**Introduction to Spring Boot framework: -**

Java Spring Framework (Spring Framework) is a popular, open source, enterprise-level framework for creating standalone, production-grade applications that run on the Java Virtual Machine (JVM).

Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities:

1. Autoconfiguration
2. An opinionated approach to configuration
3. The ability to create standalone applications

These features work together to provide you with a tool that allows you to set up a Spring-based application with minimal configuration and setup.

**Why is Spring Framework so popular?**

Spring Framework offers a *dependency injection* feature that lets objects define their own dependencies that the Spring container later injects into them. This enables developers to create modular applications consisting of loosely coupled components that are ideal for [microservices](https://www.ibm.com/cloud/learn/microservices) and distributed network applications.

Spring Framework also offers built-in support for typical tasks that an application needs to perform, such as data binding, type conversion, validation, exception handling, resource and event management, internationalization, and more. It integrates with various Java EE technologies.

**What Spring Boot adds to Spring Framework**

As capable and comprehensive as Spring Framework is, it still requires significant time and knowledge to configure, set up, and deploy Spring applications. Spring Boot mitigates this effort with three important capabilities.

### **Autoconfiguration**

Autoconfiguration means that applications are initialized with pre-set dependencies that you don't have to configure manually. As Java Spring Boot comes with built-in autoconfiguration capabilities, it automatically configures both the underlying Spring Framework and third-party packages based on your settings (and based on best practices, which helps avoid errors). Even though you can override these defaults once the initialization is complete, Java Spring Boot's autoconfiguration feature enables you to start developing your Spring-based applications fast and reduces the possibility of human errors.

### **Opinionated approach**

Spring Boot uses an opinionated approach to adding and configuring starter dependencies, based on the needs of your project. Following its own judgment, Spring Boot chooses which packages to install and which default values to use, rather than requiring you to make all those decisions yourself and set up everything manually.

You can define the needs of your project during the initialization process, during which you choose among multiple starter dependencies—called Spring Starters—that cover typical use cases. You run Spring Boot Initializr by filling out a simple web form, without any coding.

For example, the ‘Spring Web’ starter dependency allows you to build Spring-based web applications with minimal configuration by adding all the necessary dependencies—such as the Apache Tomcat web server—to your project. ‘Spring Security’ is another popular starter dependency that automatically adds authentication and access-control features to your application.

Spring Boot includes over 50 Spring Starters, and many more third-party starters are available.

### **Standalone applications**

Spring Boot helps developers create applications that just run. Specifically, it lets you create standalone applications that run on their own, without relying on an external web server, by embedding a web server such as Tomcat or Netty into your app during the initialization process. As a result, you can launch your application on any platform by simply hitting the Run command. (You can opt out of this feature to build applications without an embedded Web server.)

### **Spring Boot vs. Spring Framework**

Again, the biggest advantages of using Spring Boot versus Spring Framework alone are ease of use and faster development. In theory, this comes at the expense of the greater flexibility you get from working directly with Spring Framework.

But, in practice, unless you need or want to implement a very unique configuration, using Spring Booth is worth the tradeoff. You still are able to use Spring Framework’s very popular annotation system that lets you easily inject extra dependencies (not covered by Spring Starters) into your application. And, you still get access to all Spring Framework features, including easy event handling, validation, data binding, type conversion, and built-in security and testing capabilities. Bottom line, if your project’s scope is covered by even just one Spring Starter, Spring Boot can significantly streamline development.

**Understanding features of Spring Boot**

Features of Spring Boot:

Spring CLI 🡪 Spring CLI helps us to boost our application development process and avoid boilerplate code.

Spring Initializer 🡪 This is basically a web base application, which can create an internal project structure for you.

Spring Actuator 🡪This feature helps s while running our spring boot application.

Starter Dependencies-> Using this, spring boot will manage all recommended dependencies and add in our project

Auto Configuration 🡪 While creating spring boot application all basic and recommended configuration will be done automatically.

Logging and Security --> The functionality of logging will be available here in Spring boot and Spring boot will also provide lots of security mechanism for our application.

**Understanding Starter POMs**

Dependency management is a critical aspect of any complex project. And doing this manually is less than ideal; the more time you spent on it the less time you have on the other important aspects of the project.

Spring Boot starters were built to address exactly this problem. Starter POMs are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy-paste loads of dependency descriptors.

We have more than 30 Boot starters available – let's see some of them in the following sections.

### **The Web Starter**

First, let's look at developing the REST service; we can use libraries like Spring MVC, Tomcat and Jackson – a lot of dependencies for a single application.

Spring Boot starters can help to reduce the number of manually added dependencies just by adding one dependency. So instead of manually specifying the dependencies just add one starter as in the following example:

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-web</**artifactId**>

</**dependency**>

Now we can create a REST controller. For the sake of simplicity we won't use the database and focus on the REST controller:

### **The Test Starter**

For testing we usually use the following set of libraries: Spring Test, JUnit, Hamcrest, and Mockito. We can include all of these libraries manually, but Spring Boot starter can be used to automatically include these libraries in the following way:

1. <**dependency**>
2. <**groupId**>org.springframework.boot</**groupId**>
3. <**artifactId**>spring-boot-starter-test</**artifactId**>
4. <**scope**>test</**scope**>
5. </**dependency**>

Notice that you don't need to specify the version number of an artifact. Spring Boot will figure out what version to use – all you need to specify is the version of *spring-boot-starter-parent* artifact. If later on you need to upgrade the Boot library and dependencies, just upgrade the Boot version in one place and it will take care of the rest.

### **The Data JPA Starter**

Most web applications have some sort of persistence – and that's quite often JPA.

Instead of defining all of the associated dependencies manually – let's go with the starter instead:

1. <**dependency**>
2. <**groupId**>org.springframework.boot</**groupId**>
3. <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>
4. </**dependency**>
5. <**dependency**>
6. <**groupId**>com.h2database</**groupId**>
7. <**artifactId**>h2</**artifactId**>
8. <**scope**>runtime</**scope**>
9. </**dependency**>

Notice that out of the box we have automatic support for at least the following databases: H2, Derby and Hsqldb. In our example, we'll use H2.

### **4. The Mail Starter**

A very common task in enterprise development is sending email, and dealing directly with Java Mail API usually can be difficult.

Spring Boot starter hides this complexity – mail dependencies can be specified in the following way:

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-mail</**artifactId**>

</**dependency**>

Now we can directly use the JavaMailSender, so let's write some tests.

For the testing purpose, we need a simple SMTP server. In this example, we'll use Wiser. This is how we can include it in our POM:

<**dependency**>

<**groupId**>org.subethamail</**groupId**>

<**artifactId**>subethasmtp</**artifactId**>

<**version**>3.1.7</**version**>

<**scope**>test</**scope**>

</**dependency**>

The latest version of Wiser can be found on [Maven central repository](https://search.maven.org/classic/#search%7Cga%7C1%7Csubethasmtp).

## Conclusion

In this article we have given an overview of Starters, explained why we need them and provided examples on how to use them in your projects.

Let's recap the benefits of using Spring Boot starters:

* increase pom manageability
* production-ready, tested & supported dependency configurations
* decrease the overall configuration time for the project

**Creating First Spring Boot Application**

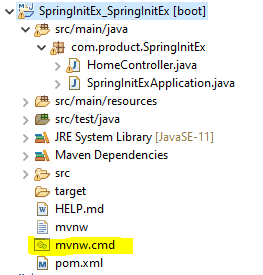
* Directly in Eclipse
* By using Spring Initialzr

**Creating Executable Jars**

We can create executable jar files using following ways

* mvnw package
* mvnw -e clean package spring-boot:repackage 🡪 if we created some test cases later on and there will be any issue we that and we will have to create jar again then we can use this command.
* Using eclipse/sts 🡪 we can create jar simply by using in built functionality of Eclipse of STS. Simply use maven build

Whenever we create a spring boot project either directly in Eclipse or using spring Initialzr we will get some standard files and folders which will help us to create executable jar files.



**Spring DevTools and Live Reload Spring Boot Fundamentals.**

1. DevTool stands for Developer tool. It has been developed to minimize development time of java application while working with spring boot application. Spring boot devtools pick up changes and restart application automatically.
2. For achieving this we will first have to add one dependency in our POM.xml by name spring-boot-devtools.

**<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-devtools -->**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-devtools</artifactId>**

**<version>2.6.3</version>**

**</dependency>**

1. After adding dev-tools in our project it will offers us some features as below:
   1. **Property default** 🡪 when sometimes we will have to use templates. Templates are nothing but just some UI pages, when we use such templates in our spring boot application at that time these templates save in our cache memory. So if you will make changes once your template saves in cache memory then changes won’t appear because you are getting view from your cache memory but when we will use DevTools then automatically the properties of cache will be set to false so our data won’t save in cache and whatever changes you will do, those all changes will appear directly which will boost your development speed.
   2. **Automatic restart** 🡪 when we will use DevTools in our project then it will keep checking class path of our project. So whenever you will do changes then it will check class path and it will restart our application automatically.
   3. **Live Reloading** 🡪 This feature will helps us to get recently modification in our UI pages that is we can say templates as soon as we will make changes. Same like Live server in VS code.
   4. It also support **Remote Application**, Remote debugging.

DI – Dependency Injection

**DI and Autowiring using Spring Boot.**

@Component

Employee

@Autowired

Address address

@Component

Address

Container

address

(“add”)

Note 🡪 @SpringBootApplication this annotation performs three tasks.

1. configuration
2. EnableAutoConfiguration
3. ComponentScan

For above all three annotation only one annotation that is @SpringBootApplication works