# **Spring Core vs. Spring Boot vs. Spring MVC**

## Spring vs. Spring Boot

**Spring:** Spring Framework is the most popular application development framework of Java. The main feature of the Spring Framework is **dependency Injection** or **Inversion of Control** (IoC). With the help of Spring Framework, we can develop a **loosely** coupled application. It is better to use if application type or characteristics are purely defined.

**Spring Boot:** Spring Boot is a module of Spring Framework. It allows us to build a stand-alone application with minimal or zero configurations. It is better to use if we want to develop a simple Spring-based application or RESTful services.

The primary comparison between Spring and Spring Boot are discussed below:

| **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 server 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. |

## 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.

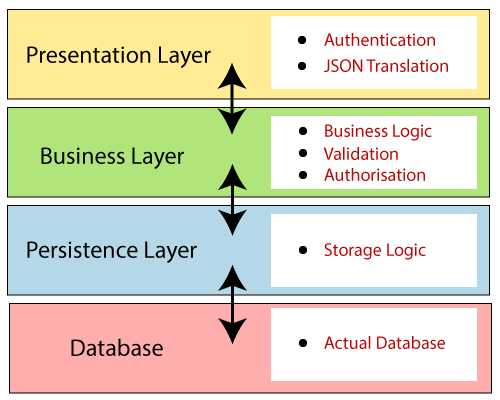
Spring Boot and Spring MVC exist for different purposes. The primary comparison between Spring Boot and Spring MVC are discussed below:

| **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. |

## Architecture

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**
* **Business Layer**
* **Persistence Layer**
* **Database Layer**



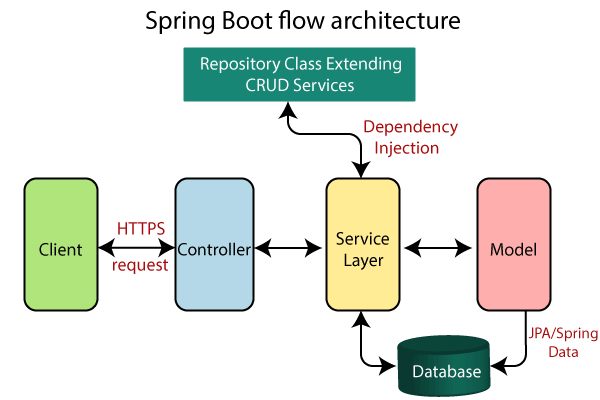
**Presentation Layer:** The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of **views** i.e., frontend part.

**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.

Steps :- Spring Boot Application using Spring Initlizer

1. Open Spring Initializer site from google (https://start.spring.io) [Spring Initializr](https://start.spring.io/))
2. Create an application with given steps
   1. Select Project Type(Maven Project)
   2. Select Prog. Lang (Java)
   3. Select Spring Boot Version (2.5.4)
   4. Now Provide Project Meta Data (Group ID, Artifact ID)
   5. Now Click Add Dependencies button to add Dependencies
   6. Now Click Generate Button

**STS - (Spring Test Suite)**

## 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.

Application.properties

server.port=9001

server.servlet.context-path=/myapp

spring.application.name=springboot1

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

spring.mvc.static-path-pattern=/resources/\*\*

spring.datasource.url=jdbc:mysql://localhost:3306/springbootdb

spring.datasource.username=root

spring.datasource.password=mysql

1. spring.jpa.hibernate.ddl-auto=create-drop

<link rel="stylesheet" href="/resources/css/style.css">

<script type="text/javascript" src="/resources/js/app.js"></script>

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.boraji.tutorial.springboot</groupId> <artifactId>spring-boot-web-application-example</artifactId> <version>0.0.1-SNAPSHOT</version> <properties> <java.version>1.8</java.version> </properties> <parent> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-parent</artifactId> <version>1.5.4.RELEASE</version> </parent>

<dependencies> <dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-web</artifactId> </dependency> *<!-- JSTL tag lib -->*

<dependency> <groupId>javax.servlet.jsp.jstl</groupId> <artifactId>javax.servlet.jsp.jstl-api</artifactId> <version>1.2.1</version> </dependency>

<dependency> <groupId>taglibs</groupId> <artifactId>standard</artifactId> <version>1.1.2</version> </dependency> *<!-- Tomcat for JSP rendering -->*

<dependency> <groupId>org.apache.tomcat.embed</groupId> <artifactId>tomcat-embed-jasper</artifactId> <scope>provided</scope> </dependency>

</dependencies> <build> <plugins> <plugin> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-maven-plugin</artifactId> </plugin> </plugins> </build> <packaging>war</packaging> </project>