SPRING BOOT WEB SERVICES

A Web Service is can be defined by following ways

It is a client-server application or application component for communication.

The method of communication between two devices over the network.

It is a software system for the interoperable machine to machine communication.

It is a collection of standards or protocols for exchanging information between two devices or application.

Types of Web Services :

- **SOAP web services.**
- **RESTful web services.**

Web Service Components

There are three major web service components.

1. SOAP
2. WSDL
3. UDDI

SOAP

SOAP is an acronym for Simple Object Access Protocol.

SOAP is a XML-based protocol for accessing web services.

SOAP is a W3C recommendation for communication between applications.

SOAP is XML based, so it is platform independent and language independent. In other words, it can be used with Java, .Net or PHP language on any platform.

WSDL

WSDL is an acronym for Web Services Description Language.

WSDL is a xml document containing information about web services such as method name, method parameter and how to access it.

WSDL is a part of UDDI. It acts as a interface between web service applications.

WSDL is pronounced as wiz-dull.

UDDI

UDDI is an acronym for Universal Description, Discovery and Integration.

UDDI is a XML based framework for describing, discovering and integrating web services.

UDDI is a directory of web service interfaces described by WSDL, containing information about web services.

SOAP vs REST Web Services

There are many differences between SOAP and REST web services. The important 10 differences between SOAP and REST are given below:

No.	SOAP	REST
1)	SOAP is a **protocol**.	REST is an **architectural style**.
2)	SOAP stands for **Simple Object Access Protocol**.	REST stands for **REpresentational State Transfer**.
3)	SOAP **can't use REST** because it is a protocol.	REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
4)	SOAP **uses services interfaces to expose the business logic**.	REST **uses URI to expose business logic**.
5)	**JAX-WS** is the java API for SOAP web services.	**JAX-RS** is the java API for RESTful web services.
6)	SOAP **defines standards** to be strictly followed.	REST does not define too much standards like SOAP.
7)	SOAP **requires more bandwidth** and resource than REST.	REST **requires less bandwidth** and resource than SOAP.
8)	SOAP **defines its own security**.	RESTful web services **inherits security measures** from the underlying transport.
9)	SOAP **permits XML** data format only.	REST **permits different** data format such as Plain text, HTML, XML, JSON etc.
10)	SOAP is **less preferred** than REST.	REST **more preferred** than SOAP.

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.

Implementing the POST Method to create User Resource

In the previous few steps, we have created simple RESTful services. In this section, we will use the POST method to post the user resource for the specific URI "**/users**."

Here we are using two annotations, **@RequestBody** and **@PathMapping**.

@RequestBody

The @RequestBody annotation maps body of the web request to the method parameter. The body of the request is passed through an HttpMessageConverter. It resolves the method argument depending on the content type of the request. Optionally, automatic validation can be applied by annotating the argument with @Valid.

@PathMapping

The @PathMapping annotation is the specialized version of the **@RequestMapping** annotation which acts as a shortcut for **@RequestMapping(method=RequestMethod=POST)**. @PostMapping method handles the Http POST requests matched with the specified URI.

Connecting RESTful Services to JPA

Creating a User Entity and some test Data

Let's create a User entity and a UserRepository so that we can access the detail of the user.

Step 1: Open **pom.xml** file and add **spring-boot-starter-data-jpa** dependency.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Step 2: Make the **User** class as an entity by adding an annotation **@Entity** just above the User class.

@Entity: Entities are nothing but POJO (Plain Old Java Object). It represents the data that can be persisted to the database. It represents a table in a database. Every instance of an entity represents a row in a table. We cannot declare an entity class as **final**.

Step 3: Make the Id primary key by adding an annotation @Id just above the Id variable. Also, add an annotation **@GeneratedValue**.

@Id: It defines that the member field below is the primary key of the current entity. Every entity must have a primary key that uniquely defines the column.

@GeneratedValue: The @GeneratedValue annotation may be applied to a primary key property or field of an entity with the **@Id** annotation. It is used to support the primary key. We have to add the **@GeneratedValue** annotation to the primary key attribute and choose a generation type. The default generation type is GenerationType.AUTO.

Before moving to the next step, **remove** or **comment** the basic security dependency in the **pom.xml**.

Step 4: Open **application.properties** file and enable the **H2 console** so that we can see what data we have inserted in the table.

```
spring.h2.console.enabled=true
```

We also need to enable the **SQL logging** to see which SQL statements are executing. It starts SQL logging in the log when the statement executes.

```
1.  spring.jpa.show-sql=true
```

Now restart the application to pick up the changes. We can see in the following image that the table is created.

Step 5: We have to create a SQL file to insert data in the user table.

Right-click on the folder **src/main/resource** -> New ->Other -> Select SQL File -> provide the file name **data.sql ->** Click on finish.

Step 6: In the data.sql file, insert the data into **user** table. We have inserted the following data:

```
1.  insert into user values(1, sysdate(), 'John');
2.  insert into user values(2, sysdate(),'Robert');
3.  insert into user values(3, sysdate(), 'Andrew');
```

Run the application.

Step 7: Open the browser and type http://localhost:8080/h2-console to connect the H2 console. Make sure that it has JDBC URL: **jdbc:h2:mem:testdb**. Do not write anything in the password field.

Step 8: Click on the **Connect** button to login. The table which we have created appears on the left-hand side of the page.

Step 10: Type the query **select * from user**;