

Spring RESTful Web Services

By Chandrasekaran Janardhanan

Contents

Session pre-requisite	4
Install and Setup JDK, Tomcat and Eclipse.....	5
Understand the need for Maven.....	7
Display Hello World in Spring Boot Application	11
Load languages for international news media website.....	14
Understanding Dependency Injection	22
Implement Dependency Injection using Autowiring.....	25
Miscellaneous topics related to Spring XML based configuration.....	27
Implement Dependency Injection using Annotations	30
Miscellaneous topics related to Spring Core Framework	34
Understand how internet works.....	37
Understanding what is a Single Page Application (SPA).....	40
Hello World Servlet	44
Understanding the Servlet Life Cycle.....	46
Display Hello World using PrintWriter.....	48
Sending values to server	49
Download text file using Servlet.....	50
Create a RESTful Web Service using Servlet	51
Create a RESTful Web service that sends Hello World greetings message	52
Create a RESTful Web service to get all languages for the media web site	55
Test the language service using MockMvc	57
Get language based on language code	59
Return appropriate response when a language code is not found.....	61
Add language using POST	63
Validate data for POST request	67
Create REST API to update language	71
Create REST API to delete language.....	73
Listing movies for a movie booking SPA	76
Best Bus Case Study	80
Get a specific movie detail for editing	81
Save movie details	82
Securing REST API with Spring Security	84
Securing REST API with Username and Password	86
Sending Mail using Spring Boot.....	88
Demonstrate Spring Boot Actuator	90
Why logging is important?	92

Miscellaneous Topics.....	93
Interview Questions.....	94

Session pre-requisite

- ❖ Core Java

Install and Setup JDK, Tomcat and Eclipse

Download JDK

- ❖ Go to <https://jdk.java.net/>
- ❖ Click the JDK under the "Ready for use" section
- ❖ Click on the link "zip" adjacent to windows/x64

Install JDK

- ❖ Open the downloaded zip file
- ❖ Right click on the folder starting with "jdk" and select "Copy"
- ❖ Paste the folder to D:
- ❖ Open Windows File Explore and navigate to the folder D:\jdk-[ver]\bin
- ❖ Click on the address bar and copy the folder location

Setup JDK

- ❖ Click "Start" button in windows
- ❖ Type "environment" and select "Edit the system environment variables"
- ❖ Click "Environment Variables"
- ❖ In "System Variables" section scroll down and find the variable named "Path"
- ❖ Select and click "Edit"
- ❖ Click "New"
- ❖ Paste the path copied after installation
- ❖ Move the path to the top using "Move Up" button
- ❖ Click OK three times on the respective window
- ❖ Click "Start" button and type "cmd"
- ❖ Select the Command Prompt option and open the command line window
- ❖ Type command "javac -version" and press enter
- ❖ If version number is displayed then JDK is successfully installed

Download Tomcat

- ❖ Go to <http://tomcat.apache.org/>
- ❖ Click on download of the latest version.
- ❖ Under Binary Distribution section click on zip to download.
- ❖ Open the zip file and select the folder starting with apache-tomcat
- ❖ Paste the folder in D:

Setup JAVA_HOME Environment Variable

- ❖ Click "Start"
- ❖ Type "environment"

- ❖ Select "Edit the system environment variables"
- ❖ Click "Environment Variables"
- ❖ If JAVA_HOME already exists:
 - Click on JAVA_HOME
 - Click "Edit"
 - Specify "Variable Value" as the folder "D:\jdk-[ver]" where JDK zip was extracted
 - Click "OK" and close the windows one by one
- ❖ If JAVA_HOME does not exist:
 - Click "New"
 - Specify "Variable Name" as "JAVA_HOME"
 - Specify "Variable Value" as the folder "D:\jdk-[ver]" where JDK zip was extracted
 - Click "OK" and close the windows one by one

Install Eclipse

- ❖ In browser go to <https://www.eclipse.org/>
- ❖ Click "Download".
- ❖ Click "Download 64 bit".
- ❖ This will download an executable installer file.
- ❖ Execute this program after download.
- ❖ On the window that appears select "Eclipse IDE for Enterprise Java Developers"
- ❖ Ensure Java 1.8+ VM has the JDK 13 folder that had been installed earlier.
- ❖ Select an appropriate installation folder.
- ❖ Ensure "Create start menu entry" and "Create desktop shortcut" boxes are checked.
- ❖ Click "INSTALL" to initiate the installation. This will take a while, wait until the installation completes.

Install Maven

- ❖ In browser go to <http://maven.apache.org/>
- ❖ Click "Download".
- ❖ Click the link adjacent to "Binary zip archive".
- ❖ Double click and open the downloaded zip file, then right click the maven folder and paste it in D drive.
- ❖ Press windows "Start" button and search for "environment"
- ❖ Select "Edit the system environment variables"
- ❖ Click "Environment Variables"
- ❖ Click "Path" in System variables section.
- ❖ Click "Edit" and include the folder "D:\[maven-folder]\bin" in the Path.
- ❖ Open command prompt and run the command "mvn -version"
- ❖ This should display the version currently installed.

Understand the need for Maven

Understanding application distribution

- ❖ Consider that you have developed an application in Java.
- ❖ This application computes a scientific calculation.
- ❖ This application has around 15 java source files.
- ❖ Now you have to send this application to the actual users.
- ❖ How do you share this application to the users? What are the options available?

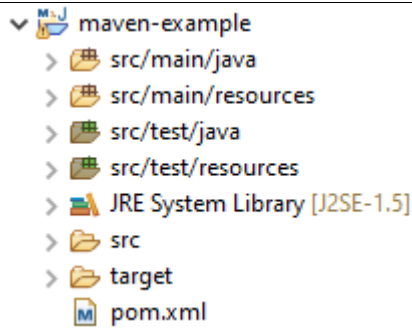
Options available to package a Java application

- ❖ Send sources files to users along with documentation to compile and run.
- ❖ Most users will not know how to compile and run.
- ❖ We can compile the classes and send them.
- ❖ Transferring multiple files will be a challenge.
- ❖ We can create a jar file that contains all the classes.
- ❖ JAR is a **Java Archive** tool that packages a set of files into a single archive file.
- ❖ Options available to create a jar file:
 - Use the export feature in Eclipse to convert the class files into a single jar file.
Eclipse > Right Click on Project > Export...
 - Use jar command in command line to create a jar file.
<https://www.codejava.net/java-core/tools/using-jar-command-examples>
 - Using third party build tools. Some examples provided below:
 - Apache Maven
 - Gradle
 - Apache Ant
- ❖ Any third-party libraries written in java is typically distributed as jar. A very good example is Apache Commons, which contains a collection of reusable classes and methods. In order to use this library, this jar can be download and included in the Eclipse project.

Activity #1 – Create a Maven Project and build

Create a sample maven project in Eclipse.

- ❖ Open Eclipse
- ❖ File > New > Maven Project
- ❖ Check the box "Create a simple project .."
- ❖ Enter Group Id as "com.company.project"
- ❖ Enter Artifact Id as "maven-example"
- ❖ Click "Finish"
- ❖ Check the progress bar in the bottom right corner and wait till the progress completes.
- ❖ The project structure should look something like this.



- ❖ Create a new class named "HelloWorld" in "src/main/java" folder.
- ❖ In the main method of HelloWorld include a system out to print "Hello World".
- ❖ Right click on "maven-example" project > Run As > Maven build
- ❖ This might fail with Java compiler error. If it fails, include the following lines just before the closure of the project tag in pom.xml file.

```
<properties>
  <maven.compiler.target>1.8</maven.compiler.target>
  <maven.compiler.source>1.8</maven.compiler.source>
</properties>
```

- ❖ Try Maven install again and check if the jar file is generated in target folder.

Activity #2 – Include Spring dependencies for the Maven project

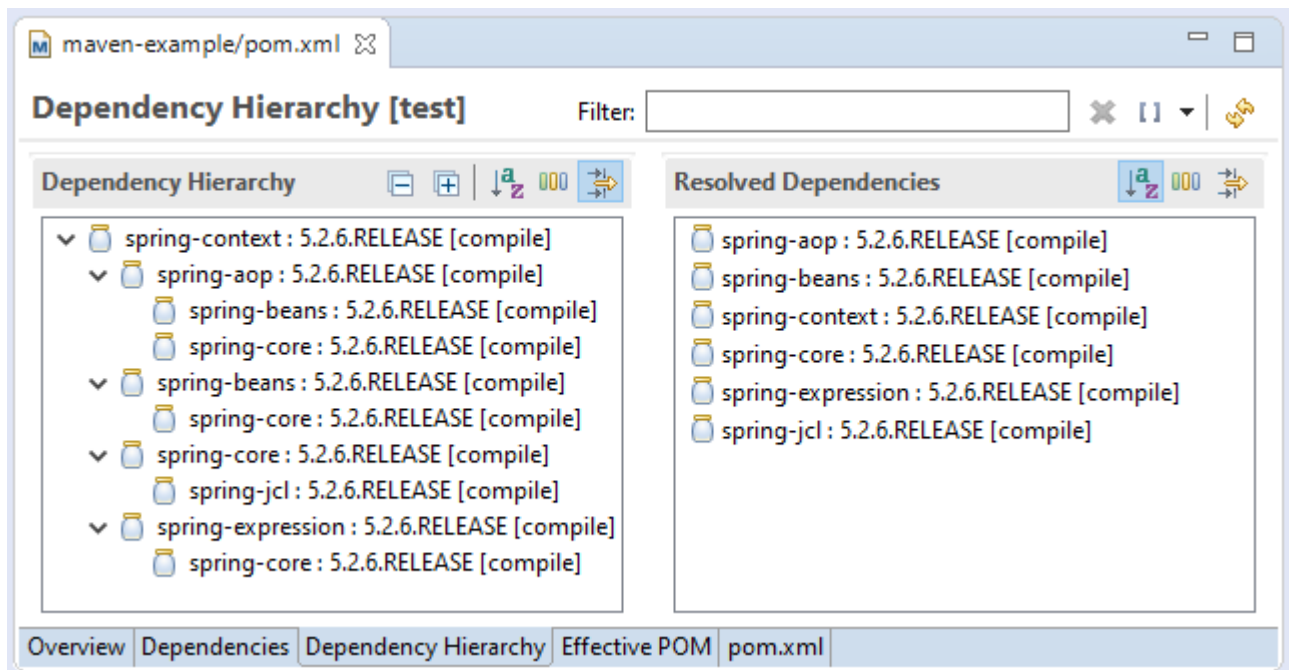
Spring Beans and Spring Context are the jar files required for learning the basics of Spring Framework. Configure these two jar files in maven-example project.

- ❖ Open Eclipse
- ❖ Open <https://mvnrepository.com/>
- ❖ Search for "spring beans"
- ❖ Click on "spring-beans"
- ❖ Click on the latest release.
- ❖ Clicking on the "jar" will download the file. But we will be skipping this step and we will do it in an alternative way.
- ❖ In maven repository site, search for "spring context"
- ❖ Click on "spring-context"
- ❖ Click on the latest version
- ❖ Scroll down to the "Compile Dependencies" sections. This section defines the list of 4 dependent jar files for spring context.
- ❖ To compile and run spring context, we need these additional jar files as well.
- ❖ It is also possible that these 4 jars might be dependent on another 4 jars, which makes it difficult to identify all dependent jars.
- ❖ We will follow the approach below:
 - Open pom.xml file in maven-example project
 - Include the following tag before the closure of the closing project tag.

```
<dependencies>
</dependencies>
```

- In the spring context jar maven repository page, go to "Maven" tab.
- Click inside the box below

- This will automatically copy the content.
- Include the copied content within the dependencies tag.
- Save the file.
- Click on "Dependency Hierarchy" tab and wait for a while till it displays content, it should look something like the below:



- "spring-context" is dependent on "spring-aop", "spring-beans", "spring-core" and "spring-expression".
 - Similarly, rest of the dependencies can be found from the Dependency Hierarchy tree view in the left-hand side.
 - The right-hand side displays consolidated list of all dependencies without hierarchy information.
 - This display means that we have successfully download all the dependent jars of spring context.
- ❖ Open windows explorer.
 - ❖ Go to "C:\Users\[your-profile]\.m2\repository"
 - ❖ By navigating the content in this folder, we can find all the jars that we had downloaded recently.
 - ❖ This folder is the local repository folder of Maven.
 - ❖ Based on the dependencies defined in pom.xml, all dependent jar files will get downloaded in the local repository.
 - ❖ This is a onetime activity, once a jar file of a specific version is downloaded this can be reused by various projects and will not get downloaded again.
 - ❖ Open windows explorer and go to the "maven-example" folder
 - ❖ In the folder address bar click on the empty space, type cmd and press enter.
 - ❖ This will open command prompt in the respective directory.
 - ❖ Execute the following command:
mvn clean package
 - ❖ This command is the same as the one done in Eclipse. The goal clean removes previous build.

About Maven

- ❖ Open Source project under the Apache group of projects
- ❖ Maven is a build automation tool
- ❖ It is primarily used to build Java projects
- ❖ It works based on the concept of Project Object Model (POM)
- ❖ Maven can manage build, reporting and documentation
- ❖ Central Repository is the remote server in internet which contains all common jar libraries used in Java Community.
- ❖ Local Repository refers to the .m2/repository folder, which contain our project specific jar files.
- ❖ Maven helps in downloading the dependencies based on the dependencies defined in pom.xml.

Check your understanding

- ❖ Maven is an open source project (true/false)
- ❖ What is the purpose of maven?
- ❖ What does POM stand for?
- ❖ What is the primary content in pom.xml?
- ❖ What is group id and artifact id?
- ❖ What is the difference between a Central Repository and Local Repository?
- ❖ What is the default folder of Local Repository?
- ❖ What is the command to build the jar file in Maven?

Display Hello World in Spring Boot Application

Activity

Display Hello World in Spring Boot Application.

Expected Output

```

  ____
 /    \
(  )   \
 \    /
  ____

:: Spring Boot ::                (v2.2.6.RELEASE)

2020-04-28 16:28:50.627 INFO 4984 --- [
2020-04-28 16:28:50.644 INFO 4984 --- [
2020-04-28 16:28:51.015 INFO 4984 --- [
2020-04-28 16:28:54.607 INFO 4984 --- [
2020-04-28 16:28:54.712 INFO 4984 --- [
Hello World
```

Solution (Step #1 – Download project using Spring Initializr)

- ❖ Go to <https://start.spring.io/>
- ❖ Change Group as "com.company"
- ❖ Change Artifact Id as "spring-learn"
- ❖ Click "GENERATE" to download the project
- ❖ Extract the zip and place the "spring-learn" in a convenient folder.
- ❖ Import the project in Eclipse
File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish
- ❖ Wait till the project progress completes. The project structure in Eclipse should look something like the below:

```

▼ spring-learn
  > src/main/java
  > src/main/resources
  > src/test/java
  > JRE System Library [JavaSE-1.8]
  > Maven Dependencies
  > src
  > target
  > HELP.md
  > mvnw
  > mvnw.cmd
  > pom.xml
```

- ❖ Expand "src/main/java"
- ❖ Open "SpringBootApplication.java" in the editor.
- ❖ Include `System.out.println("Hello World")` after the `run()` method call.
- ❖ Run `SpringBootApplication` and check the result.

Explanation (Spring Boot Application)

- ❖ Go to the browser window where we have <https://start.spring.io>
- ❖ Click "EXPLORE"
- ❖ The folder structure showed is what got downloaded as spring-learn.zip.
- ❖ Click on "src".
- ❖ We see the SpringLearnApplication.java file
- ❖ This class initializes the Spring Boot Application using @SpringBootApplication annotation.
- ❖ The run method starts the application.
- ❖ In Eclipse 'Project Explorer' expand, "spring-learn" > "src/main/java" > "com.company.springlearn". The SpringBootApplication.java file can be found here.
- ❖ The unit testing files are placed under "src/test/java".

Explanation (The dependency hierarchy)

- ❖ Open pom.xml in Eclipse
- ❖ Click on "Dependency Hierarchy"
- ❖ Collapse all items
- ❖ Expand spring-boot-starter
- ❖ Collapse all items under spring-boot-starter
- ❖ The following key libraries loaded by Spring Boot Starter:
 - Spring Boot
 - Spring Boot Autoconfigure
 - Spring Boot Starter Logging
 - Spring Core
- ❖ Spring Boot Starters provides rapid ways to include a specific feature.
- ❖ For example, Spring Boot Starter Logging loads log4j and slf4j libraries, which ensures that the latest logging libraries are included.
- ❖ Following are few starter packs available in Spring Boot:
 - spring-boot-starter – Basic level starter pack with auto-configuration and logging
 - spring-boot-starter-mail – Implement mailing functionality
 - spring-boot-starter-web – For building Web Application, RESTful Web Service using embedded tomcat container.
 - spring-boot-starter-test – Includes necessary unit testing and mock testing libraries.

Explanation (About Spring Boot)

- ❖ Spring Boot makes it easy to create stand-alone, production grade applications that can just run. For example, it is possible to run a web application using a single jar file.
- ❖ Spring Boot takes an opinionated view of libraries which helps to get started with minimum fuss. This represents the standard jars included by Spring Boot based on the type of application. In our example, it included logging and unit testing libraries with specifying them in dependencies.
- ❖ Spring Boot is a project that is built on top of Spring Framework
- ❖ It provides easier and faster ways to setup, configure and run applications.
- ❖ Spring Boot reduces cost and development time of an application.

Check your understanding

- ❖ What is Spring Boot and what are its benefits?
- ❖ What is the first step in creating a new Spring Boot Application?
- ❖ What is that annotation that is used to define a spring boot application?
- ❖ What are the dependent libraries of Spring Boot Starter?

Load languages for international news media website

Problem

An international news media website wants to support all major geographies in the world. They wanted to support the following languages. A drop down will be available in the website which displays the following languages; hence a static list of these languages must be available as an array list. Define these languages in spring xml configuration file. Each language should be stored along with 2-character international language code.

- ❖ English (en)
- ❖ Spanish (es)
- ❖ Chinese (zh)
- ❖ Hindi (hi)
- ❖ Japanese (ja)

Expected Output

```
1 en English
2 es Spanish
3 zh Chinese
4 hi Hindi
5 ja Japanese
```

Solution (Step #1 – Define a single language)

Language.java

- ❖ Open spring-learn project in Eclipse
 - ❖ Create a new package "com.company.springlearn.model"
 - ❖ Create class Language in this new package
 - ❖ Define instance variables.
- ```
private long id;
private String code;
private String name;
```
- ❖ Include an empty parameter constructor with a log to denote that the constructor is called.

```
public Language() {
 System.out.println("Inside language constructor");
}
```

- ❖ Generate getter setter methods for all the instance variables.
- ❖ Include system out for each setter method.

```
System.out.println("Inside setId()");

System.out.println("Inside setCode()");

System.out.println("Inside setName()");
```

- ❖ Generate toString() method.

#### language.xml

- ❖ Right click on src/main/resources > New > Other ...
- ❖ Type "xml" and select "XML File"
- ❖ Click Next
- ❖ Name the file as "language.xml" and click "Next"
- ❖ Click Finish

- ❖ Click Source
- ❖ Copy and paste the following code into language.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 https://www.springframework.org/schema/beans/spring-beans.xsd">

</beans>
```

- ❖ Include the following bean definition within the <beans> tag.

```
<bean id="en" class="com.company.springlearn.model.Language">
 <property name="id" value="1"/>
 <property name="code" value="en"/>
 <property name="name" value="English"/>
</bean>
```

### SpringLearnApplication.java

- ❖ Remove the hello world system out.
- ❖ Include the following code after the run method call.

```
SpringApplication.run(SpringLearnApplication.class, args);
ApplicationContext context = new ClassPathXmlApplicationContext(
 "language.xml");
Language english = context.getBean("en", Language.class);
```

- ❖ Run this class and check if similar output is seen in the console.

```
Inside language constructor
Inside setId()
Inside setCode()
Inside setName()
Language [id=1, code=en, name=English]
```

## Explanation

### Spring Framework

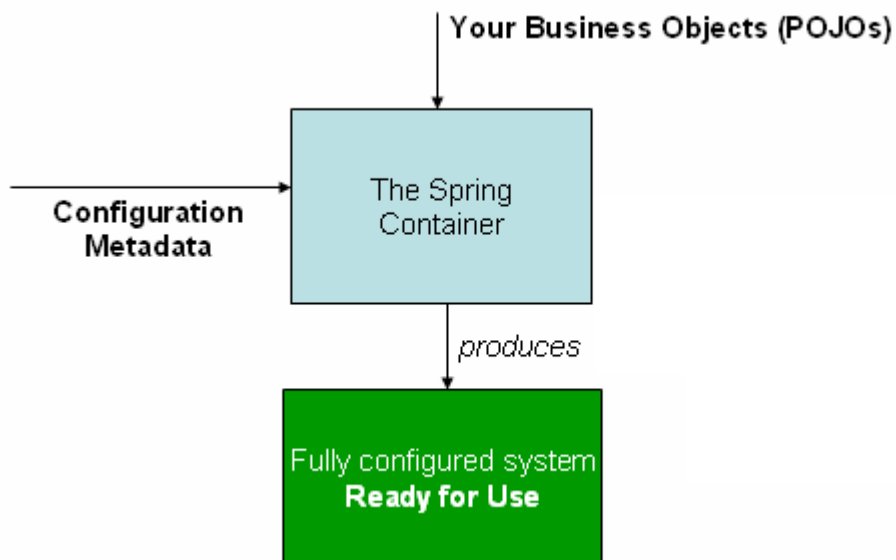
- ❖ Popular application development framework for enterprise Java
- ❖ Developers around the world use Spring Framework to create high performing, easily testable and reusable code.
- ❖ Spring Framework is open source.
- ❖ Initially written by Rod Johnson to avoid the pitfalls of EJB.
- ❖ Spring core framework is light weight.
- ❖ The core features can be used to develop any kind of Java application.

### Benefits

- ❖ Enables developers to develop enterprise-class application using POJOs.
- ❖ Spring is organized in a modular fashion, meaning that the libraries are clearly defined such that you can plug and use the modules that are relevant.
- ❖ Spring does not reinvent the wheel, but it effectively makes use of existing technologies (Example: logging framework, ORM)
- ❖ Centralized exception handling
- ❖ Consistent transaction management

## Spring Core Container

- ❖ The spring core container consists of the following key components:
  - Core – This module provides the fundamental parts of the framework
  - Bean – Manages creation of beans (In our example, it is creation of Language bean)
  - Context – Medium to access any object defined and configured
  - SpEL – Expression language to query and manipulate object graph at runtime.
- ❖ Spring container is at the core of the framework.
- ❖ The container is responsible for:
  - Object creation
  - Wiring objects together
  - Manage the object life cycle
- ❖ Container gets instruction on what bean to instantiate, configure and assemble by reading the configuration (in our example, it is language.xml)
- ❖ Apart from XML file, this can also be represented as annotations and Java code.



- ❖ The POJO in our example represents Language.java

## XML Configuration and Application Context

- ❖ File language.xml is a spring configuration xml file.
- ❖ This file defines a bean instance for Language class
- ❖ <bean> tag is used to define a bean instance.
- ❖ "id" attribute is used to uniquely identify a bean instance.
- ❖ "class" attribute defines the fully qualified class name that needs to be instantiated.
- ❖ <property> tag helps in defining values for the instance variables.
- ❖ "name" attribute specifies the instance variable name.
- ❖ "value" attribute helps defining the value for an instance variable.
- ❖ The reason for placing language.xml in resources folder is to ensure that this file is available in the classpath during runtime.
- ❖ The instance creation of ClasspathXmlApplicationContext searches for language.xml file in the classpath.
- ❖ This constructor reads the <bean> tag and performs the following steps using Java reflection:



- Create instance of Language class using the empty parameter constructor.
- Calls setId(), setCode() and setName() methods to set the values.
- Stores this language object instance under the name "english"
- ❖ The getBean() method just returns the created object based on the "english" id.

### Check your understanding

- ❖ What is Spring Framework?
- ❖ What are the benefits of using Spring Framework?
- ❖ What is POJO and how it plays a role in Spring Framework?
- ❖ What is Spring Container and what is it responsible for?
- ❖ What are the key aspects that a Spring Container is dependent on?
- ❖ What is the purpose of spring configuration xml file?
- ❖ List the attributes and purpose of them in <bean> tag?
- ❖ List the attributes and purpose of them in <property> tag?
- ❖ What does the ClasspathXmlApplicationContext do?
- ❖ How to read a bean created in spring configuration xml file?

### Activity (Understanding the Singleton and Prototype Scope)

- ❖ After getting Language class instance using getBean() include another variable to get the same instance.

```
Language english = context.getBean("en", Language.class);
Language anotherEnglish = context.getBean("en", Language.class);
```

- ❖ Execute the class and check the logs.
- ❖ The logs included in constructor or setter methods are not printed.
- ❖ Which means that new instance of Language is not created for anotherEnglish variable.
- ❖ By default, each bean defined in spring xml configuration file is a Singleton.
- ❖ A Singleton means that it is possible to create only one instance. Refer code below.

```
public class Singleton {

 private static Singleton singleton;

 private Singleton() {}

 public static Singleton getInstance() {
 if (singleton == null) {
 singleton = new Singleton();
 }
 return singleton;
 }

 public void display() {
 System.out.println("I am a Singleton");
 }

 public static void main(String args[]) {
 Singleton singleton = Singleton.getInstance();
 }

}
```

- ❖ If there is a need to create a new instance each time the `getBean()` method is called, then the scope needs to be changed as prototype.
- ❖ Open `language.xml` and modify the bean tag with attribute named "scope" with value as "prototype".
- ❖ Now execute the program and you will find that for each `getBean()` call a new instance of library is getting created.
- ❖ The following scopes are applicable only for web development:
  - request – Represents a HTTP Request.
  - session – Represents a HTTP Session.
  - global-session – Represents a global HTTP Session.

## Solution (Step #2 – Implement logging)

### Understanding the current logs

- ❖ Execute `SpringLearnApplication` and check the content in Console window.
- ❖ The lines starting with timestamp represents the logs generated internally by spring boot when starting and application.
- ❖ The text "`c.c.springlearn.SpringLearnApplication`" represents class `SpringLearnApplication` in "`com.company.springlearn`" package.
- ❖ The first 'c' represents 'com'.
- ❖ The second 'c' represents 'company'.
- ❖ Due to spacing constraints the package names are shortened.

### Generate logs from spring framework

- ❖ Open `application.properties` file in `src/main/resources` folder
- ❖ Include the line below:
 

```
logging.level.org.springframework=debug
```
- ❖ Run `SpringLearnApplication`.
- ❖ Now more logs are generated than before.
- ❖ Let us try to analyze and understand it.
- ❖ Earlier logs were displayed from the "`c.c.springlearn.SpringLearnApplication`" class
- ❖ Find below sample short form of class names in logs and the full package name.

From Log	Full Package
<code>o.s.boot.SpringApplication</code>	<code>org.springframework.boot.SpringApplication</code>
<code>o.s.b.c.c.ConfigFileApplicationListener</code>	<code>org.springframework.boot.context.config.ConfigFileApplicationListener</code>
<code>o.s.b.f.s.DefaultListableBeanFactory</code>	<code>org.springframework.beans.factory.support.DefaultListableBeanFactory</code>

- ❖ Please note that all these packages start with "`org.springframework`", the reason is because of the log configuration that we have defined in `application.properties`.
- ❖ These are logs defined within the spring framework coding, which is what is getting displayed. For troubleshooting any issues related to spring framework, these logs might help resolving the issue.
- ❖ Modify the log level to 'trace' in `application.properties`:
 

```
logging.level.org.springframework=trace
```
- ❖ Check the logs again, now even more logs are displayed at TRACE level.

- ❖ Go to the end of the logs, it clearly denotes the values read from spring xml configuration file and how it gets converted into object.
- ❖ Each line in this log can be classified as TRACE, DEBUG and INFO.
- ❖ When defining a log, each can fall under any one of the below categories:  
 INFO – Informational messages that represents high level application execution progress.  
 DEBUG – More fine-grained information that helps debugging and application.  
 TRACE – Even more fine-grained information than DEBUG  
 WARN – Represents a harmful situation  
 ERROR – An error event, but the application can still function.
- ❖ Let us include some logs to understand how log levels work. Let us make changes in SpringLearnApplication.java
- ❖ Import the following classes:  

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```
- ❖ Include a logger static variable at class level:  

```
private static final Logger LOGGER = LoggerFactory
 .getLogger(SpringLearnApplication.class);
```
- ❖ Include the following lines within the main() method:  

```
LOGGER.trace("TRACE Message");
LOGGER.debug("DEBUG Message");
LOGGER.info("INFO Message");
LOGGER.warn("WARN Message");
LOGGER.error("ERROR Message");
```
- ❖ Comment out the existing trace log and include logger for SpringLearnApplication class.  

```
#logging.level.org.springframework=trace
logging.level.com.company.springlearn=error
```
- ❖ Execute the program and check the logs in console. Notice that only ERROR log is printed.
- ❖ Change the log level to 'info' and check the result. Now INFO, WARN and ERROR is displayed.
- ❖ Change to 'trace'. Now all the logs will be displayed.

### **Include logs for Spring Learn application**

- ❖ Modify the code in SpringLearnApplication.java as specified below within the main method. Include respective logs:  

```
LOGGER.info("Start");

SpringApplication.run(SpringLearnApplication.class, args);

ApplicationContext context = new ClassPathXmlApplicationContext(
 "language.xml");
LOGGER.debug("Context is {}", context);

Language english = context.getBean("en", Language.class);
LOGGER.debug("{} ", english);

LOGGER.info("End");
```
- ❖ Include info level logs in start and end of method.
- ❖ Include debug level logs to print intermediate variables and values.
- ❖ Logs not required for bean classes. Remove the system out println in Language.java
- ❖ Copy the content of the current log into a text editor like notepad.

- ❖ The pattern in which each log statement is displayed can be modified. Please include this line below in application.properties:

```
logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} [%-10.10thread] %5p %-
30.30logger{25} %25M %m%n
```

%d{dd-MM-yy} - Date

%d{HH:mm:ss.SSS} - Time with milliseconds level precision

[%-10.10thread] - Name of the thread

%5p - Log level

%-30.30logger{25} - Class name with package

%25M - Method Name

%-3L - Execution line number in source file

%m - Log message

%n - New Line

- ❖ Displaying **method name** and **line number** are **costlier operations**, should be avoided in real time project. For learning we will include this.
- ❖ Execute the program and check the pattern. Observe the following changes:
  - Date format changed
  - Method name is included
  - Line Number
- ❖ You might still have lot of questions why we have to take so much pain to implement logging. The reason for this will be very clear when working with REST API development.

## Loading list of beans

- ❖ Open language.xml
- ❖ Create specific bean instance for each language.
- ❖ Include language list as specified below:

```
<bean id="languageList" class="java.util.ArrayList">
 <constructor-arg>
 <list>
 <ref bean="en"></ref>
 <ref bean="es"></ref>
 <ref bean="zh"></ref>
 <ref bean="hi"></ref>
 <ref bean="ja"></ref>
 </list>
 </constructor-arg>
</bean>
```

- ❖ Modify bean reading steps as specified below:

```
ArrayList<Language> languages = context.getBean("languageList", ArrayList.class);
LOGGER.debug("{} ", languages);
for (Language language : languages) {
 long id = language.getId();
 String code = language.getCode();
 String name = language.getName();
 LOGGER.debug("{} {} {}", id, code, name);
}
```

- ❖ Run the application and check if logs are displayed with each language.

## About Logging

- ❖ The actual implementation of logging is done by SLF4J.

- ❖ SLF4J stands for Simple Logging Façade for Java
- ❖ This is a wrapper framework for various logging frameworks available in Java
  - java.util.logging
  - logbac
  - log4j
- ❖ Spring Boot had loaded lo4j and SLF4J as part of spring-boot-starter.
- ❖ SLF4J does not have any implementation, but acts as a wrapper around it to incorporate a standard way to implement logging.

### About list configuration in XML

- ❖ Use constructor-arg to define constructor arguments when creating a bean.
- ❖ Use ref to reference existing beans within the context.

### Check your understanding

- ❖ List the different log levels and the order in which it is applied
- ❖ Differentiate between INFO, DEBUG, TRACE, WARN and ERROR levels.
- ❖ How to define logging levels in application.properties?
- ❖ To implement loggers in our application what are the steps involved?
- ❖ How to log a message in an application?
- ❖ How to define an array list in spring xml configuration file?

## Understanding Dependency Injection

### Problem

Create a book class and link it with language using dependency injection. The book class should have the following structure:

- ❖ Book
  - Id – long
  - Title – String
  - Language – Language
    - Id – long
    - Code – String
    - Name – String

### Expected Output

Book [id=1, title=Harry Potter and the Sorceror's Stone, language=Language [id=1, code=en, name=English]]

### Solution

#### Book.java

- ❖ Create a new class Book in com.company.springlearn.model package
- ❖ Create following instance variables.

```
private long id;
private String title;
private Language language;
```
- ❖ Generate getters / setters
- ❖ Generate toString() method

#### language.xml

- ❖ Include the following bean definition in the end of language.xml:

```
<bean id="book" class="com.company.springlearn.model.Book">
 <property name="id" value="1"/>
 <property name="title" value="Harry Potter and the Sorceror's Stone"/>
 <property name="language" ref="en"/>
</bean>
```

#### SpringLearnApplication.xml

- ❖ Include a new method displayBook() after the main() method:

```
public static void displayBook() {
 LOGGER.info("Start");
 ApplicationContext context = getLanguageContext();
 Book book = context.getBean("book", Book.class);
 LOGGER.debug("{", book);
 LOGGER.info("End");
}
```

- ❖ Invoke this method in main()

```

public static void main(String[] args) {
 LOGGER.info("Start");
 SpringApplication.run(SpringLearnApplication.class, args);

 //displayLanguages();
 //demonstratePrototype();

 displayBook();

 LOGGER.info("End");
}

```

- ❖ Run SpringLearnApplication and check if book is display with English language.

## Explanation

- ❖ Book class is dependent on Language class.
- ❖ The dependent bean can be loaded using ref attribute in property tag.
- ❖ This mechanism is called Dependency Injection.
- ❖ The act of connecting one object with other objects using a framework is called Dependency Injection.
- ❖ If we are not using Dependency Injection, the creation of the Language objects needs to be taken care by the developer.
- ❖ When using Dependency Injection, the dependent object is identified based on the spring configuration xml file and loaded automatically.
- ❖ The way in which we have applied Dependency Injection in this example is called Setter Based Dependency Injection, since Library object is set in book by invoking the setter method in Book class.
- ❖ There are two ways by which Dependency Injection can be achieved:
  - Constructor based dependency injection
  - Setter based dependency injection
- ❖ Dependency Injection is also called as Inversion of Control. Since the order of creation of objects are in reversed order, that is the Dependent objects are already loaded by Spring Container.
- ❖ To implement constructor-based implementation:

- Include constructor in Book.java

```

public Book(long id, String title, Language language) {
 super();
 this.id = id;
 this.title = title;
 this.language = language;
}

```

- Include new bean definition in language.xml

```

<bean id="bookConstructor" class="com.company.springLearn.model.Book">
 <constructor-arg type="Long" value="1"/>
 <constructor-arg type="String" value="Harry Potter and the Sorcerer's Stone"/>
 <constructor-arg ref="Language"/>
</bean>

```

- Use the new bean id during bean lookup:

```

Book book = context.getBean("bookConstructor", Book.class);

```

### Check your understanding

- ❖ What is Dependency Injection?
- ❖ What is Inversion of Control?
- ❖ What are the two ways in which dependency injection can be applied?
- ❖ List the steps to implement constructor-based dependency injection?
- ❖ List the steps to implement setter-based dependency injection?



## Implement Dependency Injection using Autowiring

### Problem

Represent the creation of book instance using auto wiring.

### Expected Output

Book [id=1, title=Harry Potter and the Sorcerer's Stone, language=Language [id=1, code=en, name=English]]

### Solution

#### language.xml

- ❖ Create a copy of bean with id as "book" and rename it as "bookAutowired", then remove the "language" property.

```
<bean id="bookAutowired" class="com.company.springlearn.model.Book" autowire="byName">
 <property name="id" value="1"/>
 <property name="title" value="Harry Potter and the Sorcerer's Stone"/>
</bean>
```

- ❖ Copy the English language bean and name the bean id as "language".

```
<bean id="language" class="com.company.springlearn.model.Language">
 <property name="id" value="1"/>
 <property name="code" value="en"/>
 <property name="name" value="English"/>
</bean>
```

#### SpringLearnApplication.xml

- ❖ Include a new method displayBookAutowired() after the main() method:

```
public static void displayBookAutowired() {
 LOGGER.info("Start");
 ApplicationContext context = getLanguageContext();
 Book book = context.getBean("bookAutowired", Book.class);
 LOGGER.debug("{ }", book);
 LOGGER.info("End");
}
```

- ❖ Invoke this method in main()

```
public static void main(String[] args) {
 LOGGER.info("Start");
 SpringApplication.run(SpringLearnApplication.class, args);

 //displayLanguages();
 //demonstratePrototype();
 //displayBook();
 displayBookAutowired();

 LOGGER.info("End");
}
```

- ❖ Run SpringLearnApplication and check if book is display with English language.

### Explanation

- ❖ The property autowire="byName" loads dependent objects based on the property name.
- ❖ "language" is a property of Book class
- ❖ So, it looks for bean with "id" as "language" and loads that bean. This decision happens because of the autowire="byName" defined in the bean with id "bookAutowired"

- ❖ To verify this, change the bean id "language" with a different name and check if it results in an exception.
- ❖ Autowiring is a concept to load dependent beans based on automatic configuration rather than depending on the bean configuration.
- ❖ Change the value byName to byType and run the application, it will result in an error.
- ❖ The byType configuration looks for a single instance of bean definition of Language class, but since there are multiple bean ids defined it fails.
- ❖ Comment out the rest of the beans and have only one Language bean, this should result in correct loading of the beans.

### **Check your understanding**

- ❖ What is autowiring?
- ❖ How does autowiring byName works?
- ❖ How does autowiring byType works?

## Miscellaneous topics related to Spring XML based configuration

### Injecting Inner Beans

- ❖ In java it is possible to define a class within a class, which is called as Inner Class
- ❖ Find below the syntax to provide bean definition for inner classes:

```
<bean id="outer" class="...">
 <!-- instead of using a reference to a target bean, simply define the target bean inline -->
 <property name="target">
 <bean class="com.example.Person"> <!-- this is the inner bean -->
 <property name="name" value="Fiona Apple"/>
 <property name="age" value="25"/>
 </bean>
 </property>
</bean>
```

### Initializing List

- ❖ Refer bean definition below to initialize any implementation of List:

```
<property name="someList">
 <list>
 <value>a list element followed by a reference</value>
 <ref bean="myDataSource" />
 </list>
</property>
```

- ❖ The first item in the list is of type string
- ❖ The second item in the list refers another bean of a different type.

### Initializing Map

- ❖ Refer bean definition below to initialize any implementation of Map:

```
<property name="someMap">
 <map>
 <entry key="an entry" value="just some string"/>
 <entry key="a ref" value-ref="myDataSource"/>
 </map>
</property>
```

- ❖ The first item in the map is of type string
- ❖ The second item in the list refers another bean of a different type.

### Initializing Set

- ❖ Refer bean definition below to initialize any implementation of Set:

```
<property name="someSet">
 <set>
 <value>just some string</value>
 <ref bean="myDataSource" />
 </set>
</property>
```

- ❖ The first item in the set is of type string

- ❖ The second item in the set refers another bean of a different type.

### Assigning Empty String

- ❖ Refer property definition below that defines an empty string:

```
<bean class="ExampleBean">
 <property name="email" value=""/>
</bean>
```

### Assigning null value

- ❖ Refer property definition below that defines a property value as null:

```
<bean class="ExampleBean">
 <property name="email">
 <null/>
 </property>
</bean>
```

### Define property values using p-namespace

- ❖ Refer property definition below that assigns value using an alternative way. The first bean definition is a comparison with the way we represent now:

```
<bean name="classic" class="com.example.ExampleBean">
 <property name="email" value="someone@somewhere.com"/>
</bean>

<bean name="p-namespace" class="com.example.ExampleBean"
 p:email="someone@somewhere.com"/>
```

### Inheritance

- ❖ It is also possible to define beans with parent and child classes, refer example below.

```
<bean id="inheritedTestBean" abstract="true"
 class="org.springframework.beans.TestBean">
 <property name="name" value="parent"/>
 <property name="age" value="1"/>
</bean>

<bean id="inheritsWithDifferentClass"
 class="org.springframework.beans.DerivedTestBean"
 parent="inheritedTestBean" init-method="initialize"> 1
 <property name="name" value="override"/>
 <!-- the age property value of 1 will be inherited from parent -->
</bean>
```

### Lazy Initialization

- ❖ Currently, when the application context is created the container is initialized with all the beans defined in the XML configuration file.
- ❖ But there might be scenarios where a bean needs to be initialized only during the first usage.
- ❖ Since the content of a bean is very heavy, it needs to be loaded if it is required.

- ❖ This can be defined using lazy-init property of the bean.

```
<bean id="lazy" class="com.something.ExpensiveToCreateBean" lazy-init="true"/>
```

### Lazy Initialization

- ❖ Currently, when the application context is created the container is initialized with all the beans defined in the XML configuration file.
- ❖ But there might be scenarios where a bean needs to be initialized only during the first usage.
- ❖ Since the content of a bean is very heavy, it needs to be loaded if it is required.
- ❖ This can be defined using lazy-init property of the bean.

```
<bean id="lazy" class="com.something.ExpensiveToCreateBean" lazy-init="true"/>
```

### Initialization and Destruction Callbacks

- ❖ If there is any initialization that needs to be done before creation of a bean in the container, the call back methods can be used. Refer code below:

```
<bean id="exampleInitBean" class="examples.ExampleBean" init-method="init"/>
```

```
public class ExampleBean {

 public void init() {
 // do some initialization work
 }
}
```

- ❖ Similarly refer code below to handle a situation when a bean is removed from the container:

```
<bean id="exampleInitBean" class="examples.ExampleBean" destroy-method="cleanup"/>
```

```
public class ExampleBean {

 public void cleanup() {
 // do some destruction work (like releasing pooled connections)
 }
}
```

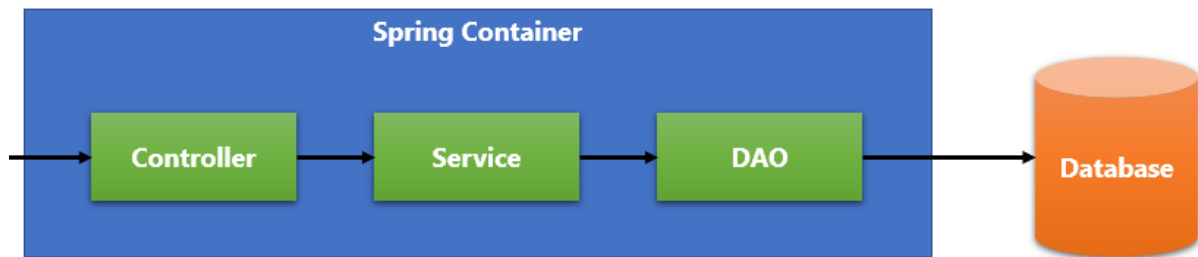
### Callbacks for container startup and shutdown

- ❖ Similar to bean initialization and cleanup call, there are callback methods available for start of container and before shutdown of the container.

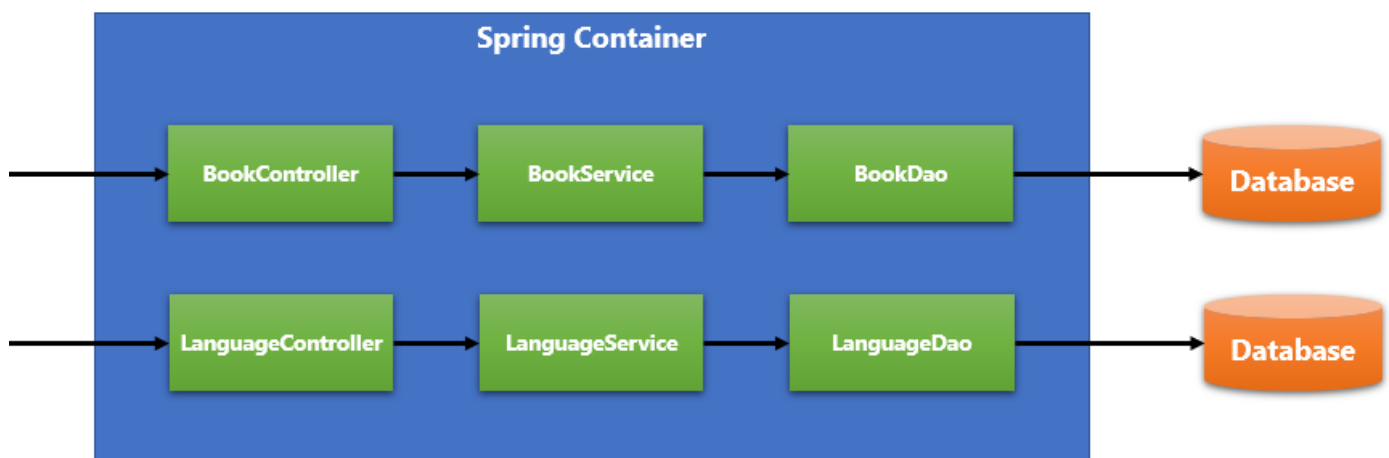
## Implement Dependency Injection using Annotations

### Problem

Implement Spring RESTful Web Service components using spring annotation configuration. Find below the general architecture for implementing a RESTful Web Service.



If we have to implement the above architecture for Book and Language, then following will be the structure.



- ❖ BookController is dependent on BookService
- ❖ BookService is dependent on BookDao
- ❖ LanguageController is dependent on LanguageService
- ❖ LanguageService is dependent on LanguageDao

### Solution

- ❖ Create a new package `com.company.springlearn.controller`
- ❖ Create new class `BookController` in this new package and define the class as specified below.

```
import org.springframework.stereotype.Component;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component
public class BookController {

 public static final Logger LOGGER = LoggerFactory.getLogger(BookController.class);

 public BookController() {
 LOGGER.debug("Inside BookController constructor");
 }
}
```

- ❖ Run Spring Learn Application and check the log output.

- ❖ In the console check if the log defined in the BookController constructor is displayed.

## Explanation

- ❖ From the output we can understand that BookController class is instantiated by the Spring Container.
- ❖ How did spring identify the BookController? Let us find that out by going through the below points.
- ❖ This is because of the @SpringBootApplication annotation defined in SpringLearnApplication class.
- ❖ Actually @SpringBootApplication encapsulates the definition of the below three annotations:
  - @Configuration
  - @EnableAutoConfiguration
  - @ComponentScan
- ❖ The @Configuration annotation represents that SpringLearnApplication class in itself is a spring configuration file.
- ❖ The @EnableAutoConfiguration enables automatic spring configuration, which means that spring will look for configurations available in various classes and it will find opportunities to find beans to be loaded into Spring Container.
- ❖ The @ComponentScan represents that spring can look for beans to be loaded in the com.company.springlearn package and all its sub-packages, so spring will scan for components available in the below list of packages:
  - com.company.springlearn
  - com.company.springlearn.model
  - com.company.springlearn.controller
- ❖ While scanning the controller package it can find the class defined with @Component annotation, so it loads this class into spring container. This is equivalent to the spring xml configuration specified below:

```
<bean id="LanguageController"
 class="com.company.springlearn.controller.LanguageController">
</bean>
```
- ❖ The @Component annotation can be applied to any class that represents implementation of a functionality and does not contain any data on its own.
- ❖ There are specific annotations to define the kind of functionality it is going to support. Find below the same:
  - @Controller – To handle request
  - @Service – To implement business logic
  - @Repository – To interact with database
- ❖ Let us change the @Component to @Controller and we should still get the same result.

```
import org.springframework.stereotype.Controller;

@Controller
public class BookController {
```

### Check your understanding

- ❖ What annotations does @SpringBootApplication encapsulate?
- ❖ Explain how component scanning happens in a spring boot application.

### Exercise Activity

- ❖ Define LanguageController using spring xml configuration.
  - Comment out the existing language definitions in language.xml
  - Comment out the existing method calls and invoke getLanguageContext() method in SpringLearnApplication main() method.
- ❖ Define com.company.springlearn.service.BookService with @Service annotation.
- ❖ Define com.company.springlearn.service.LanguageService and load it using spring xml configuration.

### Solution (Autowire Controller with Service)

- ❖ Define instance variable BookService in BookController.

```
private BookService bookService;
```

- ❖ Define the setter method and autowire the method.

```
@Autowired
public void setBookService(BookService bookService) {
 LOGGER.debug("Inside setBookService of BookController");
 this.bookService = bookService;
}
```

- ❖ This should display the log defined within the setBookService() method.

### Explanation

- ❖ The display of the log in setBookService() indicates that the instance variable bookService is injected automatically by the spring container.
- ❖ This is achieved by using the @Autowired annotation.
- ❖ Based on this annotation, the input parameter type is checked, in this case it is BookService.
- ❖ Then the container looks for any BookService instance available in the container.
- ❖ If there is only one instance of BookService available, then it picks that bean and sends it as parameter for setBookService() method.
- ❖ The @Autowired can also be applied on constructor which gives the same result.

```
@Autowired
public BookController(BookService bookService) {
 LOGGER.debug("Inside BookController constructor with bookService parameter");
 this.bookService = bookService;
}
```

- ❖ The @Autowired can also be applied on instance variable.

```
@Autowired
private BookService bookService;
```

- ❖ As part of JSR 330 definition the auto wiring feature is named as @Inject, hence @Autowired can be replaced with @Inject.



### Exercise Activity

- ❖ Define autowiring in language.xml between LanguageController and LanguageService.
- ❖ Define autowiring in language.xml between LanguageService and LanguageDao.
- ❖ Define constructor based autowiring between BookService and BookDao.

### Check your understanding

- ❖ What is auto wiring?
- ❖ How to achieve dependency injection using annotations and auto wiring? List down the steps involved.

## Miscellaneous topics related to Spring Core Framework

### Customizing bean using BeanPostProcessor

- ❖ In the examples seen so far, the instantiation of the bean is completely managed by the container
- ❖ If there is a requirement to completely customize the way by which an object needs to be instantiated can be achieved using BeanPostProcessor.

### @Required

- ❖ Makes an attribute of a class mandatory.
- ❖ This annotation defined on a setter method helps in ensuring that the specific attribute is initialized.

```
public class SimpleMovieLister {

 private MovieFinder movieFinder;

 @Required
 public void setMovieFinder(MovieFinder movieFinder) {
 this.movieFinder = movieFinder;
 }

 // ...
}
```

### Java-based configuration

- ❖ So far, we have seen two types of spring configuration:
  - XML based
  - Annotation based
- ❖ There is one more way which is called as Java-based configuration.
- ❖ This can be achieved using the annotations @Configuration, @Bean, @Import and @DependsOn. Refer sample below:

*Bean definition*

```
@Configuration
public class AppConfig {

 @Bean
 public MyService myService() {
 return new MyServiceImpl();
 }
}
```

*Bean access*

```
public static void main(String[] args) {
 ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);
 MyService myService = ctx.getBean(MyService.class);
 myService.doStuff();
}
```

## Resources

- ❖ Frequently we need to refer configuration, images or media files which is outside the spring application.
- ❖ These resources may be present in file system, classpath or URL.
- ❖ For reading each type of resource, we have to depend on a different set of API.
- ❖ This can be handled in a unified way using Resource and ResourceLoader available in Spring Core library.

### *Resource Interface*

```
public interface Resource extends InputStreamSource {

 boolean exists();

 boolean isOpen();

 URL getURL() throws IOException;

 File getFile() throws IOException;

 Resource createRelative(String relativePath) throws IOException;

 String getFilename();

 String getDescription();
}
```

- ❖ Built-in implementations of Resource interface:
  - UrlResource
  - ClassPathResource
  - FileSystemResource
  - ServletContextResource
  - InputStreamResource
  - ByteArrayResource

## Validation, Data Binding and Type Conversion

- ❖ This topic deals with conversion of data from one type to a bean. For example, HTTP form to a bean.
- ❖ This also deals with validating the data before setting the bean attributes.
- ❖ This will be dealt in more detail when working on Spring RESTful Web Service development.

## Spring Expression Language (SpEL)

- ❖ Walkthrough the examples and topics provided in the link below:

<https://www.baeldung.com/spring-expression-language>

## Aspect Oriented Programming (AOP)

- ❖ Consider the scenario of logger implementation
  - In each class we need to import the logger
  - Create a static definition for the logger

- Include logs on start of method and end of method.
- This is a manual activity that needs to be done in most of the class
- In AOP terminology, this is called as a cross-cutting concern.
- ❖ AOP is another way of thinking about program structure.
- ❖ In AOP, the unit of modularity is an aspect.
- ❖ Aspects enable modularization of cross-cutting concerns such as transaction management, logging, etc.

## Understand how internet works

### Activity

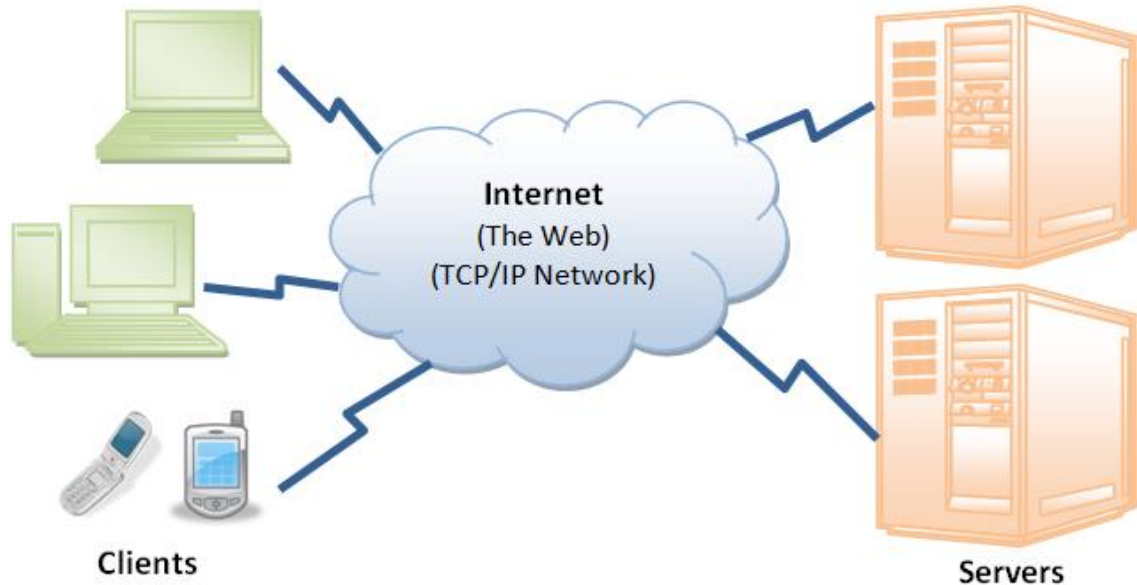
- ❖ What is your guess about how internet works?
- ❖ When [www.google.com](http://www.google.com) is accessed in the browser what happens?

### Let us understand what happens in the background

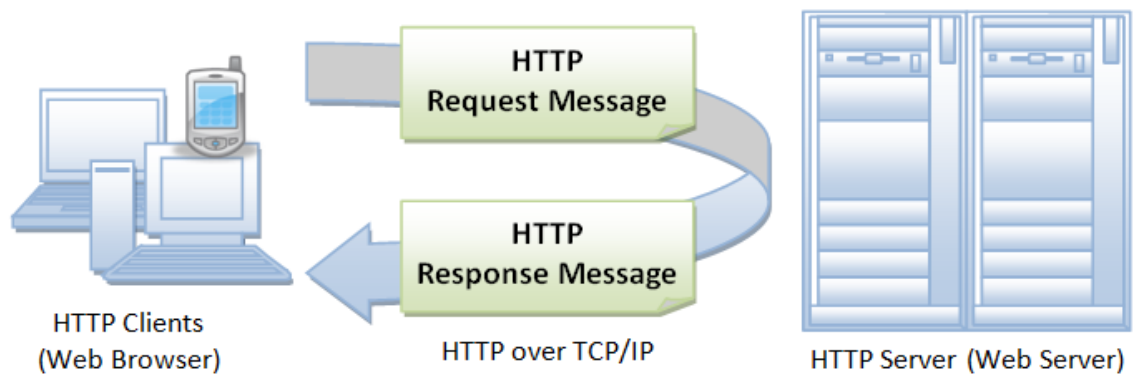
- ❖ Open Chrome Browser
- ❖ Go to menu > More tools > Developer tools
- ❖ Click "Network"
- ❖ Access <http://www.google.com>
- ❖ Within the Network window click [www.google.com](http://www.google.com)
- ❖ Click "Headers"
- ❖ Collapse "General", "Response Headers"
- ❖ The key value pairs specified in the "Request Headers" section is sent to the server that belongs to the domain [www.google.com](http://www.google.com).
- ❖ The key value pairs specified in "Response Headers" is what is received back by the browser from the server.
- ❖ The server response also contains a component called Response Body that contains the actual content for the web page, this Response Body can be seen by clicking on "Response" tab.
- ❖ This is the actual HTML returned by the server is what is rendered as a web page in the browser window.
- ❖ The other entries represent the images and other files downloaded to display the web page.
- ❖ Now type some thing and search for any keyword and press enter.
- ❖ The content in the Network tab is freshly refreshed and has new set of items.
- ❖ Click on the first link in the Network tab, we can see a fresh request and response received.

### Explanation for Web Architecture and HTTP

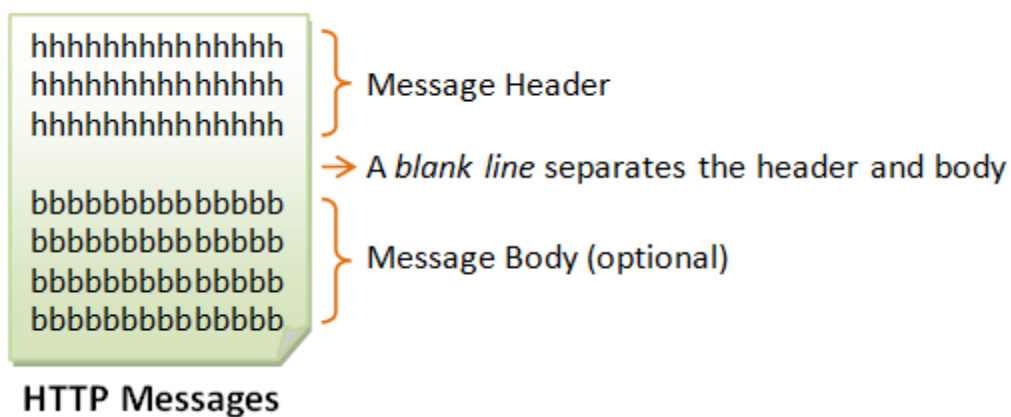
- ❖ Client/Server based Architecture
- ❖ Client initiates the request
- ❖ Server receives, processes and gives a response to the Client
- ❖ Client consumes the Server's response and converts it into display or can use the response for processing.
- ❖ Clients
  - Hardware - Mobile Phone, Laptop, Desktop, Tablet
  - Software – Browsers (Chrome, Firefox, Edge), Developer Tools (Postman, CURL), Testing Tools (Selenium, Protractor)
- ❖ Servers
  - Hardware – Servers from various vendors (Dell, HP, IBM) running on operating systems Windows Server, Linux and Unix
  - Software – IIS Server, Apache HTTP Server, Nginx, Apache Tomcat



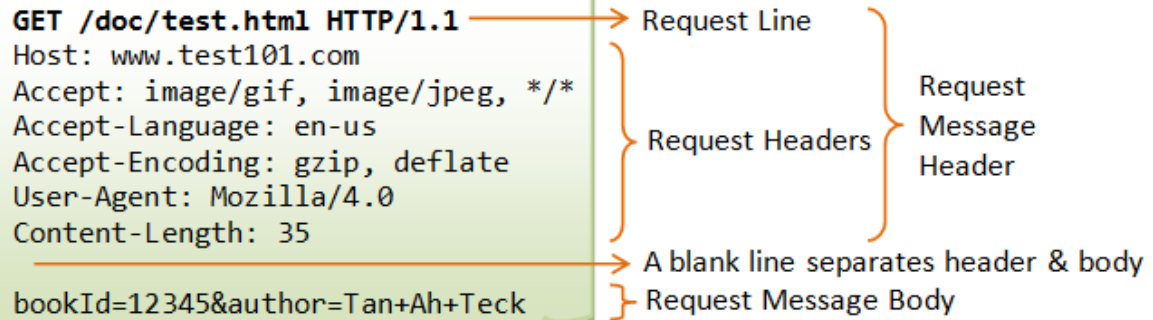
- ❖ In the activity we did, Chrome Browser our desktop is the client, Tomcat is the server
- ❖ Web Architecture Works on a Request / Response paradigm



- ❖ HTTP stands for Hyper Text Transfer Protocol
- ❖ HTTP was developed by various standards organizations like CERN, IETF and W3C
- ❖ HTTP is an Application Layer protocol for transmitting documents
- ❖ HTTP is a stateless protocol
- ❖ Composition of a HTTP request or response



- ❖ Sample HTTP request



```

GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck

```

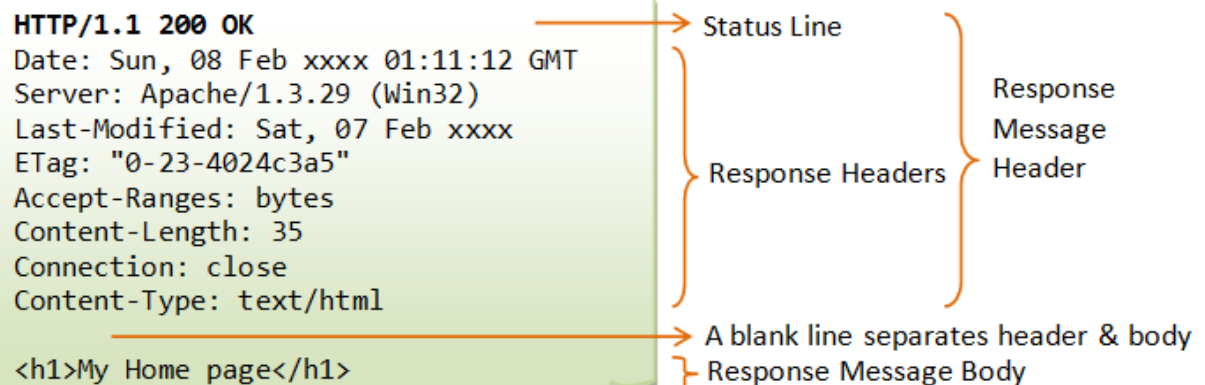
Annotations:

- Request Line: `GET /doc/test.html HTTP/1.1`
- Request Headers: `Host: www.test101.com`, `Accept: image/gif, image/jpeg, */*`, `Accept-Language: en-us`, `Accept-Encoding: gzip, deflate`, `User-Agent: Mozilla/4.0`, `Content-Length: 35`
- A blank line separates header & body
- Request Message Body: `bookId=12345&author=Tan+Ah+Teck`

❖ View HTTP request of HelloWorldServlet:

- Open the tab where hello-world servlet is displayed in Chrome browser.
- Click Menu > More tools > Developer tools
- Click Network
- Refresh the page
- Click "hello-world" in the network window
- View the content under "Request Headers" section

❖ Sample HTTP response



```

HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>

```

Annotations:

- Status Line: `HTTP/1.1 200 OK`
- Response Headers: `Date: Sun, 08 Feb xxxx 01:11:12 GMT`, `Server: Apache/1.3.29 (Win32)`, `Last-Modified: Sat, 07 Feb xxxx`, `ETag: "0-23-4024c3a5"`, `Accept-Ranges: bytes`, `Content-Length: 35`, `Connection: close`, `Content-Type: text/html`
- A blank line separates header & body
- Response Message Body: `<h1>My Home page</h1>`

❖ Check "Response Headers" section in Network window of Chrome browser.

❖ View the HTTP Response Status in "General" section

All image Courtesy: [https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP\\_Basics.html](https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

### Check your understanding

- ❖ Elaborate about the client/server architecture of web application.
- ❖ What is the primary protocol used for communication between a client and server in internet?
- ❖ What does HTTP stand for?
- ❖ Using HTTP protocol, it is possible to identify the client that is calling. (True / False)
- ❖ What is HTTP Request? What are the key components of a HTTP request?
- ❖ What is HTTP Response? What are the key components of a HTTP response?

## Understanding what is a Single Page Application (SPA)

### Activity

#### Multi-page Application

- ❖ Go to <https://www.thehindu.com/>
- ❖ Click on a news article
- ❖ Observe that the browser page goes blank and then the news article page is reloaded

#### Single Page Application

- ❖ Go to <https://translate.google.com/>
- ❖ Try translating few words
- ❖ Try changing the language in either side and translate few words
- ❖ Notice that the page is never reloaded

### Explanation

Schematic representation of single and multi-page application.

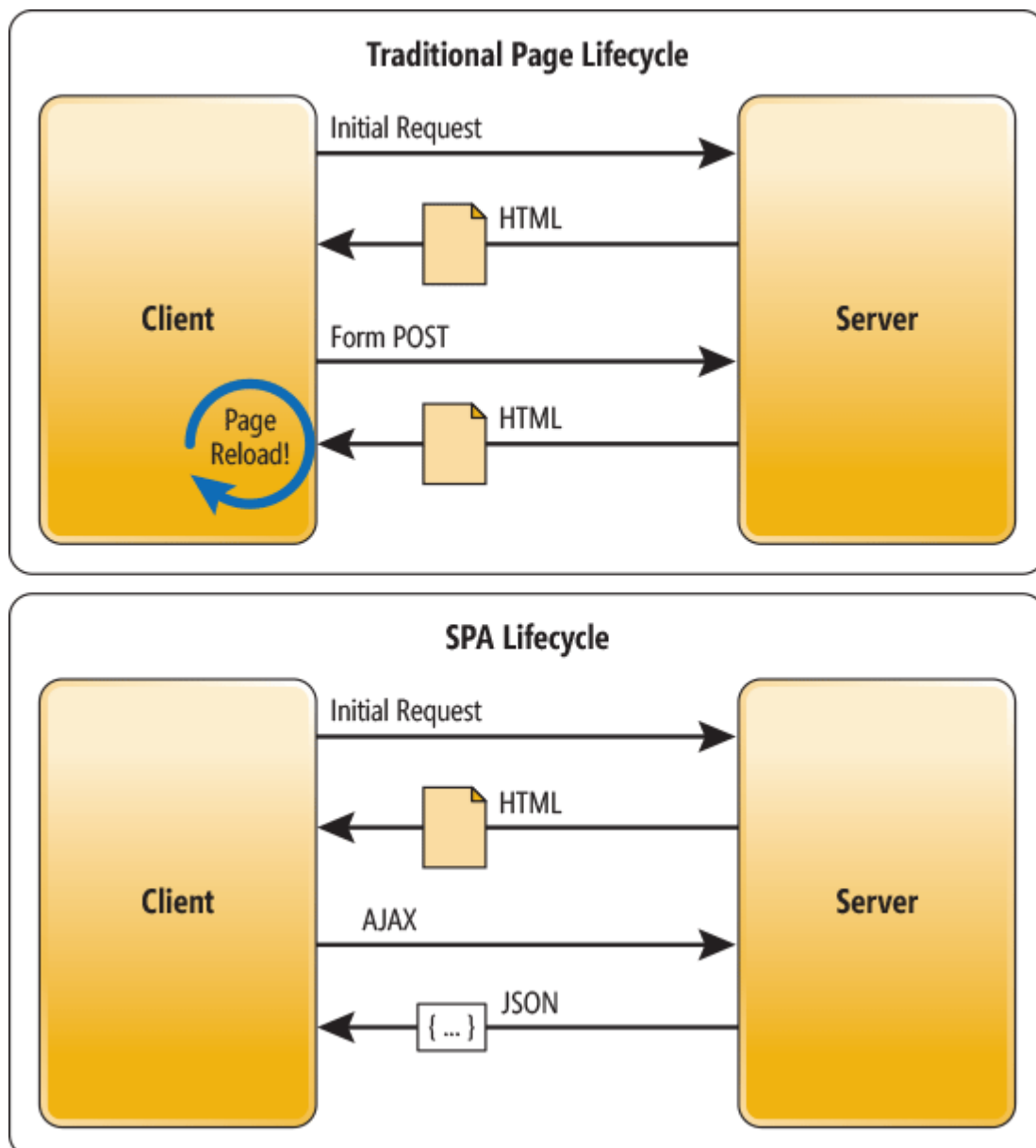


Image Courtesy: [https://www.goconqr.com/c/74455/course\\_modules/113576-single-page-vs-multi-page-architecture#](https://www.goconqr.com/c/74455/course_modules/113576-single-page-vs-multi-page-architecture#)



### **Multi-page Application**

- ❖ Traditional applications are mostly multi-page applications
- ❖ Multi page applications does not use AJAX.
- ❖ The HTTP request send receives a fresh HTML document and the entire page is cleared and freshly rendered by the browser.
- ❖ The disadvantage of multi-page application is that there is large data transferred every time a page is accessed, which puts more load on the server.
- ❖ Technologies for building multi-page application:
  - Microsoft – ASP, ASP.NET
  - Java – JSP, Servlet, Struts, Spring MVC

### **Single Page Application (SPA)**

- ❖ Single Page Application uses JavaScript and AJAX for implementation
- ❖ When the URL of Single Page Application is accessed, all the pages of the applications are loaded in a single request and response.
- ❖ After the initial load, screen navigation is done based on hiding and showing DOM elements.
- ❖ Pages that require fresh data will be using AJAX in the background to get the data from the server.
- ❖ One disadvantage of Single Page Application is that, if the website has lots of screens and images, then the initial load time of the application will be high.
- ❖ There are various JavaScript frameworks such as Angular, ReactJS and VueJS has evolved which helps in building Single Page Applications.
- ❖ These frameworks do not address the server-side part of the implementation, but only helps in building the front-end part.
- ❖ These frameworks use JavaScript and AJAX for implementation of their frameworks.

### **Get a glimpse of Angular and JSON**

- ❖ Open the below link in browser:  
<https://stackblitz.com/edit/angular-fchphp?file=src%2Fapp%2Fapp.component.html>
- ❖ This link contains an example for angular which displays a list of users
- ❖ This user list is obtained from a RESTful web service
- ❖ In Chrome Browser, Developer Tools Network tab click on "users" from the list and click "Response" tab.
- ❖ Click within the response text and press Ctrl + a and Ctrl + c to copy.
- ❖ Open <http://jsonviewer.stack.hu/> in browser
- ❖ Paste the content copy into the text area available.
- ❖ Click "Format"
- ❖ This type of representation of data is called as JSON.
- ❖ JSON stands for JavaScript Object Notation.
- ❖ JSON is a platform independent communication format for RESTful Web Service.
- ❖ Property names are represented in double quotes. (Example: "page", "per\_page" etc.)
- ❖ Text value is represented within double quotes (Example: "first\_name")
- ❖ Number and Boolean is represented without double quotes (Example: value for "total")

- ❖ The value can be a string, number, Boolean or object.
- ❖ When it is an object it is defined within open and close curly braces. (Example: similar to "data" but defined with curly braces instead of square brackets)
- ❖ Array of items are represented within square brackets. (Example: "data")

## About RESTful Web Service

- ❖ In the browser stackblitz window, go to Network tab, click "users" > Headers > Request Header, you can find the content-type property:

```
Content-Encoding: gzip
Content-Type: application/json; charset=utf-8
Date: Fri, 15 May 2020 05:30:15 GMT
```

- ❖ This represents the type of content responded. For normal web pages, this will be text/html.
- ❖ In the browser stackblitz window, go to Network tab and right click "users" > Copy > Copy link address.
- ❖ Paste this URL in Notepad, which should look like this:  
`https://reqres.in/api/users`
  - "https" represents the protocol. The "https" is same as "http", but it is secured.
  - "reqres.in" represents the server.
  - "users" represents the service that is accessed.
- ❖ If the above URL is access in the browser, then following is the JSON response received.

```
{
 "page": 1,
 "per_page": 6,
 "total": 12,
 "total_pages": 2,
 "data": [
 {
 "id": 1,
 "email": "george.bluth@reqres.in",
 "first_name": "George",
 "last_name": "Bluth",
 "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/calebogden/128.jpg"
 },
 {
 "id": 2,
 "email": "janet.weaver@reqres.in",
 "first_name": "Janet",
 "last_name": "Weaver",
 "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/josephstein/128.jpg"
 },
 {
 "id": 3,
 "email": "emma.wong@reqres.in",
 "first_name": "Emma",
 "last_name": "Wong",
 "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/olegpogodaev/128.jpg"
 },
 {
 "id": 4,
 "email": "eve.holt@reqres.in",
 "first_name": "Eve",
 "last_name": "Holt",
 "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/marcoramires/128.jpg"
 },
 {
 "id": 5,
 "email": "charles.morris@reqres.in",
 "first_name": "Charles",
 "last_name": "Morris",
 "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/stephenmoon/128.jpg"
 },
 {
 "id": 6,
 "email": "tracey.ramos@reqres.in",
 "first_name": "Tracey",
 "last_name": "Ramos",
 "avatar": "https://s3.amazonaws.com/uifaces/faces/twitter/bigmancho/128.jpg"
 }
],
 "ad": {
 "company": "StatusCode Weekly",
 "url": "http://statuscode.org/",
 "text": "A weekly newsletter focusing on software development, infrastructure, the server, performance, and the stack end of things."
 }
}
```

- ❖ As part of Spring RESTful Web Services learning we will be implementing this kind of services.
- ❖ This service can be consumed by any front-end framework like Angular, React JS or React JS.
- ❖ RESTful Web Service
  - REST stands for Representational State Transfer
  - REST is a software architectural style
  - It defines constraints for creating web services.
  - It provides interoperability between computer systems on the internet.
  - Request made responds with HTML, XML, JSON or another format.
  - Currently JSON is the popular format used for this communication.
  - RESTful Web Services are lightweight, fast performing alternative for XML

### Check your understanding

- ❖ What is multi-page application?
- ❖ What is single page application?
- ❖ Multi-page application loads only once. (True / False)
- ❖ What are the advantages of Single Page Application?
- ❖ Are there any disadvantages of Single Page Application, if so, what is it?
- ❖ List few technology frameworks available to implement multi-page application.
- ❖ What are the popular JavaScript frameworks available to implement Single Page Application?
- ❖ What technologies does implementation of Single Page Application depends on?
- ❖ What is the expanded form for JSON?
- ❖ How is a property defined in JSON?
- ❖ How is a property value defined in JSON? What are the three possible types?
- ❖ How is an object defined in JSON?
- ❖ How is an array defined in JSON?
- ❖ What is the syntax for defining JSON?
- ❖ What is the content type for HTML web page?
- ❖ What is the content type for JSON?
- ❖ What is the expanded form for REST?
- ❖ What does REST mean and what is its purpose?
- ❖ What are the advantages of using REST?
- ❖ What is the popular format used for communication in RESTful Web Services?

## Hello World Servlet

### Activity

Display "Hello World" in Tomcat Server console.

### Expected Output in Eclipse Console

```
Apr 01, 2020 4:05:38 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
Apr 01, 2020 4:05:38 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in [3,024] milliseconds
Hello World
```

### Solution

- ❖ Launch Eclipse
- ❖ Click File > New > Dynamic Web Project
- ❖ Enter "Project name:" as "jee-learn"
- ❖ Click "Next"
- ❖ Click "Next"
- ❖ Click "Finish"
- ❖ Expand "Java Resources"
- ❖ Right click on "src" and create a new package named "com.company.servlet"
- ❖ Right click on the package "com.company.servlet" and select New > Servlet
- ❖ Enter the servlet name as "HelloWorldServlet"
- ❖ Click "Next"
- ❖ Click "HelloWorldServlet" in "URL Mappings" section
- ❖ Click "Edit" and change the URL to "/hello-world"
- ❖ Click "Next"
- ❖ Uncheck "doPost"
- ❖ Click "Finish"
- ❖ Open "HelloWorldServlet.java" in the editor window.
- ❖ Remove the comments for improving the readability.
- ❖ Include code in doGet() method to display Hello World.
- ❖ Right click on "jee-learn" project and select Run As > Run on server
- ❖ In the window that opens check if the respective Tomcat Server is selected.
- ❖ Click "Next"
- ❖ Click "Finish"
- ❖ Check in console window till "Server startup" message is displayed.
- ❖ Open the URL <http://localhost:8080/jee-learn/hello-world> in the browser and check if the message "Served at: jee-learn/" is displayed.
- ❖ Open the console window and check if "Hello World" is displayed.

### Explanation for the Servlet code

- ❖ The URL used in Chrome browser is composed for the following:
  - "http" defines the HTTP protocol

- "localhost" refers to the local computer
- "8080" refers to the port number in which the Tomcat server runs. Double click on the tomcat server in Eclipse to find the port details defined.
- "jee-learn" refers to a specific web application in Tomcat Server. A single instance of Tomcat Server can run multiple web applications.
- "hello-world" refers to the URL of the servlet. This is what is defined in @WebServlet annotation at the class level of the servlet.
- ❖ A servlet class has to extend HttpServlet
- ❖ doGet() method is called whenever the URL is called from the browser.
- ❖ The parameters HttpServletRequest and HttpServletResponse helps in managing the HTTP Request and HTTP Response.
- ❖ Follow steps below to find where the compiled classes are placed:
  - Go to "Servers" window in Eclipse
  - Double click the Tomcat Server
  - In the window that opens go to the "Server Locations" section.
  - Check the folder in server path and go to that folder. The ".metadata" folder is located in the root folder of Eclipse Workspace folder.
  - Then go to the folder specified in "Deploy path"
  - Then go to folder "jee-learn\WEB-INF\classes\com\company\servlet"
  - The compile class files for the Servlet will be available in this location
  - "WEB-INF\classes" is a folder for any Java Web Application is the standard location to place the class files of the application.

### Java Web Application Architecture

- ❖ Based on the technology the web application architecture varies
- ❖ ASP.NET, PHP, Ruby on Rails, Django for Python are different technologies and frameworks available for web application development.
- ❖ Servlets and JSP forms the core technology for development of web applications in Java.
- ❖ Application developed using Servlets and JSP can be deployed in various server applications like WebLogic, WebSphere, Glass Fish, JBoss and Tomcat
- ❖ Servlet and JSP are standards defined as part of Java Community Process. The implementation of these standards is done by the servers defined above.

### Questions

- ❖ What is the main purpose of Tomcat Server?
- ❖ Brief about the @WebServlet annotation.
- ❖ What is the purpose of doGet() method?
- ❖ Explain the key components of Java Web Application architecture?

## Understanding the Servlet Life Cycle

### Activity

Demonstrate how Servlet Life Cycle methods `init()`, `service()` and `destroy()` work.

### Expected Output in Eclipse Console

```
INFO: Server startup in [3,824] milliseconds
Inside init().
Hello World
Inside service().
Hello World
Inside service().
Apr 01, 2020 9:39:19 PM org.apache.catalina.core.StandardServer await
INFO: A valid shutdown command was received via the shutdown port. Stoppi
Apr 01, 2020 9:39:19 PM org.apache.coyote.AbstractProtocol pause
INFO: Pausing ProtocolHandler ["http-nio-8080"]
Apr 01, 2020 9:39:20 PM org.apache.catalina.core.StandardService stopInte
INFO: Stopping service [Catalina]
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.catalina.loader.WebappCl
WARNING: Please consider reporting this to the maintainers of org.apache.
WARNING: Use --illegal-access=warn to enable warnings of further illegal
WARNING: All illegal access operations will be denied in a future release
Inside destroy().
Apr 01, 2020 9:39:20 PM org.apache.coyote.AbstractProtocol stop
INFO: Stopping ProtocolHandler ["http-nio-8080"]
Apr 01, 2020 9:39:20 PM org.apache.coyote.AbstractProtocol destroy
INFO: Destroying ProtocolHandler ["http-nio-8080"]
```

### Solution

- ❖ Open "HelloWorldServlet.java" in Eclipse
- ❖ Select "Source > Override/Implement Methods"
- ❖ Under `HttpServlet` select "`service(HttpServletRequestRequest, HttpServletResponse)`"
- ❖ Expand `GenericServlet` and select `destroy()` and `init()`
- ❖ Include `System.out.println()` with these three methods. Refer code below:

```
@Override
protected void service(HttpServletRequestRequest arg0, HttpServletResponse arg1)
 throws ServletException, IOException {
 super.service(arg0, arg1);
 System.out.println("Inside service().");
}

@Override
public void destroy() {
 super.destroy();
 System.out.println("Inside destroy().");
}

@Override
public void init() throws ServletException {
 super.init();
 System.out.println("Inside init().");
}
```

- ❖ If the server is already running, then tomcat would have already got started. Refer console window to check if the application is restarted:  
Apr 01, 2020 9:27:28 PM org.apache.catalina.core.StandardContext reload  
INFO: Reloading Context with name [/jee-learn] is completed
- ❖ If the server is not running, right click on "jee-learn" and select "Run As > Run on Server"

- ❖ Refer the display in Eclipse console:

```
Inside init().
Hello World
Inside service().
```

- ❖ Refresh the browser again. Then the console looks as specified below:

```
Inside init().
Hello World
Inside service().
Hello World
Inside service().
```

- ❖ Notice that the init() method is not called the second time.
- ❖ Go to Eclipse
- ❖ Go to Server window
- ❖ Right click on the tomcat server and select stop.
- ❖ Go to Console window and check the content. This should be similar to the sample output provided above.

### Explanation

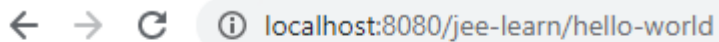
- ❖ The following are the key life cycle methods of servlet:
  - "init()" – Include any servlet initialization content in this method.
  - "service()" – This method is invoked whenever there is a request coming to the server.
  - "destroy()" – This method is invoked whenever a servlet is reloaded or before shutting down of the Tomcat Server. Include any release of resources that were created during init().

## Display Hello World using PrintWriter

### Activity

Display Hello World in the web page.

### Expected Output

A screenshot of a web browser's address bar. It shows navigation icons (back, forward, refresh) on the left, followed by an information icon and the URL "localhost:8080/jee-learn/hello-world".

# Hello World

### Solution

- ❖ Open HelloWorldServlet.java
- ❖ Remove the line starting with "response.getWriter()..."
- ❖ Include the below lines after the System.out.println():

```
PrintWriter writer = response.getWriter();
writer.println("<h1>Hello World</h1>");
```
- ❖ Import PrintWriter from java.io. [Use shortcut key Ctrl+Shift+O to automatically include imports]
- ❖ Save and run the application on the server
- ❖ In browser open the URL <http://localhost:8080/jee-learn/hello-world> and check if "Hello World" is displayed with bold and bigger font.

### Explanation

- ❖ "response" is an input parameter for doGet() method, which can be used to define the response that needs to be sent to the client.
- ❖ PrintWriter is used write HTML response to the requesting client.



## Sending values to server

### Activity

Display greeting message as displayed below. The name should be dynamically obtained from the URL.

### Expected Output

A screenshot of a web browser's address bar. It shows navigation icons (back, forward, refresh) on the left, followed by an information icon and the URL "localhost:8080/jee-learn/greeting?name=John".

# Hello John

### Solution

- ❖ Create a new servlet `com.company.servlet.GreetingServlet` with following details:
  - Class Name: `GreetingServlet`
  - URL: `/greeting`
  - Methods to implement: `doGet()`
- ❖ For the sake of readability, remove the commented lines.
- ❖ Remove line starting with `response.getWriter()`.
- ❖ In the `doGet()` method include the code below:

```
String name = request.getParameter("name");

PrintWriter writer = response.getWriter();
writer.println("<h1>Hello " + name + "</h1>");
```
- ❖ Save and open the URL <http://localhost:8080/jee-learn/greeting?name=John> in browser, which should display the expected result.

### Explanation

- ❖ Input parameters and values can be included in the URL. Refer details below:
    - Question mark after the URL represents the beginning of parameter definition.
    - After the question mark the parameter name and value can be provided using equal to.
    - There can be more than one parameter, with each parameter separated by `&`
- Sample:
- ```
http://www.xyzbank.com/savings/get-statement?account=03234&start=01/03/00&end=01/04/00
```
- ❖ "request" parameter of the `doGet()` method details of the request sent from client.
 - ❖ The `getParameter()` method of request helps in getting the parameters defined in the URL.

Download text file using Servlet

Activity

Demonstrate download of a text file using Servlet. Refer text file content below.

Expected Output

File name: greetings.txt

```
Hello World!
Welcome to programming.
```

Solution

- ❖ Create a new servlet named DownloadServlet with doGet() method.
- ❖ Remove the commented lines.
- ❖ Include the following code in doGet() method.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/plain");
    response.setHeader("Content-Disposition", "attachment;filename=greetings.txt");
    ServletOutputStream out = response.getOutputStream();
    out.println("Hello World!");
    out.println("Welcome to Programming.");
    out.close();
}
```

- ❖ Stop and start the server
- ❖ In browser give the URL <http://localhost:8080/jee-learn/download>
- ❖ This should download a file named "greetings.txt"
- ❖ Open the text file and check the content.

Explanation

- ❖ The setContentType() method in response helps to define the file type of the response.
- ❖ The setHeader() method of response helps to define a header property and value.
- ❖ The Content-Disposition header informs browser on how to handle the response.
- ❖ The value "attachment" notifies the browser to download.
- ❖ The "filename" property helps to define the name for the downloaded file.
- ❖ The getOutputStream() method provides the stream to write the response directly.
- ❖ The println() method helps in writing the actual content in the response.
- ❖ The close() helps to close the output stream.

Questions

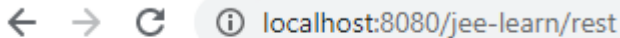
- ❖ What is the content type for sending text file in HTTP response?
- ❖ Using HttpServletResponse how to make browser initiate a download?

Create a RESTful Web Service using Servlet

Activity

Demonstrate creating a RESTful Web Service using Servlet. Respond with a greetings message.

Expected Output

A screenshot of a web browser's address bar. It shows navigation icons (back, forward, refresh) and an information icon on the left. The address bar contains the text "localhost:8080/jee-learn/rest".

```
{ "message": "hello world" }
```

Solution

- ❖ Create a new servlet named RestServlet with doGet() method.
- ❖ Remove the commented lines.
- ❖ Include the following code in doGet() method.

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("application/json");
    ServletOutputStream out = response.getOutputStream();
    out.print("{ \"message\": \"hello world\" }");
    out.close();
}
```

- ❖ Stop and start the server
- ❖ In browser give the URL <http://localhost:8080/jee-learn/rest>
- ❖ This should display the JSON greetings message in browser.

Explanation

- ❖ The setContentType() method in response helps to define the file type of the response.
- ❖ The setHeader() method of response helps to define a header property and value.
- ❖ The Content-Disposition header informs browser on how to handle the response.
- ❖ The value "attachment" notifies the browser to download.
- ❖ The "filename" property helps to define the name for the downloaded file.
- ❖ The getOutputStream() method provides the stream to write the response directly.
- ❖ The println() method helps in writing the actual content in the response.
- ❖ The close() helps to close the output stream.

Questions

- ❖ What is the content type for sending text file in HTTP response?
- ❖ Using HttpServletResponse how to make browser initiate a download?

Create a RESTful Web service that sends Hello World greetings message

Activity

Write a RESTful Web Service that returns Hello World greetings message.

Solution (Step #1 – Convert the spring-learn application into web)

- ❖ Open Eclipse
- ❖ Open pom.xml of the spring-learn project
- ❖ Copy and paste the following dependency within <dependencies>

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

- ❖ Save the file and wait for the build to complete.
- ❖ Run SpringLearnApplication and check if Tomcat Server is loaded.
- ❖ Also notice that the running application does not terminate, which means that the Tomcat Server is started and is waiting for request from client.

Solution (Step #2 – Create Controller for sending Hello World message)

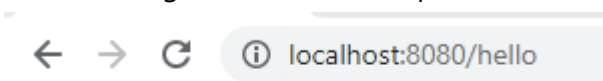
- ❖ Create a new class com.company.springlearn.controller.HelloController.
- ❖ Include LOGGER.
- ❖ Include annotation.

```
@RestController
public class HelloController {
```

- ❖ Resolve the annotation using the Ctrl+Shift+O to organize imports.
- ❖ Include method.

```
@GetMapping("/hello")
public String sayHello() {
    LOGGER.info("Start");
    return "Hello World";
}
```

- ❖ Resolve the annotation using the Ctrl+Shift+O to organize imports.
- ❖ Run SpringLearnApplication.
- ❖ Open <http://localhost:8080/hello> in browser.
- ❖ The following should be the output.



Hello World

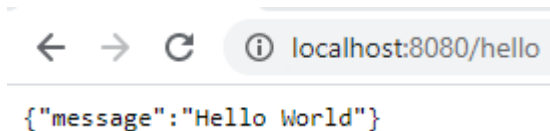
- ❖ The Network tab in Developer Tools will be display the content-type as specified below:

Content-Type: text/html;charset=UTF-8

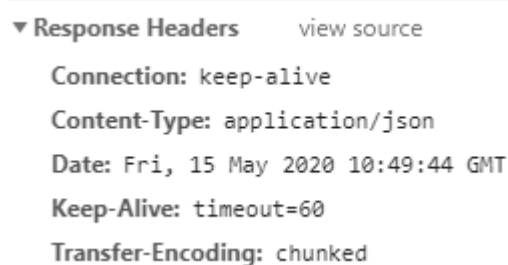
- ❖ Modify the sayHello() method as specified below.

```
@GetMapping("/hello")
public HashMap<String, String> sayHello() {
    LOGGER.info("Start");
    HashMap<String, String> message = new HashMap<>();
    message.put("message", "Hello World");
    LOGGER.info("End");
    return message;
}
```

- ❖ In the Eclipse console check if the application reloads.
- ❖ Once the application reloads, open <http://localhost:8080/hello> or refresh this page if it is already open. The output should look like this.



- ❖ The content-type in Chrome Developer Tools should look like this:

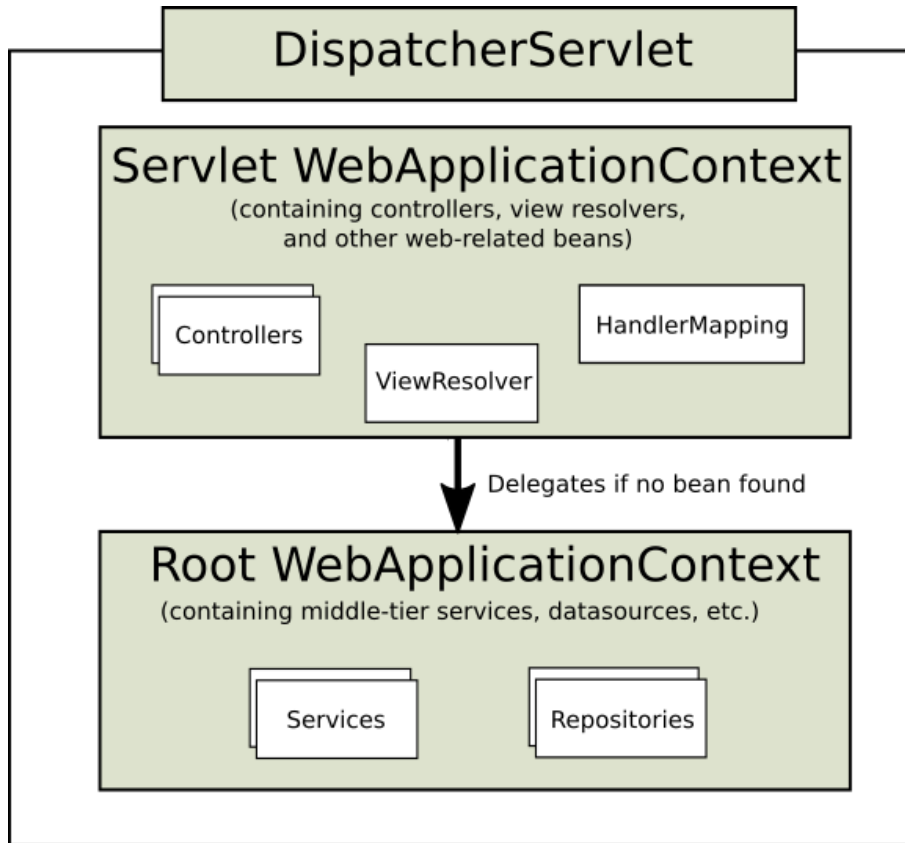


Explanation

- ❖ Open pom.xml
- ❖ Open "Dependency Hierarchy" and check the contents under spring-boot-start-web. The following are the libraries loaded:
 - spring-boot-starter-json – Spring boot assumes that we might be developing RESTful Web Services and had included this starter pack. This helps in conversion between Java objects and JSON. Jackson is one of the third-party open source libraries that does this conversion, which as well has got included as part of this package.
 - spring-boot-starter-tomcat – A web server is required for development of a web application, hence this is included. This tomcat is embedded and does not require a separate tomcat server setup.
 - spring-boot-starter-validation – This starter pack helps in validating the bean content.
 - spring-web – This is a module in Spring Framework that helps in web development and contains all library to handle HTTP.
 - spring-boot-dev-tools – This package helps in automatically restarting tomcat server if there are any changes detected.
- ❖ The @RestController indicates Spring Framework that this component is going to be deployed as a RESTful Web Service.
- ❖ The @GetMapping annotation defines a service with mapping URL "/hello".
- ❖ The DispatcherServlet of Spring MVC framework is the one that receives this request and sends it to HelloController.
- ❖ This can be found by including trace logs in application.properties and making a request.

```
logging.level.org.springframework.web.servlet.DispatcherServlet=trace
```

- ❖ Spring MVC like many other frameworks is defined with Front Controller pattern
- ❖ As per the FrontController pattern there should be a single Servlet that processes all the request.
- ❖ In Spring MVC framework, this servlet is the DispatcherServlet.
- ❖ Find below the architecture of Spring MVC framework:



- ❖ If SpringBoot is not used then this servlet needs to be configured in web.xml.
- ❖ SpringBoot does in configuration by default, hence it is not required to configure the DispatcherServlet.
- ❖ The conversion from HashMap to JSON is automatically taken care by the Jackson library, this conversion is initiated by Spring Framework.

Check your understanding

- ❖ What were the dependencies added to convert the SpringLearnApplication into a web application?
- ❖ What is the purpose of these dependencies?
- ❖ What are the behavioral changes that happened after including the above modules?
- ❖ What is the purpose of the annotation `@RestController`?
- ❖ What is the purpose of the annotation `@GetMapping`?
- ❖ How does DispatcherServlet play a role in this example?
- ❖ What is a Front Controller pattern?

Create a RESTful Web service to get all languages for the media web site

Problem

The front-end application of the media web site wants to display a drop down with list of languages, so that the user can select a language. Write a REST API to return the list of languages.

Expected Output

Access URL: <http://localhost:8080/news/api/languages>

Method Type: GET

Response:

```
[
  {"id":1,"code":"en","name":"English"},
  {"id":2,"code":"es","name":"Spanish"},
  {"id":3,"code":"zh","name":"Chinese"},
  {"id":4,"code":"hi","name":"Hindi"},
  {"id":5,"code":"ja","name":"Japanese"}
]
```

Solution

language.xml

- ❖ Modify to uncomment the five language and the language list.
- ❖ Rest all bean creation can be commented out.

LanguageService.java

- ❖ Include annotations at class level.

```
@Service
public class LanguageService {
```

- ❖ Include logger.

- ❖ Include static variables of the class.

```
private static List<Language> languageList;
```

- ❖ Include constructor.

```
public LanguageService() {
    LOGGER.info("Start");
    context = new ClassPathXmlApplicationContext("language.xml");
    languageList = context.getBean("languageList", ArrayList.class);
    LOGGER.info("End");
}
```

- ❖ Include getLanguages() method.

```
public ArrayList<Language> getLanugages() {
    LOGGER.info("Start");
    return languageList;
}
```

LanguageController.java

- ❖ Include annotations at class level.

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/news/api/languages")
public class LanguageController {
```

- ❖ Autowire Service.

```
@Autowired
private LanguageService languageService;
```

- ❖ Include new method with get mapping:

```
@GetMapping
public List<Language> getLanguages() {
    LOGGER.info("Start");
    return languageService.getLanugages();
}
```

- ❖ Save all files and run SpringLearnApplication.
- ❖ Access the URL <http://localhost:8080/news/api/languages>, this displays result as JSON.

Explanation

- ❖ When the URL <http://localhost:8080/news/api/languages> is accessed, the DispatcherServlets tries to find if there is any controller that maps to the URL news/api/languages.
- ❖ The @RequestMapping definition in this class helps to map the URL.
- ❖ Then spring web framework tries to find if there is any method that can be mapped to this HTTP request.
- ❖ When accessing any URL from the browser address bar, the browser by default sends the method type as GET.
- ❖ Since there is a method with @GetMapping, spring framework identifies this as the method to execute.
- ❖ Language list is read from spring xml.
- ❖ Since the controller is defined as @RestController, spring converts the array list into JSON data using the Jackson library already added as dependency.

Test the language service using MockMvc

Activity

Test the language list REST service using JUnit and MockMvc.

Solution (Step #1 – Load the language controller)

SpringLearnApplicationTests.java

- ❖ Open Eclipse go to Project Explore > spring-learn > src/test/java
- ❖ Open SpringLearnApplicationTests.java
- ❖ Create instance variable for controller and auto wire it:

```
@Autowired
private LanguageController languageController;
```

- ❖ Copy and paste below import:

```
import static org.junit.jupiter.api.Assertions.assertNotNull;
```

- ❖ Modify the contextLoads() method to test if the language controller is loaded.

```
@Test
void contextLoads() {
    assertNotNull(languageController);
}
```

- ❖ Run the test cases by following the steps below:
 - Right click on SpringLearnApplicationTests > Run As > JUnit Test
- ❖ Click on JUnit window and check if the test case has passed. It should display a green band if it is succeeded.

Explanation

- ❖ The assertNotNull() checks if the passed parameter is null or not.
- ❖ This method is available JUnit library.

Solution (Step #2 – Test the language REST Service)

SpringLearnApplicationTests.java

- ❖ Copy and paste below imports

```
import static org.junit.Assert.assertNotNull;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.ResultActions;
```

- ❖ Include below annotation at class level:

```
@SpringBootTest
@AutoConfigureMockMvc
class SpringLearnApplicationTests {
```

- ❖ Include instance variable for MockMvc:

```
@Autowired
private MockMvc mvc;
```

- ❖ Include test method:

```
@Test
public void getLanguage() throws Exception {
    ResultActions actions = mvc.perform(get("/news/api/languages"));
    actions.andExpect(status().isOk());
}
```

- ❖ Modify the test method to include more checks:

```
@Test
public void getLanguage() throws Exception {
    ResultActions actions = mvc.perform(get("/news/api/languages"));
    actions.andExpect(status().isOk());
    actions.andExpect(jsonPath("$").isArray());
    actions.andExpect(jsonPath("$.length()").value(5));
}
```

Explanation

- ❖ MockMvc is a class in spring framework to do end to end testing.
- ❖ @Test represents a test method.
- ❖ The perform() method invokes the URL.
- ❖ The status().isOk() method checks if the HTTP Response is 200.
- ❖ The jsonPath method helps in verifying the content of JSON response.
- ❖ The isArray() method checks whether the response is an array.
- ❖ The '\$' represents the root.
- ❖ The '\$.length()' gets the length of the array.
- ❖ The library jayway JsonPath is a separate library that helps in navigating and reading the JSON content.

Get language based on language code

Activity

Get details about a specific language based on the language code.

Expected Output

Access URL: <http://localhost:8080/news/api/languages/en>

Method Type: GET

Response:

```
{ "id": 1, "code": "en", "name": "English" }
```

Solution

LanguageService.java

- ❖ Include method below.

```
public Language getLanguage(String code) {  
    LOGGER.info("Start");  
    LOGGER.debug("Code: " + code);  
    for (Language language : languageList) {  
        LOGGER.debug("{} ", language);  
        if (language.getCode().equals(code)) {  
            return language;  
        }  
    }  
    LOGGER.info("End");  
    return null;  
}
```

LanguageController.java

- ❖ Include method below.

```
@GetMapping("/{code}")  
public @ResponseBody Language getLanguage(@PathVariable String code) {  
    LOGGER.info("Start");  
    LOGGER.debug("Code: " + code);  
    Language language = languageService.getLanguage(code);  
    LOGGER.debug("Language: {} ", language);  
    if (language == null) {  
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Language not found");  
    } else {  
        return language;  
    }  
}
```

- ❖ Save file and run SpringLearnApplication.
- ❖ Access the URL <http://localhost:8080/news/api/languages/en>.
- ❖ Try changing 'en' in the above URL with 'ja' and check if Japan language details are returned.

Explanation

- ❖ By specifying @GetMapping with "{code}" we are including a variable parameter.
- ❖ This variable name in the path is defined in the method parameter with the same name "code" and it is also defined with @PathVariable to denote that the value for this parameter has to come from the path.

- ❖ Spring framework internally reads the value from the path and calls the method passing the code as parameter.
- ❖ Notice that both the methods defined in this class are `@GetMapping`, so how does spring framework identify which method to call.
- ❖ Since there is a slight difference in the URL with `/{code}`, this new method is identified for execution.

Return appropriate response when a language code is not found

Problem

When a non-existent language code is provided in the previous REST API, it just returns a blank response. Instead, it should respond with appropriate error message with the standard HTTP Status Code 404, which means that a resource is not found.

Expected Output

Access URL: <http://localhost:8080/news/api/languages/ss>

Method Type: GET

Response:

```
{
  "timestamp": "2020-05-22T18:30:01.517+0000",
  "status": 404,
  "error": "Not Found",
  "message": "Language not found",
  "path": "/news/api/languages/ss"
}
```

HTTP Error Status should be 404 which denotes "Resource not found"

Solution

LanguageNotFoundException.java

- ❖ Create new class `LanguageNotFoundException` in `com.company.springlearn.exception` package, which extends from `Exception` class.
- ❖ Include annotation for this exception class:

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason="Language not found")
public class LanguageNotFoundException extends Exception {
}
```

LanguageController.java

- ❖ Modify `getLanguage()` with throws:

```
public Language getLanguage(@PathVariable String code)
    throws LanguageNotFoundException {
```
- ❖ Remove the 'return null' line and replace with below throw call:

```
throw new LanguageNotFoundException();
```
- ❖ Save file and run `SpringLearnApplication`.
- ❖ Access the URL <http://localhost:8080/news/api/languages/en>.
- ❖ Access the URL <http://localhost:8080/news/api/languages/ss>.
- ❖ This will result in white label error, we need additional tools to check this result.
- ❖ Download git from <https://git-scm.com/downloads> and install the same.
- ❖ Download postman from <https://www.postman.com/downloads/>. Run downloaded executable to install.
- ❖ The installation of post man can be found in `C:\Users\<user>\AppData\Local\Postman`. Create a shortcut for the executable in a convenient location or include it in Taskbar.
- ❖ Testing in git bash:

- Open "Git Bash" from Start menu.

- Run the following command:

```
curl -v http://localhost:8080/news/api/languages/ss
```

- This would result in a large stack trace in the response.
- Include below property in application.properties and check the result using curl.

```
server.error.include-stacktrace=never
```

- Check the headers in request and response:

```
> GET /news/api/languages/ss HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 404
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Fri, 22 May 2020 18:02:35 GMT
```

- Check the response message in the bottom.

❖ Testing in Postman:

- Launch Postman
- Skip the signup step
- Click on "Create a request"
- In the text box adjacent to GET provide the URL as <http://localhost:8080/news/api/languages/ss> and click Send.
- In the body section, the response error message should be displayed as specified below:

```
{
  "timestamp": "2020-05-22T18:30:01.517+0000",
  "status": 404,
  "error": "Not Found",
  "message": "Language not found",
  "path": "/news/api/languages/ss"
}
```

- Click headers to see the response headers
- Check if status is displayed as 404 Not Found.

Add language using POST

Problem

As part of expansion, the news website wants to provide news articles in German language, hence there is a need to add new language. Implement a REST API to add a new language.

Expected Input

The new language to be created is sent as JSON in the request body.

Access URL: `http://localhost:8080/news/api/languages`

Method Type: POST

Content Type: `application/json`

Request Body:

```
{"code": "de", "name": "German"}
```

Expected Output

HTTP Response:

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Tue, 01 Oct 2019 17:23:47 GMT

{"id":6,"code":"de","name":"German"}
```

Solution (Step #1 – Create a post handler method)

LanguageController.java

- ❖ Add new method.

```
@PostMapping
public void addLanguage() {
    LOGGER.info("Start");
}
```

- ❖ Save and run the application.
- ❖ Open Postman
- ❖ Click on any one of the existing request.
- ❖ Change method from GET to POST.
- ❖ Modify the URL to have the following URL:

```
http://localhost:8080/news/api/languages
```

- ❖ Click "Send"
- ❖ Check if Status is 200
- ❖ Click on code and then click HTTP. The display should look something like this.

```
POST /news/api/languages HTTP/1.1
Host: localhost:8080
```

- ❖ Make note that the method type is invoked as POST.
- ❖ Check the console to see if "Start" log in `addLanguage()` method is printed, which means that the method with `@PostMapping` is called.

Solution (Step #2 – Create a post handler method)

LanguageController.java

- ❖ Modify method addLanguage() as specified below:

```
@PostMapping
public Language addLanguage(@RequestBody Language language) {
    LOGGER.info("Start");
    LOGGER.debug("{} ", language);
    LOGGER.info("End");
    return language;
}
```

- ❖ Save the file and run the application.
- ❖ Open Postman
- ❖ Modify the existing request as specified below.
- ❖ Method: POST
- ❖ URL:

http://localhost:8080/news/api/languages

- ❖ Click on "Body" before the URL bar
- ❖ Copy and paste the JSON content below.

```
{"code": "de", "name": "German"}
```

- ❖ Click on the "Headers", the content type is defined as "text/html". This needs to be changed to "application/json".
- ❖ Click "Body"
- ❖ Then click the "Text" drop down and select "JSON".
- ❖ Check in Headers if the content type is changed accordingly.
- ❖ Click on "Code" and check if the HTTP content is displayed as specified below:

```
POST /news/api/languages HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{"code": "de", "name": "German"}
```

- ❖ Close the window and then click "Send"
- ❖ Check if Status is 200
- ❖ Check the content in the "Body" section in the bottom. It should look like the one below.

```
{
  "id": 0,
  "code": "de",
  "name": "German"
}
```

- ❖ To test using curl command, refer the command below:

```
curl -i -H 'Content-Type: application/json' -X POST -s -d
'{"code": "de", "name": "German"}' http://localhost:8080/news/api/languages
```

- ❖ Output


```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 23 May 2020 08:20:20 GMT
{"id":0,"code":"de","name":"German"}
```

Explanation

- ❖ Notice the following aspects done when sending the request:
 - Method defined as POST
 - The new language to be added is sent as JSON request.
- ❖ The `@RequestBody` annotation in the parameter of the `addLanguage()` method helps in converting the JSON received in the body into an object. Refer steps below:
 - Following is the JSON received from the request body:

```
{"code": "de", "name": "German"}
```
 - Since the parameter is of type `Language` a new instance of `Language` is created.
 - The Jackson library reads the first property `"code"`.
 - Initializes the first letter as capital which results as `"Code"`.
 - Then prefixed with `set`, which results as `"setCode"`.
 - The Jackson library then reads value of `"code"` as `"de"` and invokes the `setCode()` method on the newly created `Language` instance passing `"de"` as the parameter.
 - It repeats the above steps for each parameter.
- ❖ Then spring framework calls the `addLanguage()` method passing the newly created language object.
- ❖ The `setId()` method would not be called since it is not called available in the JSON request, hence it gets the default value 0.

Solution (Step #3 – Add item into language list)

LanguageService.java

- ❖ Include method.

```
public Language addLanguage(Language language) {
    LOGGER.info("Start");
    LOGGER.debug("{", language);
    language.setId(languageList.size() + 1);
    languageList.add(language);
    LOGGER.info("End");
    return language;
}
```

LanguageController.java

- ❖ Modify method `addLanguage()` as specified below:

```
@PostMapping
public Language addLanguage(@RequestBody @Valid Language language) {
    LOGGER.info("Start");
    LOGGER.debug("{} ", language);
    languageService.addLanguage(language);
    LOGGER.info("End");
    return language;
}
```

- ❖ Save the file and run the application.
- ❖ Open Postman and execute the previous request.

```
{
  "id": 6,
  "code": "de",
  "name": "German"
}
```

Validate data for POST request

Problem

As the POST request is a plain text, there are good possibilities to key in incorrect data. Moreover, hackers might try to pass inconsistent data which might affect the integrity of the application. Hence it becomes important that necessary checks are in place for all the fields.

Validations

- ❖ The language code is a two-character code. Implement the validation to check if the data sent is exactly two characters width.
- ❖ The language name cannot exceed 10 characters.

Expected Input

The new language to be created is sent as JSON in the request body.

Access URL: <http://localhost:8080/news/api/languages>

Method Type: POST

Content Type: application/json

Request Body:

```
{ "code": "d", "name": "German Language" }
```

Expected Output

HTTP Response

Status: 400 Bad Request

```
{
  "errors": [
    "Language code should be 2 characters",
    "Language cannot exceed 10 characters"
  ],
  "timestamp": "2020-05-23T10:00:46.151+0000",
  "status": 400
}
```

Solution (Step #1 – Respond with default error response)

Language.java

- ❖ Add below annotations:

```
@NotNull
@Size(min=2, max=2, message="Language code should be 2 characters")
private String code;

@Size(max=10, message="Language cannot exceed 10 characters")
private String name;
```

LanguageController.java

- ❖ Include @Valid annotation in to addLanguage() method parameter.

```
@PostMapping
public Language addLanguage(@RequestBody @Valid Language language) {
```

- ❖ Open Postman
- ❖ Click "Body" and modify the language code to "d"
- ❖ Click Send. The response should be like the one below:

```

{
  "timestamp": "2020-05-23T10:07:57.481+0000",
  "status": 400,
  "error": "Bad Request",
  "errors": [
    {
      "codes": [
        "Size.language.name",
        "Size.name",
        "Size.java.lang.String",
        "Size"
      ],
      "arguments": [
        {
          "codes": [
            "language.name",
            "name"
          ],
          "arguments": null,
          "defaultMessage": "name",
          "code": "name"
        },
        10,
        0
      ],
      "defaultMessage": "Language cannot exceed 10 characters",
      "objectName": "language",
      "field": "name",
      "rejectedValue": "German Language",
      "bindingFailure": false,
      "code": "Size"
    },
    {
      "codes": [
        "Size.language.code",
        "Size.code",
        "Size.java.lang.String",
        "Size"
      ],
      "arguments": [
        {
          "codes": [
            "language.code",
            "code"
          ],
          "arguments": null,
          "defaultMessage": "code",
          "code": "code"
        },
        2,
        2
      ],
      "defaultMessage": "Language code should be 2 characters",
      "objectName": "language",
      "field": "code",
      "rejectedValue": "e",
      "bindingFailure": false,
      "code": "Size"
    }
  ],
  "message": "Validation failed for object='language'. Error count: 2",
  "path": "/news/api/languages"
}

```

Explanation

- ❖ The @NotNull and @Size are from the package javax.validation.constraints. These are standards defined as part of validation.
- ❖ These annotations define the validation required for a specific field. An appropriate message can also be attached with the annotation so that it will be given as the reason in the response.
- ❖ The @Valid annotation initiates the validation on Language instance based on the validations defined in Language.java.
- ❖ If there is an validation error then the addLanguage() method itself will not be invoked. This can be verified in the logs.
- ❖ The response is constructed by the spring framework itself and provides all the details related to the validation error as response.
- ❖ The response is not in a more readable format, which will be corrected in subsequent steps.

Solution (Step #2 – Respond with customized error response)

GlobalExceptionHandler.java

- ❖ Create a new class GlobalExceptionHandler which extends from ResponseEntityExceptionHandler.
- ❖ Include annotation @ControllerAdvice at class level.
- ❖ Use Source > Override/Implement Method option to override the handleMethodArgumentNotValid method.
- ❖ Implement the coding in this method as specified below:

```
@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(
    MethodArgumentNotValidException ex, HttpHeaders headers,
    HttpStatus status, WebRequest request) {
    LOGGER.info("Start");
    Map<String, Object> body = new HashMap<>();
    body.put("timestamp", new Date());
    body.put("status", status.value());

    List<String> errors = ex.getBindingResult().getFieldErrors().stream()
        .map(x -> x.getDefaultMessage()).collect(Collectors.toList());
    body.put("errors", errors);

    LOGGER.info("End");
    return new ResponseEntity<>(body, headers, status);
}
```

- ❖ Save the file.
- ❖ Execute the same request in Postman. The result should get displayed like this.

```
{
  "errors": [
    "Language code should be 2 characters",
    "Language cannot exceed 10 characters"
  ],
  "timestamp": "2020-05-23T10:00:46.151+0000",
  "status": 400
}
```

Explanation

- ❖ The `@ControllerAdvice` annotation helps in defining a class that can handle exceptions at global application level.
- ❖ The `handleMethodArgumentNotValid()` is trigger whenever there is a failure when validating using `@Valid` tag.
- ❖ The method call `ex.getBindingResult().getFieldErrors()` returns the list of errors using an in-built data type `FieldErrors`. `Lambda Expressions` is used to iterate through the list of errors and accumulate them into a separate collection and return it as array of string.
- ❖ The `HashMap` contains the customized set of attributes and values that we want to respond as JSON.
- ❖ This class holds good for all the validation errors in this web application and this logic need not be repeated.

Create REST API to update language

Problem

If there are any changes in the language code or to correct any spelling mistakes in the language data, a update method will help in correcting the errors.

Expected Input

The new language to be created is sent as JSON in the request body.

Access URL: <http://localhost:8080/news/api/languages>

Method Type: PUT

Content Type: application/json

Request Body:

```
{ "id": 2, "code": "sp", "name": "Spanish" }
```

Expected Output

HTTP Response

Status: 200 OK

Solution

LanguageService.java

- ❖ Add new method.

```
public void updateLanguage(Language language) {  
    LOGGER.info("Start");  
    LOGGER.debug("{", language);  
    for (int i = 0; i < languageList.size(); i++) {  
        if (languageList.get(i).getId() == language.getId()) {  
            languageList.set(i, language);  
        }  
    }  
    LOGGER.info("End");  
}
```

LanguageController.java

- ❖ Add new method.

```
@PostMapping  
public void updateLanguage(@RequestBody @Valid Language language) {  
    LOGGER.info("Start");  
    LOGGER.debug("{", language);  
    languageService.updateLanguage(language);  
    LOGGER.info("End");  
}
```

- ❖ Save file.
- ❖ Open Postman.
- ❖ Modify the method type as "PUT"
- ❖ Provide below text in body:

```
{ "id": 2, "code": "sp", "name": "Spanish" }
```

- ❖ Click Send
- ❖ Check if response is with below status:

Status: 200 OK

- ❖ Open another GET request from the history and get all the languages to check if the data is modified.

Create REST API to delete language

Problem

Provide REST API to remove a language

Expected Input

The new language to be created is sent as JSON in the request body.

Access URL: <http://localhost:8080/news/api/languages/6>

Method Type: DELETE

Content Type: application/json

Expected Output

HTTP Response

Status: 200 OK

Solution

LanguageService.java

- ❖ Add new method.

```
public void deleteLanguage(Long id) {  
    LOGGER.info("Start");  
    LOGGER.debug("id: {}", id);  
    for (int i = 0; i < languageList.size(); i++) {  
        LOGGER.debug("{}", languageList.get(i));  
        if (languageList.get(i).getId() == id) {  
            languageList.remove(i);  
        }  
    }  
    LOGGER.info("End");  
}
```

LanguageController.java

- ❖ Add new method.

```
@DeleteMapping("/{id}")  
public void deleteLanguage(@PathVariable Long id) {  
    LOGGER.info("Start");  
    LOGGER.debug("id: {}", id);  
    languageService.deleteLanguage(id);  
    LOGGER.info("End");  
}
```

- ❖ Save file.
- ❖ Open Postman.
- ❖ Modify the method type as "DELETE"
- ❖ Change the URL to end with "/5"
- ❖ Remove the content in the Body
- ❖ Click Send
- ❖ This results in error.

```

{
  "timestamp": "2020-05-23T10:53:08.060+0000",
  "status": 500,
  "error": "Internal Server Error",
  "message": "Optional long parameter 'id' is present but cannot be translated into a null value due to declared as a primitive type. Consider declaring it as object wrapper for the corresponding primitive type.",
  "path": "/news/api/languages/5"
}

```

- ❖ Open Postman
- ❖ Run the delete request again, which should return the status as OK.
- ❖ Now run the GET request to get all languages and check if the item is removed.

HTTP Request Methods

Summary of HTTP method types that we have used so far.

HTTP Method	Usage Scenario
GET	Used to get data about a resource
POST	Used to create a resource
PUT	Used to update a resource
DELETE	Used to delete a resource

Based on this classification, the respective coding needs to be implemented in the server side.

REST API Naming Guidelines

- ❖ Each resource should have a unique and specific URL
- ❖ To get all resources provide the resource name in plural
- ❖ To get a specific resource provide resource name in plural followed with slash and parameter
- ❖ To create a resource the URL should be the resource name in plural and the data to create the resource should be defined in the payload
- ❖ To update a resource the URL should be the resource name in plural with data in payload
- ❖ To delete a resource the URL should be the resource name in plural followed by slash and the specific resource to delete
- ❖ Resource name with multiple words should be separated by hyphen and not with underscore. For example, if the resource is menu item implement the URL as "menu-item".

Type	URL	Description	Annotation
GET	http://sample.api.com/app-name/countries	Get all countries	@GetMapping
GET	http://sample.api.com/app-name/countries/{code}	Get a specific country	@GetMapping("/{id}")
POST	http://sample.api.com/app-name/countries	Create country based on data in post	@PostMapping
PUT	http://sample.api.com/app-name/countries	Update country based on data in post	@PutMapping
DELETE	http://sample.api.com/app-name/countries/{code}	Delete a specific country	@DeleteMapping("/{id}")

REST API Summary

- ❖ @RequestMapping – Class level URL mapping.
- ❖ MockMvc – Implemented end to end testing. Refer code snippet below.

```
@Test
public void getLanguage() throws Exception {
    ResultActions actions = mvc.perform(get("/news/api/languages"));
    actions.andExpect(status().isOk());
    actions.andExpect(jsonPath("$").isArray());
    actions.andExpect(jsonPath("$.length()").value(5));
}
```

- ❖ @PathVariable – Get the variable value from the path
- ❖ Exception Handling for Language Not Found – Exception created with @ResponseStatus annotation.

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason="Language not found")
public class LanguageNotFoundException extends Exception {
}

```

- ❖ @PostMapping
 - Handle post request to create a resource.
 - The request should set the header with content type as application/json.
 - Used @RequestBody to convert the JSON in the request to bean.
- ❖ @Valid – Validate the bean properties of the input data.
- ❖ Bean Validation

```
@NotNull
@Size(min=2, max=2, message="Language code should be 2 characters")
private String code;

@Size(max=10, message="Language cannot exceed 10 characters")
private String name;
```

Validation Annotations:

<https://docs.oracle.com/javaee/7/api/javax/validation/constraints/package-summary.html>

- ❖ Global Exception Handling for validations:
 - GlobalExceptionHandler class extending from ResponseExceptionHandler
 - @ControllerAdvice – Annotation to define global exception handling
 - Override of handleMethodArgumentNotValid() to hand @Valid errors
- ❖ @PutMapping – Handle PUT methods to update resource
- ❖ @DeleteMapping – Handle DELETE methods to delete resource.

Check your understanding

- ❖ List out the steps to create a REST API Service for reading data.
- ❖ What is the significance of @RequestMapping?
- ❖ List high level steps to implement end to end testing of REST API.
- ❖ How to implement business exceptions in REST API using Spring MVC?
- ❖ List the steps to implement validation of input.
- ❖ List the steps to implement global exception handling of validations.

Listing movies for a movie booking SPA

Problem

An online movie ticket booking web site is going to be developed in Angular or ReactJS. This application is going to be dependent on a REST API service that will return a list of movies that are currently screened. The reference to the page layout of this application is available in the link below:

<https://drive.google.com/file/d/1iE1mqFTWwrB4rg6ajVztp8hbA1LEcWEX/view?usp=sharing>

List of Genres

- ❖ Action, Adventure, Drama, Comedy, Horror, Thriller

List of MPAA Ratings

- ❖ G, PG-13, PG-18, R, NC-17

Solution (Step #1 - Data Model definition)

- ❖ Based on the above screen layout identify the model classes and attributes.

Solution (Step #2 - Data Model class definition)

- ❖ Create class `com.company.springlearn.model.Genre`. Generate getters/setters, `toString()`.

```
public class Genre {  
  
    private long id;  
    private String name;  
}
```

- ❖ Create class `com.company.springlearn.model.MpaaRating`. Generate getters/setters, `toString()`.

```
public class MpaaRating {  
  
    private long id;  
    private String name;  
}
```

- ❖ Create class `com.company.springlearn.model.Movie`. Generate getters/setters, `toString()`.

```
public class Movie {  
  
    private long id;  
    private String title;  
    private int duration;  
    private long budget;  
    private boolean bookingsOpen;  
    private float rating;  
    private Date releaseDate;  
    private MpaaRating mpaaRating;  
    private List<Genre> genres;  
}
```

- ❖ Create new file `movie.xml` in `src/main/resources` folder.
- ❖ Copy and paste the xml content from the link
<https://drive.google.com/file/d/1sgNmeNgzASB0oujSMaRtknzw7IsRG6Mg/view?usp=sharing>

Solution (Step #3 – Implement Service)

- ❖ Create class `com.company.springlearn.service.Service` with `@Service` annotation.

```
@Service  
public class MovieService {  
  
}
```

- ❖ Include logger.
- ❖ Include constructor

```

public MovieService() {
    LOGGER.info("Start");
    ApplicationContext context = new ClassPathXmlApplicationContext("movie.xml");
    movieList = context.getBean("movieList", ArrayList.class);
    LOGGER.info("End");
}

```

- ❖ Include method below.

```

public List<Movie> getMovies() {
    LOGGER.info("Start");
    return movieList;
}

```

Solution (Step #4 – Implement Controller)

- ❖ Create class `com.company.springlearn.controller.MovieController` with request mapping URL, logger and autowire `MovieService`.

```

@RestController
@RequestMapping("/booking/api/movies")
public class MovieController {
    public static final Logger LOGGER = LoggerFactory.getLogger(MovieController.class);

    @Autowired
    private MovieService movieService;
}

```

- ❖ Include method for getting all movie data.

```

@GetMapping
public List<Movie> getMovies() {
    LOGGER.info("Start");
    return movieService.getMovies();
}

```

- ❖ Start the application and test if the URL <http://localhost:8080/booking/api/movies> in browser displays the list of movies as JSON.
- ❖ Notice in the response that the release date is returned as specified below:

```
"releaseDate": "2019-08-30T18:30:00.000+0000"
```

- ❖ Instead it should be returned in "dd/MM/yyyy" format.
- ❖ Include the following annotation in `releaseDate` property of `Movie` class.

```

@JsonFormat(pattern = "dd/MM/yyyy")
private Date releaseDate;

```

- ❖ Save and check the JSON response if the `releaseDate` is returned in the expected format.
- ❖ Copy and paste the JSON text in an online viewer and check if the results are as expected.
- ❖ Check how genres and MPAA Rating is returned.

Solution (Step #5 – Access REST API from browser)

- ❖ Create a new HTML file named movies.html, the copy and paste the HTML Code below:

```
<!doctype html>
<html>
  <body>
    <h1>Movie List</h1>
    <code style="width:100%; height: 100px" id="movies"></code>
  </body>
  <script language="JavaScript">
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open('GET', 'http://localhost:8080/booking/api/movies', true);
    xmlHttp.send();
    xmlHttp.onreadystatechange = function () {
      if (xmlHttp.readyState == 4) {
        if (xmlHttp.status == 200) {
          var responseJson = xmlHttp.responseText;
          document.getElementById("movies").innerHTML = responseJson;
        }
      }
    };
  </script>
</html>
```

- ❖ Check the developer tools console in the browser, it will be displaying an error related to CORS.
- ❖ Create a new class com.company.springlearn.WebConfig.
- ❖ Implement the code below:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**").allowedMethods("*").allowedOrigins("*");
    }
}
```

- ❖ Now check in the browser if the JSON data is returned and displayed in the browser.

Explanation

- ❖ The expanded form for CORS is Cross-Origin Resource Sharing.
- ❖ This is a header in HTTP response, which denotes the browser to run the web application on one origin. Follow steps below to check in the browser:
 - Chrome Browser > Menu > More Tools > Developer tools > Network
 - Refresh the page and click on 'movies' in Network window. Check the headers for 'cors'
- ❖ This header ensures that hackers does not inject any JavaScript that belongs to a different domain in internet.
- ❖ In WebConfig class the CORS mapping setting to allow any URL and any method and origin. After this configuration the browser is able to read this data.
- ❖ Refer below image for understanding CORS.

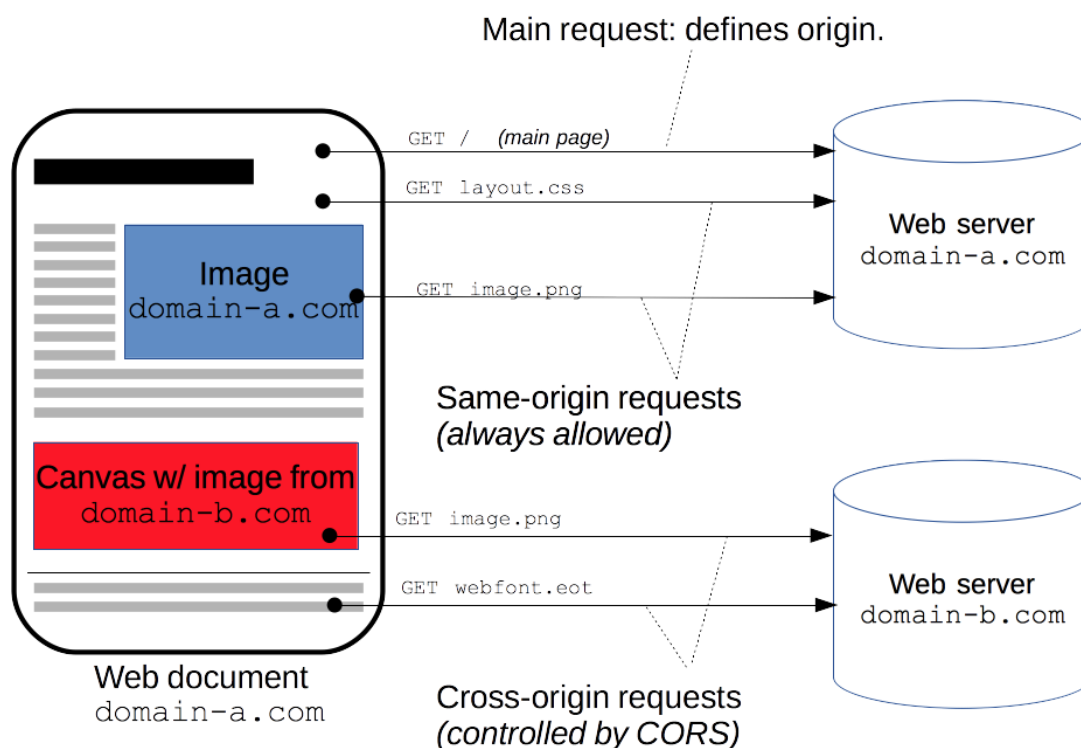


Image Courtesy: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

Best Bus Case Study

Implement a case study as part of learning Spring REST API development. Go through the case study link provided below:

Case Study Specification

<https://drive.google.com/drive/folders/1UoP-6kex-rMixitv3EcSYXk612VXcWQH>

Sample User Interface

<https://drive.google.com/file/d/1s6ZCgXnJ9muoBLBw69as2hxUVmKAWGGg/view?usp=sharing>

Next Steps

- ❖ Create a new spring boot application named "best-bus"
- ❖ Create model classes based on best bus requirement
- ❖ Create spring xml configuration file with data for best bus case study
- ❖ Implement REST API service for listing of bus schedule

Get a specific movie detail for editing

Problem

Clicking Edit link in movie list page should display the page to edit the movie details. Refer screen layouts below:

Movie List

<https://drive.google.com/file/d/1iE1mqFTWwrB4rg6ajVztp8hbA1LEcWEX/view?usp=sharing>

Edit Movie

<https://drive.google.com/file/d/17vuoi0o0Yelogo6TGxGDEtsrHZJyMq4E/view?usp=sharing>

Implement the following REST API services to display the Edit Movie form:

- ❖ Get information about a single movie based on the movie id.
- ❖ Get all MPAA Ratings to display the MPAA Rating drop down.
- ❖ Get all Genres to display the list of checkboxes.

Solution (Step #1 – Implement getting movie based on movie id)

- ❖ Open MovieService and include a new method.

```
public List<Movie> getMovies() {  
    LOGGER.info("Start");  
    return movieList;  
}
```
- ❖ Open MovieController and include a new method.

```
@GetMapping("/{movieId}")  
public Movie getMovie(@PathVariable long movieId) {  
    LOGGER.info("Start");  
    return movieService.getMovie(movieId);  
}
```
- ❖ Save and check the URL <http://localhost:8080/booking/api/movies/1> to see if it returns the respective movie details.
- ❖ Try changing the URL with different movie id and check if it returns the respective movie details.

Solution (Step #2 – Get all MPAA Rating)

- ❖ Create MpaaRatingService class and implement a method to return all MPAA Ratings from the xml configuration file.
- ❖ Create MpaaRatingController class that autowires MpaaRatingService and create a new method to get all MPAA Ratings from the service.
- ❖ Define the @RequestMapping URL as booking/api/mpaa-ratings

Solution (Step #3 – Get all Genres)

- ❖ Create GenreService class and implement a method to return all MPAA Ratings from the xml configuration file.
- ❖ Create GenreController class that autowires GenreService and create a new method to get all MPAA Ratings from the service.
- ❖ Define the @RequestMapping URL as booking/api/genres

Save movie details

Problem

After user modifies data in Edit Movie and clicks Save, the updated movie details of the form will be sent as a REST API call with PUT method. Refer sample movie JSON data below.

```
{
  "id": 2,
  "title": "The Lord of the Rings: The Fellowship of the Ring",
  "genres": [
    {
      "id": 3,
      "name": "Drama",
      "selected": true
    },
    {
      "id": 4,
      "name": "Comedy",
      "selected": true
    }
  ],
  "releaseDate": "19/12/2002",
  "mpaaRating": {
    "id": "3",
    "name": "PG-13"
  },
  "rating": "8.9",
  "duration": "180",
  "bookingOpen": false,
  "budget": "800000000",
  "favorite": false
}
```

Solution

- ❖ Open MovieService and include a new method.

```
@PutMapping
public void updateMovie(@RequestBody Movie movie) {
    LOGGER.info("Start");
    LOGGER.debug("{", movie);
    movieService.updateMovie(movie);
    LOGGER.info("End");
}
```

- ❖ Open MovieController and include a new method.

```
@PutMapping
public void updateMovie(@RequestBody Movie movie) {
    LOGGER.info("Start");
    LOGGER.debug("{", movie);
    movieService.updateMovie(movie);
    LOGGER.info("End");
}
```

- ❖ Save the file
- ❖ In Postman try test the implementation with the following options.
 - URL: <http://localhost:8080/booking/api/movies>
 - Method: PUT
 - Body > Content Type > JSON
 - Body: Copy and paste the JSON provided in the problem statement
- ❖ Click "Send" and check if the response status is 200.
- ❖ Make another REST API call from Postman to get all movies and verify if the data for movie with id 2 have different values.

Securing REST API with Spring Security

Problem

Implement Basic HTTP based authentication for all the REST API methods implemented so far in spring-learn application.

Solution

- ❖ Open pom.xml, then copy and paste code below within dependencies:

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

- ❖ Save the file and run the SpringLearnApplication
- ❖ Check the console logs.
- ❖ Notice that password is automatically generated when the server starts:

```
04-06-20 16:11:59.388 [tartetMain] INFO etailsServiceAutoConfiguration
Using generated security password: 7249d2e5-e33d-4de5-8808-075c730bf546
04-06-20 16:12:01.003 [tartetMain] INFO s.w.DefaultSecurityFilterChain
```

- ❖ Copy the password and save it in a text file for future reference.
- ❖ In Postman get movie list with method type GET.
- ❖ Response Status should be 401, which represents Unauthorized.
- ❖ JSON Response:

```
{
  "timestamp": "2020-06-04T10:49:48.473+0000",
  "status": 401,
  "error": "Unauthorized",
  "message": "Unauthorized",
  "path": "/booking/api/movies"
```

- ❖ Follow steps below the send the request with credentials.
 - In browser, go to <https://www.base64encode.org/>
 - Copy and paste the password in the text box, then prefix the password with "user:"

Encode to Base64 format

Simply enter your data then push the encode button.

user:7249d2e5-e33d-4de5-8808-075c730bf546

- Click "Encode" button
- Copy the encoded value generated in the bottom text box.

> ENCODE <

Encodes your data into the textarea below.

dXNlcj03MjQ5ZDJINS1IMzNkLTRkZTU0ODgwOC0wNzVjNzMwYmY1NDY=

- Open Postman
- Click "Headers"
- Click "Key" and enter the value as "Authorization"
- In adjacent "Value" section type "Basic ". Include a space after "Basic". Then paste the encoded password generated earlier.

☒ Authorization Basic dXNlcjo3MjQ5ZDJINS1IMzNkLTRkZTUtODgwOC0wNzVjNzMwYmY1NDY= ...

- Now click "Send", it should return the movie list successfully with Status code as 200.
- Try making a small change in the credentials to check if authentication fails.

Explanation

- ❖ In HTTP there is a pre-defined list of authentication schemes. Find below the list of the authentication schemes:

Authentication Scheme Name	Reference
Basic	[RFC7617]
Bearer	[RFC6750]
Digest	[RFC7616]
HOBA	[RFC7486, Section 3]
Mutual	[RFC8120]
Negotiate	[RFC4559, Section 3]
OAuth	[RFC5849, Section 3.5.1]
SCRAM-SHA-1	[RFC7804]
SCRAM-SHA-256	[RFC7804]
vapid	[RFC 8292, Section 3]

Reference Link: <https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml>

- ❖ In our example, we have implemented the "Basic" authentication.
- ❖ As per the Basic authentication scheme we need to use the request header parameter "Authorization".
- ❖ In this header we need to pass the value in the below specified format:

Basic {base64-encoded-credentials}

- ❖ Base64 encoding format converts text into non readable format based on an algorithm.
- ❖ Base64 encoding does not help in securely sending the credentials, it just helps in hiding the password for human eyes. Whereas based on the encoding algorithm, the Base64 encoded value can be decoded back to original value.
- ❖ We used a third-party website to do the conversion from text to Base64.
- ❖ The implementation in the spring framework converts the Base64 encoding and checks if it matches with the generated password.

Securing REST API with Username and Password

Problem

Implement security with an Username and Password instead of randomly generated password.

Solution

- ❖ Create a class 'com.company.springlearn.security' with class 'SecurityConfig'
- ❖ Annotate @Configuration and @EnableWebSecurity and extend WebSecurityConfigurerAdapter
- ❖ Include the below specified function:

```
@Bean
public PasswordEncoder passwordEncoder() {
    LOGGER.info("Start");
    return new BCryptPasswordEncoder();
}
```

- ❖ Include the below specified function:

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    LOGGER.info("Start");
    auth.inMemoryAuthentication()
        .withUser("user")
        .password(passwordEncoder().encode("pwd"))
        .roles("USER");
    LOGGER.info("End");
}
```

- ❖ Save the file and start the application.
- ❖ Check the logs to see if SecurityConfig class is initialized.
- ❖ Also notice that the configure() method and passwordEncoder() methods are called.
- ❖ Try accessing the REST API to get all movies and it should fail with status code 401.
- ❖ In browser, go to <https://www.base64encode.org/>
- ❖ Provide "user:pwd" as input and the encoded value from the box below.
- ❖ Paste encoded value after Basic in the Authorization header in Postman and click "Send".
- ❖ This should return the list of movies.

Explanation

- ❖ The BCryptPasswordEncoder class helps in Base64 conversion.
- ❖ The configure() method defines one user credential with the role as 'USER'. Further fine grained authorization can be implemented using the role. Refer code snippet below.

```
1. @Override
2. protected void configure(HttpSecurity http) throws Exception {
3.     http
4.         .csrf().disable()
5.         .authorizeRequests()
6.
7.         .antMatchers("/managers/status/check").hasAnyAuthority("DELETE_USER_AUTHORITY")
8.         .antMatchers("/users/status/check").hasRole("USER")
9.         .anyRequest().authenticated()
10.        .and()
11.        .httpBasic()
12.        .and()
13.        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
14. }
```

- ❖ For learning reasons, we are creating user credentials use the options provided in spring framework, ideally the username and password has to come from the database.

Sending Mail using Spring Boot

Activity

Send a test mail using Spring Boot Mail Starter.

Steps

- ❖ Open pom.xml
- ❖ Copy and paste an existing spring boot starter dependency
- ❖ Modify the end part of the starter name as "mail"
- ❖ Save the pom.xml
- ❖ Wait for the build to complete
- ❖ Open Dependency Hierarchy and look through the content for mail dependencies
- ❖ The jakarta-mail and jakarta-activation are the core libraries to send and receive mails.
- ❖ Open application.properties. Copy and paste the code below:

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=user@gmail.com
spring.mail.password=password
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

- ❖ Modify username and password with your respective gmail id and password.
- ❖ Create new controller class com.company.springlearn.controller.MailController
- ❖ Include @RestController annotation
- ❖ Include LOGGER
- ❖ Include below auto wiring.

```
@Autowired
private JavaMailSender mailSender;
```

- ❖ Include below method.

```
@GetMapping("/send-test-mail")
public void sendMail() {
    LOGGER.info("Start");
    SimpleMailMessage message = new SimpleMailMessage();
    message.setTo("jjchandra@gmail.com");
    message.setSubject("Test Mail");
    message.setText("Test Message");

    mailSender.send(message);
    LOGGER.info("End");
}
```

- ❖ Save and start the application.
- ❖ Open Postman
- ❖ Modify the get movie URL as <http://localhost:8080/send-test-mail>. Using previous post is to ensure that we are sending the authentication credentials.
- ❖ Click "Send"
- ❖ This should return error code 500 which means "Internal Server Error".
- ❖ Check the application logs. It should contain exception trace with 'bad credentials' message.
- ❖ Open the link below in browser:

<https://myaccount.google.com/lesssecureapps?pli=1>

- ❖ Enable the setting "Allow less secure apps"
- ❖ Retry the request in Postman
- ❖ If the request is success with status 200 and blank response, check if there is a test mail in your gmail inbox, which means the mail is sent successfully.
- ❖ Go to myaccount link above again and change the settings back to OFF state.

Explanation

- ❖ JavaMailSender and SimpleMailMessage are implementations from spring framework, which helps in simplifying the mail sending steps. Refer code below, if we have to use the core library.

```
String to = "to@gmail.com";
final String from = "from@gmail.com";

Properties properties = new Properties();
properties.put("mail.smtp.starttls.enable", "true");
properties.put("mail.smtp.auth", "true");
properties.put("mail.smtp.host", "smtp.gmail.com");
properties.put("mail.smtp.port", "587");

Session session = Session.getInstance(properties,
    new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(from, "password");
        }
    });

MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
message.setSubject("Hello");
message.setText("What's up?");

Transport.send(message);
```

- ❖ SMTP refers to Simple Mail Transfer Protocol
- ❖ The gmail server connection details are provided in application.properties.
- ❖ TLS stands for Transport Layer Security, which ensures that the information sent is secured.
- ❖ We are enabling secured data transferring using "mail.smtp.starttls.enable"
- ❖ Using "mail.smtp.auth", the authentication is mandated.
- ❖ The gmail server is referred using "spring.mail.host" and "spring.mail.port".
- ❖ The JavaMailSender class helps in preparing the configuration to send and sends the mail.
- ❖ The SimpleMailMessage class helps to define the mail content.

Demonstrate Spring Boot Actuator

Activity

Demonstrate usage of Spring Boot Actuator

Steps

- ❖ Stop the application
- ❖ Open pom.xml
- ❖ Copy and paste below dependencies:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-rest-hal-browser</artifactId>
</dependency>
```

- ❖ Open application.properties. Copy and paste the code below:

```
management.security.enabled=false
management.security.roles=ADMIN
security.basic.enabled=true
security.user.name=admin
security.user.password=admin
```

- ❖ Save and start the application.
- ❖ Open Postman
- ❖ Modify the previous URL as <http://localhost:8080/actuator>. Using previous post is to ensure that we are sending the authentication credentials.
- ❖ Click "Send" and refer expected output below.

```
{
  "_links": {
    "self": {
      "href": "http://localhost:8080/actuator",
      "templated": false
    },
    "health-path": {
      "href": "http://localhost:8080/actuator/health/{*path}",
      "templated": true
    },
    "health": {
      "href": "http://localhost:8080/actuator/health",
      "templated": false
    },
    "info": {
      "href": "http://localhost:8080/actuator/info",
      "templated": false
    }
  }
}
```

- ❖ Modify the previous URL as <http://localhost:8080/actuator/health>.
- ❖ Click "Send", the output should look something like the below.

```
{
  "status": "UP"
}
```

Explanation

- ❖ Spring Boot Actuator is a sub project of Spring Boot
- ❖ Actuator helps to monitor and manage Spring Boot Application
- ❖ The [link](#) contains details about all the end points.
- ❖ HAL stands for JSON Hypertext Application Language
- ❖ This methodology helps to self explain about the various endpoints. Notice actuator link returns the list of possible other links available.
- ❖ When security is enabled in the properties file it helps to secure the actuator links.

Why logging is important?



Miscellaneous Topics

URI Patterns

Various ways to define patterns:

<https://docs.spring.io/spring/docs/5.3.0-SNAPSHOT/spring-framework-reference/web.html#mvc-ann-requestmapping-uri-templates>

Method Arguments

<https://docs.spring.io/spring/docs/5.3.0-SNAPSHOT/spring-framework-reference/web.html#mvc-ann-arguments>

Request Param

<https://docs.spring.io/spring/docs/5.3.0-SNAPSHOT/spring-framework-reference/web.html#mvc-ann-requestparam>

Request Header

<https://docs.spring.io/spring/docs/5.3.0-SNAPSHOT/spring-framework-reference/web.html#mvc-ann-requestheader>

Request Header

<https://docs.spring.io/spring/docs/5.3.0-SNAPSHOT/spring-framework-reference/web.html#mvc-ann-requestheader>

Exception Handling

<https://docs.spring.io/spring/docs/5.3.0-SNAPSHOT/spring-framework-reference/web.html#mvc-ann-rest-exceptions>

Interview Questions

Spring Core

<https://www.greycampus.com/blog/programming/top-spring-interview-questions-and-answers>

Spring Boot

<https://www.javatpoint.com/spring-boot-interview-questions>

Spring REST

<https://dzone.com/articles/top-20-spring-mvc-interview-questions-answers>

