UNIT 9 INTRODUCTION TO SPRING BOOT

Structure **Page No.**

9.0 INTRODUCTION

Fast, secured and cost-effective development of software plays a vital role in the growth of an organization. Java has been the popular choice for many enterprises and programmers since the mid 90's for designing and developing software. Hence, lots of frameworks are being developed to ease the software development using Java.

In order to make the software development easy, fast and effective several frameworks have been introduced such as Struts, Spring and ORM tools such as Toplink, Hibernate, Eclipse Link, Ibatis.

Spring based enterprise application requires many configurations and complicated dependency management which make the set up tedious and error-prone. Each time spring-based enterprise application requires repetition of the same configuration steps:

- Based on application type, import the required dependencies such as spring mvc, spring jpa, spring jdbc, spring rest etc.

- Import the specified Spring version compatible third-party libraries such as hibernate, Jackson etc.
- Configure web layer beans such as view resolver, resource manager etc.
- Configure DAO layers such as data source, transaction management, entity manager etc.
- Import web container libraries in the case of web applications.

The spring team considered simplifying the routine configuration and facilitating the developers with some utility that automates the configuration process and speeds up the spring-based application's build and deployment process. Spring has evolved a lot in recent years and introduced many new modules such as Spring Boot, Spring Security, Spring Cloud, Spring Data etc.

Spring Boot is a utility project which enables an organization to develop production-ready spring-based applications and services with less effort, reduced cost and minimal configuration. ***Spring Boot*** facilitates the developers to create a Spring based web application with minimum lines of code since it provides auto-configuration out-of-the-box. In this unit Tomcat and Tomcat are used interchangeably.

9.1 OBJECTIVES

After going through this unit, you will be able to:

- describeSpring Boot and its features,
- describeSpring Boot starters and runners,
- use Spring Boot annotations,
- set application properties and profile-based application properties,
- use DevTools and Actuator,
- perform command line Spring Boot application execution, and
- perform step by step development and execution of Spring Boot web app.

9.2 SPRING BOOT:AN OVERVIEW

Spring Boot is an extension of the Spring framework that takes one step ahead and simplifies the configuration in order to fasten the development and execution of the spring application. It eliminates the boilerplate configuration required for a spring application. It is a module that enriches the Spring framework with Rapid Application Development (RAD) feature. It provides an easy way to create a stand-alone and production ready spring application with minimum configurations. Spring Boot provides comprehensive infrastructure support for the development and monitoring of the enterprise-ready applications based on micro services. Spring Boot is a combination of **Spring framework with auto-configuration** and **embedded Servers**.

Spring Boot provides a vast number of features and benefits. A few of them are as follows:

1) Everything is auto-configured in Spring Boot.

2) Spring Boot starter eases the dependency management and application configuration
3) It simplifies the application deployment by using an embedded server
4) Production ready features to monitor and manage applications, such as health checks, metrics gathering etc.
5) Reduces the application development time and run the application independently
6) Very easy to understand and develop Spring application
7) Increases productivity and reduces the cost

9.2.1 Spring Boot Working and Annotations

A class with the main method and annotated with **@SpringBootApplication** is the entry point of the Spring Boot application. Spring Boot auto-configures all required configurations. It performs auto-configuration by scanning the classes in class-path annotated with @Component or @Configuration. @SpringBootApplication annotation comprising the following three annotations with their default values-

- @EnableAutoConfiguration
- @ComponentScan
- @Configuration

@EnableAutoConfiguration

As the name implies, it enables auto-configuration. It means **Spring Boot** looks for auto-configuration beans on its class-path and automatically applies them. For example, if H2 dependency is added and you have not manually configured any database connection beans, then Spring will auto-configure H2 as an in-memory database.

The *@EnableAutoConfiguration* annotation should be applied in the root package so that every sub-packages and class can be examined.

@ComponentScan

The ***@ComponentScan*** annotation enables Spring to scan for things like configurations, controllers, services, and other components we define. By default, the scanning starts from the package of the class declaring @ComponentScan. The component scanning can be started from specified packages by defining ***basePackageClasses***, ***basePackages*** attributes into **@ComponentScan**. The **@ComponentScan** annotation is used with **@Configuration.**

@Configuration

The @Configuration annotation is used for annotation-based configuration into the Spring application. A class annotated with **@Configuration** indicates that the class declares one or more **@Bean** methods and can be processed by Spring container to generate bean definitions and service requests for those beans at runtime.

Other available annotations in the Spring Boot are described below:

@ConditionalOnClass and @ConditionalOnMissingClass

Spring @ConditionalOnClass and @ConditionalOnMissingClass annotations
let @Configuration classes be included based on the presence or absence of specific

classes. Hence, @ConditionalOnClass loads a bean only if a certain class is on the classpath and @ConditionalOnMissingClass loads a bean only if a certain class is not on the classpath.

@CoditionalOnBean and @ConditionalOnMissingBean

These annotations are used to define conditions based on the **presence or absence** of a specific bean.

@ConditionalOnProperty

The annotation is used to conditionally create a Spring bean depending on the configuration of a property.

```
@Bean
@ConditionalOnProperty(name = "usepostgres", havingValue = "local")
DataSourcedataSource()
{
   // ...
}
```

The DataSource bean will be created only if property **usepostgres** exists and it has value as **local.**

@ConditionalOnResource

The annotation is used to conditionally create a Spring bean based on the presence or absence of resources. The SpringConfig class beans will be created if the resource log4j.properties file is present.

```
@Configuration
@ConditionalOnResource(resources = { "log4j.properties" })
class SpringConfig
{
        @Bean
        public Log4j log4j()
        {
                return new Log4j();
        }
}
```

9.2.2 Spring Boot Starter

Before describing Spring Boot Starter, the following section explains all the required things to develop a web application.

Development of a web application using Spring MVC will require identifying all the required dependencies, compatible versions and how to connect them together. Followings are some of the dependencies which are used in Spring MVC web application. For all the dependencies we need to choose the compatible version dependencies.

```
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>4.2.2.RELEASE</version>
</dependency>

<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.5.3</version>
</dependency>

<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-validator</artifactId>
<version>5.0.2.Final</version>
</dependency>
<dependency>
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.17</version>
</dependency>
```

Dependency management is a very important and complex process in any project. Manual dependency management is error-prone, non-ideal and non-suggestible. To focus more on aspects apart from dependency management in web application development will require some solution that addresses compatible dependency management problems.

Spring Boot starters are the dependency descriptor that addresses the problem of compatible versions of dependency management. Spring Boot Starter POMs are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need from Spring Boot starters. Spring Boot provides a number of starters that make development easier and rapid. Spring Boot starters follow a similar naming pattern as spring-boot-starter-*, where * denotes a particular type of application. For instance, developing a rest service that requires libraries like Spring MVC, Tomcat, Log and Jackson, a single Spring Boot starter named spring-boot-starter-web needs to be included. Spring Boot provides many numbers starter, and a few of them are listed below-

spring-boot-starter-web	It is used for building web applications, including RESTful applications using Spring MVC. It uses Tomcat as the default embedded container.
spring-boot-starter-jdbc	It is used for JDBC with the Tomcat JDBC connection pool.
spring-boot-starter-validation	It is used for Java Bean Validation with Hibernate Validator.
spring-boot-starter-security	It is used for Spring Security.
spring-boot-starter-data-jpa	It is used for Spring Data JPA with Hibernate.
spring-boot-starter	It is used as a core starter, including auto-configuration support, logging etc.
spring-boot-starter-test	It is used to test Spring Boot applications with libraries, including JUnit, Hamcrest, and Mockito.
spring-boot-starter-thymeleaf	It is used to build MVC web applications using Thymeleaf views.

9.2.3 Spring Boot Project Structure

Spring Boot provides a high degree of flexibility in code layout. However, there are certain best practices that will help to organize the code for better readability.

In Spring Boot, default package declaration is not recommended since it can cause issues such as malfunctioning of auto-configuration or Component Scan. Spring Boot annotationslike:

@ComponentScan,

@EntityScan,

@ConfigurationPropertiesScan, and

@SpringBootApplication

use packages to define scanning locations. Thus, the default package should be avoided.

The *@SpringBootApplication* annotation triggers component scanning for the current package and its sub-packages. Therefore, the main class of the project should reside in the base package to use the implicit components scan of Spring Boot.

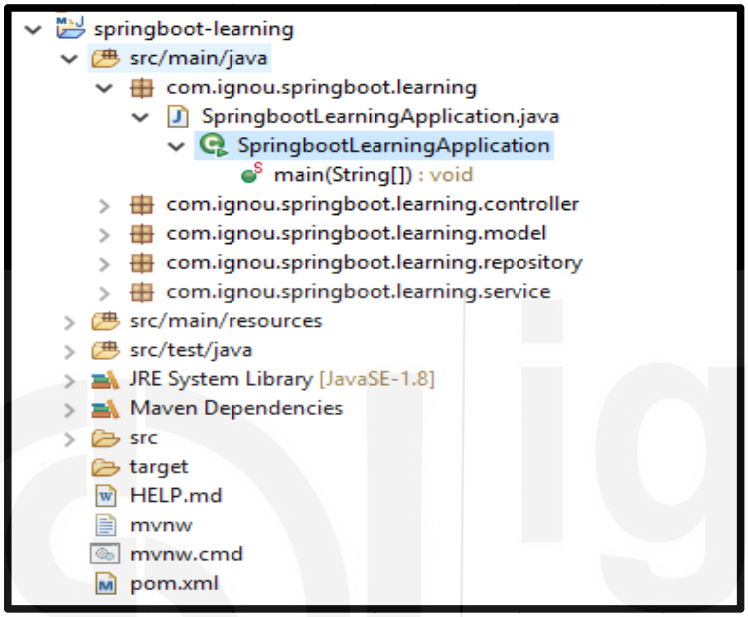


Figure 9.1: SpringBoot Project Structure For Default Scanning

In the above code structure, controller, model, repository and service are sub-package of base package **com.ignou.springboot.learning.** All the components and configurations are eligible for implicit scanning with main class as below:

```
package com.ignou.springboot.learning;

importorg.springframework.boot.SpringApplication;

importorg.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringbootLearningApplication

{
	public static void main(String[] args)
	{

	SpringApplication.run(SpringbootLearningApplication.class,
args);
	}
}
```

Spring Boot has a high degree of flexibility. Thus, it allows us to relocate the scanning elsewhere by specifying the base package manually.

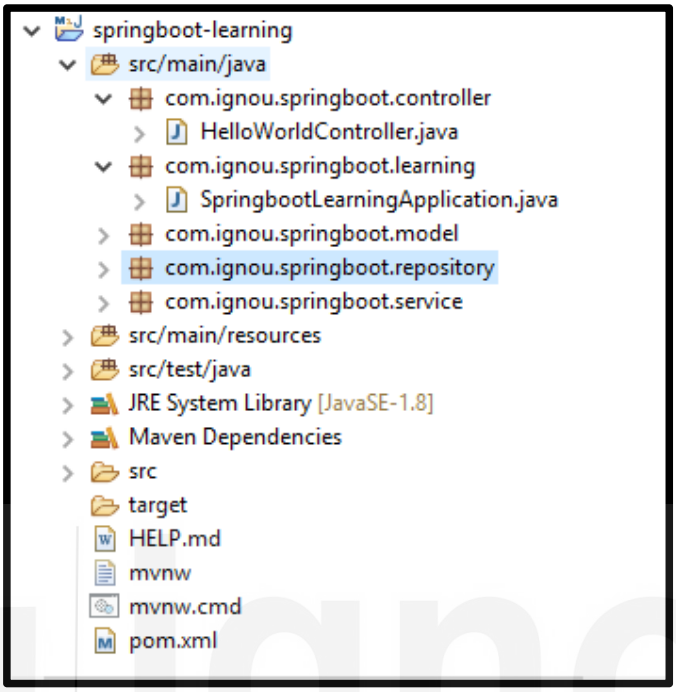


Figure9.2: SpringBoot Project Structure For Custom Scan Package

```
package com.ignou.springboot.learning;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication(scanBasePackages= {"com.ignou.springboot"})

public class SpringbootLearningApplication
{
	public static void main(String[] args)
	{
		SpringApplication.run(SpringbootLearningApplication.class, args);
	}
}
```

In above package structure, component and configuration packages are not sub-package of main class. Thus, scanning location has been explicitly defined as @SpringBootApplication(scanBasePackages= {"com.ignou.springboot"})

9.2.4 Spring Boot Runners

Spring Boot provides two runner interfaces named as ApplicationRunner and CommandLineRunner. These interfaces enable you to execute a piece of code just after the Spring Boot application is started.

Both interfaces are Functional Interface. If any piece of code needs to be executed when Spring Boot Application starts, we can implement either of these functional interfaces and override the single method**run.**

Application Runner

Spring bean which implements the **ApplicationRunner**interfaces in a Spring Boot Application, the bean gets executed when application is started. ApplicationRunner example is as following-

```
packagecom.ignou.springboot.learning;

importorg.springframework.boot.ApplicationArguments;

importorg.springframework.boot.ApplicationRunner;

importorg.springframework.stereotype.Component;

@Component

publicclassApplicationRunnerImplimplementsApplicationRunner
{
	@Override
	publicvoid run(ApplicationArgumentsargs) throws Exception
	{
		System.out.println("Application Runner Executed.");
	}
}
```

The execution result of Spring Boot application having the above component is shown below, and you can observe that the bean implementing ApplicationRunner is executed just after the application starts.

Figure 9.3: Execution Result of ApplicationRunner

Command Line Runner

The ***Command LineRunner*** interface is also meant for the same purpose as ApplicationRunner interface. Spring bean which implements the ***CommandLineRunner*** interface in a Spring Boot Application, the bean gets executed when the application is started. ***CommandLineRunner*** example is as following-

```
packagecom.ignou.springboot.learning;

importorg.springframework.boot.CommandLineRunner;

importorg.springframework.stereotype.Component;

@Component
```

```
publicclassCommandLineRunnerImplimplementsCommandLineRunner

{
	@Override
	publicvoid run(String... args) throws Exception
	{
		System.out.println("Command Line Runner executed");
	}
}
```

Execution result of Spring Boot application with the above component is shown below and you can observe that the bean implementing ApplicationRunner is executed just after application starts.

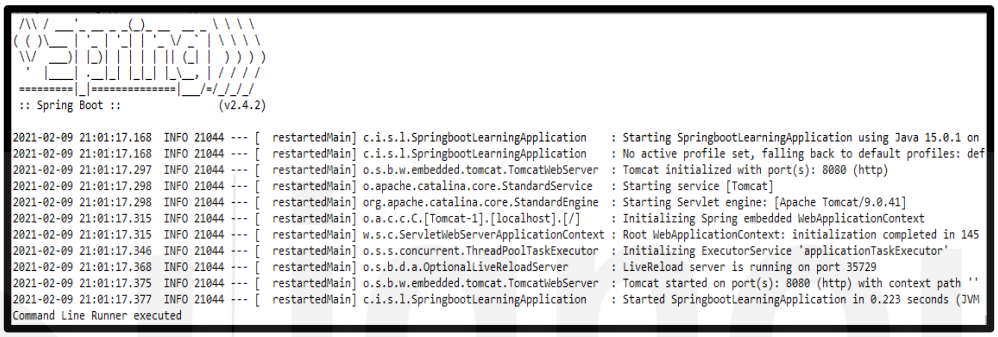


Figure 9.4: Execution Result Of CommandLineRunner

9.2.5 Spring Boot Web Application Using Thymeleaf

The previous sections have described the basic concepts of Spring Boot. This section explains the process of creating a "Hello World" website using the Spring Boot concepts. Required tools and software are as follows:

- Eclipse
- Maven
- Spring Boot
- JDK 8 or above

Step 1: Create a Spring Boot Project using Spring Initializr. Visit at **https://start.spring.io/** and generate the Spring Boot project with added dependency. For the project,**Spring Web** and **Thymeleaf** dependency will be used.

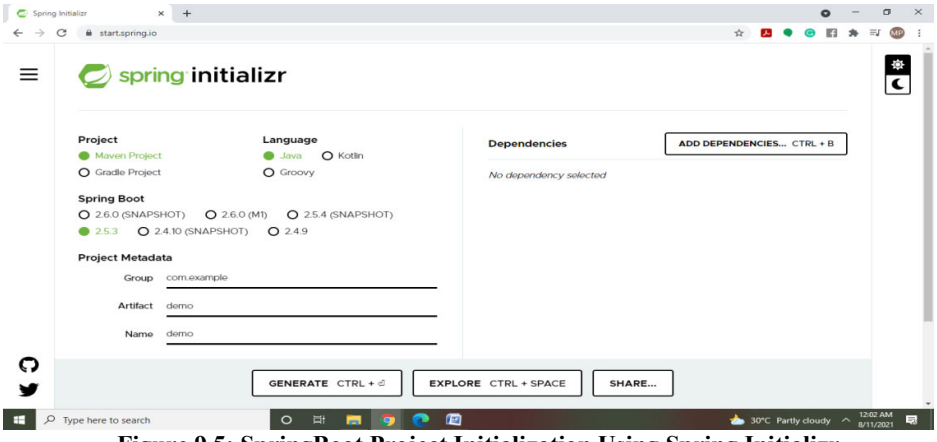


Figure 9.5: SpringBoot Project Initialization Using Spring Initializr

Step 2: Extract the generated project and Import this into eclipse as maven project. Project structure is shown below.

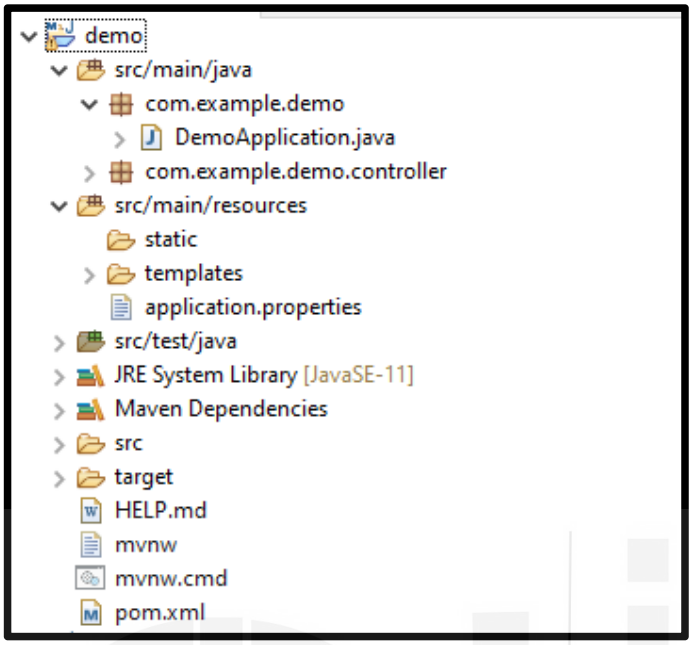


Figure 9.6: SpringBoot Project Structure

The controller is a sub-package of the base package. Thus, implicit scanning of components and configuration will be used.

Step 3: Add a HomeController into the controller package. HomeController has two endpoints:

- "/"
- "/greeting?name=Jack"

```
packagecom.example.demo.controller;
importorg.springframework.stereotype.Controller;
importorg.springframework.ui.Model;
importorg.springframework.web.bind.annotation.GetMapping;
importorg.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestParam;
@Controller

publicclassHomeController
{
	@RequestMapping(value="/")
	public String home()
	{
		return"home";
	}
	@GetMapping("/greeting")
	public String greeting(@RequestParam(name="name",
required=false,defaultValue="World") String name, Model model)
	{
		model.addAttribute("name", name);
		return"greeting";
```

```
        }
}
```

Step 4: Thymeleaf is used as a view technology in this example. For both the end point, add the corresponding HTML template file into src/main/resources/templates

home.html

```
<!DOCTYPEhtml>

<html>

<head>

<metacharset="ISO-8859-1"/>

<title>Spring Boot Web Application</title>

</head>

<body>

<h1>Welcome to Thymeleaf Spring Boot web application</h1>

</body>

</html>
```

greeting.html

```
<!DOCTYPEHTML>

<htmlxmlns:th="http://www.thymeleaf.org">

<head>

<title>Getting Started: Serving Web Content</title>

<metahttp-equiv="Content-Type"content="text/html; charset=UTF-8"/>

</head>

<body>

<pth:text="'Hello, ' + ${name} + '!'"/>

</body>

</html>
```

Step 5: Now the project is ready to run. Run **DemoApplication.java** file as Java Application in eclipse as shown below.

Access the endpoints, as mentioned in step 3, into the browser. You will get the

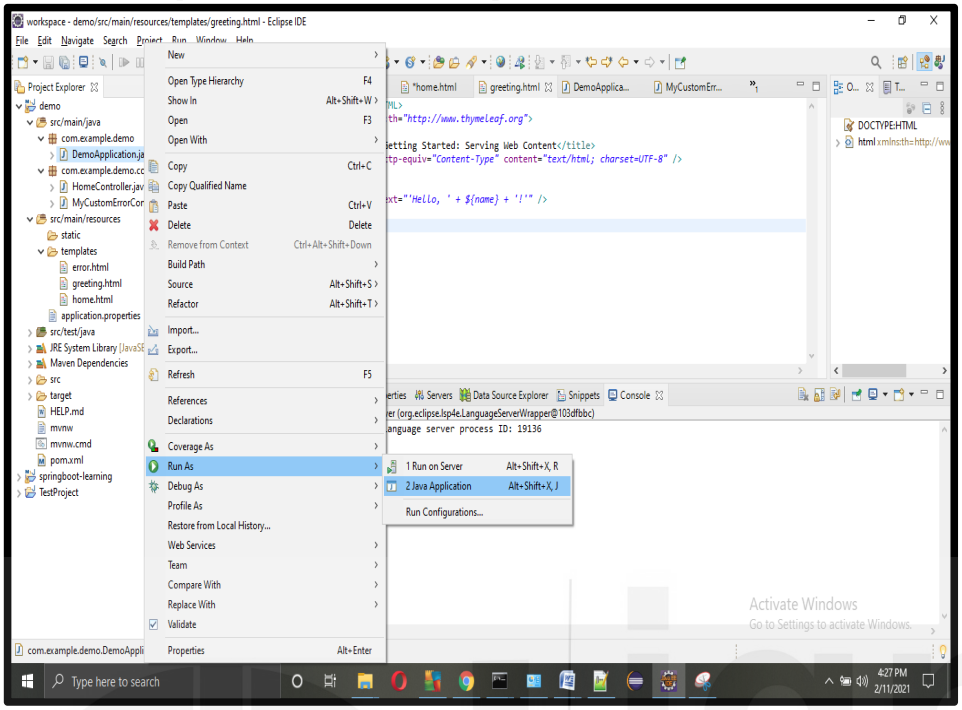


Figure 9.6: SpringBoot Project Execution Into Eclipse

following output in the browser.

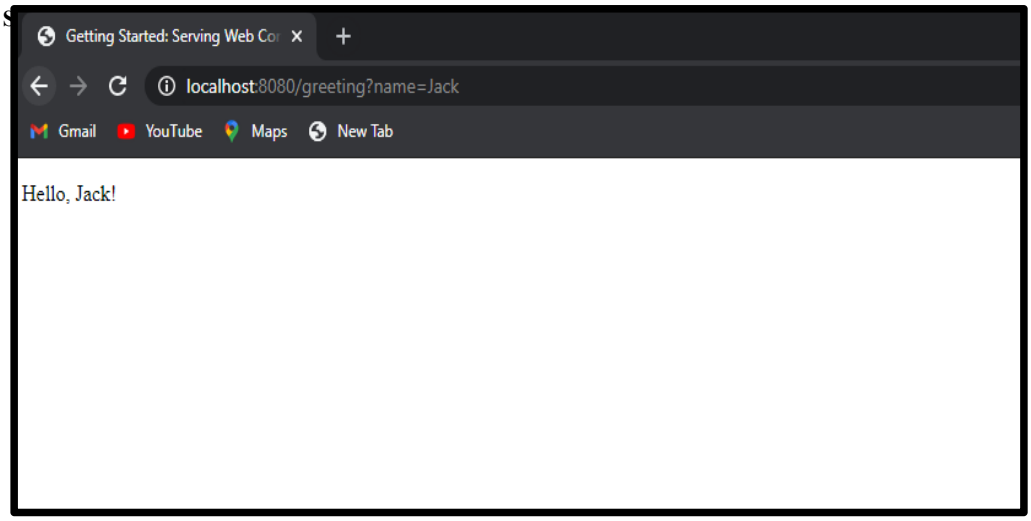


Figure9.7: Execution Result 1

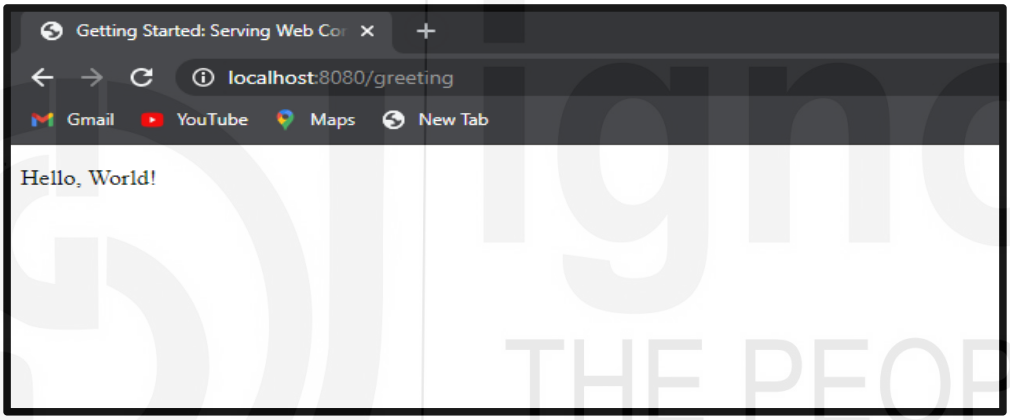


Figure 8.9: Execution Result 2

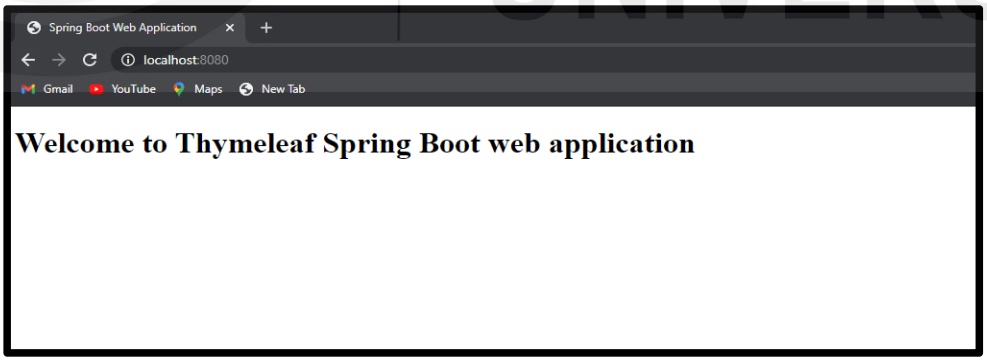


Figure 9.9: Execution Result 3

☛Check Your Progress 1

1) What is Spring Boot?Explain its need.

……
……
……
……
……

2) Describe Spring Boot working with @SpringBootApplication.

……
……
……
……
……

3) What are the Spring Boot starters, and what are the available starters?

……
……
……
……
……

4) What do you understand by auto-configuration in Spring Boot and how to disable the auto-configuration?

……
……
……
……
……

5) What is the need of a Spring Boot runner?

……
……
……
……
……

9.3 SPRING BOOT DEVTOOLS AND SPRING BOOT ACTUATOR

Spring Boot DevTools and Spring Boot Actuators are very important modules that enhance the development experience and increase productivity. These modules reduce the developers' effort and thus reducethe cost of the project.

9.3.1 Spring Boot DevTools

Spring Boot DevTools was released with Spring Boot 1.3. DevTools stands for developer tool. The aim of DevTools module is to enhance the application development experience by reducing the development time of the Spring Boot Application. During a web application development, a developer changes the code many times and then restarts the application to verify the changed code. DevToolsreduces the developer effort. It detects the code changes and restarts the application automatically. DevTools can be integrated into a Spring Boot application just by adding the following dependency into pom.xml.

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-devtools</artifactId>
	<scope>runtime</scope>
	<optional>true</optional>
</dependency>
```

Spring Boot DevTools provides the following important features –

- Automatic Restart
- Live Reload
- Property defaults

Automatic Restart: Auto-restart refers to the reloading of Java classes and configures it at the server-side. While developing an application, a developer changes the code frequently and to verify these changes, the steps build, deploy and restart of the server is required. Spring Boot DevTools automates the build and deploys process, which increases productivity. Files change in the class-path triggers Spring Boot DevTools to restart the application. This auto-restart process reduces the time significantly to verify the changes. Spring Boot uses two types of ClassLoaders:

- **Base ClassLoader:** This ClassLoader loads the classes which do not change. E.g. Third-party jars
- **Restart ClassLoader:** This ClassLoader loads the classes which we are actively developing

Live Reload: Live Reload or auto-refresh is also a very important feature provided by Spring Boot DevTools. Spring Boot DevTools module also comes with an embedded LiveReload server that can be used to trigger a browser refresh whenever a resource is changed. For example, when a developer makes the changes into templates or other resources, he/she has to refresh the browser to verify the changes. With the Live Reload/Auto Refresh feature, developers do not need to press F5 to refresh the browser. Thus, it enhances the development experience and increases productivity.

Enabling Live Reload in Spring Boot Application is pretty easy. Following steps are required to enable it.

1) Add the dependency spring-boot-devtools to your project's build file (pom.xml).

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
```

Once the application is started, one can verify the LiveReload server into log.

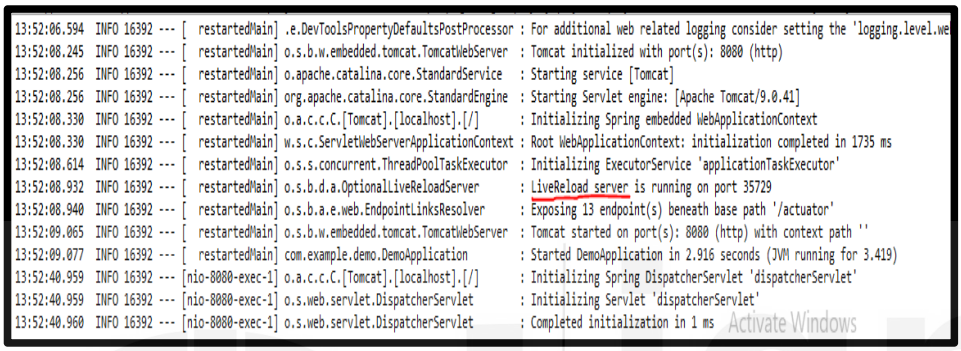
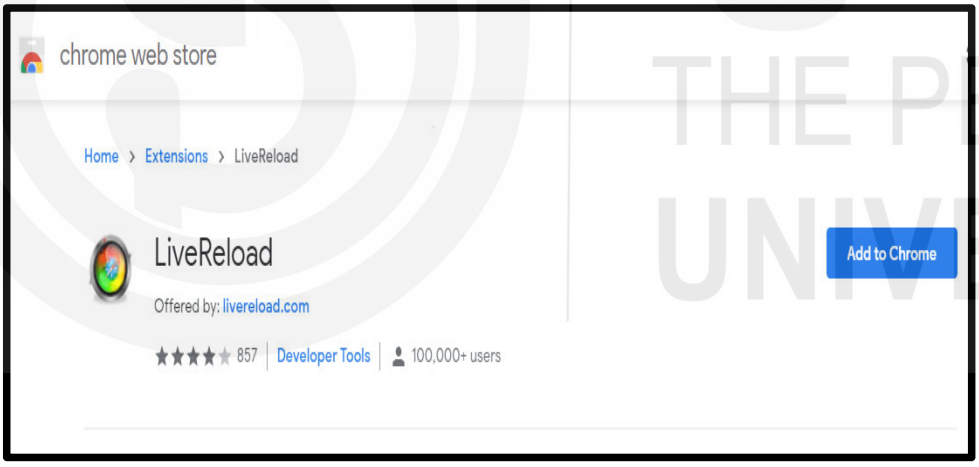


Figure 9.10: Live Reload

2) Install LiveReload extension for the browser. Go to http://livereload.com/extensions/ and click on the link that relates to your



browser.

Figure 9.11: Live Reload Extension

You can enable and disable the live reload by clicking on LiveReload as shown in screenshot.

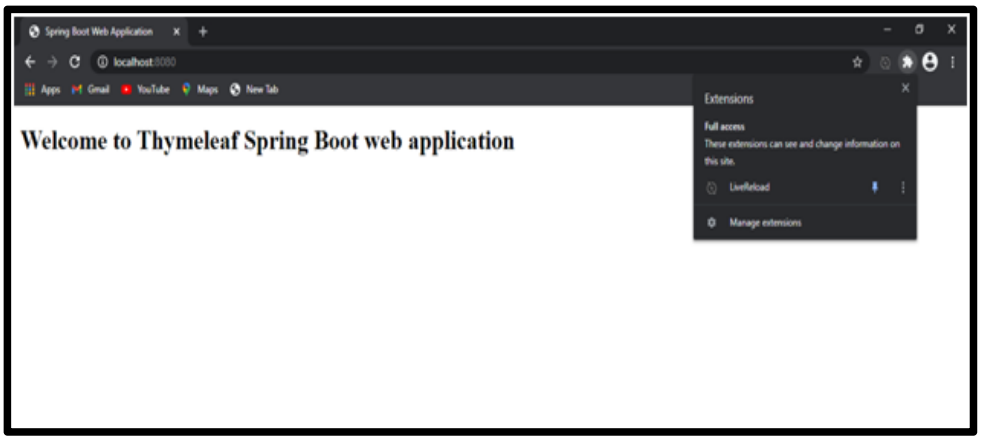


Figure 9.12: Enable/Disable Live Relaod

Update of home.html triggers the browser to auto-refresh. There is no need to press F5. Output is shown below:

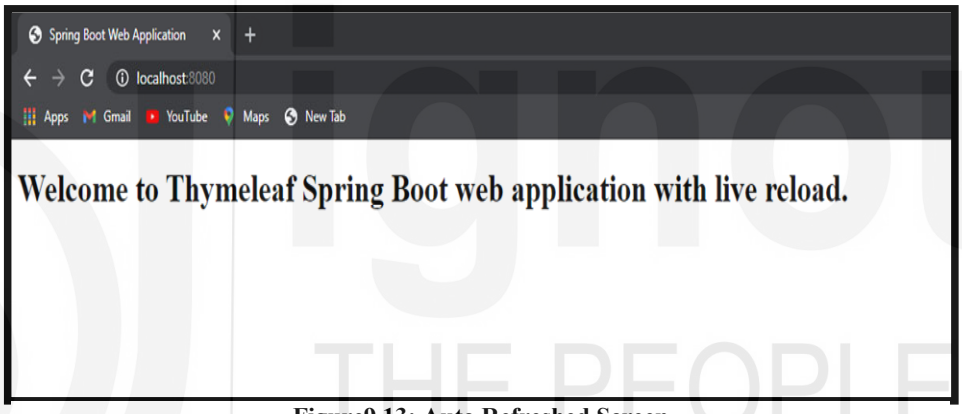


Figure9.13: Auto Refreshed Screen

LiveReload works on the following path:

- /static
- /public
- /resources
- /templates
- /META-INF/maven
- /META-INF/resources

The following properties can be used to disable and enable the LiveReload feature.

```
# Disable LiveReload for following path

spring.devtools.restart.exclude=public/**, static/**, templates/**

# Enable LiveReload for additional path

spring.devtools.restart.additional-paths=/path-to-folder
```

Property Defaults: Spring Boot does a lot of auto configuration. It also includes caching for performance improvement. Template technology Thymeleaf contains the property **spring.thymeleaf.cache.** DevTools disables the caching and allows us to update pages without restarting the application. During the development, caching for Thymeleaf, Freemarker, Groovy Templates are automatically disabled by DevTools.

9.3.2 Spring Boot Actuator

Spring Boot Actuator is a sub-project of the Spring Boot framework. The Actuator provides production-ready features such as application monitoring, Network traffic, State of database and many more. Without any implementation, Actuator provides production-grade tools. In Spring Boot application, Actuator is primarily used to expose operational information about the running application such as info, health, dump, metrics, env etc. It provides **HTTP** and **JMX** endpoints to manage and monitor the Spring Boot application. There are three main features of Spring Boot Actuator-

- Endpoints
- Metrics
- Audit

Endpoints: The actuator endpoints enable us to monitor and interact with the application. In Spring Boot 2.x, most of the endpoints of the Actuator are disabled. By default, only two endpoints **/health and /info** are available. Other required endpoints can be enabled by adding ***management.endpoints.web.exposure.include*** property into application.properties. **By default, all Actuator endpoints are now placed under the *actuator* path.** Some of the important endpoints are listed below –

- **/health** provides the health status of the application
- **/info** provides general information about the application. It might be build information or the latest commit information.
- **/metrics** provides metrics of application. Returned metrics include generic metrics as well as a custom metric.
- **/env** provides the current environment properties.

Metrics: Spring Boot integrates the micrometer to provide dimensional metrics. The micrometer is the instrumentation library that empowers the delivery of application metrics from Spring. Metrics of the application can be accessed by **/metrics** endpoint.

Audit: Spring Boot provides a flexible audit framework that publishes events to an AuditEventRepository. It automatically publishes the authentication events if spring-security is in execution.

9.3.3 Spring Boot Actuator Example

This section explains the required steps to integrate the Spring Boot Actuator into the previously developed website into section 9.2.5. Add the spring-boot-starter-actuator dependency into pom.xml –

```
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>
```

Start the web application and access the **http://localhost:8080/actuator** endpoint. By default, only two end points are enabled. Result is as below:

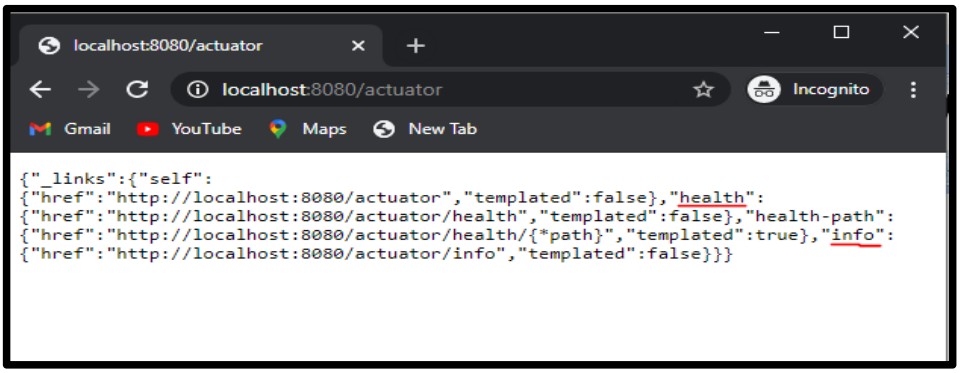


Figure 9.14: Default Enabled Actuator Endpoints

Add the following property in application.properties file and access the **http://localhost:8080/actuator** to get all available endpoints.

```
management.endpoints.web.exposure.include= *
```

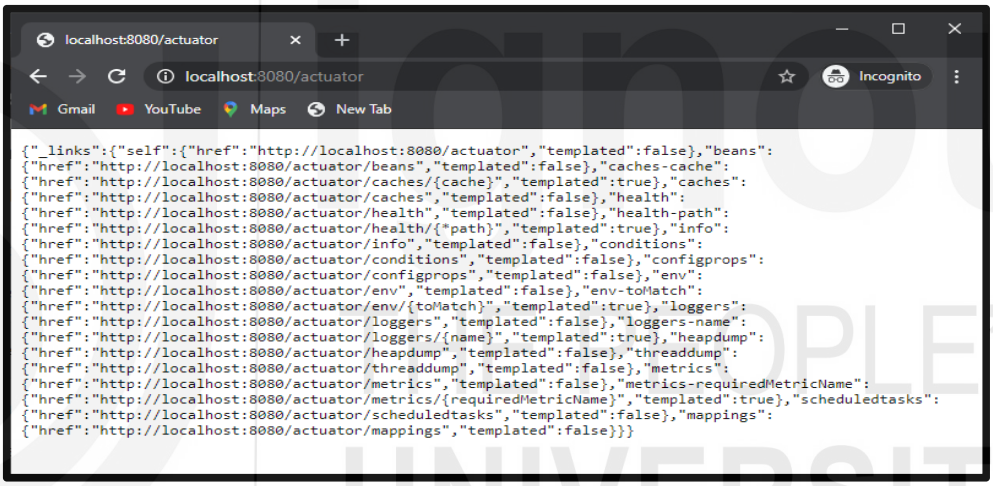


Figure 9.15: All Endpoints of Actuator

Output after adding above property is shown below:
Output for endpoint **http://localhost:8080/actuator/metrics** is shown below -

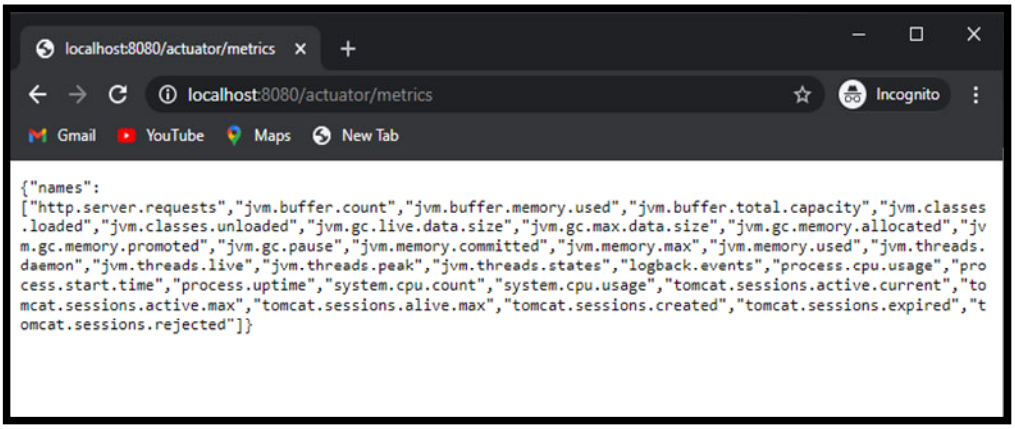


Figure 9.16: Autuator Metrics Result

The above output shows all available metrics. Metric **http.server.requests**show the following output :

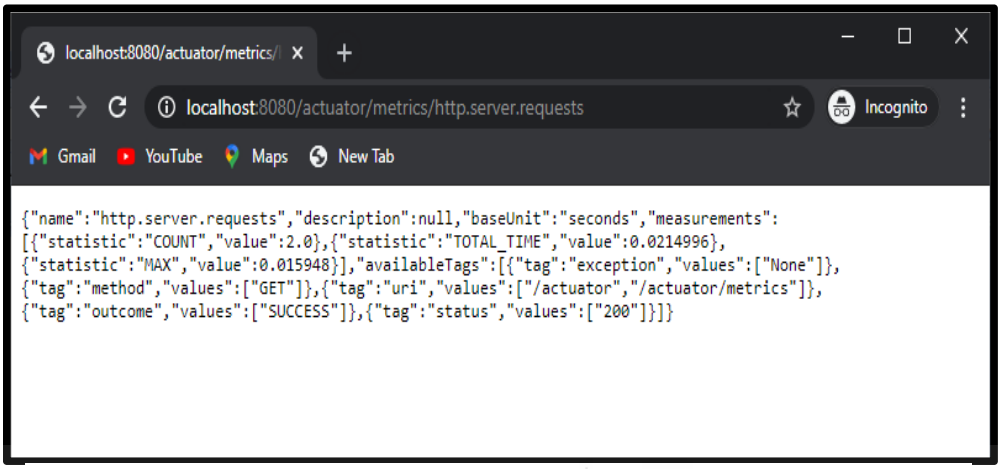


Figure 9.17: Metric http.server.requests

Output for **/health** endpoint is shown below-

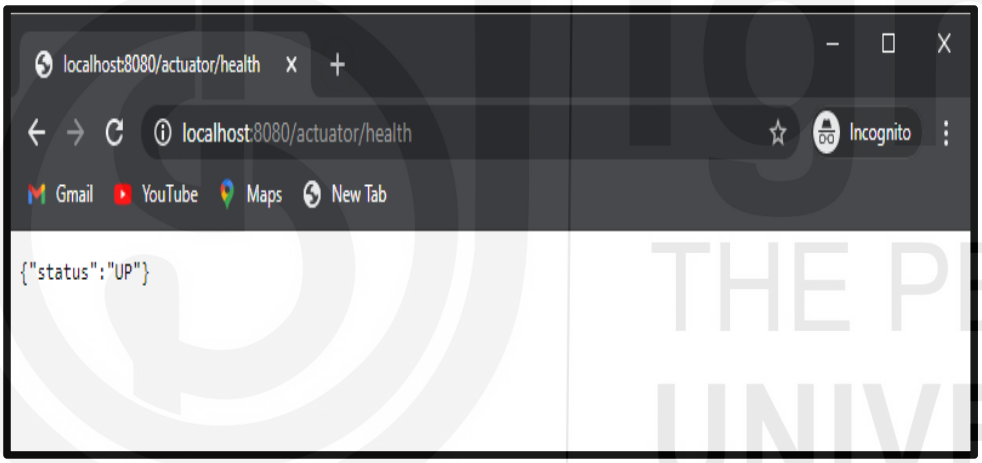


Figure9.18: Health Status of Server

☛Check Your Progress 2

1) What is the need of Spring Boot DevTools?

……

……

……

……

……

2) Explain Spring Boot Actuator and its advantages.

……

……

……

……

……..

3) Write a simple "Hello World" Spring Boot rest application.

……

……

……

……

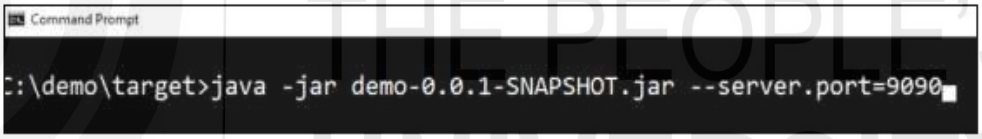
……..

9.4 SPRING BOOT- APPLICATION PROPERTIES

Application Properties enable us to work in different environments such as Prod, Dev, Test. This section explains how to set up and use properties in Spring Boot via Java configuration and @PropertySource. Spring Boot application properties can be set into properties files, YAML files, command-line arguments and environment variables.

9.4.1 Command Line Properties

Spring Boot Environment properties can be passed via command-line properties. Command-line properties take precedence over the other property sources. The following screenshot shows the command line property ***–server.port=9090*** to change the port number



Note: Double hyphen (--) can be used as a delimiter to pass multiple command-line properties.

9.4.2 Properties File

By default, Spring Boot can access configurations available in an **application.properties** file kept under the classpath. The application.properties file should be kept in **src/main/resources** directory. Thesample application.properties file is shown below-

```
server.port = 9090

spring.application.name = demoservice
```

Property server.port changes the port number on which the web application runs.

9.4.3 YAML File

Spring Boot also supports YAML based properties configuration. Properties file application.yml can be used instead of application.properties. **YAML is a convenient format for specifying hierarchical configuration data**. The application.yml file also should be kept in **src/main/resources** directory. Sample application.yml file is shown below-

```
server:
port: 9090
spring:
application:
name: demoservice
```

9.4.4 Externalized Properties

Spring Boot supports keeping properties file at any location. We can externalize the properties file so that if any property is changed, a new build is not required. Just application restart will take effect on the changed property. While executing the application jar file, the following command-line argument is used to specify the location of the property file.

```
-Dspring.config.location = C:\application.properties
```

9.4.5 @Value annotation

Properties, defined for environment or application, can be fetched in java code using **@Value** annotation. Syntax to use @Value annotation is shown below –

```
@Value("${property_key}")
```

If a property **spring.application.name** is defined into application.properties or application.yml, it can be accessed into java code using as-

```
@Value("${spring.application.name}")

String appName;
```

While running, if the property is not found, an exception is thrown. The default value can be set while fetching the value using @Value annotation as follows-

```
@Value("${property_key:default_value}")
```

9.4.6 Active Profile

An application is executed in multiple environments such as test, development, production etc. To modify application.properties file based on the environment is not

an ideal approach. There must be multiple properties files corresponding to each environment. At run time, Spring Boot must be able to select the desired environment properties file.

Spring Boot supports different properties based on the Spring active profile. Consider that there are separate properties file for each environment as shown below:

application.properties

```
server.port = 9090

spring.application.name = demoservice
```

application-dev.properties

```
server.port = 9090

spring.application.name = demoservice
```

application-prod.properties

```
server.port = 9090

spring.application.name = demoservice
```

By default, Spring Boot uses application.properties file. But Spring Boot allows to set the active profile and corresponding properties file that is used by Spring Boot. The following command shows how to set an active profile while starting an application from the command- line.

While running an application from eclipse IDE, the active profile can be set as shown in the screenshot.

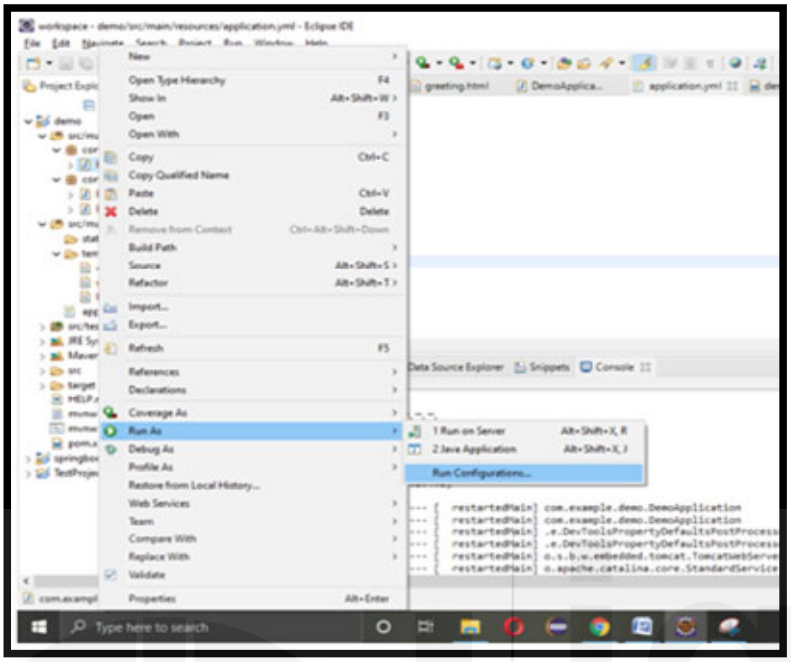


Figure 9.19: Set Active Profile in Eclipse Project 1

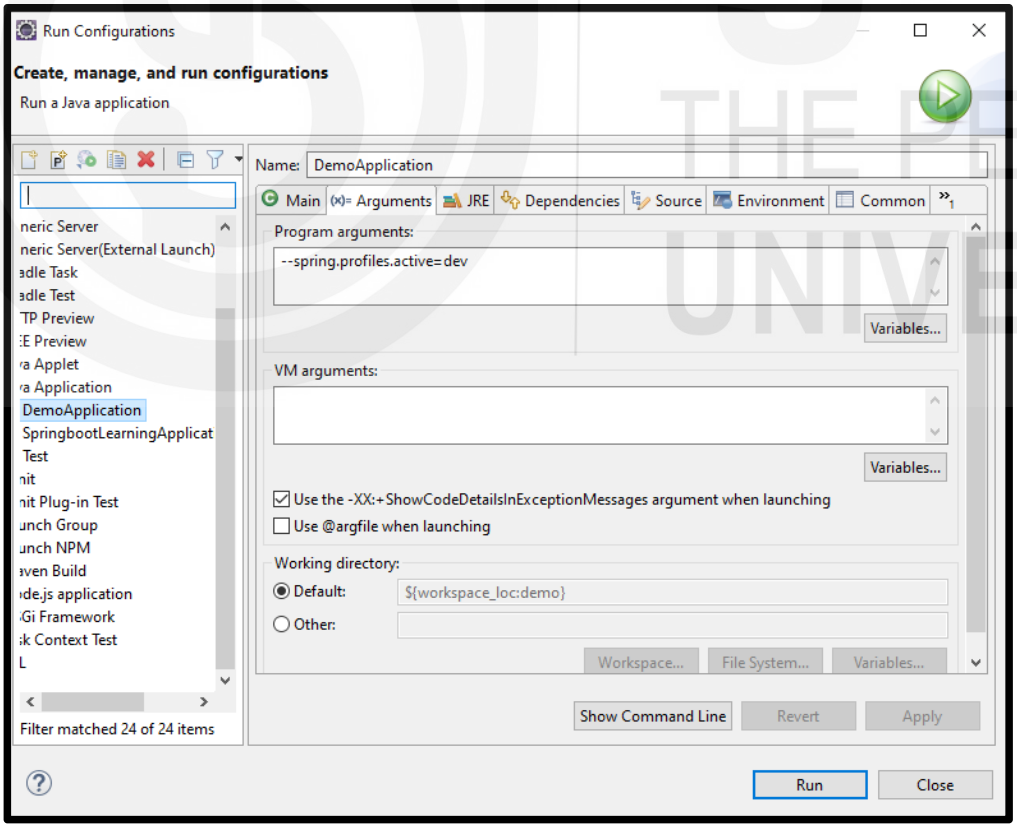


Figure9.20: Set Active Profile in Eclipse Project 2

Active profile can be checked in the log while starting the application.

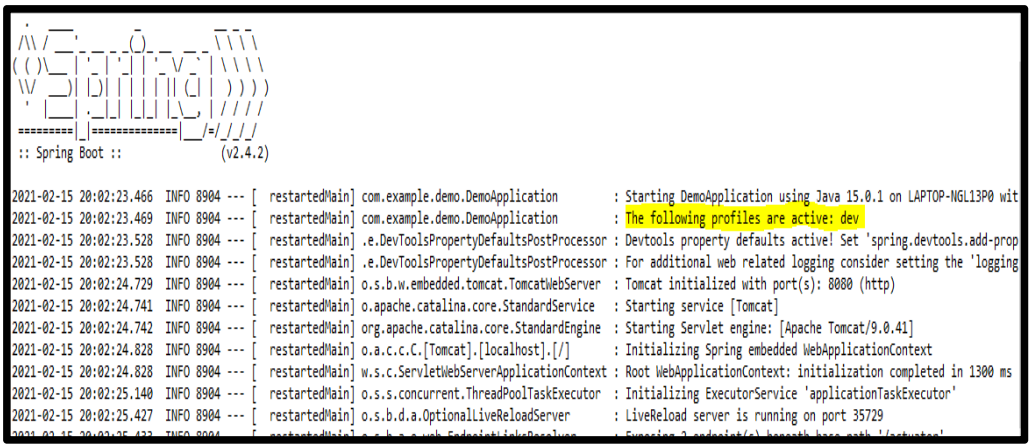


Figure 9.21: SpringBoot Application Execution Log

9.4.7 Spring Active Profile in application.yml

YAML file allows active profile parameters to be kept into a single application.yml file. Unlike multiple application.properties files, there is a single application.yml. Delimiter (---) is used to separate each profile in an application.yml file. Sample application.yml file is shown with active profile dev and prod.

```
server:
port: 9090
spring:
application:
name: demoservice

---
server:
port: 8080
spring:
config:
activate:
on-profile: dev
application:
name: demoservice
---
server:
port: 8080
spring:
config:
activate:
on-profile: prod
application:
name: demoservice
```

9.5 RUNNING SPRING BOOT APPS FROM COMMAND LINE

This section describes a couple of ways to run a Spring Boot app from a command line in a terminal window. Later it explains how to package the app as war and which can be deployed on any application server such as Apache Tomcat, WebLogic, JBoss, etc.

The **Spring Boot Maven** plugin is the recommended tool to build, test and package the Spring Boot Application code. The plugin is configured by adding it into pom.xml.

```
<build>
<plugins>
    ...
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
    ...
</plugins>
</build>
```

The plugin comes with lots of convenient features such as –

- It can package all our dependencies (including an embedded application server if needed) in a single, runnable fat jar/war
- It resolves the correct dependency versions

9.5.1 Running the Code with Maven in Exploded Form

The Spring Boot Maven plugin has the ability to automatically deploy the web application in an embedded application server. If **spring-boot-starter-web** dependency has been included, plugin knows that Tomcat is required to run the code. On execution of **mvnspring-boot:run** command in the project root folder, the plugin reads the pom configuration. If a web application container is required, the plugin triggers the download of Apache Tomcat and initializes the startup of Tomcat. The command to run the Spring Boot application using maven plugin is as following –

```
mvnspring-boot:run
```

Execution of the above command starts the Spring Boot application and the produced log is shown.

9.5.2 Running the Code as a Stand-Alone Packaged Application

Once the development phase is over, the application is moved to production, and we need to package the application. Just include the **Spring Boot Maven** plugin and execute the following command in order to package the application.

```
mvn clean package spring-boot:repackage
```

The execution of the above command packages the application and produces the jar

Figure 9.22: SpringBoot Application Execution From Command Line

file into the target folder. The generated jar file can be executed using the following command.

```
java –jar <File Name>
```

As you can notice, that –cp option and main class has been skipped into java –jar command because the Spring Boot Maven plugin takes care of all these configurations into the manifest file.

9.5.3 Application Packaging as WAR and Deployment on Tomcat

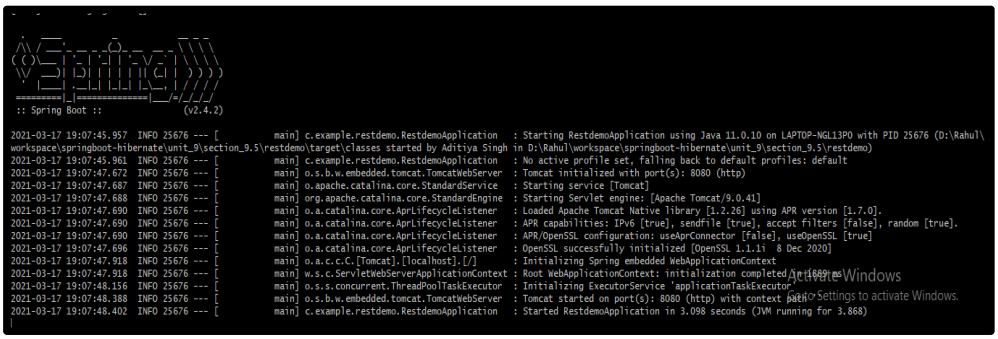
By default, Spring Boot builds a standalone Java application that can run as a desktop application. The standalone Java application is not suitable for the environment where installation of a new service or manual execution of an application is not allowed, such as Production environment.

The Servlet containers require the applications to meet some contracts to be deployed. For Tomcat the contract is the Servlet API 3.0. This section considers the example used in section 9.2.5 and explains how the application can be packaged as WAR and deployed into Tomcat.

First, change the packaging type as war into pom.xml with the following content.

```
<packaging>war</packaging>
```

Initialize the Servlet context required by Tomcat by extending



the *SpringBootServletInitializer* class*:*

```
@SpringBootApplication

publicclassDemoApplicationextendsSpringBootServletInitializer

{
	publicstaticvoid main(String[] args)
	{
		SpringApplication.run(DemoApplication.class, args);
	}
}
```

By default, generated war file name includes version number also. The Name of the generated war can be modified with following –

```
<build>

	<finalName>${artifactId}</finalName>

	...

</build>
```

Execute the following command in order to build the Tomcat deployable war if artifact id is **demo,** war file generated at target/demo.war. Follow the below steps in order to deploy the generated war file on Tomcat.

- Download Apache Tomcat and unpack it into the tomcat folder.
- Copy the generated war file from target/demo.war to tomcat/webapps/demo.war
- Go to bin dir of Tomcat and start the tomcat using ***catalina.bat start*** (on windows) or ***catalina.sh start*** (on Unix)
- Access the application using http://localhost:8080/demo

☛Check Your Progress 3

1) Mention the possible sources of external configuration.

……

……

……

……

……

2) Can we change the port of the embedded Tomcat server in Spring boot?

……

……

……

……

……

3) Explain the concept of profile in Spring Boot and say how it is useful,

……

……

……

……

……

4) Explain the Spring Boot application execution with Maven.

……

……

……

……

……

5) What are the steps to deploy Spring Boot web applications as JAR and WAR files?

……

……

……

……

……

9.6 SUMMARY

This unit has described Spring Boot facilitates the developers to create a Spring based web application with minimum lines of code since it provides auto configuration **out-of-the-box**. This unit has explained the following:

- A class with the main method and annotated with **@SpringBootApplication** is the entry point of the Spring Boot application.
- Spring Boot starters are the dependency descriptor that addresses the problem of compatible versions of dependency management. It is also a one-stop-shop for all the Spring and related technology that you need.
- Spring Boot provides two runner interfaces named as ***ApplicationRunner*** and ***CommandLineRunner***. These functional interfaces enable you to execute a piece of code just after the Spring Boot application is started.
- Spring Boot ***DevTools*** important features such as Automatic restart, Live Reload, Property Defaults such as cache disable and allows us to update pages without restarting the application
- ***Actuator*** provides production-ready features such as application monitoring, Network traffic, State of database and many more. Without any implementation, Actuator provides production-grade tools.
- By default, only two endpoints ***/health*** and ***/info*** are available. Other required endpoints can be enabled by adding ***management.endpoints.web.exposure.include*** property into application.properties.
- Various ways to define application properties such as command-line argument, application.properties file and application.yml file.

- Spring Boot supports keeping properties files at any location. We can externalize the properties file so that if any property is changed, a new build is not required.
- Spring Boot supports the concept of an active profile. There can be multiple properties files corresponding to each environment. At run time, Spring Boot selects the desired environment properties file based **on active profile**.
- Spring Boot application can be executed from command line with Spring Boot Maven plugin using **mvnspring-boot:run** and can be executed as a standalone application by **java -jar <fileName>**

9.7 SOLUTIONS/ANSWER TO CHECK YOUR PROGRESS

☛Check Your Progress 1

1) ***Spring Boot*** is a utility project which enables an organization to develop production-ready spring-based applications and services with less effort, reduced cost and minimal configuration. ***Spring Boot***facilitates the developers to create a Spring based web application with minimum lines of code since it provides auto configuration out-of-the-box.

   It is a module that enriches the Spring framework with Rapid Application Development (RAD) feature. It provides an easy way to create a stand-alone and production ready spring application with minimum configurations. Spring Boot is a combination of Spring framework with auto-configuration and embedded Servers.

   Spring Boot provides a vast number of features and benefits. A few of them are as follows:

   - Everything is auto-configured in Spring Boot.
   - Spring Boot starter eases the dependency management and application configuration
   - It simplifies the application deployment by using an embedded server
   - Production-ready features to monitor and manage applications such as health checks, metrics gathering etc.
   - Reduces the application development time and run the application independently
   - Very easy to understand and develop Spring application

2) Spring Boot auto configures all required configurations. It performs auto-configuration by scanning the classes in classpath annotated with @Component or @Configuration. @SpringBootApplication annotation comprises the following three annotations with their default values-
   - @EnableAutoConfiguration
   - @ComponentScan
   - @Configuration

   For details, check section 9.2.1

3) Spring Boot starters are the dependency descriptor that addresses the problem of compatible versions of dependency management. Starter POMs are a set of convenient dependency descriptors that you can include in your application.

You get a one-stop-shop for all the Spring and related technology that you need from Spring Boot starters. Spring Boot provides a number of starters that make development easier and rapid. Spring Boot provides many numbers starter, and a few of them are listed below-

spring-boot-starter-web	It is used for building web applications, including RESTful applications using Spring MVC. It uses Tomcat as the default embedded container.
spring-boot-starter-jdbc	It is used for JDBC with the Tomcat JDBC connection pool.
spring-boot-starter-validation	It is used for Java Bean Validation with Hibernate Validator.
spring-boot-starter-security	It is used for Spring Security.
spring-boot-starter-data-jpa	It is used for Spring Data JPA with Hibernate.

4) As the name implies, it is used to automatically configure the required configuration for the application. It means **Spring Boot** looks for auto-configuration beans on its classpath and automatically applies them. For example, if H2 dependency is added and you have not manually configured any database connection beans, then Spring will auto-configure H2 as an in-memory database.
To disable the auto-configuration property, you have to use exclude attribute of @EnableAutoConfiguration. For example, Data Source autoconfiguration can be disabled as:
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
The property ***spring.autoconfigure.exclude*** property can be used in application.properties or application .yaml file to mention the exclude list of auto-configuration classes.

5) Spring Boot provides two runner interfaces named as ApplicationRunner and CommandLineRunner. These interfaces enable you to execute a piece of code just after the Spring Boot application is started.
Both interfaces are Functional Interface. If any piece of code needs to be executed when Spring Boot Application starts, we can implement either of these functional interfaces and override the single method orun.
Check the details of ApplicationRunner and CommandLineRunner in section 9.2.4

☛Check Your Progress 2

1) Spring Boot DevTools was released with Spring Boot 1.3. DevToolsstands for developer tool. Aim of DevTools module is to enhance the application development experience by improving the development time of the Spring Boot Application. During a web application development, a developer changes the code many times and then restarts the application to verify the code changes. DevTools reduces the developer effort. It detects the code changes and restarts the application. DevTools can be integrated into a Spring Boot application just by adding the following dependency into pom.xml.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
</dependency>
```

Spring Boot DevTools provides the following important features –

- Automatic Restart
- Live Reload
- Property defaults

2) Spring Boot Actuator is a sub-projectof the Spring Boot framework. The Actuator provides production-ready features such as application monitoring, Network traffic, State of database and many more. Without any implementation, Actuator provides production-grade tools. In Spring Boot application, Actuator is primarily used to expose operational information about the running application such as info, health, dump, metrics, env etc. It provides **HTTP** and **JMX** endpoints to manage and monitor the Spring Boot application. There are three main features of the Spring Boot Actuator-
   - Endpoints
   - Metrics
   - Audit

   By default, only two endpoints **/health and /info** are available. Other required endpoints can be enabled by adding ***management.endpoints.web.exposure.include*** property into application.properties. **By default, all Actuator endpoints are now placed under the *actuator* path.** Some of the important endpoints are listed below –

   - **/health** provides the health status of application
   - **/info** provides general information about the application. It might be build information or the latest commit information.
   - **/metrics** provides metrics of application. Returned metrics include generic metrics as well as custom metric.
   - **/env** provides the current environment properties.

3) To create a simple Hello World rest application using Spring Boot, we need to perform the below steps:

   - Create a Spring Boot Project using Spring Initializr. Visit at https://start.spring.io/ and generate the Spring Boot project with **Spring Web** dependency.
   - Import the project in Eclipse IDE as a maven **project**.
   - Create a controller package and add a **HomeController** class annotated with **@RestController**.

```
@RestController

publicclassHomeController {

	@GetMapping(value="/")

	public String index() {

		return"Hello World!!";

	}

}
```

- From the root of the project, execute command **mvnspring-boot:run** in order to run the application.
- Access the application with **http://localhost:8080**

☛Check Your Progress 3

1) Application Properties enable us to work in different environments such as Prod, Dev, Test. Spring Boot application properties can be set into properties files, YAML files, command-line arguments and environment variables.
Spring Boot supports keeping properties file at any location. We can externalize the properties file so that if any property is changed, a new build is not required. Just application restart will take effect on the changed property. While executing the application jar file, the following command line argument is used to specify the location of the property file.

```
-Dspring.config.location = C:\application.properties
```

2) Yes, we can change the port of the embedded tomcat server by using the application properties file. In application.properties file, you must add a property of "**server.port**" and assign it to any port you wish to. For example, if you want to assign it to 8081, then you have to mention **server.port=8081**. Once you mention the port number, the application properties file will be automatically loaded by Spring Boot, and the required configurations will be applied to the application.

3) An application is executed in multiple environments such as test, development, production. To modify application.properties file based on the environment is not an ideal approach. There must be multiple properties files corresponding to each environment. At run time Spring Boot must be able to select the desired environment properties file.
Spring Boot supports different properties based on the Spring active profile. By default, Spring Boot uses application.properties file. But Spring Boot allows to set the active profile and corresponding properties file that is used by Spring Boot. Following command shows to set **dev** as active profile while starting the application from command line, and the execution will read the application-dev.properties file since active profile is set as **dev**.
java –jar demo.jar --spring.profiles.active=dev

4) Check section 9.5.1
5) Check section 9.5.3

9.8 REFERENCES/FURTHER READING

- Craig Walls, “Spring Boot in action” Manning Publications, 2016. https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf)
- Christian Bauer, Gavin King, and Gary Gregory, “**J**ava Persistence with Hibernate”,Manning Publications, 2015.
- Ethan Marcotte, “Responsive Web Design”, Jeffrey Zeldman Publication, 2011(http://nadin.miem.edu.ru/images_2015/responsive-web-design-2nd-edition.pdf)
- Tomcy John, “Hands-On Spring Security 5 for Reactive Applications”,Packt Publishing,2018
- https://spring.io/guides/gs/spring-boot/
- https://dzone.com/articles/introducing-spring-boot
- https://www.baeldung.com/spring-boot
- https://docs.spring.io/spring-boot/docs/1.5.4.RELEASE/reference/pdf/spring-boot-reference.pdf
- https://www.baeldung.com/spring-boot-starters
- https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf
- https://www.baeldung.com/spring-boot-devtools
- https://docs.spring.io/spring-boot/docs/1.5.16.RELEASE/reference/html/using-boot-devtools.html
- https://howtodoinjava.com/spring-boot/developer-tools-module-tutorial/
- https://www.baeldung.com/spring-boot-run-maven-vs-executable-jar