RESTful Web services using Spring

Definition of a Web Service

W3C: Software system designed to support interoperable machine-to-machine interaction over a network

- Platform Agnostic
- Allow communication over a network

> *NOTE: A ServiceDefinition comprises of 4 important parts*
>
> - *Request Response Format*
> - *Request Structure*
> - *Response Structure*
> - *Endpoint*

NOTE: REST was introduced by Roy Fielding, who also created HTTP

Spring Autowiring

In order to inject Spring into a Java Application, we need to ask the following questions

- *What are Beans ?*
- *What are the dependencies of the Beans ?*
- *Where to search for the Beans ?*

What are the Beans ?

In the Context of Spring, You mark a Class as a Bean, by using annotation `@Component`

```java
package com.udemy.spring.essential.intro;

import com.udemy.spring.essential.intro.interfaces.Sortable;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
```

```
// In spring - Bean is annotated with @Component
@Component
public class BinarySearchImpl {

    Logger logger = LoggerFactory.getLogger(BinarySearchImpl.class);

   // Dependencies (Interfaces) are defined using the annotation
@Autowired
    @Autowired
    Sortable sortAlgorithm;


   // constructor injection
    public BinarySearchImpl (Sortable algorithm) {
        this.sortAlgorithm = algorithm;
    }

    public int binarySearch(int [] arr, int num) {

        // Sort the Arr
        logger.info("Sorting the array: " + arr);
        sortAlgorithm.sort(arr);

        // Binary Search

        // Return the number
        return num;
    }

}
```

What are the dependences of a Bean ?

In the Context of Spring, You mark a dependency of a Bean, by using annotation `@Autowired` on the field

```
    ...
    ...
    @Autowired
    Sortable sortAlgorithm;

    // constructor injection
    public BinarySearchImpl (Sortable algorithm) {
        this.sortAlgorithm = algorithm;
    }
```

Where to search for the Bean ?

You can inform Spring to search for the bean by using the `ApplicationContext` class.

For the main application, you get the `` `ApplicationContext `` using -

```
ApplicationContext appContext =
SpringApplication.run(IntroApplication.class, args);
```

You can then inform Spring to search for dependencies using the `getBean()` method

```
package com.udemy.spring.essential.intro;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;



@SpringBootApplication
public class IntroApplication {

  public static void main(String[] args) {
    ApplicationContext appContext =
SpringApplication.run(IntroApplication.class, args);

    // Binary Search Implementation
    // BinarySearchImpl binarySearch = new BinarySearchImpl(new
BubbleSortAlgorithm());

    // Application Context - which will maintain all the beans
    BinarySearchImpl binarySearch =
appContext.getBean(BinarySearchImpl.class);

    int result = binarySearch.binarySearch(new int[] {1,2,3}, 3);
    System.out.println("Completed run: " + result);
  }

}
```

> *NOTE: By default in Spring, it searches for the dependencies in the root package & it's sub-packages*

Injection - Constructor & Setter Injection

By **default**, all beans in Spring are created as **singletons**, which means they will be created in a container once and the same object will be **injected** anywhere it is requested.

- Constructor - this uses the Bean constructor and is explicitly mentioned in the logs
- Setter (default) - You can either specify the setter OR spring will by default inject

> *All mandatory dependencies SHOULD be defined using Constructor injection*

Constructor Injection	Setter Injection
Constructor injection uses the constructor to inject dependency on any Spring-managed bean.	Setter injection in Spring uses setter methods like setDependency() to inject dependency on any bean managed by Spring's IOC container.
Constructor injection, since it uses an `index` as position to inject the dependency, it's not as readable as setter injection and you need to refer either Java documentation or code to find which index corresponds to which property.	Setter Injection in more readable than constructor injection in Spring configuration file usually the `applicationContext.xml`
Constructor injection ensures - Dependency injection, as the bean will not get created until Dependencies have been injected ie., constructor Injection does not allow you to construct an object until your dependencies are ready.	One of the drawbacks of setter injection is that it does not ensures dependency Injection
Constructor injection is preferred as more **secure** because every time you call the constructor, a new object is gets created and therefore you cannot override the class	One more drawback of setter Injection is Security. By using setter injection, you can override certain dependency which is not possible with constructor injection.

Intro to Spring Boot

Important Links

- Spring boot tutorial
- Spring boot interview Questions
- Spring vs Spring Boot vs Spring MVC
- Adding Swagger Documentation to Spring MVC project

First 10 Steps in Spring Boot

- **Step 1** : Introduction to Spring Boot - Goals and Important Features
- **Step 2** : Developing Spring Applications before Spring Boot
- **Step 3** : Using Spring Initializr to create a Spring Boot Application
- **Step 4** : Creating a Simple REST Controller
- **Step 5** : What is Spring Boot Auto Configuration?
- **Step 6** : Spring Boot vs Spring vs Spring MVC

- **Step 7** : Spring Boot Starter Projects - Starter Web and Starter JPA
- **Step 8** : Overview of different Spring Boot Starter Projects
- **Step 9** : Spring Boot Actuator
- **Step 10** : Spring Boot Developer Tools
- **Spring Boot** - Conclusion

Spring Boot Features - What is, What not

Enable building production ready applications quickly
Provide common non-functional features

- embedded servers - *Tomcat, Jetty & Undertow*
- metrics
- health checks - *through actuators*
- externalized configuration - *Quickly create property files*
- You can quickly start working with starter templates - *for Web / JPA*

What Spring Boot is NOT

- There is **ZERO** code generation in Spring boot
- Spring boot is *not an Application Server nor is it a Web Server*

What is Auto-Configuration

In Spring when we use the annotation `@SpringBootApplication` . It does 3 things -

- Indicates its a Spring context file
- Enables Auto-configuration
- Enables Component-Scanner

> *NOTE:* `SpringApplication.run(...)` *returns an* ***ApplicationContext*** *instance*

```
ApplicationContext appContext =
SpringApplication.run(WebApplication.class, args);

// print all the beans - for Each loop
for (String name: appContext.getBeanDefinitionNames()) {
  System.out.println("Bean names: " + name);
}
```

How does Spring boot auto-configure my project ?

Spring boot looks at

- Frameworks available on the CLASSPATH
- Existing configuration for the application

Based on these factors, Spring boot provides basic configuration needed to configure the project with these frameworks. This is called *Auto-configuration.*

In order to enable server side logging, got to application properties file in resources

```
logging.level.org.springframework=debug
```

Spring vs Spring boot vs Spring MVC

Spring Boot

Spring Boot does not compete with Spring or Spring MVC. It makes it easy to use them.

Spring Framework

Most important feature of Spring Framework is Dependency Injection.

At the core of all Spring Modules is Dependency Injection or IOC Inversion of Control.

Spring MVC

Spring MVC Framework provides decoupled way of developing web applications. With simple concepts like Dispatcher Servlet, ModelAndView and View Resolver, it makes it easy to develop web applications.

None of the above are competing among themselves. They each solve a specific problem

Swagger Documentation

Adding Swagger Documentation to our Spring MVC

To have springdoc-openapi automatically generate the OpenAPI 3 specification docs for our API, we simply add the springdoc-openapi-ui dependency to our *pom.xml*:

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.5.2</version>
</dependency>
```

Then when we run our application, the OpenAPI descriptions will be available at the path */v3/api-docs* by default:

```
http://localhost:8080/v3/api-docs/
```

To use a custom path, we can indicate in the *application.properties* file:

```
springdoc.api-docs.path=/api-docs
```

Now we'll be able to access the docs at:

```
http://localhost:8080/api-docs/
```

The OpenAPI definitions are in JSON format by default. For *yaml* format, we can obtain the definitions at:

```
http://localhost:8080/api-docs.yaml
```

Now we can access the Swagger API documentation at:

```
http://localhost:8080/swagger-ui.html
```

Spring Boot Actuator

Actuator brings in - a lot of monitoring around the Web Application

- In Actuator, you would be able to read alot of meta-data around the application
- Number of times a specific service is called
- Number of times a service has failed

You need the following 2 Dependencies, to activate your actuator

```
    <!-- Logging & Monitoring services -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <!-- HAL browser -->
    <dependency>
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-rest-hal-browser</artifactId>
      <version>3.3.6.RELEASE</version>
    </dependency>
```

You need to also enable the following configurations in the **application.properties** file

```
# logging.level.spring.framework = DEBUG
# management.security.enabled = FALSE

# Exposing on http all the management endpoints
management.endpoints.web.exposure.include=*
```

The HAT Browser can be accessed on restarting your Spring application

```
http://localhost:8080/browser/index.html#/
```

Spring Boot Dev Tools

You can also enable Sprint boot Dev Tools (maven dependency) which behaves very similar to Node JS Nodemon

```
<!-- Spring boot dev tools -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dev-tools</artifactId>
    </dependency>
```