${book.bookPrice}" />
            <td>
                <a
th:href="@{/setupUpdateBook(code=${book.bookCode})}">Update</a>
                |
                <a
th:href="@{/deleteBook(code=${book.bookCode})}">Delete</a>
            </td>
        </tr>
        </table>
        <center><a href="setupaddbook"><input
type="submit" value="Create New Book" class="btn btn-
success"></a> </center>

    </body>
</html>

```

## addbook.jsp

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
    <title>Create New Book Form</title>

    <link th:href="@{/resources/css/bootstrap.min.css}"
rel="stylesheet" />

    </head>
    <body>
        <form id="form" class="form-horizontal"
action="addbook" method="post" th:object="${bookBean}">
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">
                    <h2 style="text-align: center">Create New
Book</h2>

```

```

        </div>
    </div>
    <hr/>
    <div class="form-group">
        <div class="col-sm-4"></div>
        <div class="col-sm-4">
            <label for="user-name">Book Code</label>
            <input type="text" class="form-control"
th:field="*{bookCode}"></input>
            <label
th:if="${#fields.hasErrors('bookCode')}}"
th:errors="*{bookCode}" style="color:red;">Error</label>

        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-4"></div>
        <div class="col-sm-4">
            <label for="user-name">Book Title</label>
            <input type="text" class="form-
control" th:field="*{bookTitle}"></input>
            <label
th:if="${#fields.hasErrors('bookTitle')}}"
th:errors="*{bookTitle}" style="color:red;">Error</label>
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-4"></div>
        <div class="col-sm-4">
            <label for="user-name">Book
Author</label>
            <input type="text" class="form-
control" th:field="*{bookAuthor}"></input>
            <label
th:if="${#fields.hasErrors('bookAuthor')}}"
th:errors="*{bookAuthor}" style="color:red;">Error</label>
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-4"></div>
        <div class="col-sm-4">
            <label for="user-name">Book Price</label>
            <input type="text" class="form-
control" th:field="*{bookPrice}"></input>
            <label
th:if="${#fields.hasErrors('bookPrice')}}"
th:errors="*{bookPrice}" style="color:red;">Error</label>
        </div>
    </div>

```

```

        <div class="form-group">
            <div class="col-sm-4"></div>
            <div class="col-sm-4">
                <div class="col-sm-2">
                    <input type="submit" value="Save"
class="btn btn-success" style="width: 80px;">
                </div>
                <div class="col-sm-1">
                </div>
                <div class="col-sm-2">
                    <a href="/" class="btn btn-
primary">List All Book</a>
                </div>
            </div>
        </div>
    </form>
</body>
</html>

```

### updatebook.jsp

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
    <title>Create New Book Form</title>

    <link th:href="@{/resources/css/bootstrap.min.css}"
rel="stylesheet" />

    </head>
    <body>
        <form id="form" class="form-horizontal"
action="updatebook" method="post" th:object="${bookBean}">
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">
                    <h2 style="text-align: center">Update
Book</h2>
                </div>
            </div>
            <hr/>
            <div class="form-group">
                <div class="col-sm-4"></div>
                <div class="col-sm-4">
                    <label for="user-name">Book Code</label>
                    <input type="text" class="form-control"
th:field="*{bookCode}"></input>

```

```

        <label
th:if="${#fields.hasErrors('bookCode')}}"
th:errors="*{bookCode}" style="color:red;">Error</label>

    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <label for="user-name">Book Title</label>
        <input type="text" class="form-
control" th:field="*{bookTitle}"></input>
        <label
th:if="${#fields.hasErrors('bookTitle')}}"
th:errors="*{bookTitle}" style="color:red;">Error</label>
    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <label for="user-name">Book
Author</label>
        <input type="text" class="form-
control" th:field="*{bookAuthor}"></input>
        <label
th:if="${#fields.hasErrors('bookAuthor')}}"
th:errors="*{bookAuthor}" style="color:red;">Error</label>
    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <label for="user-name">Book Price</label>
        <input type="text" class="form-
control" th:field="*{bookPrice}"></input>
        <label
th:if="${#fields.hasErrors('bookPrice')}}"
th:errors="*{bookPrice}" style="color:red;">Error</label>
    </div>
</div>
<div class="form-group">
    <div class="col-sm-4"></div>
    <div class="col-sm-4">
        <div class="col-sm-2">
            <input type="submit" value="Update"
class="btn btn-success" style="width: 80px;">
        </div>
    </div>
</div>

```

```
                <div class="col-sm-2">
                    <a href="/" class="btn btn-
primary">List All Book</a>
                </div>
            </div>
        </form>
    </body>
</html>
```

### application.properties

```
# datasource
spring.datasource.url=jdbc:mysql://localhost:3306/test
spring.datasource.username=root
spring.datasource.password=root

#view
spring.thymeleaf.suffix:.jsp

#validation
NotEmpty.bookBean.bookCode=Book Code must not be empty.
NotEmpty.bookBean.bookTitle=Book Title must not be empty.
NotEmpty.bookBean.bookAuthor=Book Author must not be empty.
NotEmpty.bookBean.bookPrice=Book Price must not be empty.
```