|  |  |
| --- | --- |
| **Spring** | **Spring Boot** |
| **Spring Framework** is a widely used Java EE framework for building applications. | **Spring Boot Framework** is widely used to develop **REST APIs**. |
| It aims to simplify Java EE development that makes developers more productive. | It aims to shorten the code length and provide the easiest way to develop **Web Applications**. |
| The primary feature of the Spring Framework is **dependency injection**. | The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement. |
| It helps to make things simpler by allowing us to develop **loosely coupled** applications. | It helps to create a **stand-alone** application with less configuration. |
| The developer writes a lot of code (**boilerplate code**) to do the minimal task. | It **reduces** boilerplate code. |
| To test the Spring project, we need to set up the sever explicitly. | Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc. |
| It does not provide support for an in-memory database. | It offers several plugins for working with an embedded and **in-memory** database such as **H2**. |
| Developers manually define dependencies for the Spring project in **pom.xml**. | Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement. |

**Difference between Spring & Spring Boot**

Spring Boot vs. Spring MVC

**Spring Boot:** Spring Boot makes it easy to quickly bootstrap and start developing a Spring-based application. It avoids a lot of boilerplate code.

It hides a lot of complexity behind the scene so that the developer can quickly get started and develop Spring-based applications easily.

**Spring MVC:** Spring MVC is a Web MVC Framework for building web applications. It contains a lot of configuration files for various capabilities.

It is an HTTP oriented web application development framework.

Spring Boot and Spring MVC exist for different purposes. The primary comparison between Spring Boot and Spring MVC are discussed below:

|  |  |
| --- | --- |
| **Spring Boot** | **Spring MVC** |
| **Spring Boot** is a module of Spring for packaging the Spring-based application with sensible defaults. | **Spring MVC** is a model view controller-based web framework under the Spring framework. |
| It provides default configurations to build **Spring-powered** framework. | It provides **ready to use** features for building a web application. |
| There is no need to build configuration manually. | It requires build configuration manually. |
| There is **no requirement** for a deployment descriptor. | A Deployment descriptor is **required**. |
| It avoids boilerplate code and wraps dependencies together in a single unit. | It specifies each dependency separately. |
| It **reduces** development time and increases productivity. | It takes **more** time to achieve the same. |

Before understanding the **Spring Boot Architecture**, we must know the different layers and classes present in it.

There are **four** layers in Spring Boot are as follows:



**Presentation Layer:** The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of **views** i.e., frontend part.

**Business Layer:** The business layer handles all the **business logic**. It consists of service classes and uses services provided by data access layers. It also performs **authorization** and **validation**.

**Persistence Layer:** The persistence layer contains all the **storage logic** and translates business objects from and to database rows.

**Database Layer:** In the database layer, **CRUD** (create, retrieve, update, delete) operations are performed.

|  |
| --- |
| Spring Boot Flow Architecture     * Now we have validator classes, view classes, and utility classes. * Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc. The architecture of Spring Boot is the same as the architecture of Spring MVC, * except one thing: there is no need for **DAO** and **DAOImpl** classes in Spring boot. * Creates a data access layer and performs CRUD operation. * The client makes the HTTP requests (PUT or GET). * The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required. * In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes. * A JSP page is returned to the user if no error occurred |

|  |
| --- |
| Spring Initializr Modules  Spring Initializr has the following module:   * **initializr-actuator:** It provides additional information and statistics on project generation. It is an optional module. * **initializr-bom:** In this module, **BOM** stands for **Bill Of Materials**. In Spring Boot, BOM is a special kind of **POM** that is used to   control the **versions** of a project's **dependencies**.   * It provides a central place to define and update those versions. It provides flexibility to add a dependency in our module without worrying about the versions. * Outside the software world, the **BOM** is a list of parts, items, assemblies, and other materials required to create products.   It explains **what, how,** and **where** to collect required materials.   * **initializr-docs:** It provides documentation. * **initializr-generator:** It is a core project generation library. * **initializr-generator-spring:** * **initializr-generator-test:** It provides a test infrastructure for project generation. * **initializr-metadata:** It provides metadata infrastructure for various aspects of the projects. * **initializr-service-example:** It provides custom instances. * **initializr-version-resolver:** It is an optional module to extract version numbers from an arbitrary POM. * **initializr-web:** It provides web endpoints for third party clients.   Supported Interface   * It supports **IDE STS, IntelliJ IDEA Ultimate, NetBeans, Eclipse**. You can download the plugin from <https://github.com/AlexFalappa/nb-springboot>. * If you are using VSCode, download the plugin from <https://github.com/microsoft/vscode-spring-initializr>. * Use Custom Web UI [http://start.spring.io](http://start.spring.io/) or [https://start-scs.cfapps.io](https://start-scs.cfapps.io/). * It also supports the command-line with the **Spring Boot CLI** or **cURL** or **HTTPie**.   The following image shows the Spring Initializr UI:    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**. The latest version is **2.2.2**. * **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. |

Spring Tool Suite is an IDE to develop Spring applications. It is an Eclipse-based development environment. It provides a ready-to-use environment to implement, run, deploy, and debug the application. It validates our application and provides quick fixes for the applications.

### Installing STS

**Step 1:** Download Spring Tool Suite from [https://spring.io/tools3/sts/all](https://spring.io/tools3/sts/all" \t "_blank). Click on the platform which you are using.

**Step 2:** Extract the **zip** file and install the STS.

**Step 3:** Spring Tool Suite 3 Launcher dialog box appears on the screen. Click on the **Launch** button. You can change the Workspace if you want.

**Step 4:** It starts launching the STS.

Following are the steps to create a simple Spring Boot Project.

**Step 1:** Open the Spring initializr [https://start.spring.io](https://start.spring.io/" \t "_blank).

**Step 2:** Provide the **Group** and **Artifact** name. We have provided Group name **com.example** and Artifact **spring-boot-example**.

**Step 3:** Now click on the **Generate** button.

When we click on the Generate button, it starts packing the project in a **.rar** file and downloads the project.

**Step 4:** Extract the **RAR** file.

**Step 5:** **Import** the folder.

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

It takes some time to import the project. When the project imports successfully, we can see the project directory in the **Package Explorer**. The following image shows the project directory:

**SpringBootExampleApplication.java**

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

**pom.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
4. **<parent>**
5. **<groupId>**org.springframework.boot**</groupId>**
6. **<artifactId>**spring-boot-starter-parent**</artifactId>**
7. **<version>**2.2.2.BUILD-SNAPSHOT**</version>**
8. **<relativePath/>** <!-- lookup parent from repository -->
9. **</parent>**
10. **<groupId>**com.example**</groupId>**
11. **<artifactId>**spring-boot-example**</artifactId>**
12. **<version>**0.0.1-SNAPSHOT**</version>**
13. **<name>**spring-boot-example**</name>**
14. **<description>**Demo project for Spring Boot**</description>**
15. **<properties>**
16. **<java.version>**1.8**</java.version>**
17. **</properties>**
18. **<dependencies>**
19. **<dependency>**
20. **<groupId>**org.springframework.boot**</groupId>**
21. **<artifactId>**spring-boot-starter**</artifactId>**
22. **</dependency>**
23. **<dependency>**
24. **<groupId>**org.springframework.boot**</groupId>**
25. **<artifactId>**spring-boot-starter-test**</artifactId>**
26. **<scope>**test**</scope>**
27. **<exclusions>**
28. **<exclusion>**
29. **<groupId>**org.junit.vintage**</groupId>**
30. **<artifactId>**junit-vintage-engine**</artifactId>**
31. **</exclusion>**
32. **</exclusions>**
33. **</dependency>**
34. **</dependencies>**
35. **<build>**
36. **<plugins>**
37. **<plugin>**
38. **<groupId>**org.springframework.boot**</groupId>**
39. **<artifactId>**spring-boot-maven-plugin**</artifactId>**
40. **</plugin>**
41. **</plugins>**
42. **</build>**
43. **<repositories>**
44. **<repository>**
45. **<id>**spring-milestones**</id>**
46. **<name>**Spring Milestones**</name>**
47. **<url>**https://repo.spring.io/milestone**</url>**
48. **</repository>**
49. **<repository>**
50. **<id>**spring-snapshots**</id>**
51. **<name>**Spring Snapshots**</name>**
52. **<url>**https://repo.spring.io/snapshot**</url>**
53. **<snapshots>**
54. **<enabled>**true**</enabled>**
55. **</snapshots>**
56. **</repository>**
57. **</repositories>**
58. **<pluginRepositories>**
59. **<pluginRepository>**
60. **<id>**spring-milestones**</id>**
61. **<name>**Spring Milestones**</name>**
62. **<url>**https://repo.spring.io/milestone**</url>**
63. **</pluginRepository>**
64. **<pluginRepository>**
65. **<id>**spring-snapshots**</id>**
66. **<name>**Spring Snapshots**</name>**
67. **<url>**https://repo.spring.io/snapshot**</url>**
68. **<snapshots>**
69. **<enabled>**true**</enabled>**
70. **</snapshots>**
71. **</pluginRepository>**
72. **</pluginRepositories>**
73. **</project>**

**Step 6:** Run the **SpringBootExampleApplication.java** file.

Right-click on the file -> Run As -> Java Applications

-=-=-=-=-=-=-=-=-=-

# Creating a Spring Boot Project Using STS

We can also use Spring Tool Suite to create a Spring project. In this section, we will create a **Maven Project** using **STS**.

**Step 1:** Open the Spring Tool Suite.

**Step 2:** Click on the File menu -> New -> Maven Project

|  |
| --- |
| It shows the New Maven Project wizard. Click on the **Next** button.    **Step 3:** Select the **maven-archetype-quickstart** and click on the **Next** button.  **Step 4:** Provide the **Group Id** and **Artifact Id**. We have provided Group Id **com.example** and Artifact Id **spring-boot-example-sts**.  Now click on the **Finish** button. |

**Step 5:** Open the **App.java** file. We found the following code that is by default.

**App.java**

1. **package** com.example;
2. **public** **class** App
3. {
4. **public** **static** **void** main( String[] args )
5. {
6. System.out.println( "Hello World!" );
7. }
8. }

The Maven project has a **pom.xml** file which contains the following default configuration.

**pom.xml**

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
4. **<groupId>**com.example**</groupId>**
5. **<artifactId>**spring-boot-example-sts**</artifactId>**
6. **<version>**0.0.1-SNAPSHOT**</version>**
7. **<packaging>**jar**</packaging>**
8. **<name>**spring-boot-example-sts**</name>**
9. **<url>**http://maven.apache.org**</url>**
10. **<properties>**
11. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
12. **</properties>**
13. **<dependencies>**
14. **<dependency>**
15. **<groupId>**junit**</groupId>**
16. **<artifactId>**junit**</artifactId>**
17. **<version>**3.8.1**</version>**
18. **<scope>**test**</scope>**
19. **</dependency>**
20. **</dependencies>**
21. **</project>**

**Step 6:** Add **Java version** inside the **<properties>** tag.

1. <java.version>1.8</java.version>

**Step 7:** In order to make a Spring Boot Project, we need to configure it. So, we are adding **spring boot starter parent** dependency in **pom.xml** file. Parent is used to declare that our project is a child to this parent project.

1. **<dependency>**
2. **<groupId>**org.springframework.boot**</groupId>**
3. **<artifactId>**spring-boot-starter-parent**</artifactId>**
4. **<version>**2.2.1.RELEASE**</version>**
5. **<type>**pom**</type>**
6. **</dependency>**

**Step 8:** Add the **spring-boot-starter-web** dependency in **pom.xml** file.

1. **<dependency>**
2. **<groupId>**org.springframework.boot**</groupId>**
3. **<artifactId>**spring-boot-starter-web**</artifactId>**
4. **<version>**2.2.1.RELEASE**</version>**
5. **</dependency>**

|  |
| --- |
| Note: When we add the dependencies in the pom file, it downloads the related jar file.  We can see the downloaded jar files in the Maven Dependencies folder of the project directory.    After adding all the dependencies, the pom.xml file looks like the following:  **pom.xml**   * **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" * xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>** * **<modelVersion>**4.0.0**</modelVersion>** * **<groupId>**com.example**</groupId>** * **<artifactId>**spring-boot-example-sts**</artifactId>** * **<version>**0.0.1-SNAPSHOT**</version>** * **<packaging>**jar**</packaging>** * **<name>**spring-boot-example-sts**</name>** * **<url>**http://maven.apache.org**</url>** * **<properties>** * **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>** * **<java.version>**1.8**</java.version>** * **</properties>** * **<dependencies>** * **<dependency>** * **<groupId>**org.springframework.boot**</groupId>** * **<artifactId>**spring-boot-starter-parent**</artifactId>** * **<version>**2.2.1.RELEASE**</version>** * **<type>**pom**</type>** * **</dependency>** * **<dependency>** * **<groupId>**org.springframework.boot**</groupId>** * **<artifactId>**spring-boot-starter-web**</artifactId>** * **<version>**2.2.1.RELEASE**</version>** * **</dependency>** * **<dependency>** * **<groupId>**junit**</groupId>** * **<artifactId>**junit**</artifactId>** * **<version>**3.8.1**</version>** * **<scope>**test**</scope>** * **</dependency>** * **</dependencies>** * **</project>**   **Step 9:** Create a class with the name **SpringBootExampleSts** in the package **com.example**.  Right-click on the package name -> New -> Class -> provide the class name -> Finish |

**Step 10:** After creating the class file, call the static method **run()** of the SpringApplication class. In the following code, we are calling the run() method and passing the class name as an argument.

1. SpringApplication.run(SpringBootExampleSts.**class**, args);

**Step 11:** Annotate the class by adding an annotation **@SpringBootApplication**.

**@SpringBootApplication**

A single @SpringBootApplication annotation is used to enable the following annotations:

* **@EnableAutoConfiguration:** It enables the Spring Boot auto-configuration mechanism.
* **@ComponentScan:** It scans the package where the application is located.
* **@Configuration:** It allows us to register extra beans in the context or import additional configuration classes.

**SpringBootApplicationSts.java**

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

**Step:** Run the file **SpringBootExampleSts.java**, as Java Application. It displays the console.

The line Started SpringBootExampleSts in 5.038 seconds (JVM running for 6.854) in the console shows that the application is up and running.