Spring Boot

Basic Spring 5.0

Lesson Objectives

- What is Spring Boot
- How Spring Boot works
- Developing web application using Spring Boot
- Spring Boot integration with Spring Data JPA

Spring Boot

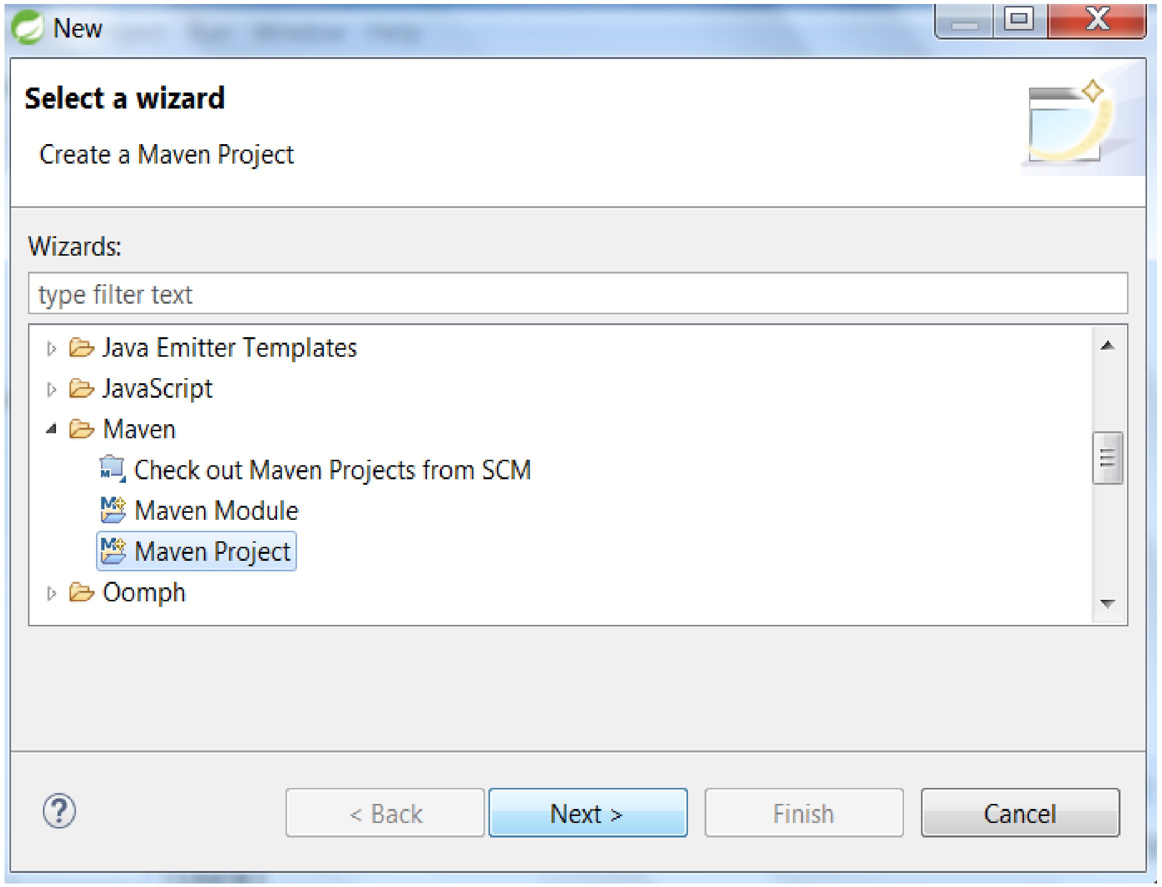
Prerequisites to start working with Spring Boot

- Knowledge of basic spring concepts
- jdk 1.8 or higher
- IDE i.e Spring STS (has maven built into it)

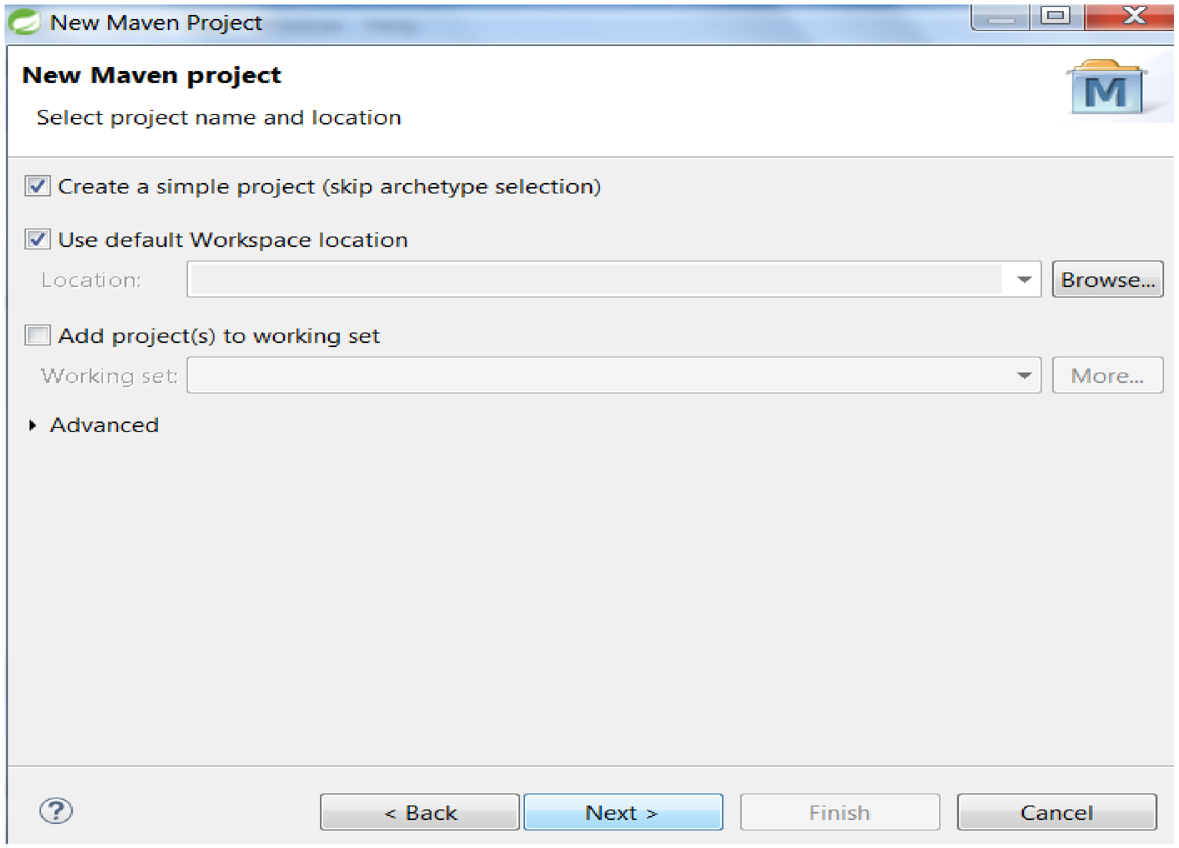
Ways to create Spring Boot project

1. Using the Spring Tool Suite IDE (STS)
2. Spring Initializer
3. Spring command line interface

Creating a spring boot application using STS IDE
Click on menu , File →New –Other - Maven -Maven Project- Click on Next



Creating a spring boot application using STS IDE
Select the checkbox, “Create a simple project” and Click On Next-



New Maven Project

New Maven project

Select project name and location

[✓] Create a simple project (skip archetype selection)

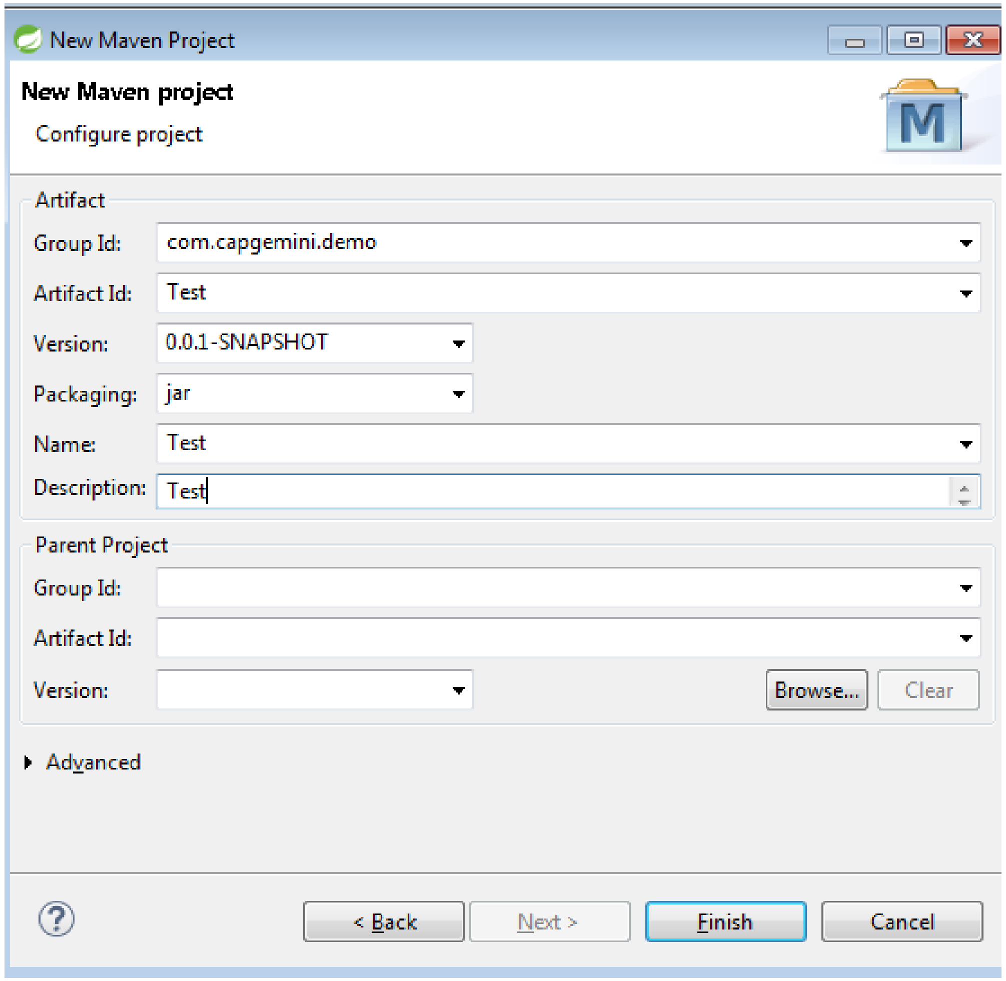
[✓] Use default Workspace location

Location:

[ ] Add project(s) to working set

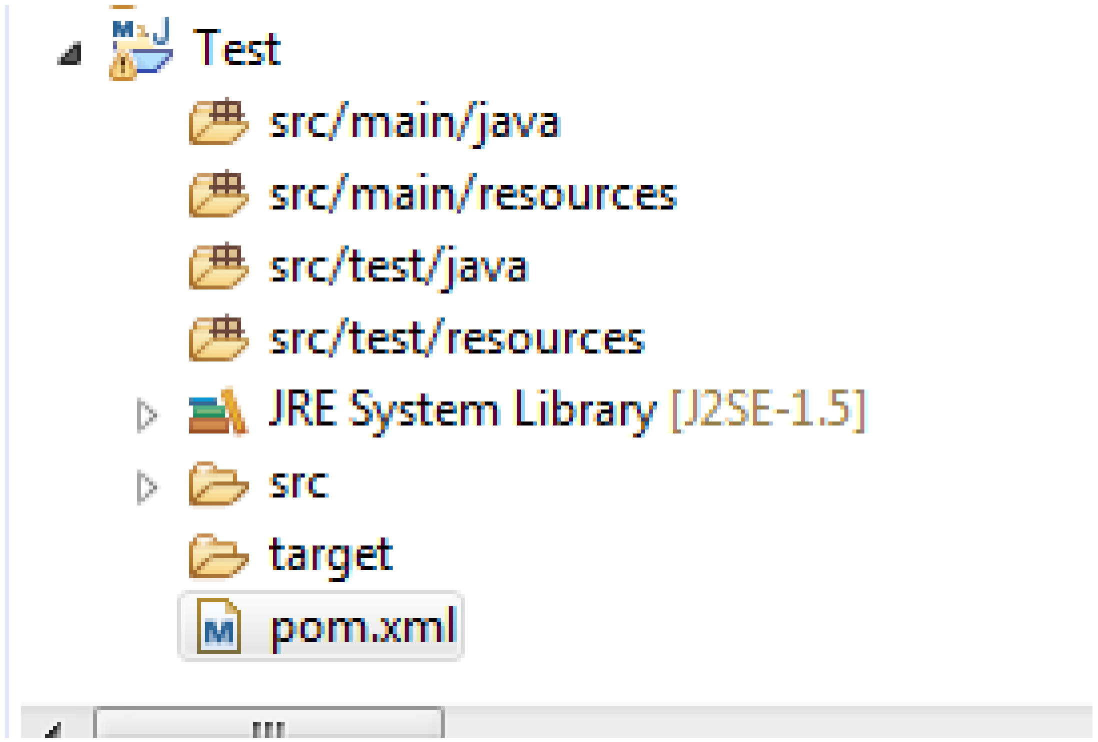
Working set:

Specify the group id, artifact Id, name and description
Click On Finish. Observe the folder structure of the newly created project

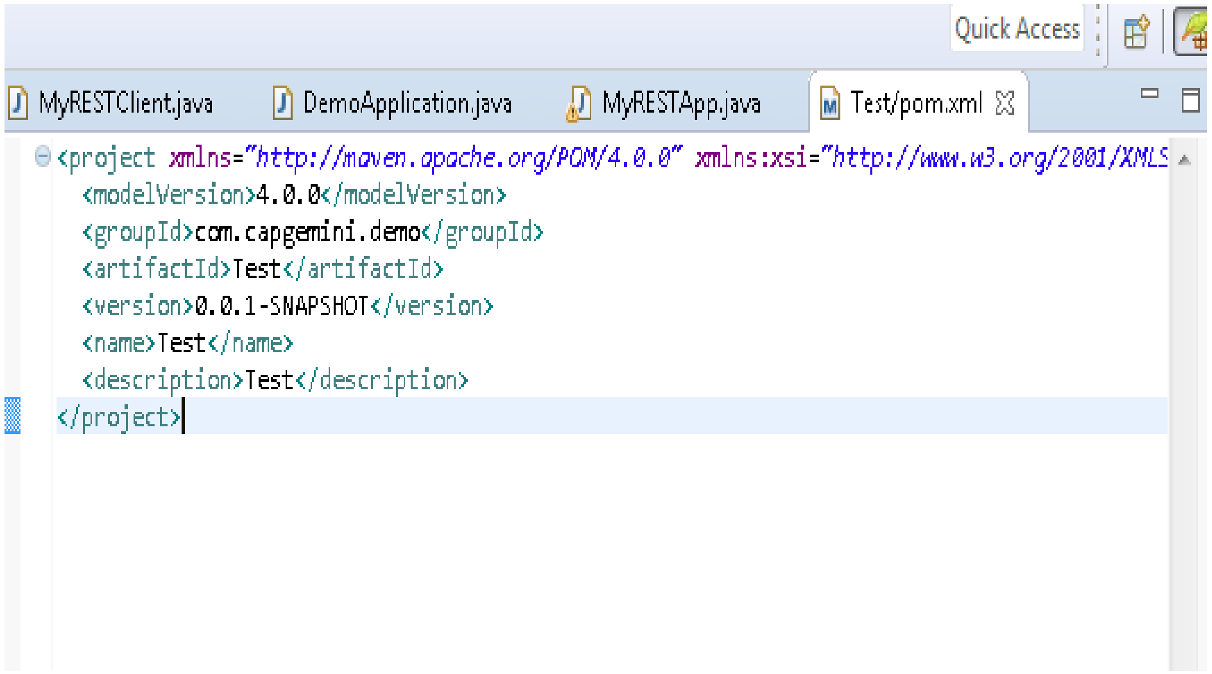


Group Id:	com.capgemini.demo
Artifact Id:	Test
Version:	0.0.1-SNAPSHOT
Packaging:	jar
Name:	Test
Description:	Test

Double click on the generated pom.xml file



The default pom.xml is shown below

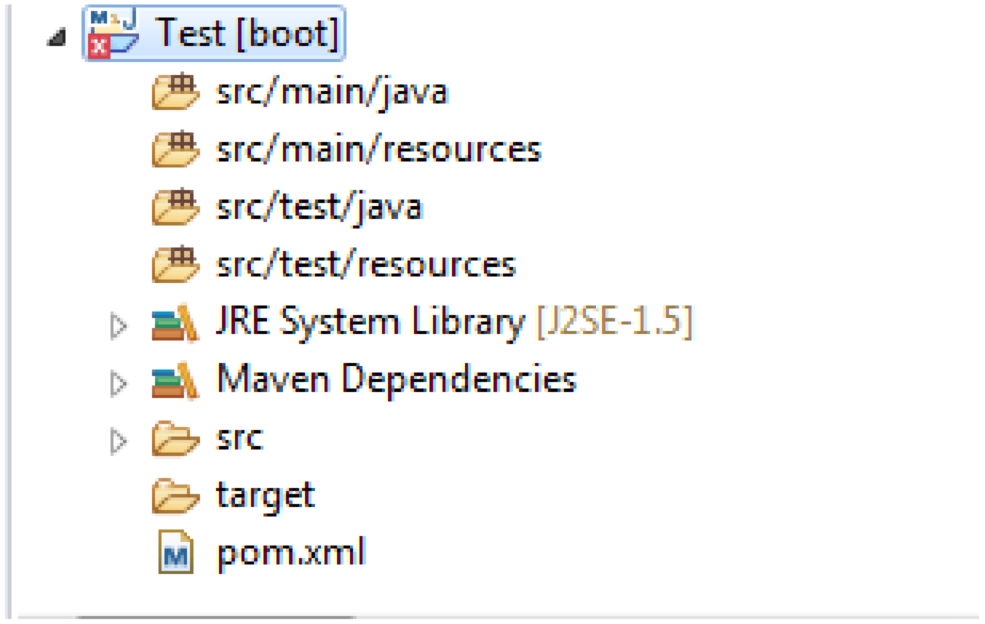


```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLS
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.capgemini.demo</groupId>
  <artifactId>Test</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Test</name>
  <description>Test</description>
</project>
```

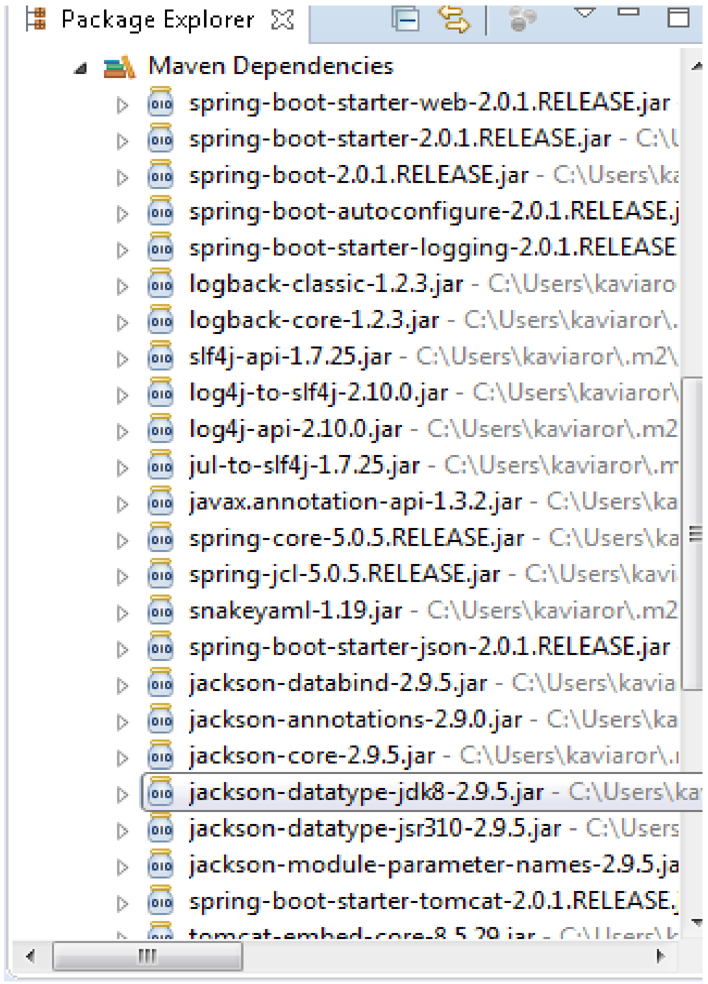
Add the following code in the pom.xml under the <description> tag

```
<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.0.1.RELEASE</version>
		<relativePath /> <!-- lookup parent from repository -->
	</parent>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
	</dependencies>
```

Observe that “Maven Dependencies” has been included into the project


Test [boot]
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml

Without Spring Boot, these jar files are among those that you would have had to copy physically into the project


Package Explorer

Maven Dependencies
- spring-boot-starter-web-2.0.1.RELEASE.jar
- spring-boot-starter-2.0.1.RELEASE.jar - C:\U
- spring-boot-2.0.1.RELEASE.jar - C:\Users\ka
- spring-boot-autoconfigure-2.0.1.RELEASE.j
- spring-boot-starter-logging-2.0.1.RELEASE
- logback-classic-1.2.3.jar - C:\Users\kaviaro
- logback-core-1.2.3.jar - C:\Users\kaviaror\.
- slf4j-api-1.7.25.jar - C:\Users\kaviaror\.m2\
- log4j-to-slf4j-2.10.0.jar - C:\Users\kaviaror\
- log4j-api-2.10.0.jar - C:\Users\kaviaror\.m2
- jul-to-slf4j-1.7.25.jar - C:\Users\kaviaror\.m
- javax.annotation-api-1.3.2.jar - C:\Users\ka
- spring-core-5.0.5.RELEASE.jar - C:\Users\ka
- spring-jcl-5.0.5.RELEASE.jar - C:\Users\kavi
- snakeyaml-1.19.jar - C:\Users\kaviaror\.m2
- spring-boot-starter-json-2.0.1.RELEASE.jar
- jackson-databind-2.9.5.jar - C:\Users\kavia
- jackson-annotations-2.9.0.jar - C:\Users\ka
- jackson-core-2.9.5.jar - C:\Users\kaviaror\.
- jackson-datatype-jdk8-2.9.5.jar - C:\Users\ka
- jackson-datatype-jsr310-2.9.5.jar - C:\Users
- jackson-module-parameter-names-2.9.5.ja
- spring-boot-starter-tomcat-2.0.1.RELEASE.
- tomcat-embed-core-8.5.29.jar - C:\Users\k

Create a new java class having the following code

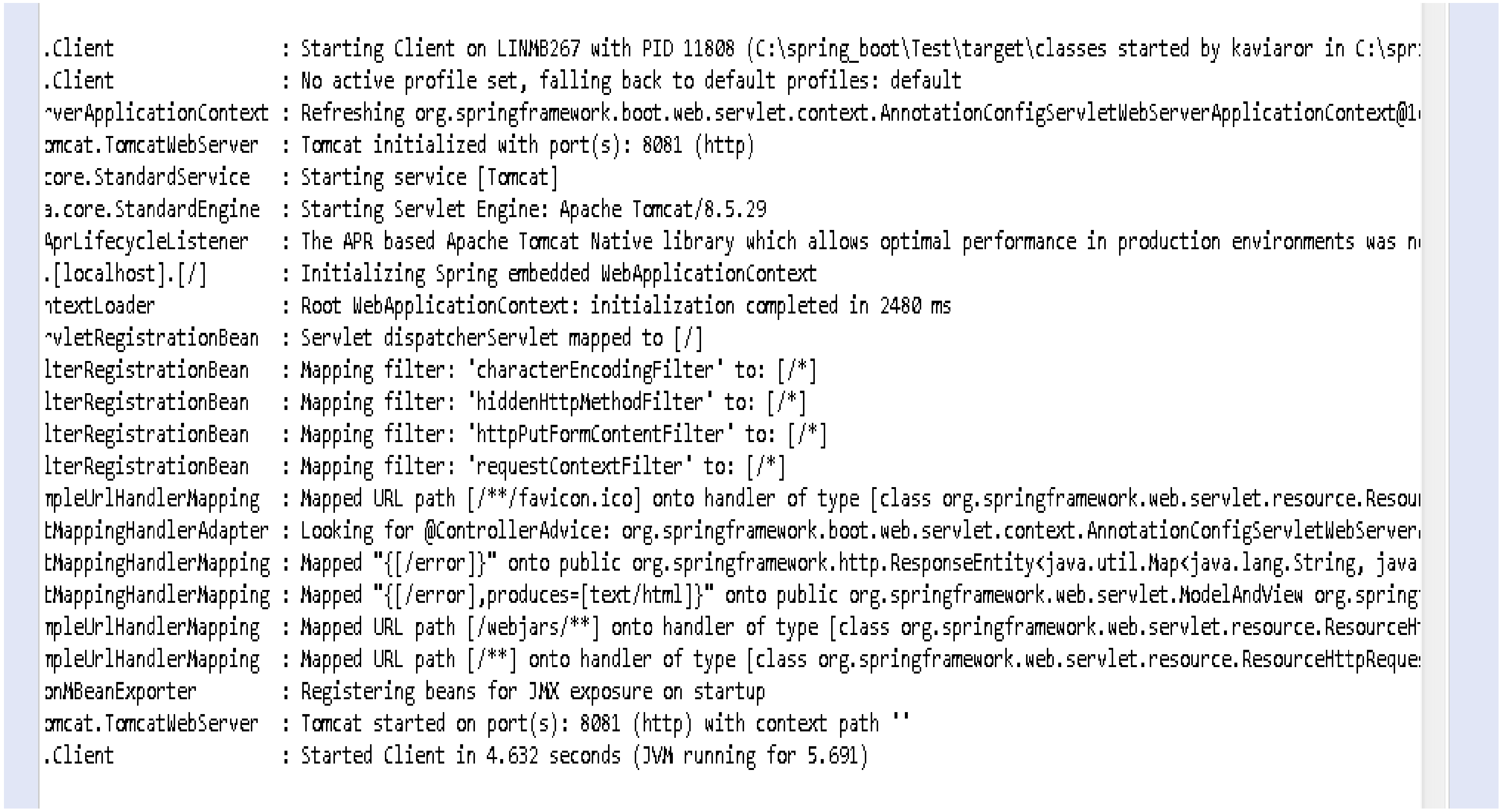
```
@SpringBootApplication
public class Client {

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

}
```

Run the above program as a regular java application

There is no need to deploy this application on any external server

Note: this class must be kept in the topmost package.

Run the application as a java application and observe the console as shown below



```
.Client                  : Starting Client on LINMB267 with PID 11808 (C:\spring_boot\Test\target\classes started by kaviaror in C:\spr
.Client                  : No active profile set, falling back to default profiles: default
rverApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@1
omcat.TomcatWebServer  : Tomcat initialized with port(s): 8081 (http)
core.StandardService   : Starting service [Tomcat]
a.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.29
AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal performance in production environments was n
.[localhost].[/]       : Initializing Spring embedded WebApplicationContext
ntextLoader            : Root WebApplicationContext: initialization completed in 2480 ms
rvletRegistrationBean  : Servlet dispatcherServlet mapped to [/]
lterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
lterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
lterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
lterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
mpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.Resou
tMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServer
tMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java
tMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.spring
mpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceH
mpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpReque
onMBeanExporter        : Registering beans for JMX exposure on startup
omcat.TomcatWebServer  : Tomcat started on port(s): 8081 (http) with context path ''
.Client                  : Started Client in 4.632 seconds (JVM running for 5.691)
```

Create a class which acts as a controller

```
@RestController
public class HelloController {
        @RequestMapping("/hello")
        public String sayHi() {
                return "Hi";
        }
}
```

As we have not mapped any URLs to methods in the controller class, this step becomes necessary

Creating a spring boot application using STS IDE

After adding the controller class, navigate to browser and type http://localhost:8081/hello

And observe the “Hi” message displayed on the browser page

We have a fully running Java spring web application developed using Spring boot

Rapid application development is what Spring boot is about.

How Spring Boot works

1. The applicaton is started from the Java main class

2. Spring boot initialises Spring context that comprises the Spring app and honours autoconfig initialisers, configuration and annotations which direct how to initialise and startup the spring context

3. Embedded server container is started and autoconfigured

This removes the need for web.xml

Spring has chosen “Tomcat” as the default container

How Spring Boot works

@SpringBootApplication

A convenience annotation that wraps commonly used annotations.

Used in place of the following 3 different annotations

1. @configuration : Instructs that a Spring configuration class is being used instead of XML to define the components

2. @EnableAutoconfiguraton : is a Spring boot specific annotation

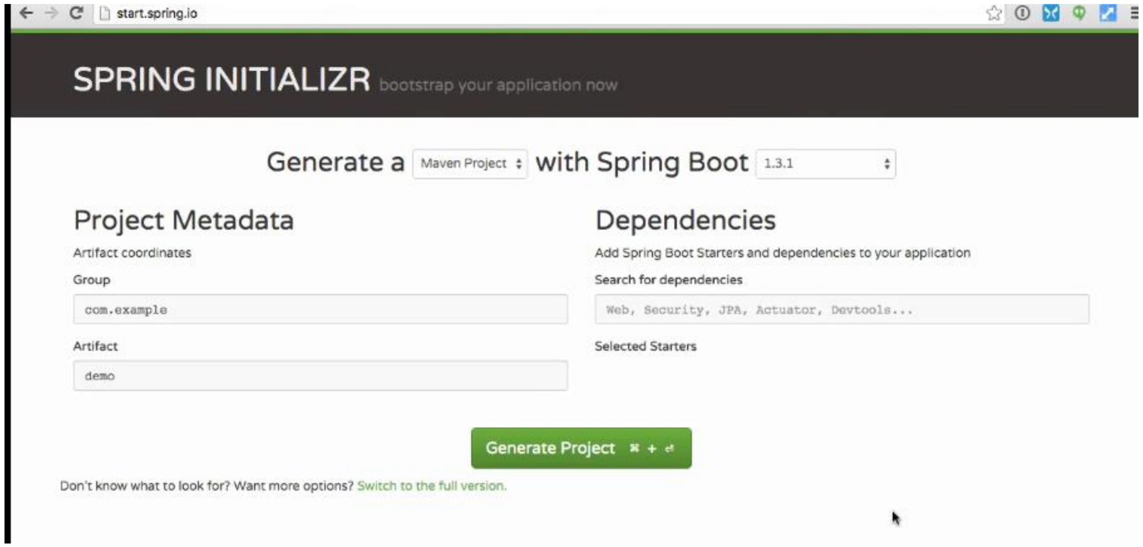
Instructs that the application should auto configure the other frameworks included as dependency with Spring.

3. @ComponentScan : Scans project for Spring components annotated with @Service, @Repository, @Component

Spring Initializer

Navigate to the following URL

start.spring.io



start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Spring Boot 1.3.1

Project Metadata

Artifact coordinates

Group: com.example

Artifact: demo

Dependencies

Add Spring Boot Starters and dependencies to your application

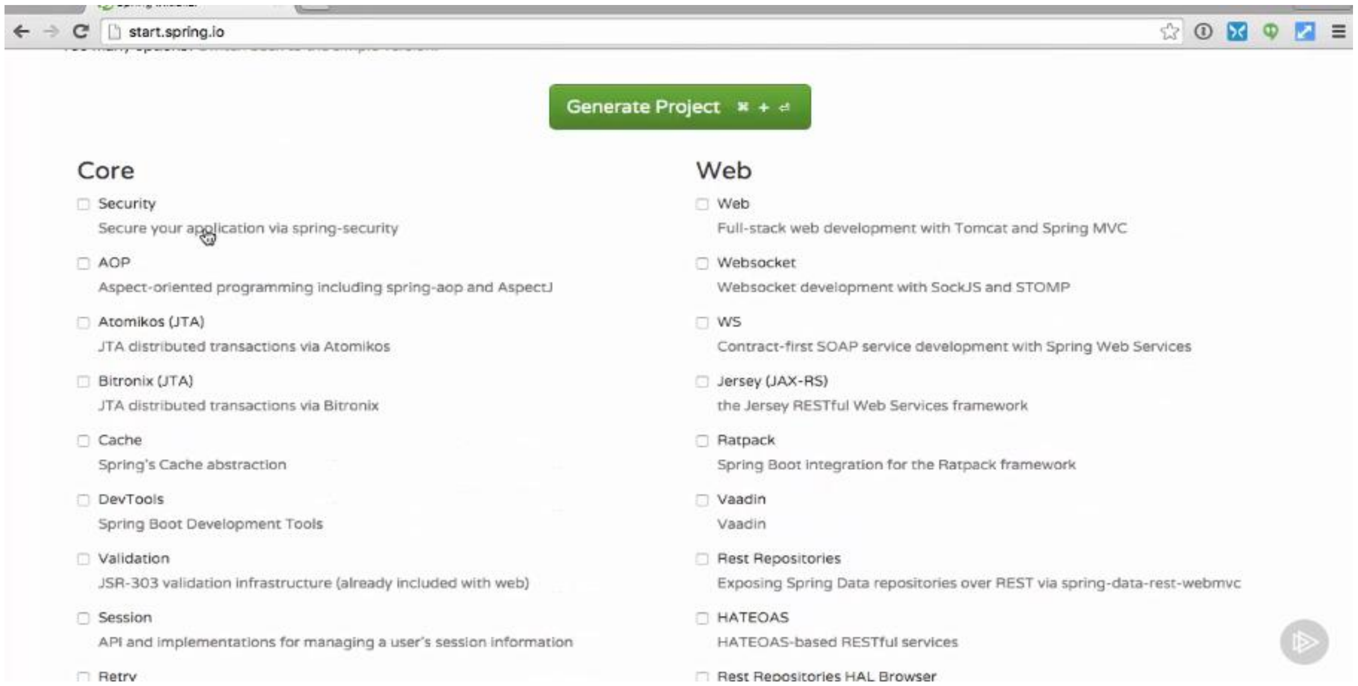
Search for dependencies: Web, Security, JPA, Actuator, Devtools...

Selected Starters

Generate Project

Don't know what to look for? Want more options? Switch to the full version.

Click on "switch to full version" link


start.spring.io

Generate Project ⌘ + ⏎

Core

- ☐ Security
  Secure your application via spring-security
- ☐ AOP
  Aspect-oriented programming including spring-aop and AspectJ
- ☐ Atomikos (JTA)
  JTA distributed transactions via Atomikos
- ☐ Bitronix (JTA)
  JTA distributed transactions via Bitronix
- ☐ Cache
  Spring's Cache abstraction
- ☐ DevTools
  Spring Boot Development Tools
- ☐ Validation
  JSR-303 validation infrastructure (already included with web)
- ☐ Session
  API and implementations for managing a user's session information
- ☐ Retry

Web

- ☐ Web
  Full-stack web development with Tomcat and Spring MVC
- ☐ Websocket
  Websocket development with SockJS and STOMP
- ☐ WS
  Contract-first SOAP service development with Spring Web Services
- ☐ Jersey (JAX-RS)
  the Jersey RESTful Web Services framework
- ☐ Ratpack
  Spring Boot integration for the Ratpack framework
- ☐ Vaadin
  Vaadin
- ☐ Rest Repositories
  Exposing Spring Data repositories over REST via spring-data-rest-webmvc
- ☐ HATEOAS
  HATEOAS-based RESTful services
- ☐ Rest Repositories HAL Browser

Select appropriate checkboxes which represent the different dependencies
 you want to include in the project and then click on "Generate Project"

Observe the zip file created for you.
This contains the folder structure of the project

Spring boot command line interface

The Spring Boot CLI is a command line tool.

You don’t necessarily need to use the CLI to work with Spring Boot

You can download the Spring CLI distribution from the Spring software repository

spring-boot-cli-xxx.BUILD-SNAPSHOT-bin.zip

Once downloaded, follow the instructions written in install.txt

Thoughts to ponder

Why move to containerless deployment

Why run the application as a plain Java program

Container deployments

Make a jar file of the application and deploy on the container

- Pre setup and configuration required
- Need to use files like web.xml to tell the container how to work
- Environmental configuration may be required. eg JNDI

Application deployments

When container is bundled inside the application, it is a better choice as

- The applications runs anywhere that Java is setup
- No need to find hosting environment
- Container is embedded inside the application which tells the container how to set up the app so that it can be access via HTTP
- Environmental configuration is internal to the application

Demo

1. Simple Java application using Spring Boot
2. Restful web application using Spring Boot
3. Spring boot application which integrates with Spring Data JPA

Summary

What we have seen so far:

- What is Spring Boot and how it works
- Create a Java application up and running using Spring Boot
- Create a Restful web application using Spring Boot
- Create a Spring boot application which integrates with Spring Data JPA

Review Question

Question 1:

-

Question 2:

-

Question 3:

-