**What is Spring Boot?**

Spring Boot is a Framework from “The Spring Team” to ease the bootstrapping and development of new Spring Applications. It provides defaults for code and annotation configuration to quick start new Spring projects within no time

### Advantages of Spring Boot:

* It is very easy to develop Spring Based applications with Java or Groovy.
* It reduces lots of development time and increases productivity.
* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
* It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
* It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.
* It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle
* It provides lots of plugins to work with embedded and in-memory Databases very easily.

In Simple Terminology, What Spring Boot means[What Is Spring Boot, Spring Boot Tutorial](https://journaldev.nyc3.digitaloceanspaces.com/2015/05/WhatIsSpringBoot1.png)That means Spring Boot is nothing but existing Spring Framework + Some Embedded HTTP Servers (Tomcat/Jetty etc.) - XML or Annotations Configurations. Here minus means we don’t need to write any XML Configuration and few Annotations only.

**Main Goal of Spring Boot:**

The main goal of Spring Boot Framework is to reduce Development, Unit Test and Integration Test time and to ease the development of Production ready web applications very easily compared to existing Spring Framework, which really takes more time.

* To avoid XML Configuration completely
* To avoid defining more Annotation Configuration(It combined some existing Spring Framework Annotations to a simple and single Annotation)
* To avoid writing lots of import statements

SpringBoot Annotations:

* **Basic Setup**
  + @SpringBootApplication
  + @Configuration
  + @ComponentScan
  + @EnableAutoConfiguration
* **Request Responses**
  + @GetMapping
  + @RequestMapping
  + @RequestParam
* **Component Types**
  + @Component
  + @Service
  + @Repository
  + @Controller
  + @RestController

**@EnableAutoConfiguration:** It auto-configures the bean that is present in the classpath .

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

**@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**.

Example:

1. @Controller
2. @RequestMapping("/book")
3. **public** **class** BooksController
4. {
5. @RequestMapping("/computer-science/books")
6. **public** String getAllBooks(Model model)
7. {
8. //application code
9. **return** "bookList";
10. }

**@RestController:** It can be considered as a combination of **@Controller** +Restfull webservices methods

* **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
* **@PostMapping:** It maps the **HTTP POST**requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
* **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
* **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes**a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**

**@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request.

**@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

**@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable.

**@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**.

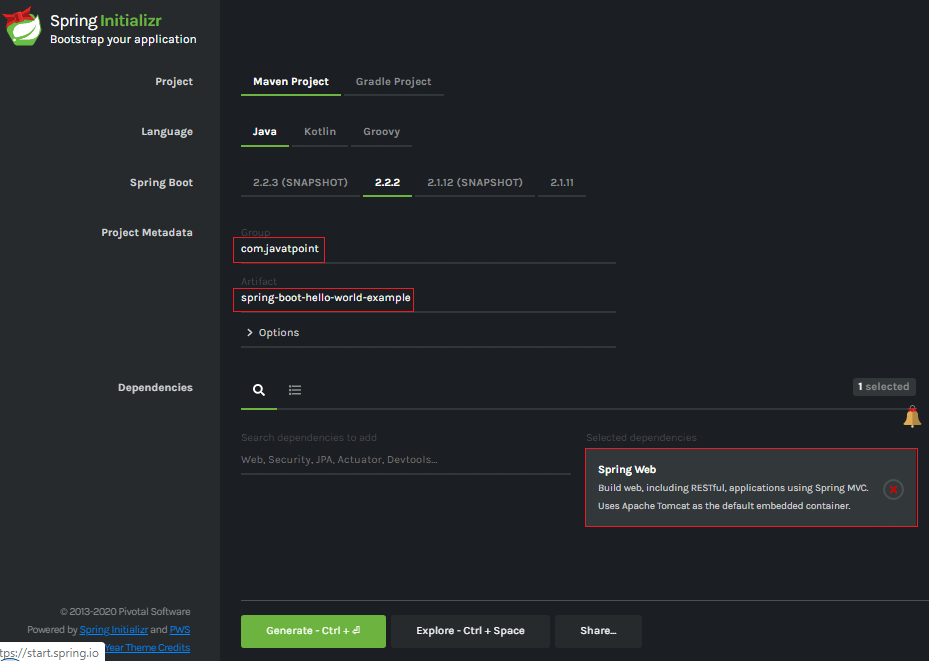
**Step 1:** Open Spring Initializr <https://start.spring.io/>.

**Step 2:** Provide the **Group** name. We have provided **com.dxe.**

**Step 3:** Provide the **Artifact** Id. We have provided the **spring-boot-hello-world-example.**

**Step 4:** Add the dependency **Spring Web.**

**Step 5:** Click on the **Generate** button. When we click on the Generate button, it wraps all the specifications into a jar file and downloads it to our local system.

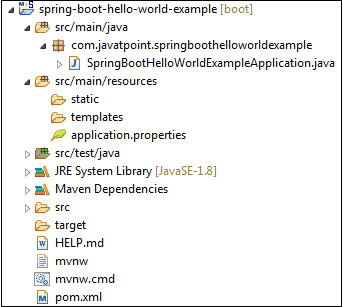


**Step 6: Extract** the RAR file.

**Step 7: Import** the project folder by using the following steps:

File -> Import -> Existing Maven Project -> Next -> Browse -> Select the Project Folder -> Finish

When the project imports successfully, it shows the following project directory in the Package Explorer section of the IDE.



**Step 8:** Create a package with the name **com.dxe.controller** inside the folder **src/main/java.**

**Step 9:** Create a Controller class with the name **HelloWorldController.**

**Step 10:** Create a method named **hello()**that returns a String.

**HelloWorldController.java**

**import** org.springframework.web.bind.annotation.RequestMapping;

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

@RestController

**public** **class** HelloWorld {

@RequestMapping("/")

String demo()

{

**return** "demodataaaaa";

}

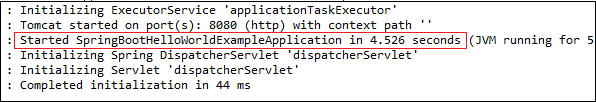
}

**Step 11:** Run the **SpringBootHelloWorldExampleApplication.java** file.

**SpringBootHelloWorldExampleApplication.java**

1. **package** com.javatpoint;
2. **import** org.springframework.boot.SpringApplication;
3. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. **public** **class** SpringBootHelloWorldExampleApplication
6. {
7. **public** **static** **void** main(String[] args)
8. {
9. SpringApplication.run(SpringBootHelloWorldExampleApplication.**class**, args);
10. }
11. }

When the application runs successfully, it shows a massage in the console, as shown in the following figure.



**Step 12:** Open the browser and invoke the URL **https://localhost:8080**. It returns a String that we have specified in the Controller.

