





# 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)



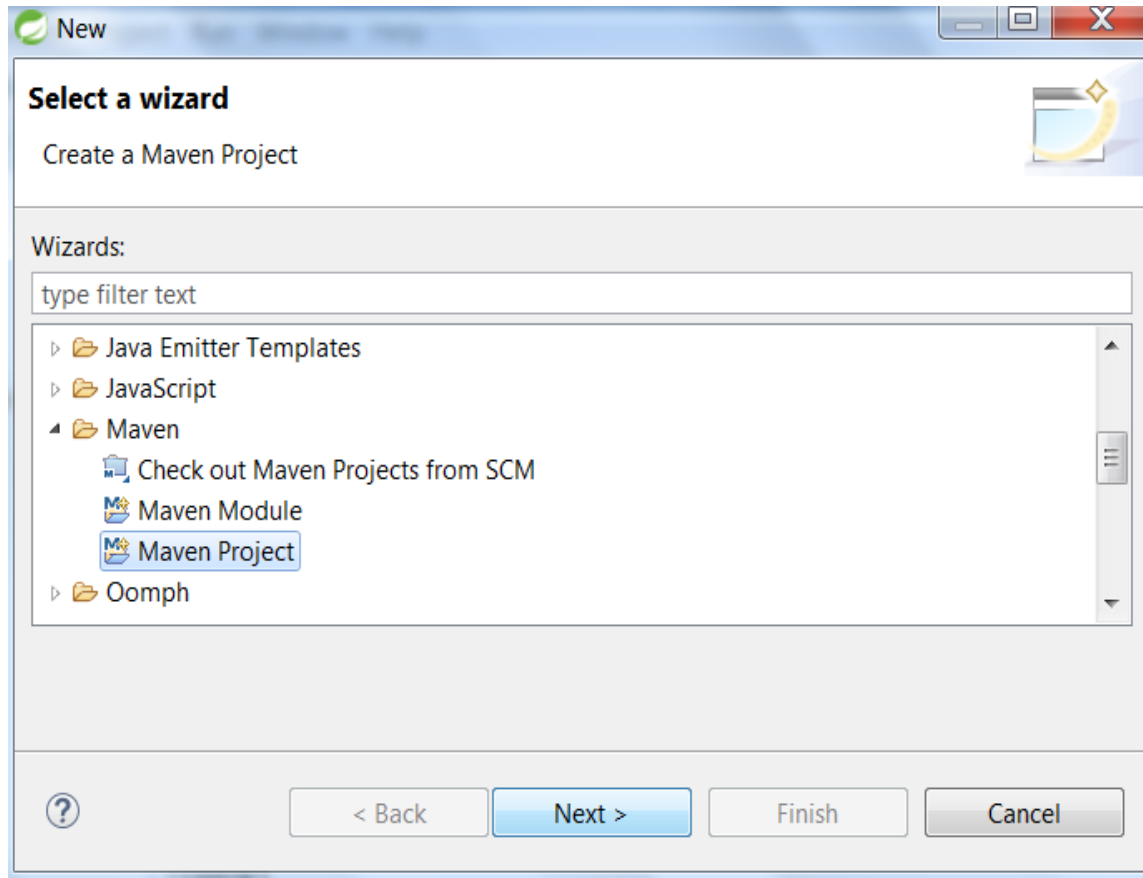
# Ways to create Spring Boot project

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**  
Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location:  **Browse...**

☐ Add project(s) to working set

Working set:  **More...**

► **Advanced**

**< Back** **Next >** **Finish** **Cancel**



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

**New Maven Project**  
Configure project

**Artifact**

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

**Parent Project**

Group Id:

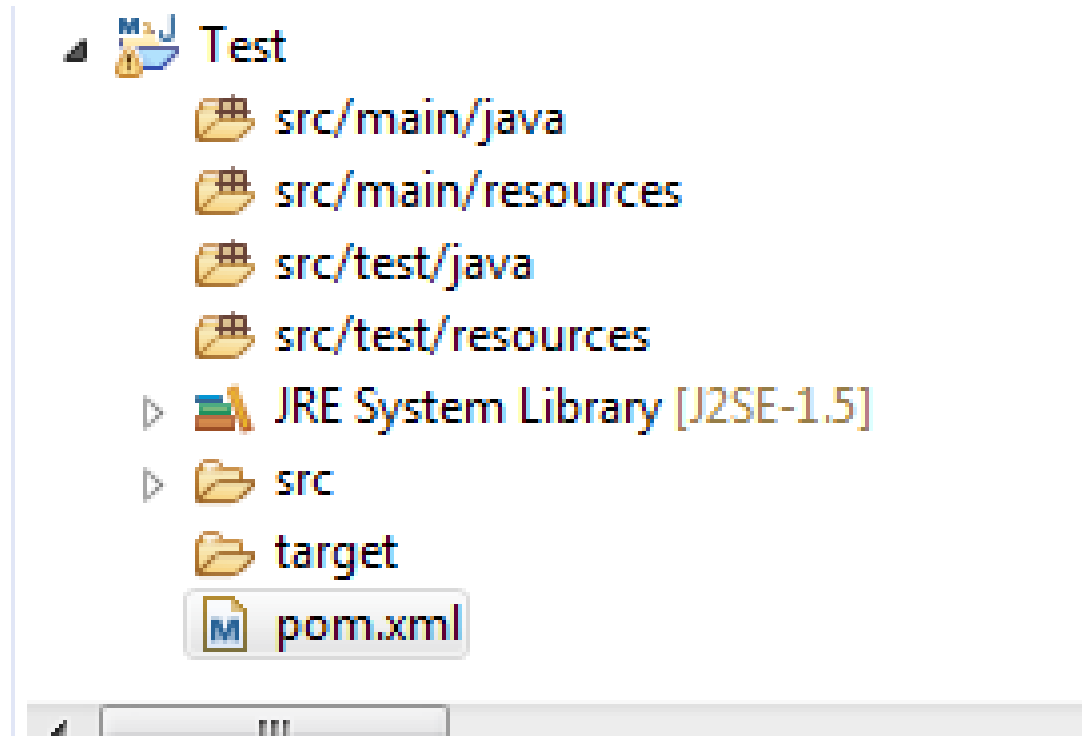
Artifact Id:

Version:

► **Advanced**



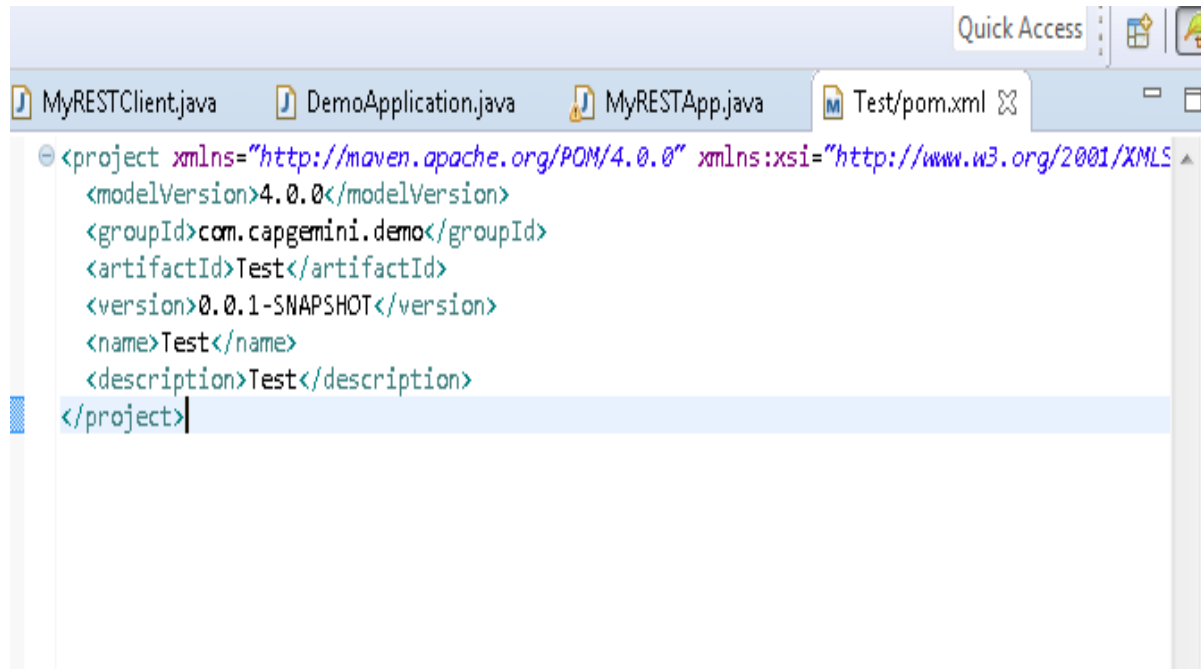
Double click on the generated pom.xml file







The default pom.xml is shown below



```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.cappgemini.demo</groupId>
    <artifactId>Test</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Test</name>
    <description>Test</description>
</project>
```

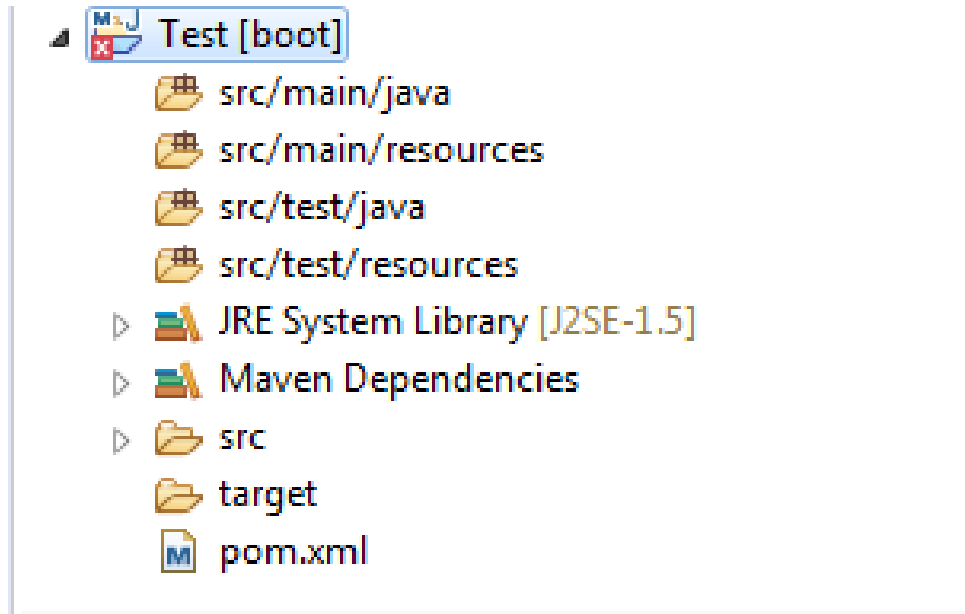


Add the following code in the pom.xml under the <description> tag

```
<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>
```

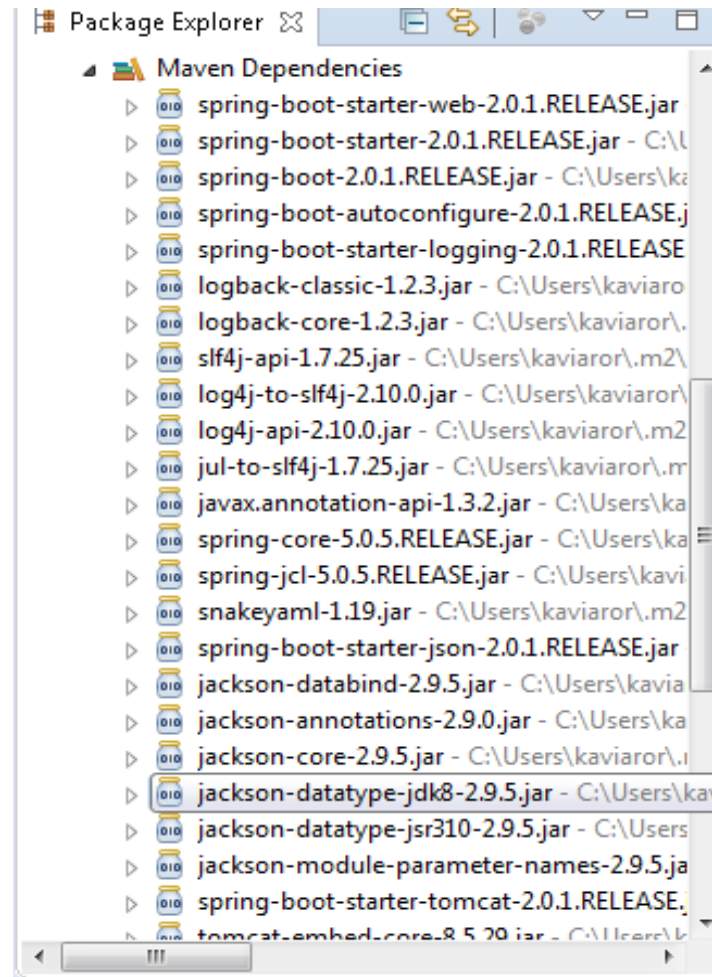


Observe that “Maven Dependencies” has been included into the project





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





## 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
erverApplicationContext : 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
rtextLoader          : Root WebApplicationContext: initialization completed in 2480 ms
vletRegistrationBean  : 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.Resour
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 application 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. @EnableAutoconfiguration : 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

The screenshot shows the Spring Initializer web application in a browser window. The address bar displays 'start.spring.io'. The page has a dark header with the text 'SPRING INITIALIZR bootstrap your application now'. Below the header, there's a form to generate a project. At the top of the form, it says 'Generate a' followed by a dropdown menu showing 'Maven Project', then 'with Spring Boot' followed by a dropdown menu showing '1.3.1'. The form is divided into two main sections: 'Project Metadata' and 'Dependencies'. Under 'Project Metadata', there's a label 'Artifact coordinates' and a 'Group' input field containing 'com.example', and an 'Artifact' input field containing 'demo'. Under 'Dependencies', there's a label 'Add Spring Boot Starters and dependencies to your application' and a 'Search for dependencies' input field containing 'Web, Security, JPA, Actuator, Devtools...'. Below the search field, there's a 'Selected Starters' section which is currently empty. At the bottom of the form, there's a green button labeled 'Generate Project' with a small icon. Below the button, there's a link that says '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
- ☐ Retrv

**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:

-