UNIT - 5

Spring Boot

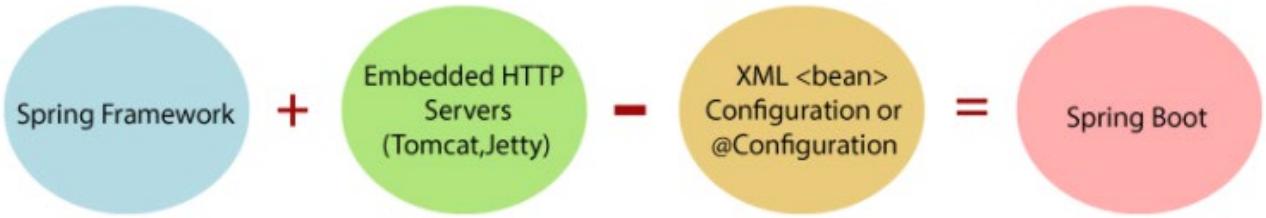
Spring Boot is a project that is built on the top of the Spring Framework.

It provides an **easier and faster** way to set up, configure, and run both simple and web-based applications.

Spring Boot is built on **top of** the conventional spring framework. So, it provides all the **features of spring** and is yet easier to use than spring.

Spring Boot is a **microservice-based framework** and making a production-ready application in very less time. In Spring Boot everything is **auto-configured**. We just need to use proper configuration for utilizing a particular functionality. Spring Boot is very useful if we want to develop **REST API**.

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.



In short, Spring Boot is the combination of **Spring Framework and Embedded Servers**.

In Spring Boot, there is **no requirement** for **XML configuration** (deployment descriptor). It uses **convention** over configuration **software design paradigm** that means it decreases the effort of the developer.

We can use **Spring STS IDE or Spring Initializr** to develop Spring Boot Java applications.

Why should we use Spring Boot Framework?

We should use Spring Boot Framework because:

- The **dependency injection** approach is used in Spring Boot.

- It contains powerful **database transaction management capabilities**.
- It **simplifies integration** with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
- It **reduces** the cost and development time of the application.

Advantages of Spring Boot

- It creates **stand-alone Spring applications** that can be started using Java -jar.
- It tests web applications easily with the help of **different Embedded HTTP servers** such as Tomcat, Jetty, etc. We don't need to deploy WAR files.
- It provides **opinionated 'starter' POMs** to simplify our Maven configuration.
- It provides **production-ready features** such as metrics, health checks, and externalized configuration.
- There is **no requirement** for XML configuration.
- It offers a **CLI tool** for developing and testing the Spring Boot application.
- It offers the number of **plug-ins**.
- It also **minimizes** writing multiple boilerplate codes (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
- It **increases productivity** and **reduces development time**.

Limitations of Spring Boot

- Spring Boot can use dependencies that are not going to be used in the application. These dependencies **increase** the **size of the application**.

Difference between Spring and Spring Boot: -

S.No.	Spring	Spring Boot
1.	Spring is an open-source lightweight framework widely used to develop **enterprise applications**.	Spring Boot is built on top of the conventional spring framework, widely used to develop **REST APIs**.

2.	The most important feature of the Spring Framework is **dependency injection**.	The most important feature of the Spring Boot is **Autoconfiguration**.
3.	It helps to create a **loosely coupled application**.	It helps to create a **stand-alone application**.
4.	To run the Spring application, we need to **set** the **server explicitly**.	Spring Boot **provides embedded servers** such as Tomcat and Jetty etc.
5.	To run the Spring application, a **deployment descriptor is required**.	There is **no requirement** for a deployment descriptor.
6.	To create a Spring application, the developers write **lots of code**.	It **reduces** the lines of code.
7.	It **doesn't provide support** for the **in-memory database**.	It **provides support** for the in-memory database such as H2.

Goals of Spring Boot

The main goal of Spring Boot is to reduce development, unit test, and integration test time.

- Provides Opinionated Development approach
- Avoids defining more Annotation Configuration
- Avoids writing lots of import statements

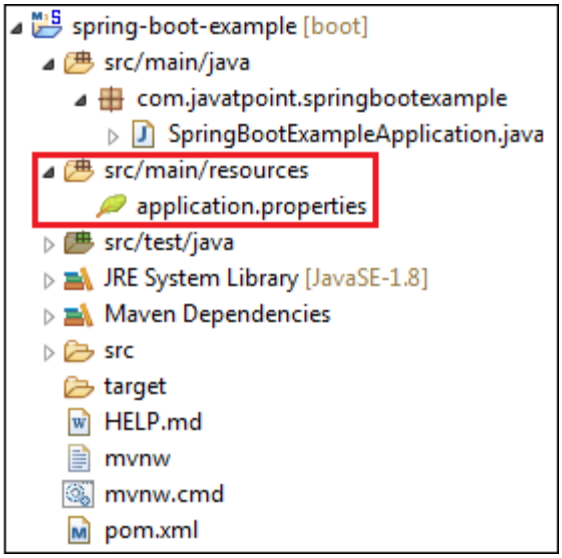
- Avoids XML Configuration.

Spring Boot Features

- Web Development
- Spring Application
- Application events and listeners
- Admin features
- Externalized Configuration
- Properties Files
- YAML Support
- Type-safe Configuration
- Logging
- Security

Spring Boot Application Properties

Spring Boot Framework comes with a built-in mechanism for application configuration using a file called **application.properties**. It is located inside the **src/main/resources** folder, as shown in the following figure.



Spring Boot provides various properties that can be configured in the **application. properties** file. The properties have default values. We can set a property(s) for the Spring Boot application. Spring Boot also allows us to define our own property if required.

The application.properties file allows us to run an application in a **different environment.** In short, we can use the application. properties file to:

- Configure the Spring Boot framework
- define our application custom configuration properties

Example of application.properties

1. #configuring application name

   spring.application.name = demoApplication
2. #configuring port

   server.port = 8081

In the above example, we have configured the **application name** and **port**. The port 8081 denotes that the application runs on port **8081**.

YAML Properties File

Spring Boot provides another file to configure the properties is called **yml** file. The **Yaml file** works because the **Snake YAML** jar is present in the classpath. Instead of using the application.properties file, we can also use the application.yml file, but the **Yml** file should be present in the classpath.

Example of application.yml

1. spring:
2. application:
3. name: demoApplication

4. server:
5. port: 8081

Spring Boot Actuator provides secured endpoints for monitoring and managing your Spring Boot application. By default, all actuator endpoints are secured.

Enabling Spring Boot Actuator

To enable Spring Boot actuator endpoints to your Spring Boot application, we need to add the Spring Boot Starter actuator dependency in our build configuration file.

Maven users can add the below dependency in your pom.xml file.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
        <groupId>com.squareup.picasso</groupId>
        <artifactId>picasso</artifactId>
        <version>(insert latest version)</version>
</dependency>
```

Gradle users can add the below dependency in your **build.gradle** file.

```
compile group: 'org.springframework.boot', name: 'spring-boot-starter-actuator'
```

In the application.properties file, we need to disable the security for actuator endpoints.

```
management.security.enabled = false
```

```
implementation 'com.squareup.picasso:picasso:(insert latest version)'
```

Some important Spring Boot Actuator endpoints are given below. You can enter them in your web browser and monitor your application behavior.

ENDPOINTS	USAGE
/metrics	To view the application metrics such as memory used, memory free, threads, classes, system uptime etc.
/env	To view the list of Environment variables used in the application.
/beans	To view the Spring beans and its types, scopes and dependency.
/health	To view the application health
/info	To view the information about the Spring Boot application.
/trace	To view the list of Traces of your Rest endpoints.

HTTP Methods in Spring RESTful Services

Representational state transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services (or simply **RESTful services**).

RESTful services enable us to develop any kind of application involving all possible CRUD (create, retrieve, update, delete) operations. We should utilize the different HTTP verbs which correspond to CRUD operations. The primary or most-commonly-used HTTP methods are GET, POST, PUT, PATCH, and DELETE. In performing these operations in **RESTful** services, there are guidelines or principles that suggest using a specific HTTP method on a specific type of call made to the server.

Below is a table summarizing primary HTTP methods and it's recommendations for **RESTful** services:

HTTP Verb	**CRUD**	**Sample**	**Description**	**Result**
GET	Read	/books	list all books. Pagination, sorting and filtering is used for big lists.	**200 (OK)**
		/books/{id}	get specific book by id	**200 (OK)** **404 (Not Found)** - id not found or invalid
POST	Create	/books	create new book	**201 (Created)** - return a Location header with a link to the newly-created resource /books/{id}. **409 (Conflict)** - indicated that resource already exists
		/books/id	Return **405 (Method Not Allowed)**, avoid using POST on single resource	
PUT	Update/Replace	/books	Return **405 (Method Not Allowed)**, unless you want to **replace every resource** in the entire collection of resource - use with caution.	
		/books/{id}	Update a book	**200 (OK)** or **204 (No Content)** - indicate successful completion of the request **201 (Created)** - if new resource is created and creating new resource is allowed **404 (Not Found)** - if id not found or invalid and creating new resource is not allowed.
PATCH	Partial Update	/books	Return **405 (Method Not Allowed)**, unless you want to modify the collection itself..	
		/books/{id}	Partial update a book	**200 (OK)** or **204 (No Content)** - indicate successful completion of the request **404 (Not Found)** - if id not found or invalid
DELETE	Delete	/books	Return **405 (Method Not Allowed)**, unless you want	

			to **delete the whole collection** - use with caution	
		/books/{id}	Delete a book	**200 (OK)** or **204 (No Content)** or **202 (Accepted)** **404 (Not Found)** - if id not found or invalid