**What is Spring Boot?**

Spring Boot is a Framework from “The Spring Team” to ease the bootstrapping and development of new Spring Applications. It provides defaults for code and annotation configuration to quick start new Spring projects within no time

### Advantages of Spring Boot:

* It is very easy to develop Spring Based applications with Java or Groovy.
* It reduces lots of development time and increases productivity.
* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
* It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
* It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.
* It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle
* It provides lots of plugins to work with embedded and in-memory Databases very easily.

In Simple Terminology, What Spring Boot means[What Is Spring Boot, Spring Boot Tutorial](https://journaldev.nyc3.digitaloceanspaces.com/2015/05/WhatIsSpringBoot1.png)That means Spring Boot is nothing but existing Spring Framework + Some Embedded HTTP Servers (Tomcat/Jetty etc.) - XML or Annotations Configurations. Here minus means we don’t need to write any XML Configuration and few Annotations only.

**Main Goal of Spring Boot:**

The main goal of Spring Boot Framework is to reduce Development, Unit Test and Integration Test time and to ease the development of Production ready web applications very easily compared to existing Spring Framework, which really takes more time.

* To avoid XML Configuration completely
* To avoid defining more Annotation Configuration(It combined some existing Spring Framework Annotations to a simple and single Annotation)
* To avoid writing lots of import statements

SpringBoot Annotations:

* **Basic Setup**
  + @SpringBootApplication
  + @Configuration
  + @ComponentScan
  + @EnableAutoConfiguration
* **Request Responses**
  + @GetMapping
  + @RequestMapping
  + @RequestParam
* **Component Types**
  + @Component
  + @Service
  + @Repository
  + @Controller
  + @RestController

**@EnableAutoConfiguration:** It auto-configures the bean that is present in the classpath .

**@SpringBootApplication:** It is a combination of three annotations **@EnableAutoConfiguration, @ComponentScan,** and **@Configuration**.

**@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**.

Example:

1. @Controller
2. @RequestMapping("/book")
3. **public** **class** BooksController
4. {
5. @RequestMapping("/computer-science/books")
6. **public** String getAllBooks(Model model)
7. {
8. //application code
9. **return** "bookList";
10. }

**@RestController:** It can be considered as a combination of **@Controller** +Restfull webservices methods

* **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
* **@PostMapping:** It maps the **HTTP POST**requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
* **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
* **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes**a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**

**@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request.

**@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

**@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable.

**@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**.

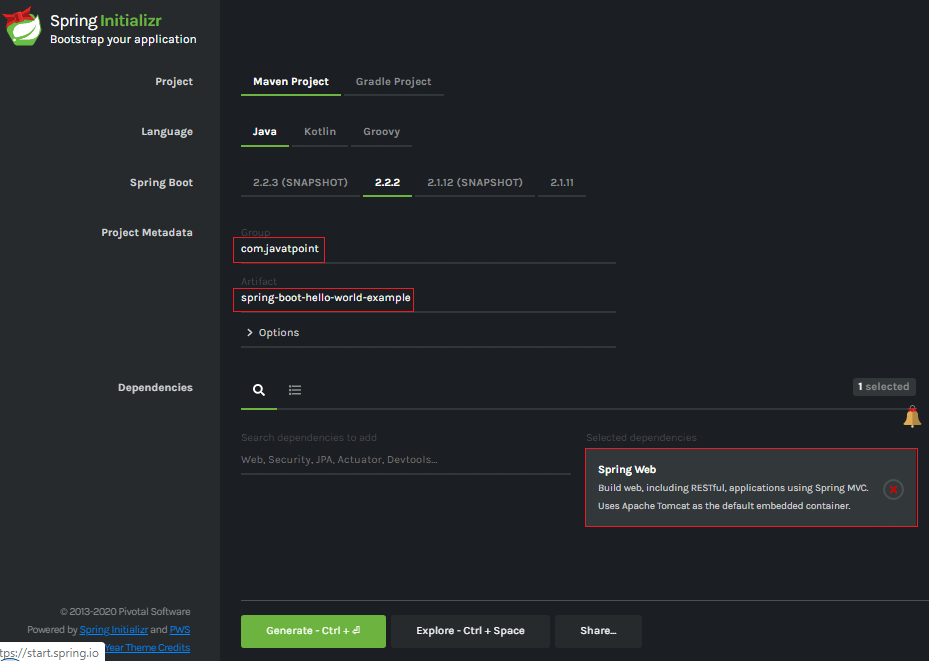
**Step 1:** Open Spring Initializr <https://start.spring.io/>.

**Step 2:** Provide the **Group** name. We have provided **com.dxe.**

**Step 3:** Provide the **Artifact** Id. We have provided the **spring-boot-hello-world-example.**

**Step 4:** Add the dependency **Spring Web.**

**Step 5:** Click on the **Generate** button. When we click on the Generate button, it wraps all the specifications into a jar file and downloads it to our local system.

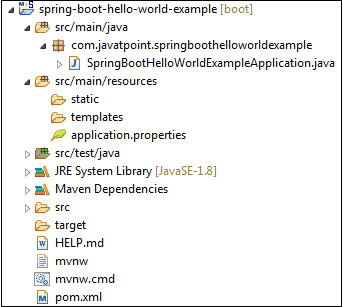


**Step 6: Extract** the RAR file.

**Step 7: Import** the project folder by using the following steps:

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

When the project imports successfully, it shows the following project directory in the Package Explorer section of the IDE.



**Step 8:** Create a package with the name **com.dxe.controller** inside the folder **src/main/java.**

**Step 9:** Create a Controller class with the name **HelloWorldController.**

**Step 10:** Create a method named **hello()**that returns a String.

**HelloWorldController.java**

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@RestController

**public** **class** HelloWorld {

@RequestMapping("/")

String demo()

{

**return** "demodataaaaa";

}

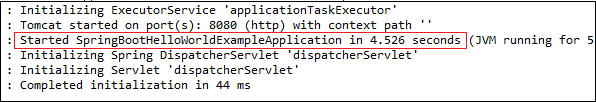
}

**Step 11:** Run the **SpringBootHelloWorldExampleApplication.java** file.

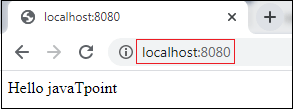
**SpringBootHelloWorldExampleApplication.java**

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

When the application runs successfully, it shows a massage in the console, as shown in the following figure.



**Step 12:** Open the browser and invoke the URL **https://localhost:8080**. It returns a String that we have specified in the Controller.



Github:

https://github.com/goud89105/springprojects

RestFull WebServices:

RESTful Web Services are client and server applications that communicate over the WWW. RESTful Web Services are REST Architecture based Web Services. In REST Architecture, everything is a resource. RESTful Web Services provides communication between software applications running on different platforms and frameworks. We can consider web services as code on demand.

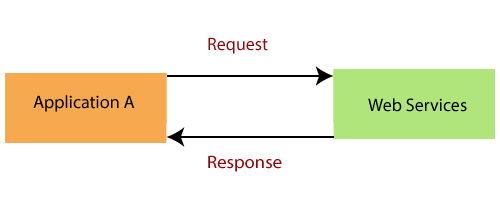
## Components of Web Services

The web services must be able to fulfill the following conditions:

* The web service must be accessible over the internet.

## How does data exchange between applications?

Suppose, we have an **Application A** which create a request to access the **web services**. The web services offer a list of services. The web service process the **request** and sends the **response** to the Application A. The input to a web service is called a request, and the output from a web service is called response. The web services can be called from different platforms.



# Introduction to RESTful Web Services

REST stands for **REpresentational State Transfer**. It is developed by **Roy Thomas Fielding**, who also developed HTTP. The main goal of RESTful web services is to make web services **more effective**. RESTful web services try to define services using the different concepts that are already present in HTTP. REST is an **architectural approach**, not a protocol.

It does not define the standard message exchange format. We can build REST services with both XML and JSON. JSON is more popular format with REST. The **key abstraction** is a resource in REST. A resource can be anything. It can be accessed through a **Uniform Resource Identifier (URI)**. For example:

The resource has representations like XML, HTML, and JSON. The current state capture by representational resource. When we request a resource, we provide the representation of the resource. The important methods of HTTP are:

* **GET:** It reads a resource.
* **PUT:** It updates an existing resource.
* **POST:** It creates a new resource.
* **DELETE:** It deletes the resource.

For example, if we want to perform the following actions in the social media application, we get the corresponding results.

**POST /users:** It creates a user.

**GET /users/{id}:** It retrieves the detail of a user.

**GET /users:** It retrieves the detail of all users.

**DELETE /users:** It deletes all users.

**DELETE /users/{id}:** It deletes a user.

**GET /users/{id}/posts/post\_id:** It retrieve the detail of a specific post.

**POST / users/{id}/ posts:** It creates a post of the user.

Further, we will implement these URI in our project.

HTTP also defines the following standard status code:

* **404:** RESOURCE NOT FOUND
* **200:** SUCCESS
* **201:** CREATED
* **401:** UNAUTHORIZED
* **500:** SERVER ERROR

## RESTful Service Constraints

* There must be a service producer and service consumer.
* The service is stateless.
* The service result must be cacheable.
* The interface is uniform and exposing resources.
* The service should assume a layered architecture.

## Advantages of RESTful web services

* RESTful web services are **platform-independent**.
* It can be written in any programming language and can be executed on any platform.
* It provides different data format like **JSON, text, HTML,** and **XML**.
* It is fast in comparison to SOAP because there is no strict specification like SOAP.
* These are **reusable**.
* They are **language neutral**.

# Key Terminology of Web Services

* Request and Response
* Message Exchange Format: XML and JSON
* Service Provider or Server
* Service Consumer or Client
* Service Definition
* Transport: HTTP and MQ

**Request and Response:** Request is the input to a web service, and the response is the output from a web service.

**Message Exchange Format:** It is the format of the request and response. There are two popular message exchange formats: **XML** and **JSON**.

**Service Provider or Server:** Service provider is one which hosts the web service.

**Service Consumer or Client:** Service consumer is one who is using the web service.

JSON:

## What is JSON?

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight data-interchange format
* JSON is plain text written in JavaScript object notation
* JSON is used to send data between computers

## JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

* Data is in name/value pairs
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

# JSON Example

JSON example can be created by object and array. Each object can have different data such as text, number, boolean etc. Let's see different JSON examples using object and array.

### JSON Object Example

A JSON object contains data in the form of key/value pair. The keys are strings and the values are the JSON types. Keys and values are separated by colon. Each entry (key/value pair) is separated by comma.

The **{** (curly brace) represents the JSON object.

1. {
2. "employee": {
3. "name":       "sonoo",
4. "salary":      56000,
5. "married":    **true**
6. }
7. }

### JSON Array example

The **[** (square bracket) represents the JSON array. A JSON array can have values and objects.

Let's see the example of JSON array having values.

1. ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

Let's see the example of JSON array having objects.

1. [
2. {"name":"Ram", "email":"Ram@gmail.com"},
3. {"name":"Bob", "email":"bob32@gmail.com"}
4. ]

## JSON Example 1

1. {"employees":[
2. {"name":"Shyam", "email":"shyamjaiswal@gmail.com"},
3. {"name":"Bob", "email":"bob32@gmail.com"},
4. {"name":"Jai", "email":"jai87@gmail.com"}
5. ]}

The XML representation of above JSON example is given below.

1. **<employees>**
2. **<employee>**
3. **<name>**Shyam**</name>**
4. **<email>**shyamjaiswal@gmail.com**</email>**
5. **</employee>**
6. **<employee>**
7. **<name>**Bob**</name>**
8. **<email>**bob32@gmail.com**</email>**
9. **</employee>**
10. **<employee>**
11. **<name>**Jai**</name>**
12. **<email>**jai87@gmail.com**</email>**
13. **</employee>**
14. **</employees>**

## JSON Example 2

1. {"menu": {
2. "id": "file",
3. "value": "File",
4. "popup": {
5. "menuitem": [
6. {"value": "New", "onclick": "CreateDoc()"},
7. {"value": "Open", "onclick": "OpenDoc()"},
8. {"value": "Save", "onclick": "SaveDoc()"}
9. ]
10. }
11. }}

The XML representation of above JSON example is given below.

1. **<menu** id="file" value="File"**>**
2. **<popup>**
3. **<menuitem** value="New" onclick="CreateDoc()" **/>**
4. **<menuitem** value="Open" onclick="OpenDoc()" **/>**
5. **<menuitem** value="Save" onclick="SaveDoc()" **/>**
6. **</popup>**
7. **</menu>**

# JSON Object

JSON object holds key/value pair. Each key is represented as a string in JSON and value can be of any type. The keys and values are separated by colon. Each key/value pair is separated by comma.

The curly brace **{** represents JSON object.

Let's see an example of JSON object.

1. {
2. "employee": {
3. "name":       "sonoo",
4. "salary":      56000,
5. "married":    **true**
6. }
7. }

In the above example, employee is an object in which "name", "salary" and "married" are the key. In this example, there are string, number and boolean value for the keys.

## JSON Object with Strings

The string value must be enclosed within double quote.

1. {
2. "name":       "sonoo",
3. "email":      "sonoojaiswal1987@gmail.com"
4. }

## JSON Object with Numbers

JSON supports numbers in double precision floating-point format. The number can be digits (0-9), fractions (.33, .532 etc) and exponents (e, e+, e-,E, E+, E-).

1. {
2. "integer": 34,
3. "fraction": .2145,
4. "exponent": 6.61789e+0
5. }

## JSON Object with Booleans

JSON also supports boolean values true or false.

1. {
2. "first": **true**,
3. "second": **false**
4. }

## JSON Nested Object Example

A JSON object can have another object also. Let's see a simple example of JSON object having another object.

1. {
2. "firstName": "Sonoo",
3. "lastName": "Jaiswal",
4. "age": 27,
5. "address" : {
6. "streetAddress": "Plot-6, Mohan Nagar",
7. "city": "Ghaziabad",
8. "state": "UP",
9. "postalCode": "201007"
10. }
11. }

# JSON Array

JSON array represents ordered list of values. JSON array can store multiple values. It can store string, number, boolean or object in JSON array.

In JSON array, values must be separated by comma.

The **[** (square bracket) represents JSON array.

## JSON Array of Strings

Let's see an example of JSON arrays storing string values.

x

1. ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

## JSON Array of Numbers

Let's see an example of JSON arrays storing number values.

1. [12, 34, 56, 43, 95]

## JSON Array of Booleans

Let's see an example of JSON arrays storing boolean values.

1. [**true**, **true**, **false**, **false**, **true**]

## JSON Array of Objects

Let's see a simple JSON array example having 4 objects.

1. {"employees":[
2. {"name":"Ram", "email":"ram@gmail.com", "age":23},
3. {"name":"Shyam", "email":"shyam23@gmail.com", "age":28},
4. {"name":"John", "email":"john@gmail.com", "age":33},
5. {"name":"Bob", "email":"bob32@gmail.com", "age":41}
6. ]}

# JSON Comments

JSON doesn't support comments. It is not a standard.

But you can do some tricks such as adding extra attribute for comment in JSON object, for example:

1. {
2. "employee": {
3. "name":       "Bob",
4. "salary":      56000,
5. "comments":    "He is a nice man"
6. }
7. }