Spring 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 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.

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.

REST API

Representational **S**tate **T**ransfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. **REST API** is a way of accessing web services in a simple and flexible way without having any processing.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth, simple and flexible making it more suitable for internet usage. It's used to fetch or give some information from a web service. All communication done via REST API uses only HTTP request.

Working: A request is sent from client to server in the form of a web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from the server in the form of a resource which can be anything like HTML, XML, Image, or JSON. But now JSON is the most popular format being used in Web Services.

In **HTTP** there are five methods that are commonly used in a REST-based Architecture i.e., POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations respectively. There are other methods which are less frequently used like OPTIONS and HEAD.

- **GET:** The HTTP GET method is used to **read** (or retrieve) a representation of a resource. In the safe path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

- **POST:** The POST verb is most often utilized to **create** new resources. In particular, it's used to create subordinate resources. That is, subordinate to some other (e.g. parent) resource. On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.

NOTE: *POST is neither safe nor idempotent.*

- **PUT:** It is used for **updating** the capabilities. However, PUT can also be used to **create** a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID. On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. PUT is not safe operation but it's idempotent.
- **PATCH:** It is used to **modify** capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource. This resembles PUT, but the body contains a set of instructions describing how a resource currently residing on the server should be modified to produce a new version. This means that the PATCH body should not just be a modified part of the resource, but in some kind of patch language like JSON Patch or XML Patch. PATCH is neither safe nor idempotent.
- **DELETE:** It is used to **delete** a resource identified by a URI. On successful deletion, return HTTP status 200 (OK) along with a response body.

Idempotence: An idempotent HTTP method is a HTTP method that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over. The result should be the same. Again, this only applies to the result, not the resource itself.

Example:

- C

1. a = 4 // It is Idempotence, as final value(a = 4)

   // would not change after executing it multiple

   // times

2. a++ // It is not Idempotence because the final value

   // will depend upon the number of times the

   // statement is executed.

Request and Response

Now we will see how request and response work for different **HTTP** methods. Let's assume we have an **API(**https://www.geeksforgeeks.org/api/students**)** for all students data of gfg.

- **GET:** Request for all Students.

  Request

  GET:/api/students

- **POST:** Request for Posting/Creating/Inserting Data

  Request

  POST:/api/students

  {"name":"Raj"}

- **PUT or PATCH:** Request for Updating Data at id=1

  Request

  PUT or PATCH:/api/students/1

  {"name":"Raj"}

- **DELETE:** Request for Deleting Data of id=1

  Request

  DELETE:/api/students/1

RESTful web services are very popular because they are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications.