in28minutes

Spring Boot - Getting Started

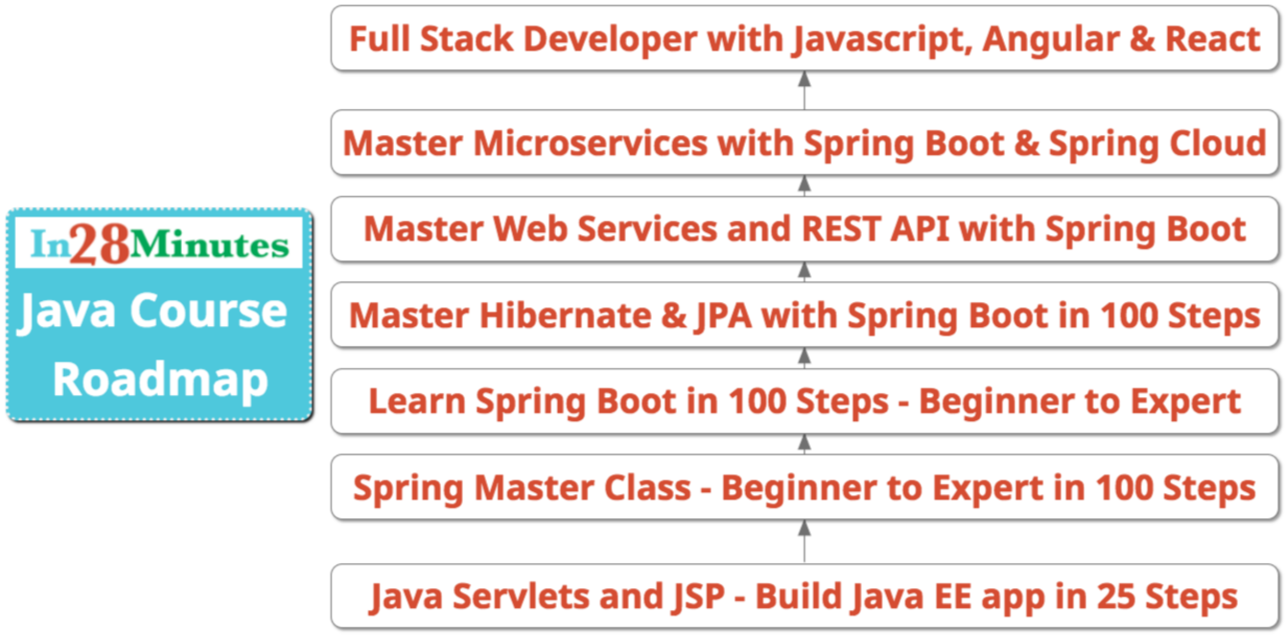
Get started with the most popular Java framework to develop awesome micro services!

Table of Contents

Congratulations

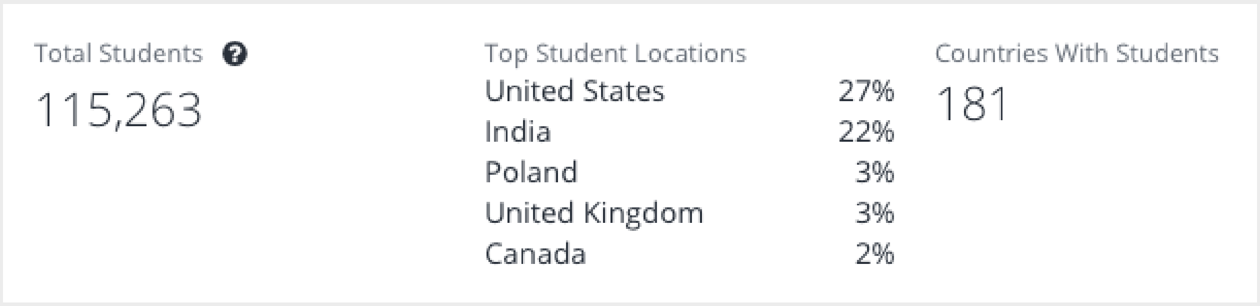
You have made a great choice in learning with in28Minutes. You are joining 100,000+ Learners learning everyday with us.

100,000+ Java beginners are learning from in28Minutes to become experts on APIs, Web Services and Microservices with Spring, Spring Boot and Spring Cloud.


In28Minutes
Java Course Roadmap
Full Stack Developer with Javascript, Angular & React
Master Microservices with Spring Boot & Spring Cloud
Master Web Services and REST API with Spring Boot
Master Hibernate & JPA with Spring Boot in 100 Steps
Learn Spring Boot in 100 Steps - Beginner to Expert
Spring Master Class - Beginner to Expert in 100 Steps
Java Servlets and JSP - Build Java EE app in 25 Steps

About in28Minutes

How did in28Minutes get to 100,000 learners across the world?



Total Students

115,263

Top Student Locations

United States	27%
India	22%
Poland	3%
United Kingdom	3%
Canada	2%

Countries With Students

181

We are focused on creating the awesome course (learning) experiences. Period.

An awesome learning experience?

What's that?

You need to get insight into the in28Minutes world to answer that.

You need to understand "***The in28Minutes Way***"

- What are our beliefs?
- What do we love?
- Why do we do what we do?
- How do we design our courses?

Let's get started on "***The in28Minutes Way***"!

Important Components of "The in28Minutes Way"

- Continuous Learning
- Hands-on
- We don't teach frameworks. We teach building applications!
- We want you to be strong on the fundamentals

- Step By Step

- Efficient and Effective
- Real Project Experiences
- Debugging and Troubleshooting skills
- Modules - Beginners and Experts!
- Focus on Unit Testing
- Code on Github

- Design and Architecture
- Modern Development Practices
- Interview Guides
- Bring the technology trends to you
- Building a connect
- Socially Conscious
- We care for our learners
- We love what we do

Troubleshooting Guide

We love all our 100,000 learners. We want to help you in every way possible.

We do not want you to get stuck because of a simple error.

This 50 page troubleshooting guide and faq is our way of thanking you for choosing to learn from in28Minutes.

.in28Minutes Trouble Shooting Guide

Spring Boot vs Spring MVC vs Spring

What is Spring Boot? What is Spring MVC? What is Spring Framework? What are their goals? How do they compare?

You will learn

- Get an overview of Spring Framework
- What are the problems that Spring Framework wanted to solve?
- Get an overview of Spring MVC Framework
- What are the problems that Spring MVC Framework wanted to solve?
- Get an overview of Spring Boot Framework
- What are the problems that Spring Boot wants to solve?
- Compare Spring vs Spring Boot vs Spring MVC
- Most important thing that you will learn is Spring, Spring MVC and Spring Boot are not competing for the same space. They solve different problems and they solve them very well.

What is the core problem that Spring Framework solves?

Think long and hard. What's the problem Spring Framework solves?

> *Most important feature of Spring Framework is Dependency Injection. At the core of all Spring Modules is Dependency Injection or IOC Inversion of Control.*

Why is this important? Because, when DI or IOC is used properly, we can develop loosely coupled applications. And loosely coupled applications can be easily unit tested.

Let’s consider a simple example:

Example without Dependency Injection

Consider the example below: WelcomeController depends on WelcomeService to get the welcome message. What is it doing to get an instance of WelcomeService? `WelcomeService service = new WelcomeService();` . It’s creating an instance of it. And that means they are tightly coupled. For example : If I create an mock for WelcomeService in a unit test for WelcomeController, How do I make WelcomeController use the mock? Not easy!

```
@RestController
public class WelcomeController {

    private WelcomeService service = new WelcomeService();

        @RequestMapping("/welcome")
        public String welcome() {
                return service.retrieveWelcomeMessage();
        }
}
```

Same Example with Dependency Injection

World looks much easier with dependency injection. You let the spring framework do the hard work. We just use two simple annotations: @Component and @Autowired.

- Using `@Component` , we tell Spring Framework - Hey there, this is a bean that you need to manage.
- Using `@Autowired` , we tell Spring Framework - Hey find the correct match for this specific type and autowire it in.

In the example below, Spring framework would create a bean for WelcomeService and autowire it into WelcomeController.

In a unit test, I can ask the Spring framework to auto-wire the mock of WelcomeService into WelcomeController. (Spring Boot makes things easy to do this with @MockBean. But, that's a different story altogether!)

```
@Component
public class WelcomeService {
    //Bla Bla Bla
}
 @RestController
public class WelcomeController {

    @Autowired
    private WelcomeService service;

        @RequestMapping("/welcome")
        public String welcome() {
                return service.retrieveWelcomeMessage();
        }
}
```

What else does Spring Framework solve?

Problem 1 : Duplication/Plumbing Code

Does Spring Framework stop with Dependency Injection? No. It builds on the core concept of Dependeny Injection with a number of Spring Modules

- Spring JDBC
- Spring MVC
- Spring AOP
- Spring ORM
- Spring JMS
- Spring Test

Consider Spring JMS and Spring JDBC for a moment.

Do these modules bring in any new functionality? No. We can do all this with J2EE or JEE. So, what do these bring in? They bring in simple abstractions. Aim of these abstractions is to

- Reduce Boilerplate Code/ Reduce Duplication
- Promote Decoupling/ Increase Unit Testablity

For example, you need much less code to use a JDBCTemplate or a JMSTemplate compared to traditional JDBC or JMS.

Problem 2 : Good Integration with Other Frameworks.

Great thing about Spring Framework is that it does not try to solve problems which are already solved. All that it does is to provide a great integration with frameworks which provide great solutions.

- Hibernate for ORM
- iBatis for Object Mapping
- JUnit & Mockito for Unit Testing

What is the core problem that Spring MVC Framework solves?

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

Why do we need Spring Boot?

Spring based applications have a lot of configuration.

When we use Spring MVC, we need to configure component scan, dispatcher servlet, a view resolver, web jars(for delivering static content) among other things.

```
<bean
```

```
 class="org.springframework.web.servlet.view.InternalResour
ceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
             <value>.jsp</value>
        </property>
   </bean>

   <mvc:resources mapping="/webjars/**"
location="/webjars/"/>
```

Below code snippet shows typical configuration of a dispatcher servlet in a web application.

```
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>

org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/todo-servlet.xml</param-
value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

When we use Hibernate/JPA, we would need to configure a datasource, an entity manager factory, a transaction manager among a host of other things.

```
    <bean id="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource"
        destroy-method="close">
        <property name="driverClass" value="${db.driver}"
/>
        <property name="jdbcUrl" value="${db.url}" />
        <property name="user" value="${db.username}" />
        <property name="password" value="${db.password}" />
    </bean>

    <jdbc:initialize-database data-source="dataSource">
        <jdbc:script location="classpath:config/schema.sql"
/>
        <jdbc:script location="classpath:config/data.sql"
/>
    </jdbc:initialize-database>

    <bean

class="org.springframework.orm.jpa.LocalContainerEntityMana
gerFactoryBean"
        id="entityManagerFactory">
        <property name="persistenceUnitName"
value="hsql_pu" />
        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory"
ref="entityManagerFactory" />
        <property name="dataSource" ref="dataSource" />
```

```
 </bean>

    <tx:annotation-driven transaction-
manager="transactionManager"/>
```

Problem #1 : Spring Boot Auto Configuration : Can we think different?

Spring Boot brings in new thought process around this.

> *Can we bring more intelligence into this? When a spring mvc jar is added into an application, can we auto configure some beans automatically?*

- How about auto configuring a Data Source if Hibernate jar is on the classpath?
- How about auto configuring a Dispatcher Servlet if Spring MVC jar is on the classpath?

There would be provisions to override the default auto configuration.

> *Spring Boot looks at a) Frameworks available on the CLASSPATH b) Existing configuration for the application. Based on these, Spring Boot provides basic configuration needed to configure the application with these frameworks. This is called `Auto Configuration`.*

Problem #2 : Spring Boot Starter Projects : Built around well known patterns

Let's say we want to develop a web application.

First of all we would need to identify the frameworks we want to use, which versions of frameworks to use and how to connect them together.

All web application have similar needs. Listed below are some of the dependencies we use in our Spring MVC Course. These include Spring MVC, Jackson Databind (for data binding), Hibernate-Validator (for server side validation using Java Validation API) and Log4j (for logging). When creating this course, we had to choose the compatible versions of all these frameworks.

```
<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>
```

Here's what the Spring Boot documentations says about starters.

Starters 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, without having to hunt through sample code and copy paste loads of dependency descriptors. For example, if you *want to get started using Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go.*

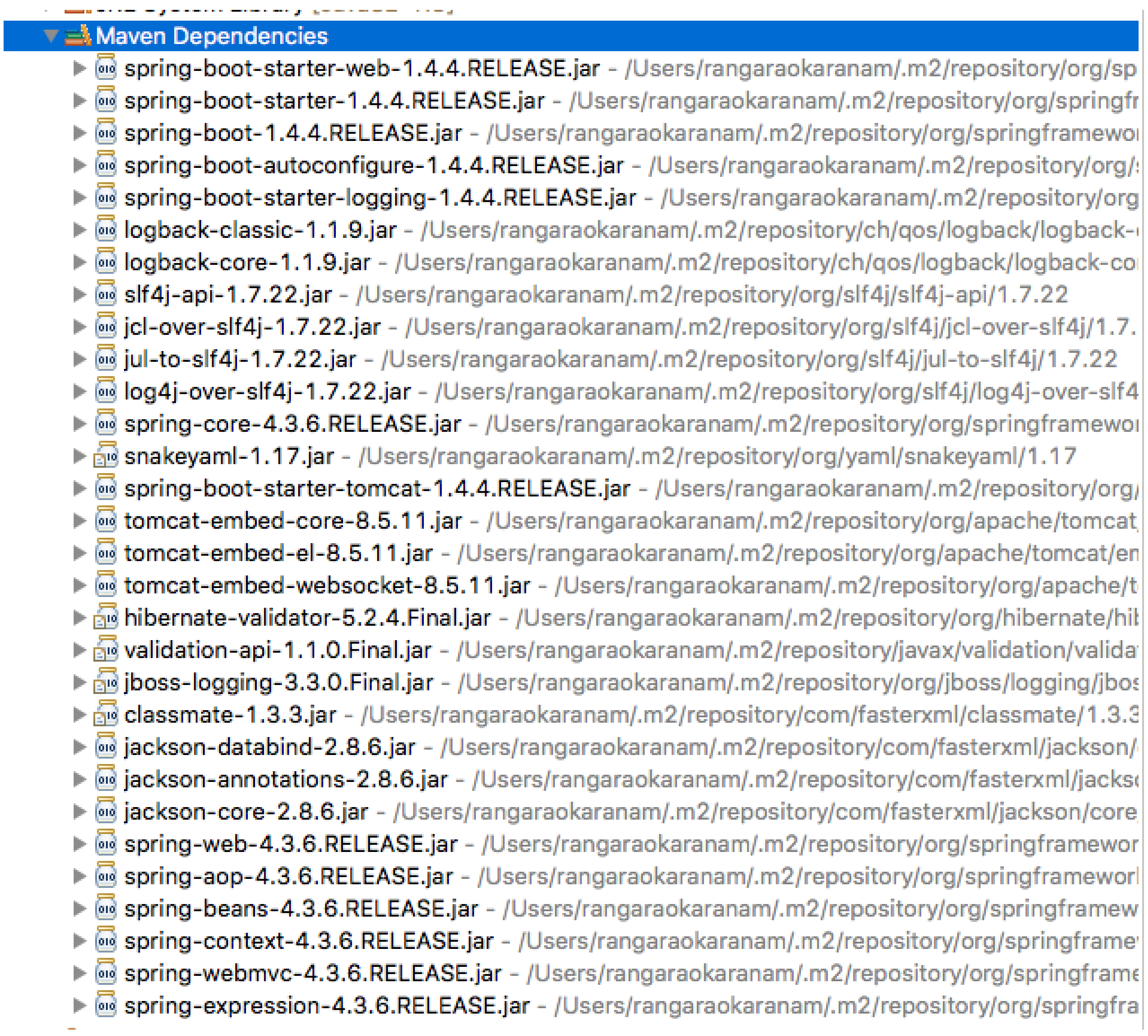
Let’s consider an example starter - Spring Boot Starter Web.

If you want to develop a web application or an application to expose restful services, Spring Boot Start Web is the starter to pick. Lets create a quick project with Spring Boot Starter Web using Spring Initializr.

Dependency for Spring Boot Starter Web

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

Following screenshot shows the different dependencies that are added in to our application



```
▼ Maven Dependencies
  ▶ spring-boot-starter-web-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/sp
  ▶ spring-boot-starter-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springfr
  ▶ spring-boot-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewo
  ▶ spring-boot-autoconfigure-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/
  ▶ spring-boot-starter-logging-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org
  ▶ logback-classic-1.1.9.jar - /Users/rangaraokaranam/.m2/repository/ch/qos/logback/logback-
  ▶ logback-core-1.1.9.jar - /Users/rangaraokaranam/.m2/repository/ch/qos/logback/logback-co
  ▶ slf4j-api-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/slf4j-api/1.7.22
  ▶ jcl-over-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/jcl-over-slf4j/1.7.
  ▶ jul-to-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/jul-to-slf4j/1.7.22
  ▶ log4j-over-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/log4j-over-slf4
  ▶ spring-core-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewo
  ▶ snakeyaml-1.17.jar - /Users/rangaraokaranam/.m2/repository/org/yaml/snakeyaml/1.17
  ▶ spring-boot-starter-tomcat-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/
  ▶ tomcat-embed-core-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat
  ▶ tomcat-embed-el-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat/en
  ▶ tomcat-embed-websocket-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/t
  ▶ hibernate-validator-5.2.4.Final.jar - /Users/rangaraokaranam/.m2/repository/org/hibernate/hil
  ▶ validation-api-1.1.0.Final.jar - /Users/rangaraokaranam/.m2/repository/javax/validation/valida
  ▶ jboss-logging-3.3.0.Final.jar - /Users/rangaraokaranam/.m2/repository/org/jboss/logging/jbos
  ▶ classmate-1.3.3.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/classmate/1.3.3
  ▶ jackson-databind-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/
  ▶ jackson-annotations-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jacks
  ▶ jackson-core-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/core
  ▶ spring-web-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewor
  ▶ spring-aop-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewor
  ▶ spring-beans-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframew
  ▶ spring-context-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframe
  ▶ spring-webmvc-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframe
  ▶ spring-expression-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springfra
```

Dependencies can be classified into:

- Spring - core, beans, context, aop
- Web MVC - (Spring MVC)
- Jackson - for JSON Binding
- Validation - Hibernate Validator, Validation API
- Embedded Servlet Container - Tomcat
- Logging - logback, slf4j

Any typical web application would use all these dependencies. Spring Boot Starter Web comes pre packaged with these. As a developer, I would not need to worry about either these dependencies or their compatible versions.

Spring Boot Starter Project Options

As we see from Spring Boot Starter Web, starter projects help us in quickly getting started with developing specific types of applications.

- spring-boot-starter-web-services - SOAP Web Services
- spring-boot-starter-web - Web & RESTful applications
- spring-boot-starter-test - Unit testing and Integration Testing
- spring-boot-starter-jdbc - Traditional JDBC
- spring-boot-starter-hateoas - Add HATEOAS features to your services
- spring-boot-starter-security - Authentication and Authorization using Spring Security
- spring-boot-starter-data-jpa - Spring Data JPA with Hibernate
- spring-boot-starter-cache - Enabling Spring Framework's caching support
- spring-boot-starter-data-rest - Expose Simple REST Services using Spring Data REST

Other Goals of Spring Boot

There are a few starters for technical stuff as well

- spring-boot-starter-actuator - To use advanced features like monitoring & tracing to your application out of the box
- spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat - To pick your specific choice of Embedded Servlet Container
- spring-boot-starter-logging - For Logging using logback
- spring-boot-starter-log4j2 - Logging using Log4j2

Spring Boot aims to enable production ready applications in quick time.

- Actuator : Enables Advanced Monitoring and Tracing of applications.
- Embedded Server Integrations - Since server is integrated into the application, I would NOT need to have a separate application server installed on the server.
- Default Error Handling

Spring Boot Project with Eclipse and Maven

You will learn

- How to bootstrap a simple project with Spring Initializr?
- How to use Spring Starter Eclipse Plugin to create a simple project with Spring Boot, Maven and Eclipse?
- How to create a Spring Boot Project manually step by step?

Introduction to Maven

Q : Why Maven?

You don’t want to store all the libraries in your project!

You want to tell I need A, B, C and you would want the tool to download the libraries and make them available to you.

That’s Maven. The tool which you use to manage the libraries.

If you need a new version of the library, you can change the version and your project is ready!

Also, You don’t need to worry about what libraries your library needs to work. For example, Spring might need other libaries - logging, xml etc.

Once you declare a dependency on Spring, Maven would download

- Spring
- And all dependencies of Spring

Isn’t that cool?

Big Picture of Maven

Defining what Maven does is very difficult.

Every Day Developer does a lot of things

- Manages Dependencies
  - Web Layer (Spring MVC)
  - Data Layer (JPA - Hibernate) etc.

- Build a jar or a war or an ear
- Run the application locally
  - Tomcat or Jetty

- Deploy to a T environment
- Add new dependencies to a project
- Run Unit Tests
- Generate Projects
- Create Eclipse Workspace

> *Maven helps us do all these and more…*

Naming a project
You define dependencies in your pom.xml.

Maven would download the dependencies and make them available for use in your project.

But, how does Maven know what dependency to download?

You need to tell it by giving the details of the dependency.

Just like you can identify a Java class with a class name and a package name, you can identify a maven artifact by a GroupId and an ArtifactId.

```
     <groupId>com.in28minutes.learning.maven</groupId>
    <artifactId>maven-in-few-steps</artifactId>
```

Declaring Dependencies
Dependencies are frameworks that you would need to develop your project.

In the example below we are adding two dependencies.

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

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

Maven Build Life Cycle

When we run "mvn clean install", we are executing the complete maven build life cycle.

Build LifeCycle is a sequence of steps

- Validate
- Compile
- Test
- Package
- Integration Test
- Verify
- Install
- Deploy

Maven follows Convention over Configuration.

Pre defined folder structure

- Source Code
    - ${basedir}/src/main/java
    - ${basedir}/src/main/resources

- Test Code
    - ${basedir}/src/test

How does Maven Work?

Maven Repository contains all the jars indexed by artifact id and group id.

Once we add a dependency to our pom.xml, maven asks the maven repository for the jar dependencies giving group id and the artifact id as the input.

- Maven repository stores all the versions of all dependencies. JUnit 4.2,4.3,4.4

The jar dependencies are stored on your machine in a folder called maven local repository. All our projects would refer to the jars from the maven local repository.

Local Repository : a temp folder on your machine where maven stores the jar and dependency files that are downloaded from Maven Repository.

Important Maven Commands

- mvn –version -> Find the maven version
- mvn compile -> compiles source files
- mvn test-compile -> compiles test files - one thing to observe is this also compiles source files
- mvn clean -> deletes target directory
- mvn test -> run unit tests
- mvn package -> creates a jar for the project
- help:effective-settings -> Debug Maven Settings
- help:effective-pom -> Look at the complete pom after all inheritances from parent poms are resolved

-
- dependency:tree -> Look at all the dependencies and transitive dependencies
- dependency:sources -> Download source code for all dependencies
- –debug -> Debug flag. Can be used with all the above commands

Creating Spring Boot Projects with Eclipse and Maven

There are three options to create Spring Boot Projects with Eclipse and Maven

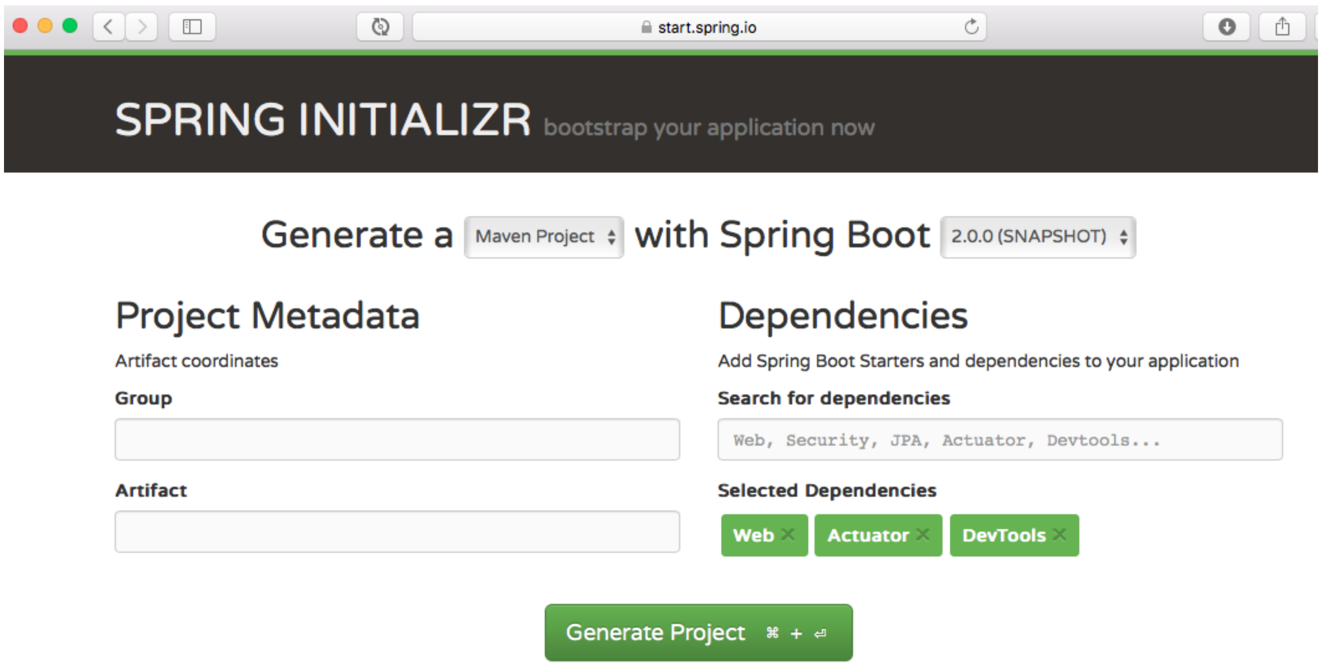
- Spring Initializr - https://start.spring.io
- Use STS or STS Eclipse Plugin and Create a Spring Boot Maven Project directly from Eclipse
- Manually Create a Maven Project and add Spring Boot Starter Dependencies.

We will use a Spring Boot Starter Web as an example.

Option 1 - Bootstrapping Spring Boot Project with Spring Initializr

Creating a Web application with Spring Initializr is a cake walk. We will use Spring Web MVC as our web framework.

> *Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.*


start.spring.io
SPRING INITIALIZR bootstrap your application now
Generate a Maven Project with Spring Boot 2.0.0 (SNAPSHOT)
Project Metadata
Artifact coordinates
Group
Artifact
Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies
Web
Actuator
DevTools
Generate Project ⌘ + ⏎

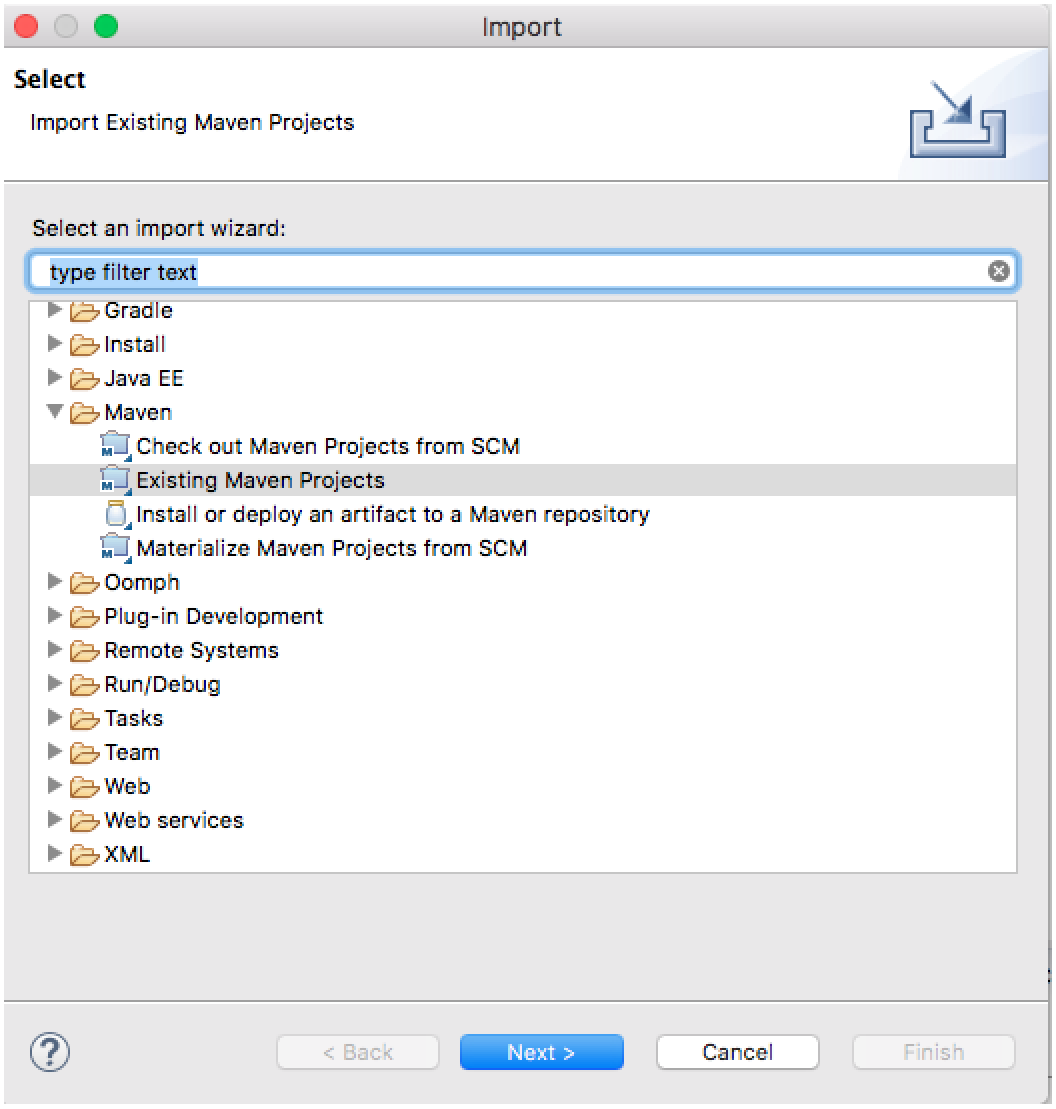
As shown in the image above, following steps have to be done

- Launch Spring Initializr and choose the following
    - Choose `com.in28minutes.springboot` as Group
    - Choose `student-services` as Artifact
    - Choose following dependencies
        - Web
        - Actuator
        - DevTools
- Click Generate Project.

This would download a zip file to your local machine.

Unzip the zip file and extract to a folder.

In Eclipse, Click File -> Import -> Existing Maven Project as shown below.



Import

Select

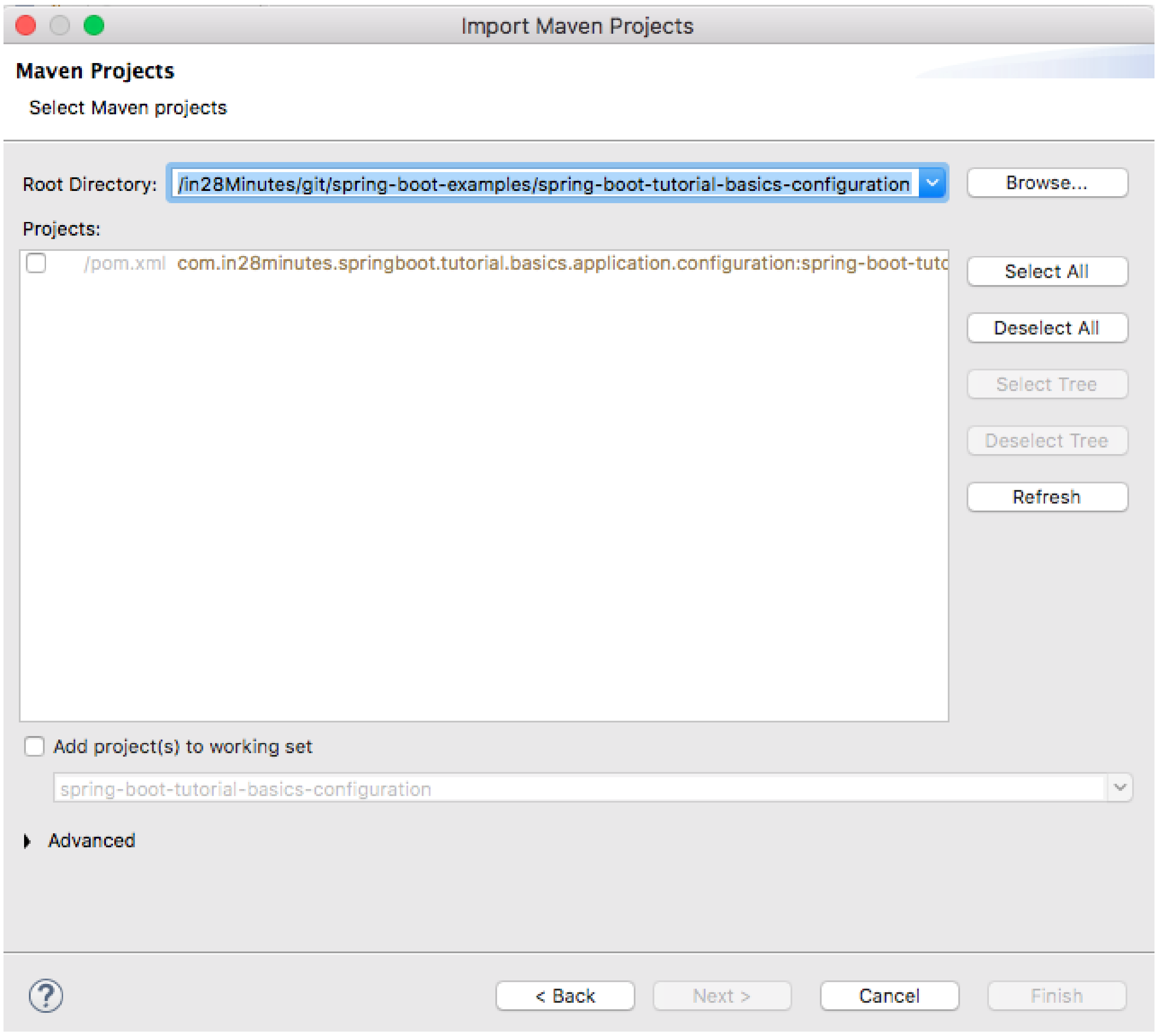
Import Existing Maven Projects

Select an import wizard:

type filter text

- Gradle
- Install
- Java EE
- Maven
  - Check out Maven Projects from SCM
  - Existing Maven Projects
  - Install or deploy an artifact to a Maven repository
  - Materialize Maven Projects from SCM
- Oomph
- Plug-in Development
- Remote Systems
- Run/Debug
- Tasks
- Team
- Web
- Web services
- XML

Navigate or type in the path of the folder where you extracted the zip file to in the next screen.



Root Directory: /in28Minutes/git/spring-boot-examples/spring-boot-tutorial-basics-configuration

/pom.xml com.in28minutes.springboot.tutorial.basics.application.configuration:spring-boot-tuto

spring-boot-tutorial-basics-configuration

Once you click Finish, Maven would take some time to download all the dependencies and initialize the project.

That's it. Your first Spring Project is Ready.

Follow these links to understand more about the project that is created - Spring Boot vs Spring vs Spring MVC, Auto Configuration, Spring Boot Starter Projects, Spring Boot Starter Parent, Spring Boot Initializr

Option 2 - Using STS or STS Eclipse Plugin to create Spring Boot Maven Project

With Spring tool suite, you can directly create a spring boot project from Eclipse.

You should either download the complete installation of STS or You can install the STS Eclipse plugin

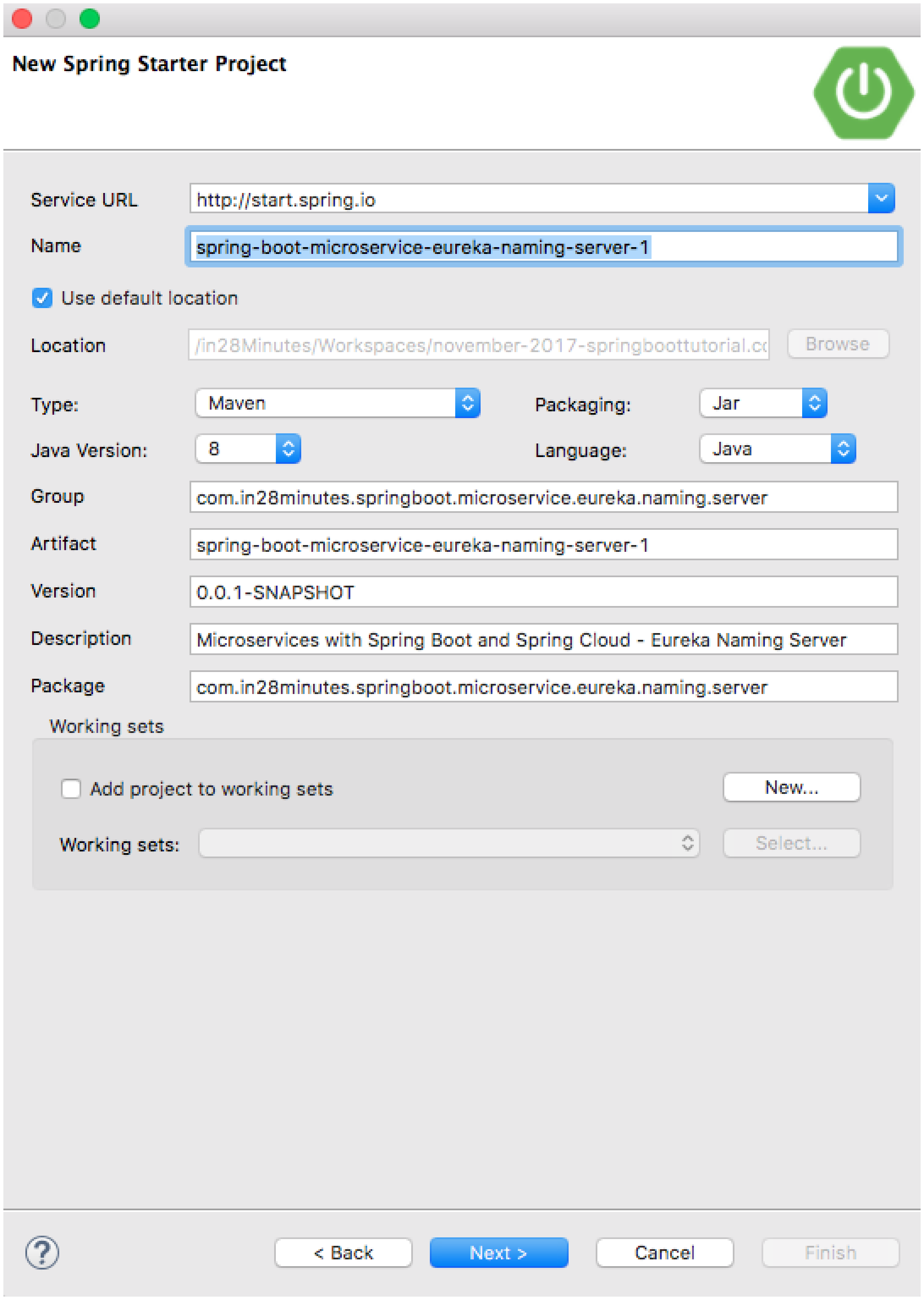
> *https://spring.io/tools/sts/all provides the complete download of STS as well as the Update Sites for STS Eclipse Plugin.*

In Eclipse/STS, start with File -> New -> Spring Starter Project as shown below.
Image

In the next screen, you can choose the following for your project.

- Group ID
- Artifact ID
- Root Package
- Version
- Description
- Java Version
- Language
- Packaging

Make sure you choose Maven as Type.



Service URL: http://start.spring.io

Name: spring-boot-microservice-eureka-naming-server-1

☑ Use default location

Location: /in28Minutes/Workspaces/november-2017-springboottutorial.co

Type: Maven

Packaging: Jar

Java Version: 8

Language: Java

Group: com.in28minutes.springboot.microservice.eureka.naming.server

Artifact: spring-boot-microservice-eureka-naming-server-1

Version: 0.0.1-SNAPSHOT

Description: Microservices with Spring Boot and Spring Cloud - Eureka Naming Server

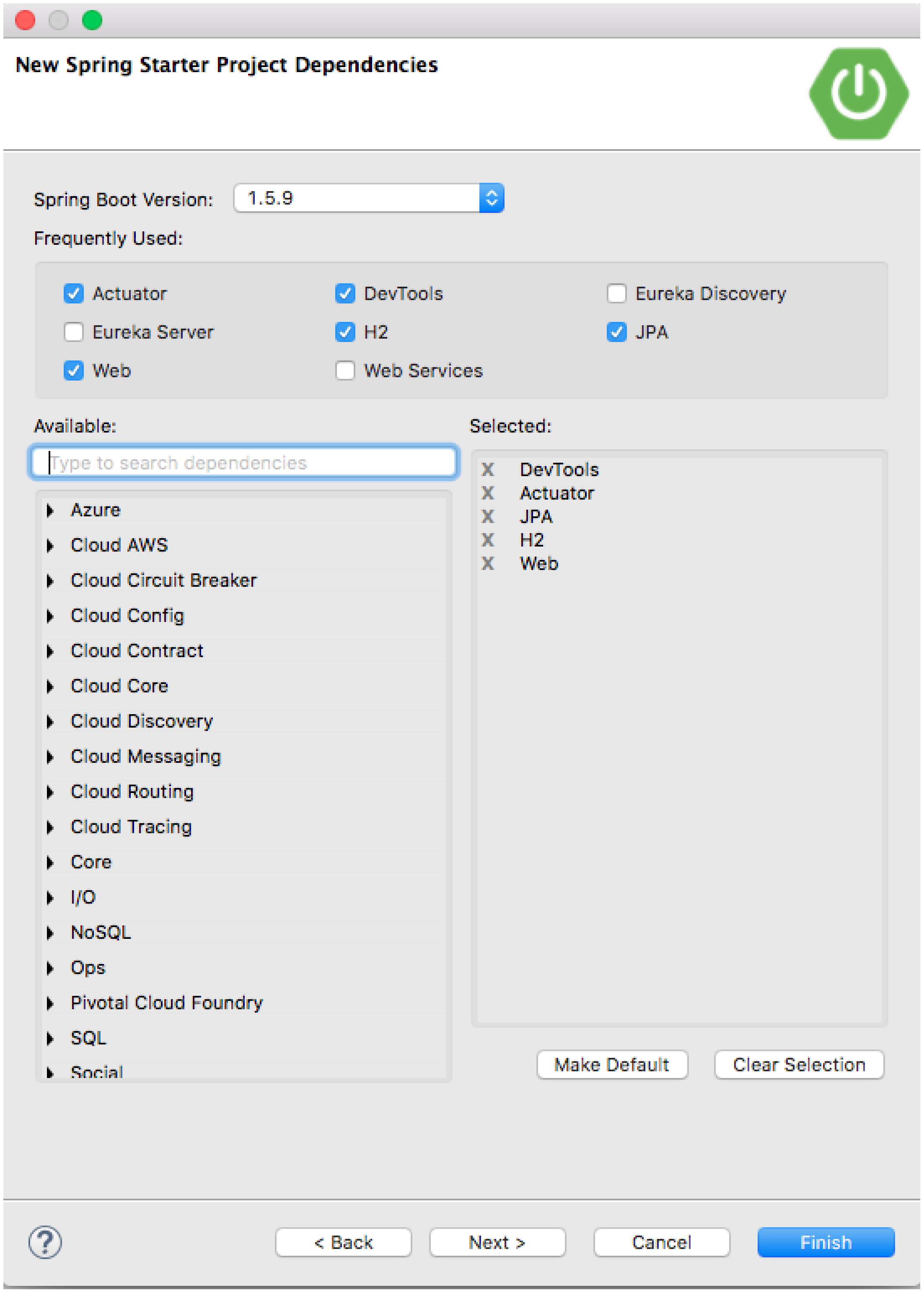
Package: com.in28minutes.springboot.microservice.eureka.naming.server

Working sets

☐ Add project to working sets

Working sets:

In the next screen, you can choose the dependencies that you would want to add to your Spring Boot project.


New Spring Starter Project Dependencies

Spring Boot Version: 1.5.9

Frequently Used:

- [✓] Actuator
- [✓] DevTools
- [ ] Eureka Discovery
- [ ] Eureka Server
- [✓] H2
- [✓] JPA
- [✓] Web
- [ ] Web Services

Available:

Type to search dependencies

- Azure
- Cloud AWS
- Cloud Circuit Breaker
- Cloud Config
- Cloud Contract
- Cloud Core
- Cloud Discovery
- Cloud Messaging
- Cloud Routing
- Cloud Tracing
- Core
- I/O
- NoSQL
- Ops
- Pivotal Cloud Foundry
- SQL
- Social

Selected:

- X DevTools
- X Actuator
- X JPA
- X H2
- X Web

Once you click Finish, Maven would take some time to download all the dependencies and initialize the project.

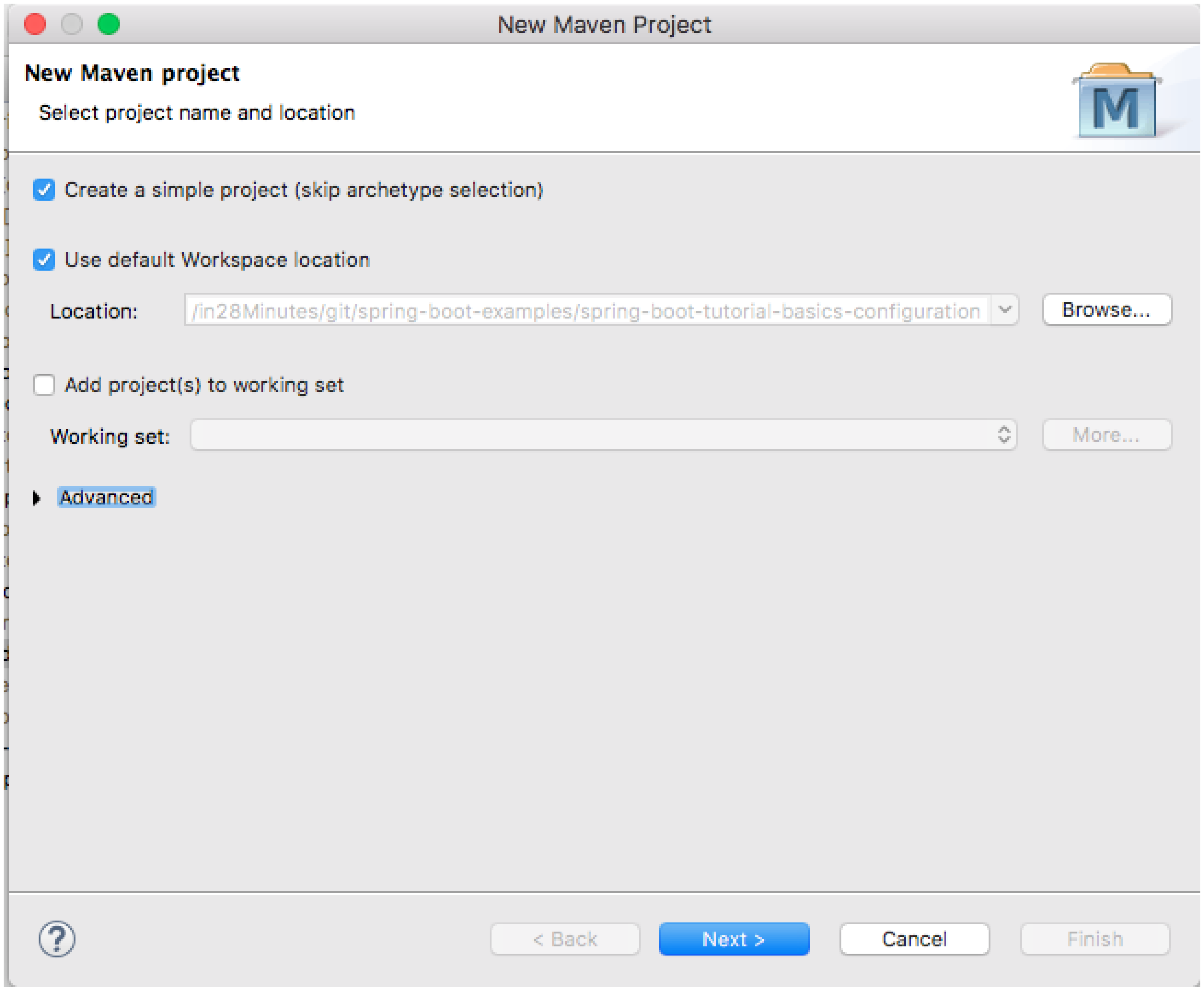
That’s it. Your first Spring Project is Ready.

Option 3 - Manually Create a Maven Spring Boot Project

The last option is to create the project manually.

In Eclipse, start with File > New > Maven Project

Choose Create a simple project as shown in the screenshot below:


New Maven Project

New Maven project

Select project name and location

☑ Create a simple project (skip archetype selection)

☑ Use default Workspace location

Location: /in28Minutes/git/spring-boot-examples/spring-boot-tutorial-basics-configuration

☐ Add project(s) to working set

Working set:

In the next screen, provide these details for your project and click Finish.

- Group ID
- Artifact ID
- Version

This would create a basic Maven project with Zero dependencies.

Next add in the appropriate Spring Boot Starters into the pom.xml

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

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

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

Starter Web is used for developing Spring Boot Web Applications or RESTful Services.

Starter Test provides unit testing and integration test capablities with Spring Test, Mockito and JUnit.

One this we are missing is the version for these dependencies.

We will add Spring Boot Starter Parent as the parent pom in the pom.xml

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.M6</version>
    <relativePath /> <!-- lookup parent from repository -->
</parent>
```

Let's configure the Java version to use as 1.8

```
<properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
</properties>
```

Next step is to create a Spring Boot Application class which will be launching point of the web application.

/src/main/java/com/in28minutes/springboot/tutorial/SpringBootWebApplication.java

```
package com.in28minutes.springboot.tutorial;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;

@SpringBootApplication
public class SpringBootWebApplication {

    public static void main(String[] args) {

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

All that you need to do is to add `@SpringBootApplication` and use `SpringApplication.run()` static method to launch the Spring Application context.

When you launch this class a java application, you would see that an embedded tomcat server would launch up and you are ready to add in features to this application.

Summary

In this article, we looked at the different options to creating Spring Boot Projects with Maven and Eclipse. I love option 2 of creating the project directly from Eclipse using STS plugin. But, you might have your own preference.

Spring Initializr - Bootstrap Your Spring Boot Applications at F1 speed!

Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.

It allows you to create varied range of Spring Boot based Applications from a very simple UI. Some of the types of applications you can bootstrap are:

- Web Applications
- Restful Applications
- Batch Applications

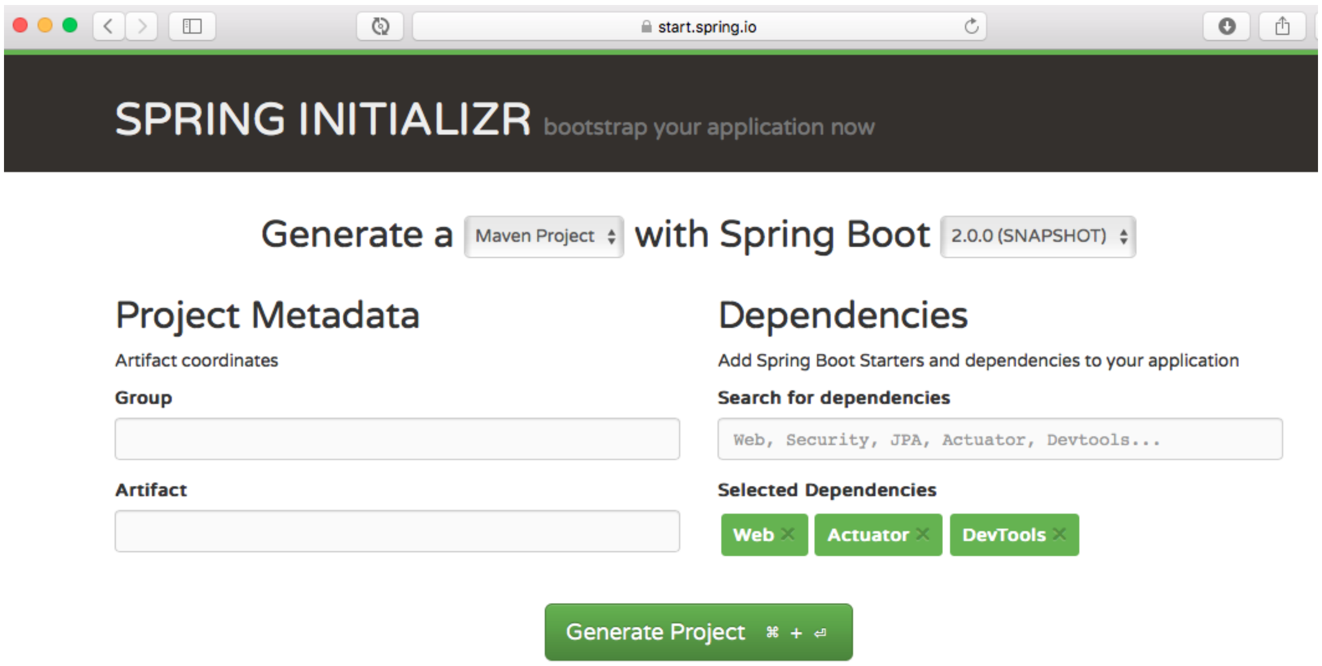
Spring Boot provides a wide range of starter projects. Spring Initializr suppports all of them and more. Among a varied range of starter projects and options supported are:

- spring-boot-starter-web-services : For building applications exposing SOAP Web Services
- spring-boot-starter-web - Build Web applications & RESTful applications
- spring-boot-starter-test - Write great Unit and Integration Tests
- spring-boot-starter-jdbc - Traditional JDBC Applications
- spring-boot-starter-hateoas - Make your services more RESTful by adding HATEOAS features
- spring-boot-starter-security - Authentication and Authorization using Spring Security
- spring-boot-starter-data-jpa - Spring Data JPA with Hibernate
- spring-boot-starter-cache - Enabling Spring Framework's caching support
- spring-boot-starter-data-rest - Expose Simple REST Services using Spring Data REST

In this guide, lets consider creating a simple web application with Spring Initializr.

Bootstrapping a Web application with Spring Initializr

Creating a Web application with Spring Initializr is very simple.


start.spring.io
SPRING INITIALIZR bootstrap your application now
Project Metadata
Artifact coordinates
Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Generate Project ⌘ + ⏎

Generate a Maven Project with Spring Boot 2.0.0 (SNAPSHOT)

Group

Artifact

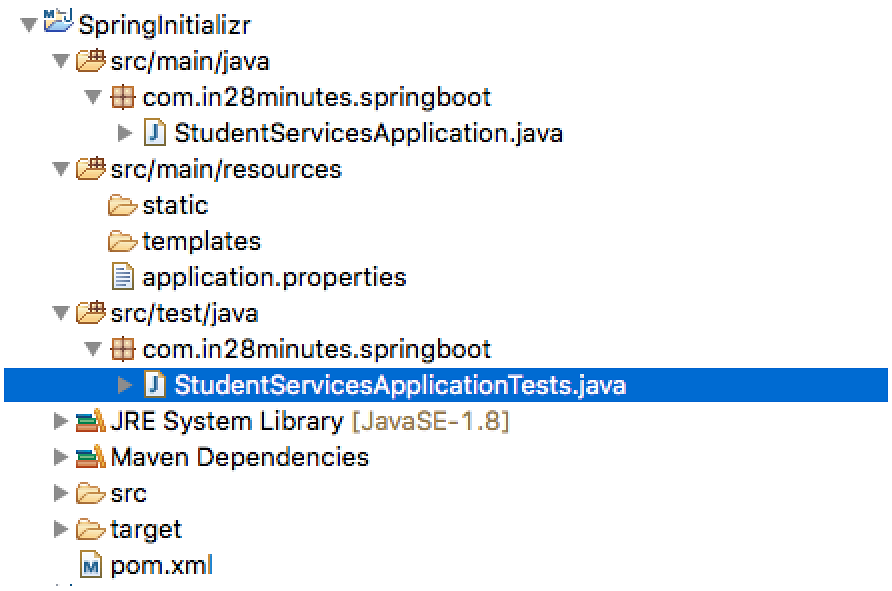
Selected Dependencies: Web, Actuator, DevTools

As shown in the image above, following steps have to be done

- Launch Spring Initializr http://start.spring.io/ and choose the following
  - Choose `com.in28minutes.springboot` as Group
  - Choose `student-services` as Artifact
  - Choose following dependencies
    - Web
- Click Generate Project button at the bottom of the page.
- Import the project into Eclipse.

Structure of the project created

Screenshot shows the project structure of the imported maven project.



```
SpringInitializr
  src/main/java
    com.in28minutes.springboot
      StudentServicesApplication.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
    com.in28minutes.springboot
      StudentServicesApplicationTests.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
```

- StudentServicesApplication.java - Spring Boot Launcher. Initializes Spring Boot Auto Configuration and Spring Application Context.
- application.properties - Application Configuration file.
- StudentServicesApplicationTests.java - Simple launcher for use in unit tests.
- pom.xml - Included dependencies for Spring Boot Starter Web. Uses Spring Boot Starter Parent as parent pom.

Complete Code Generated

Lets look at each of the file that is generated

/pom.xml

Three important things that are configured in pom.xml.

- Spring Boot Parent Pom - You can read more about Spring Boot Starter Parent here - http://www.springboottutorial.com/spring-boot-starter-parent.
- Spring Boot Starter Web - You can read more about Spring Boot Starter Web here - http://www.springboottutorial.com/spring-boot-starter-projects.
- Spring Boot Starter Plugin

```
<?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/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>com.in28minutes.springboot</groupId>
        <artifactId>student-services-
initializr</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>

        <name>student-services</name>
        <description>Demo project for Spring
Boot</description>

        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-
parent</artifactId>
                <version>1.4.4.RELEASE</version>
                <relativePath/> <!-- lookup parent from
repository -->
        </parent>

        <properties>
                <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
                <java.version>1.8</java.version>
        </properties>

        <dependencies>
                <dependency>

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

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

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

                <dependency>

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

<groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>

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

```
</project>
```

/src/main/java/com/in28minutes/springboot/StudentServicesApplication.java

- `@SpringBootApplication` - Initializes Spring Boot Auto Configuration and Spring application context. Read more about Auto Configuration here - http://www.springboottutorial.com/spring-boot-auto-configuration.
- `SpringApplication.run` - Static method to launch a Spring Boot Application.

```
package com.in28minutes.springboot;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplicatio
n;

@SpringBootApplication
public class StudentServicesApplication {

        public static void main(String[] args) {

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

/src/main/resources/application.properties

```
```

/src/test/java/com/in28minutes/springboot/StudentServicesApplicationTests.java

- Integration test launches the complete Spring Boot Application. Read more about integration tests here -

- http://www.springboottutorial.com/integration-testing-for-spring-boot-rest-services.

```
package com.in28minutes.springboot;

import org.junit.Test;
import org.junit.runner.RunWith;
import
org.springframework.boot.test.context.SpringBootTest;
import
org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class StudentServicesApplicationTests {

        @Test
        public void contextLoads() {
        }

}
```

Running the application

Following log is generate when you run StudentServicesApplication.java as a Java Application.

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.4.4.RELEASE)

2017-01-28 17:37:28.970  INFO 4311 --- [  restartedMain]
```

```
 c.i.s.StudentServicesApplication         : Starting
StudentServicesApplication on Rangas-MacBook-Pro.local with
PID 4311
(/in28Minutes/Workspaces/SpringBootTutorialWebsite2017Jan/c
reating-web-project/target/classes started by
rangaraokaranam in
/in28Minutes/Workspaces/SpringBootTutorialWebsite2017Jan/cr
eating-web-project)
2017-01-28 17:37:28.978  INFO 4311 --- [  restartedMain]
c.i.s.StudentServicesApplication         : No active
profile set, falling back to default profiles: default
2017-01-28 17:37:29.179  INFO 4311 --- [  restartedMain]
ationConfigEmbeddedWebApplicationContext : Refreshing
org.springframework.boot.context.embedded.AnnotationConfigE
mbeddedWebApplicationContext@3c9199ed: startup date [Sat
Jan 28 17:37:29 IST 2017]; root of context hierarchy
2017-01-28 17:37:30.970  INFO 4311 --- [  restartedMain]
o.s.core.annotation.AnnotationUtils      : Failed to
introspect annotations on [class
org.springframework.boot.actuate.autoconfigure.EndpointWebM
vcHypermediaManagementContextConfiguration]:
java.lang.ArrayStoreException:
sun.reflect.annotation.TypeNotPresentExceptionProxy
2017-01-28 17:37:30.972  INFO 4311 --- [  restartedMain]
o.s.core.annotation.AnnotationUtils      : Failed to
introspect annotations on [class
org.springframework.boot.actuate.autoconfigure.EndpointWebM
vcHypermediaManagementContextConfiguration]:
java.lang.ArrayStoreException:
sun.reflect.annotation.TypeNotPresentExceptionProxy
2017-01-28 17:37:33.167  INFO 4311 --- [  restartedMain]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat
initialized with port(s): 8080 (http)
2017-01-28 17:37:33.224  INFO 4311 --- [  restartedMain]
o.apache.catalina.core.StandardService   : Starting service
Tomcat
2017-01-28 17:37:33.226  INFO 4311 --- [  restartedMain]
```

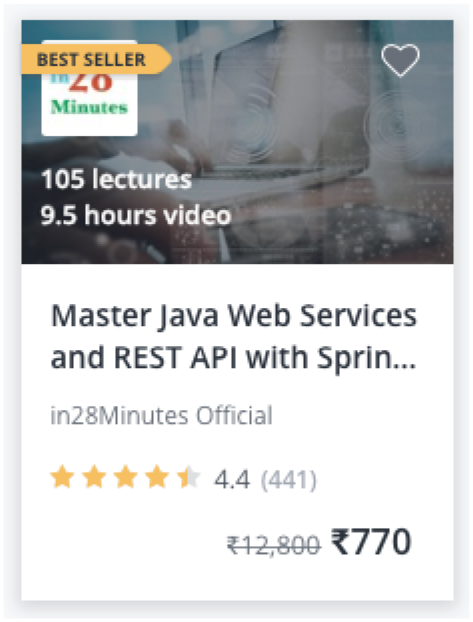
```
org.apache.catalina.core.StandardEngine  : Starting Servlet
Engine: Apache Tomcat/8.5.11
2017-01-28 17:37:33.437  INFO 4311 --- [ost-startStop-1]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing
Spring embedded WebApplicationContext
2017-01-28 17:37:33.438  INFO 4311 --- [ost-startStop-1]
o.s.web.context.ContextLoader            : Root
WebApplicationContext: initialization completed in 4296 ms
2017-01-28 17:37:33.905  INFO 4311 --- [ost-startStop-1]
o.s.b.w.servlet.ServletRegistrationBean  : Mapping servlet:
'dispatcherServlet' to [/]
2017-01-28 17:37:33.911  INFO 4311 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter:
'metricsFilter' to: [/*]
2017-01-28 17:37:33.912  INFO 4311 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter:
'characterEncodingFilter' to: [/*]
2017-01-28 17:37:33.912  INFO 4311 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter:
'hiddenHttpMethodFilter' to: [/*]
2017-01-28 17:37:33.912  INFO 4311 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter:
'httpPutFormContentFilter' to: [/*]
2017-01-28 17:37:33.912  INFO 4311 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter:
'requestContextFilter' to: [/*]
2017-01-28 17:37:33.913  INFO 4311 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter:
'webRequestLoggingFilter' to: [/*]
2017-01-28 17:37:33.913  INFO 4311 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter:
'applicationContextIdFilter' to: [/*]
2017-01-28 17:37:34.546  INFO 4311 --- [  restartedMain]
s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for
@ControllerAdvice:
org.springframework.boot.context.embedded.AnnotationConfigE
mbeddedWebApplicationContext@3c9199ed: startup date [Sat
Jan 28 17:37:29 IST 2017]; root of context hierarchy
2017-01-28 17:37:34.653  INFO 4311 --- [  restartedMain]
```

```
s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "
{[/error]}" onto public
org.springframework.http.ResponseEntity<java.util.Map<java.
lang.String, java.lang.Object>>
org.springframework.boot.autoconfigure.web.BasicErrorContro
ller.error(javax.servlet.http.HttpServletRequest)
2017-01-28 17:37:34.654  INFO 4311 --- [  restartedMain]
s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "
{[/error],produces=[text/html]}" onto public
org.springframework.web.servlet.ModelAndView
org.springframework.boot.autoconfigure.web.BasicErrorContro
ller.errorHtml(javax.servlet.http.HttpServletRequest,javax.
servlet.http.HttpServletResponse)
2017-01-28 17:37:34.717  INFO 4311 --- [  restartedMain]
o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path
[/webjars/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpReques
tHandler]
2017-01-28 17:37:34.717  INFO 4311 --- [  restartedMain]
o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path
[/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpReques
tHandler]
2017-01-28 17:37:34.785  INFO 4311 --- [  restartedMain]
o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path
[/**/favicon.ico] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpReques
tHandler]
2017-01-28 17:37:35.364  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/dump
|| /dump.json],methods=[GET],produces=[application/json]}"
onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.366  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "
{[/metrics/{name:.*}],methods=[GET],produces=
[application/json]}" onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.MetricsMvcEnd
```

```
point.value(java.lang.String)
2017-01-28 17:37:35.367  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "
{[/metrics || /metrics.json],methods=[GET],produces=
[application/json]}" onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.369  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/trace
|| /trace.json],methods=[GET],produces=[application/json]}"
onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.370  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "
{[/configprops || /configprops.json],methods=
[GET],produces=[application/json]}" onto public
java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.371  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/info
|| /info.json],methods=[GET],produces=[application/json]}"
onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.373  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "
{[/autoconfig || /autoconfig.json],methods=[GET],produces=
[application/json]}" onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.375  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "
{[/heapdump || /heapdump.json],methods=[GET],produces=
[application/octet-stream]}" onto public void
org.springframework.boot.actuate.endpoint.mvc.HeapdumpMvcEn
dpoint.invoke(boolean,javax.servlet.http.HttpServletRequest
,javax.servlet.http.HttpServletResponse) throws
```

```
java.io.IOException,javax.servlet.ServletException
2017-01-28 17:37:35.378  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/beans
|| /beans.json],methods=[GET],produces=[application/json]}"
onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.384  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "
{[/env/{name:.*}],methods=[GET],produces=
[application/json]}" onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EnvironmentMv
cEndpoint.value(java.lang.String)
2017-01-28 17:37:35.384  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "{[/env
|| /env.json],methods=[GET],produces=[application/json]}"
onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.385  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "
{[/health || /health.json],produces=[application/json]}"
onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.HealthMvcEndp
oint.invoke(java.security.Principal)
2017-01-28 17:37:35.387  INFO 4311 --- [  restartedMain]
o.s.b.a.e.mvc.EndpointHandlerMapping     : Mapped "
{[/mappings || /mappings.json],methods=[GET],produces=
[application/json]}" onto public java.lang.Object
org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAd
apter.invoke()
2017-01-28 17:37:35.653  INFO 4311 --- [  restartedMain]
o.s.b.d.a.OptionalLiveReloadServer       : LiveReload
server is running on port 35729
2017-01-28 17:37:35.832  INFO 4311 --- [  restartedMain]
o.s.j.e.a.AnnotationMBeanExporter        : Registering
beans for JMX exposure on startup
2017-01-28 17:37:35.850  INFO 4311 --- [  restartedMain]
o.s.c.support.DefaultLifecycleProcessor  : Starting beans
```

```
in phase 0
2017-01-28 17:37:36.111  INFO 4311 --- [  restartedMain]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started
on port(s): 8080 (http)
2017-01-28 17:37:36.119  INFO 4311 --- [  restartedMain]
c.i.s.StudentServicesApplication         : Started
StudentServicesApplication in 8.193 seconds (JVM running
for 10.023)
```


BEST SELLER

in 28 Minutes

105 lectures

9.5 hours video

Master Java Web Services and REST API with Sprin...

in28Minutes Official

★★★★☆ 4.4 (441)

~~₹12,800~~ ₹770

Learn Java Web Services and REST API with Spring Boot

Checkout the Course Now!

Spring Boot Auto Configuration

You will learn

- Why do we need Auto Configuration?
- What is Auto Configuration?
- A few examples of Spring Boot Auto Configuration
- How is Auto Configuration implemented in Spring Boot?
- How to debug Auto Configuration?

Why do we need Spring Boot Auto Configuration?

Spring based applications have a lot of configuration.

When we use Spring MVC, we need to configure component scan, dispatcher servlet, a view resolver, web jars(for delivering static content) among other things.

```
   <bean

class="org.springframework.web.servlet.view.InternalResourc
eViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
   </bean>

   <mvc:resources mapping="/webjars/**"
```

```
 location="/webjars/"/>
```

Below code snippet shows typical configuration of a dispatcher servlet in a web application.

```
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>

org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/todo-servlet.xml</param-
value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

When we use Hibernate/JPA, we would need to configure a datasource, an entity manager factory, a transaction manager among a host of other things.

```
    <bean id="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource"
        destroy-method="close">
        <property name="driverClass" value="${db.driver}"
/>
        <property name="jdbcUrl" value="${db.url}" />
        <property name="user" value="${db.username}" />
        <property name="password" value="${db.password}" />
```

```
 </bean>

    <jdbc:initialize-database data-source="dataSource">
        <jdbc:script location="classpath:config/schema.sql"
/>
        <jdbc:script location="classpath:config/data.sql"
/>
    </jdbc:initialize-database>

    <bean

class="org.springframework.orm.jpa.LocalContainerEntityMana
gerFactoryBean"
        id="entityManagerFactory">
        <property name="persistenceUnitName"
value="hsql_pu" />
        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory"
ref="entityManagerFactory" />
        <property name="dataSource" ref="dataSource" />
    </bean>

    <tx:annotation-driven transaction-
manager="transactionManager"/>
```

Above examples are typical with any Spring framework implementation or integration with other frameworks.

Spring Boot : Can we think different?

Spring Boot brings in new thought process around this.

Can we bring more intelligence into this? When a spring mvc jar is added into an application, can we auto configure some beans automatically?

- How about auto configuring a Data Source if Hibernate jar is on the classpath?
- How about auto configuring a Dispatcher Servlet if Spring MVC jar is on the classpath?

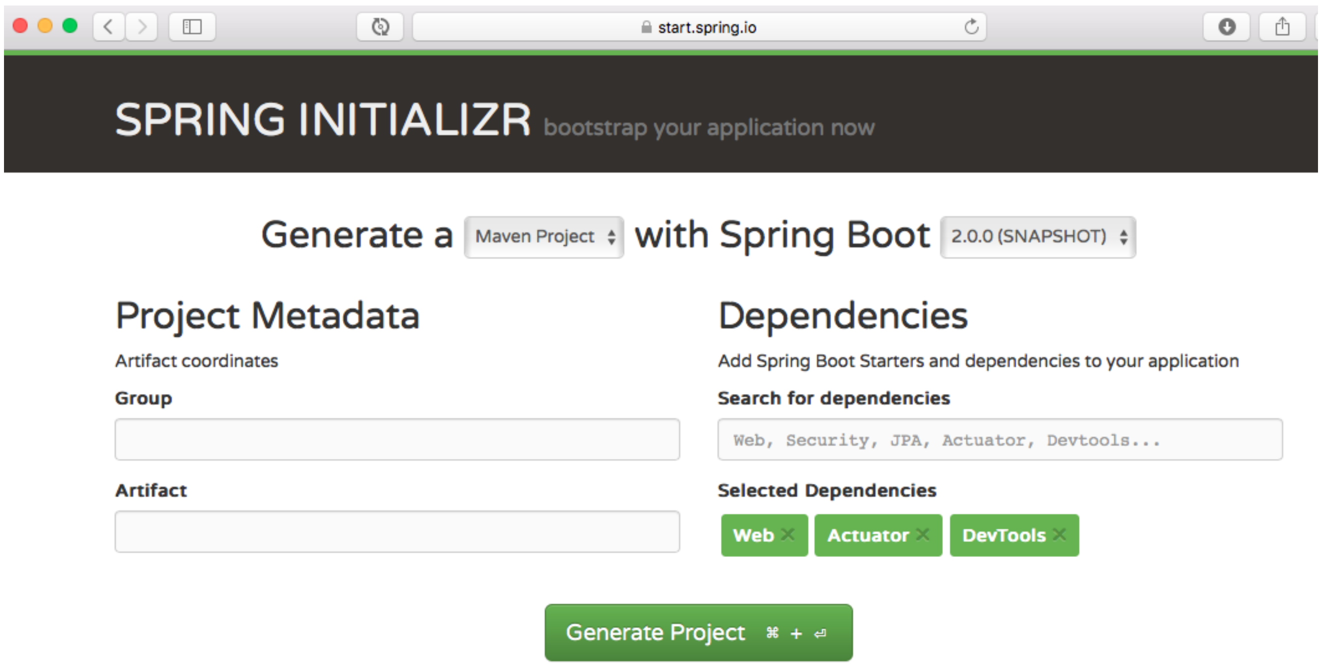
There would be provisions to override the default auto configuration.

Spring Boot looks at a) Frameworks available on the CLASSPATH b) Existing configuration for the application. Based on these, Spring Boot provides basic configuration needed to configure the application with these frameworks. This is called `Auto Configuration`.

To understand Auto Configuration further, lets bootstrap a simple Spring Boot Application using Spring Initializr.

Creating REST Services Application with Spring Initializr

Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.


start.spring.io
SPRING INITIALIZR bootstrap your application now
Generate a Maven Project with Spring Boot 2.0.0 (SNAPSHOT)
Project Metadata
Artifact coordinates
Group
Artifact
Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies
Web
Actuator
DevTools
Generate Project ⌘ + ⏎

As shown in the image above, following steps have to be done.

- Launch Spring Initializr and choose the following
  - Choose `com.in28minutes.springboot` as Group
  - Choose `student-services` as Artifact
  - Choose following dependencies
    - Web
    - Actuator
    - DevTools

- Click Generate Project.
- Import the project into Eclipse.
- If you want to understand all the files that are part of this project, you can go here.

Spring Boot Auto Configuration in action.

When we run StudentServicesApplication.java as a Java Application, you will see a few important things in the log.

```
Mapping servlet: 'dispatcherServlet' to [/]

Mapped "{[/error]}" onto public
org.springframework.http.ResponseEntity<java.util.Map<java.
lang.String, java.lang.Object>>
```

```
 org.springframework.boot.autoconfigure.web.BasicErrorContr
oller.error(javax.servlet.http.HttpServletRequest)

Mapped URL path [/webjars/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpReques
tHandler]
```

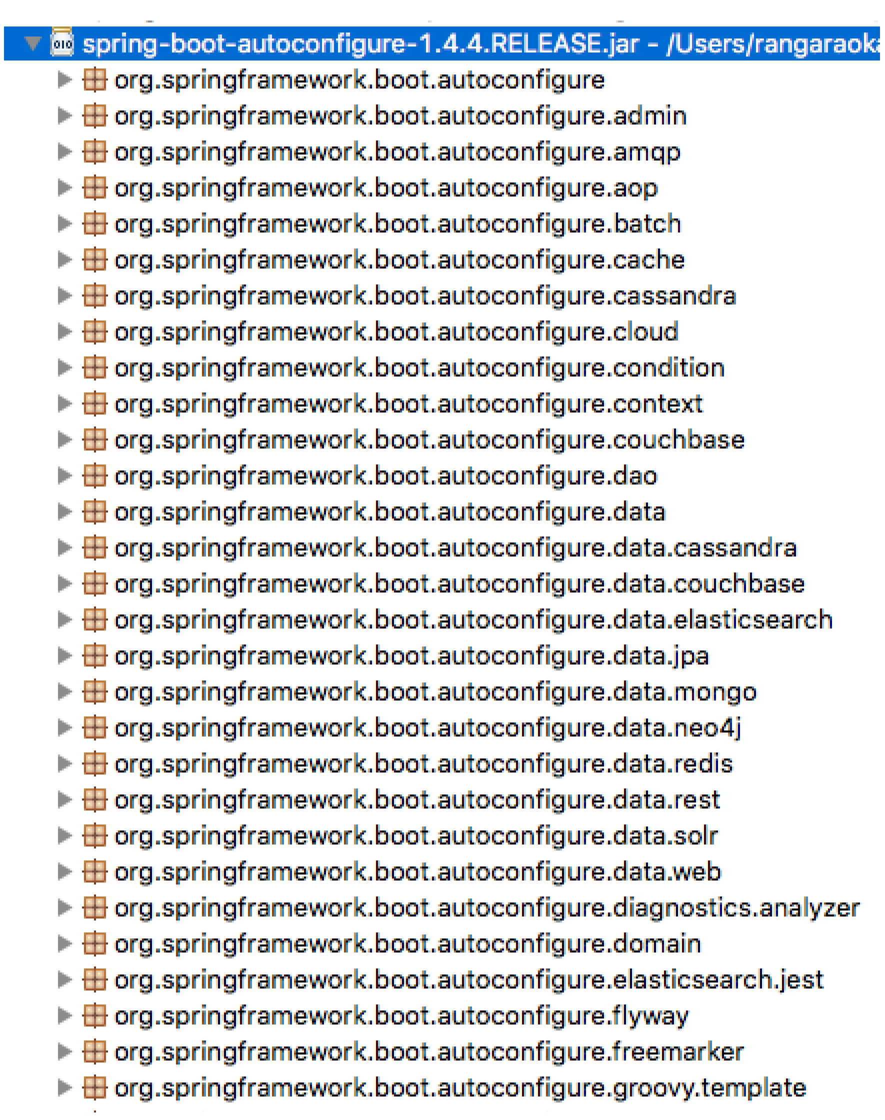
Above log statements are good examples of `Spring Boot Auto Configuration` in action.

As soon as we added in Spring Boot Starter Web as a dependency in our project, Spring Boot Autoconfiguration sees that Spring MVC is on the classpath. It autoconfigures dispatcherServlet, a default error page and webjars.

If you add Spring Boot Data JPA Starter, you will see that Spring Boot Auto Configuration auto configures a datasource and an Entity Manager.

Where is Spring Boot Auto Configuration implemented?

All auto configuration logic is implemented in `spring-boot-autoconfigure.jar`. All auto configuration logic for mvc, data, jms and other frameworks is present in a single jar.


spring-boot-autoconfigure-1.4.4.RELEASE.jar - /Users/rangaraok

```
org.springframework.boot.autoconfigure
org.springframework.boot.autoconfigure.admin
org.springframework.boot.autoconfigure.amqp
org.springframework.boot.autoconfigure.aop
org.springframework.boot.autoconfigure.batch
org.springframework.boot.autoconfigure.cache
org.springframework.boot.autoconfigure.cassandra
org.springframework.boot.autoconfigure.cloud
org.springframework.boot.autoconfigure.condition
org.springframework.boot.autoconfigure.context
org.springframework.boot.autoconfigure.couchbase
org.springframework.boot.autoconfigure.dao
org.springframework.boot.autoconfigure.data
org.springframework.boot.autoconfigure.data.cassandra
org.springframework.boot.autoconfigure.data.couchbase
org.springframework.boot.autoconfigure.data.elasticsearch
org.springframework.boot.autoconfigure.data.jpa
org.springframework.boot.autoconfigure.data.mongo
org.springframework.boot.autoconfigure.data.neo4j
org.springframework.boot.autoconfigure.data.redis
org.springframework.boot.autoconfigure.data.rest
org.springframework.boot.autoconfigure.data.solr
org.springframework.boot.autoconfigure.data.web
org.springframework.boot.autoconfigure.diagnostics.analyzer
org.springframework.boot.autoconfigure.domain
org.springframework.boot.autoconfigure.elasticsearch.jest
org.springframework.boot.autoconfigure.flyway
org.springframework.boot.autoconfigure.freemarker
org.springframework.boot.autoconfigure.groovy.template
```

Other important file inside spring-boot-autoconfigure.jar is /META-INF/spring.factories. This file lists all the auto configuration classes that should be enabled under the EnableAutoConfiguration key. A few of the important auto configurations are listed below.

```
org.springframework.boot.autoconfigure.EnableAutoConfigurat
ion=\
org.springframework.boot.autoconfigure.aop.AopAutoConfigura
tion,\
org.springframework.boot.autoconfigure.MessageSourceAutoCon
figuration,\
org.springframework.boot.autoconfigure.PropertyPlaceholderA
utoConfiguration,\
org.springframework.boot.autoconfigure.jackson.JacksonAutoC
onfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceAutoC
onfiguration,\
org.springframework.boot.autoconfigure.jdbc.JdbcTemplateAut
oConfiguration,\
org.springframework.boot.autoconfigure.jdbc.JndiDataSourceA
utoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.XADataSourceAut
oConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceTrans
actionManagerAutoConfiguration,\
org.springframework.boot.autoconfigure.security.SecurityAut
oConfiguration,\
org.springframework.boot.autoconfigure.security.SecurityFil
terAutoConfiguration,\
org.springframework.boot.autoconfigure.web.DispatcherServle
tAutoConfiguration,\
org.springframework.boot.autoconfigure.web.EmbeddedServletC
ontainerAutoConfiguration,\
org.springframework.boot.autoconfigure.web.ErrorMvcAutoConf
iguration,\
```

Example Auto Configuration

We will take a look at DataSourceAutoConfiguration.

Typically all Auto Configuration classes look at other classes available in the classpath. If specific classes are available in the classpath, then configuration for that functionality is enabled through auto configuration. Annotations like @ConditionalOnClass, @ConditionalOnMissingBean help in providing these features!

`@ConditionalOnClass({ DataSource.class, EmbeddedDatabaseType.class })` : This configuration is enabled only when these classes are available in the classpath.

```
@Configuration
@ConditionalOnClass({ DataSource.class,
EmbeddedDatabaseType.class })
```

```
@EnableConfigurationProperties(DataSourceProperties.class)
@Import({ Registrar.class,
DataSourcePoolMetadataProvidersConfiguration.class })
public class DataSourceAutoConfiguration {
```

`@ConditionalOnMissingBean` : This bean is configured only if there is no other bean configured with the same name.

```
@Bean
@ConditionalOnMissingBean
public DataSourceInitializer dataSourceInitializer() {
        return new DataSourceInitializer();
}
```

Embedded Database is configured only if there are no beans of type DataSource.class or XADataSource.class already configured.

```
@Conditional(EmbeddedDatabaseCondition.class)
@ConditionalOnMissingBean({ DataSource.class,
XADataSource.class })
@Import(EmbeddedDataSourceConfiguration.class)
protected static class EmbeddedDatabaseConfiguration {
}
```

Debugging Auto Configuration

There are two ways you can debug and find more information about auto configuration.

- Turning on debug logging
- Using Spring Boot Actuator

Debug Logging

You can turn debug logging by adding a simple property value to application.properties. In the example below, we are turning on Debug level for all logging from org.springframework package (and sub packages).

```
logging.level.org.springframework: DEBUG
```

When you restart the application, you would see an auto configuration report printed in the log. Similar to what you see below, a report is produced including all the auto configuration classes. The report separates the positive matches from negative matches. It will show why a specific bean is auto configured and also why something is not auto configured.

```
=========================
AUTO-CONFIGURATION REPORT
=========================

Positive matches:
-----------------
DispatcherServletAutoConfiguration matched
 - @ConditionalOnClass classes found:
org.springframework.web.servlet.DispatcherServlet
(OnClassCondition)
 - found web application StandardServletEnvironment
(OnWebApplicationCondition)


Negative matches:
-----------------
ActiveMQAutoConfiguration did not match
 - required @ConditionalOnClass classes not found:
javax.jms.ConnectionFactory,org.apache.activemq.ActiveMQCon
nectionFactory (OnClassCondition)

AopAutoConfiguration.CglibAutoProxyConfiguration did not
match
 - @ConditionalOnProperty missing required properties
spring.aop.proxy-target-class (OnPropertyCondition)
```

Spring Boot Actuator

Other way to debug auto configuration is to add spring boot actuator to your project. We will also add in HAL Browser to make things easy.

```
<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>
```

HAL Browser auto configuration http://localhost:8080/actuator/#http://localhost:8080/autoconfig would show the details of all the beans which are auto configured and those which are not.

```
  "negativeMatches": {
    "CacheStatisticsAutoConfiguration": [
      {
        "condition": "OnBeanCondition",
        "message": "@ConditionalOnBean (types: org.springframework.cache.C
acheManager; SearchStrategy: all) found no beans"
      }
    ],

"CacheStatisticsAutoConfiguration.CaffeineCacheStatisticsProviderConfigura
tion": [
      {
        "condition": "OnClassCondition",
        "message": "required @ConditionalOnClass classes not found: com.gi
thub.benmanes.caffeine.cache.Caffeine,org.springframework.cache.caffeine.C
affeineCacheManager"
      },
      {
        "condition": "ConditionEvaluationReport.AncestorsMatchedCondition"
,
        "message": "Ancestor 'org.springframework.boot.actuate.autoconfigu
re.CacheStatisticsAutoConfiguration' did not match"
      }
    ],
{
  "positiveMatches": {
    "AuditAutoConfiguration#auditListener": [
      {
        "condition": "OnBeanCondition",
        "message": "@ConditionalOnMissingBean (types: org.springframework.
boot.actuate.audit.listener.AbstractAuditListener; SearchStrategy: all) fo
und no beans"
      }
    ],
    "AuditAutoConfiguration#authenticationAuditListener": [
      {
        "condition": "OnClassCondition",
        "message": "@ConditionalOnClass classes found: org.springframework
.security.authentication.event.AbstractAuthenticationEvent"
      },
      {
        "condition": "OnBeanCondition",
        "message": "@ConditionalOnMissingBean (types: org.springframework.
boot.actuate.security.AbstractAuthenticationAuditListener; SearchStrategy:
all) found no beans"
      }
    ],
    "AuditAutoConfiguration#authorizationAuditListener": [
      {
        "condition": "OnClassCondition",
        "message": "@ConditionalOnClass classes found: org.springframework
.security.access.event.AbstractAuthorizationEvent"
      },
      {
```

Spring Boot Starters - Web and JPA

You will learn

- What features are provided by Spring Boot Starter Projects?
- We will look at an example of Starter Projects
- We will look at Spring Boot Starter Web
- Get an overview of different starter projects provided by Spring Boot.

Why do we need Spring Boot Starter Projects?

To understand what starter projects provide, let's consider an example project without using a starter.

What if we do not have starter projects?

Let's say we want to develop a web application with Spring MVC.

First of all we would need to identify the frameworks we want to use, which versions of frameworks to use and how to connect them together.

Listed below are some of the dependencies we use in our Spring MVC Course. These include Spring MVC, Jackson Databind (for data binding), Hibernate-Validator (for server side validation using Java Validation API) and Log4j (for logging). When creating this course, we had to choose the compatible versions of all these frameworks.

```
<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>
```

We had to add configuration to get all the stuff wired together. Configuration for dispatcher servlet, view resolver, error page, web jars among other configuration.

```
<bean

class="org.springframework.web.servlet.view.InternalResourc
eViewResolver">
    <property name="prefix">
        <value>/WEB-INF/views/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>

<bean id="messageSource"
```

```
 class="org.springframework.context.support.ReloadableResou
rceBundleMessageSource">
    <property name="basename" value="classpath:messages" />
    <property name="defaultEncoding" value="UTF-8" />
</bean>

<mvc:resources mapping="/webjars/**" location="/webjars/"
/>

<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/todo-servlet.xml</param-
value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

When using JPA, we would need to similar stuff. We need to add the jars and the configuration for datasource, entity manager, transaction manager etc.

```
<bean id="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">
    <property name="driverClass" value="${db.driver}" />
    <property name="jdbcUrl" value="${db.url}" />
    <property name="user" value="${db.username}"
```

```
 />
    <property name="password" value="${db.password}" />
</bean>

<jdbc:initialize-database data-source="dataSource">
    <jdbc:script location="classpath:config/schema.sql" />
    <jdbc:script location="classpath:config/data.sql" />
</jdbc:initialize-database>

<bean

class="org.springframework.orm.jpa.LocalContainerEntityMana
gerFactoryBean"
    id="entityManagerFactory">
    <property name="persistenceUnitName" value="hsql_pu" />
    <property name="dataSource" ref="dataSource" />
</bean>

<bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory"
ref="entityManagerFactory" />
    <property name="dataSource" ref="dataSource" />
</bean>

<tx:annotation-driven transaction-
manager="transactionManager"/>
```

Spring Boot Starter Projects

Here’s what the Spring Boot documentations says about starters.

> *Starters 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, without having to hunt through sample*

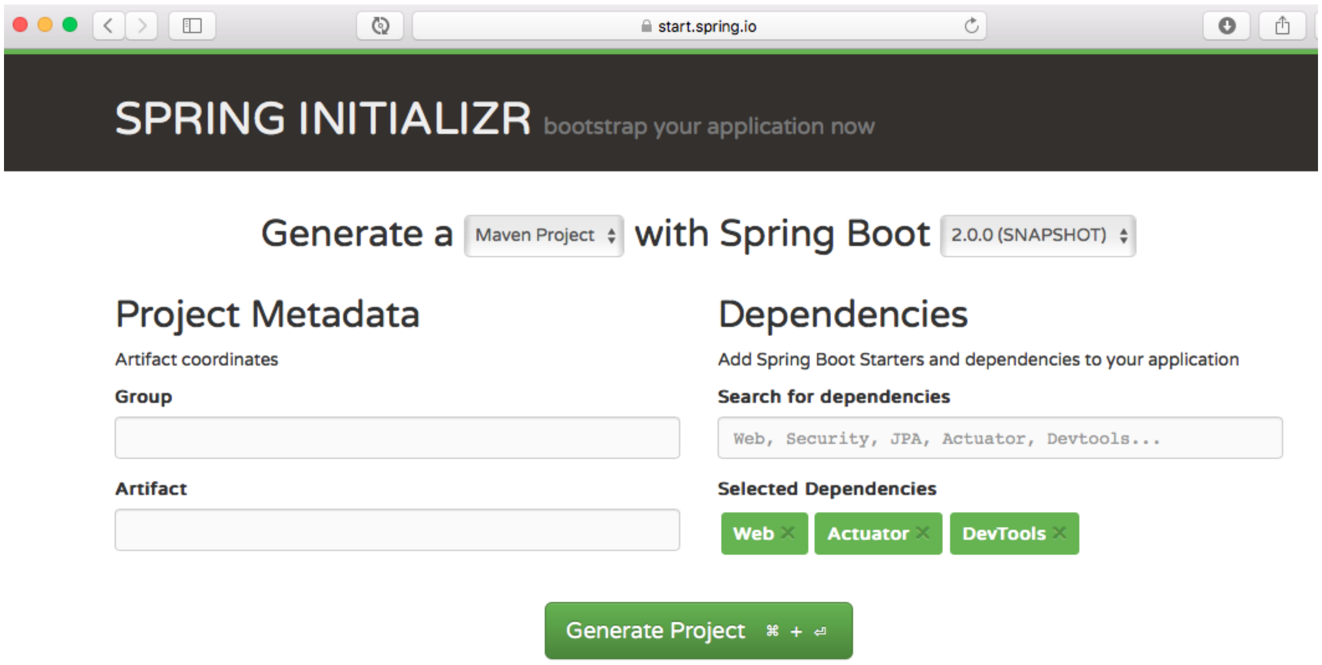
code and copy paste loads of dependency descriptors. For example, if you want to get started using Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go.

Let's consider an example starter - Spring Boot Starter Web.

If you want to develop a web application or an application to expose restful services, Spring Boot Start Web is the starter to pick. Lets create a quick project with Spring Boot Starter Web using Spring Initializr.

Creating REST Services Application with Spring Initializr

Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.


start.spring.io
SPRING INITIALIZR bootstrap your application now
Generate a Maven Project with Spring Boot 2.0.0 (SNAPSHOT)
Project Metadata
Artifact coordinates
Group
Artifact
Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies
Web
Actuator
DevTools
Generate Project ⌘ + ⏎

As shown in the image above, following steps have to be done.

- Launch Spring Initializr and choose the following
  - Choose `com.in28minutes.springboot` as Group

- - Choose `student-services` as Artifact
  - Choose following dependencies
    - Web

- Click Generate Project.
- Import the project into Eclipse.
- If you want to understand all the files that are part of this project, you can go here.

Spring Boot Starter Web

Spring Boot Starter Web brings in 2 important features

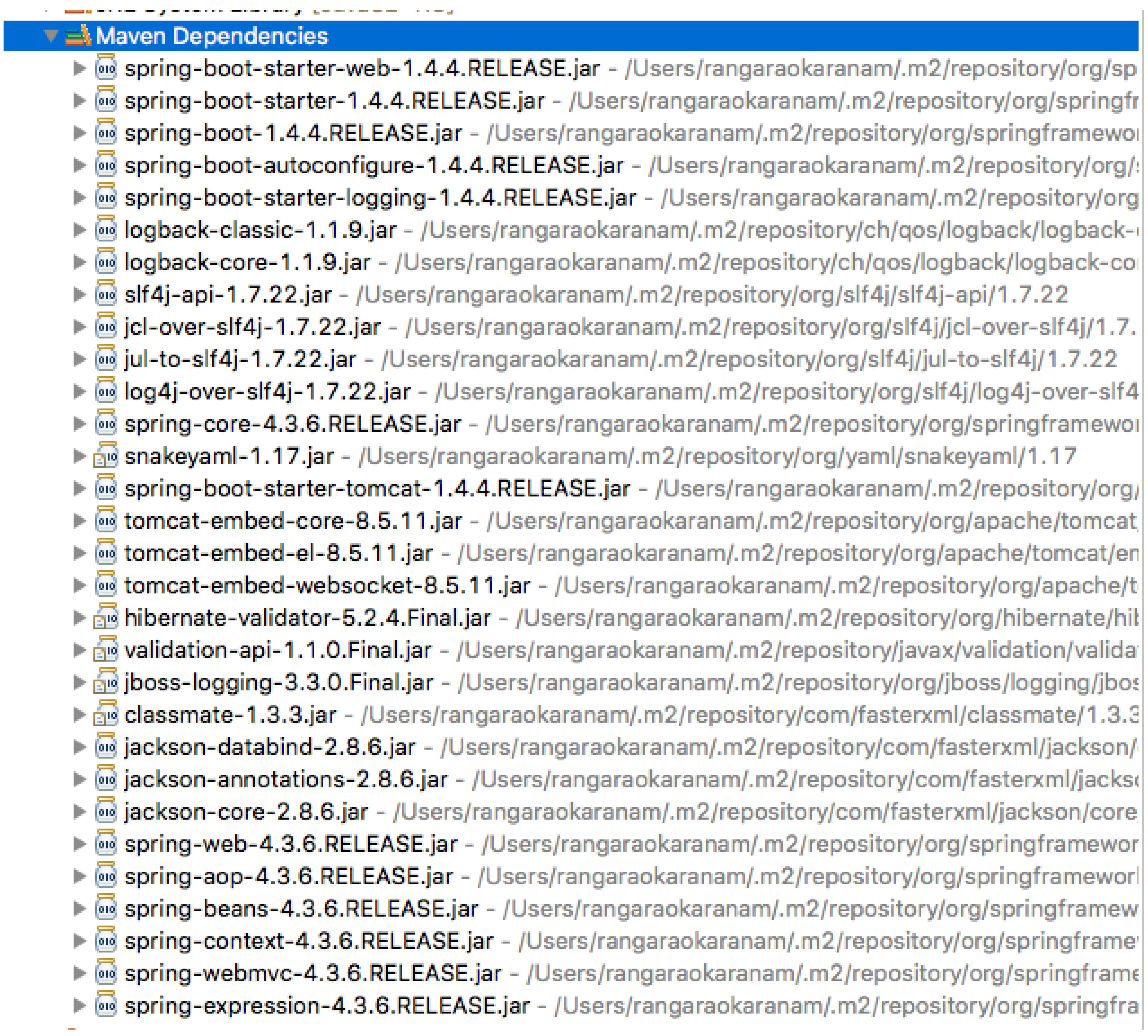
- Compatible Dependencies that are needed to develop web applications
- Auto Configuration

Dependency for Spring Boot Starter Web

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

Dependencies

Following screenshot shows the different dependencies that are added in to our application



```
▼ Maven Dependencies
  ▶ spring-boot-starter-web-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/sp
  ▶ spring-boot-starter-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springfr
  ▶ spring-boot-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewo
  ▶ spring-boot-autoconfigure-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/
  ▶ spring-boot-starter-logging-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org
  ▶ logback-classic-1.1.9.jar - /Users/rangaraokaranam/.m2/repository/ch/qos/logback/logback-
  ▶ logback-core-1.1.9.jar - /Users/rangaraokaranam/.m2/repository/ch/qos/logback/logback-co
  ▶ slf4j-api-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/slf4j-api/1.7.22
  ▶ jcl-over-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/jcl-over-slf4j/1.7.
  ▶ jul-to-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/jul-to-slf4j/1.7.22
  ▶ log4j-over-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/log4j-over-slf4
  ▶ spring-core-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewo
  ▶ snakeyaml-1.17.jar - /Users/rangaraokaranam/.m2/repository/org/yaml/snakeyaml/1.17
  ▶ spring-boot-starter-tomcat-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/
  ▶ tomcat-embed-core-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat
  ▶ tomcat-embed-el-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat/en
  ▶ tomcat-embed-websocket-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/t
  ▶ hibernate-validator-5.2.4.Final.jar - /Users/rangaraokaranam/.m2/repository/org/hibernate/hil
  ▶ validation-api-1.1.0.Final.jar - /Users/rangaraokaranam/.m2/repository/javax/validation/valida
  ▶ jboss-logging-3.3.0.Final.jar - /Users/rangaraokaranam/.m2/repository/org/jboss/logging/jbos
  ▶ classmate-1.3.3.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/classmate/1.3.3
  ▶ jackson-databind-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/
  ▶ jackson-annotations-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jacks
  ▶ jackson-core-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/core
  ▶ spring-web-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewor
  ▶ spring-aop-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframewor
  ▶ spring-beans-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframew
  ▶ spring-context-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframe
  ▶ spring-webmvc-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframe
  ▶ spring-expression-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springfra
```

Dependencies can be classified into:

- Spring - core, beans, context, aop
- Web MVC - (Spring MVC)
- Jackson - for JSON Binding
- Validation - Hibernate Validator, Validation API
- Embedded Servlet Container - Tomcat
- Logging - logback, slf4j

Any typical web application would use all these dependencies. Spring Boot Starter Web comes pre packaged with these. As a developer, I would not need to worry about either these dependencies or their compatible versions.

Auto Configuration

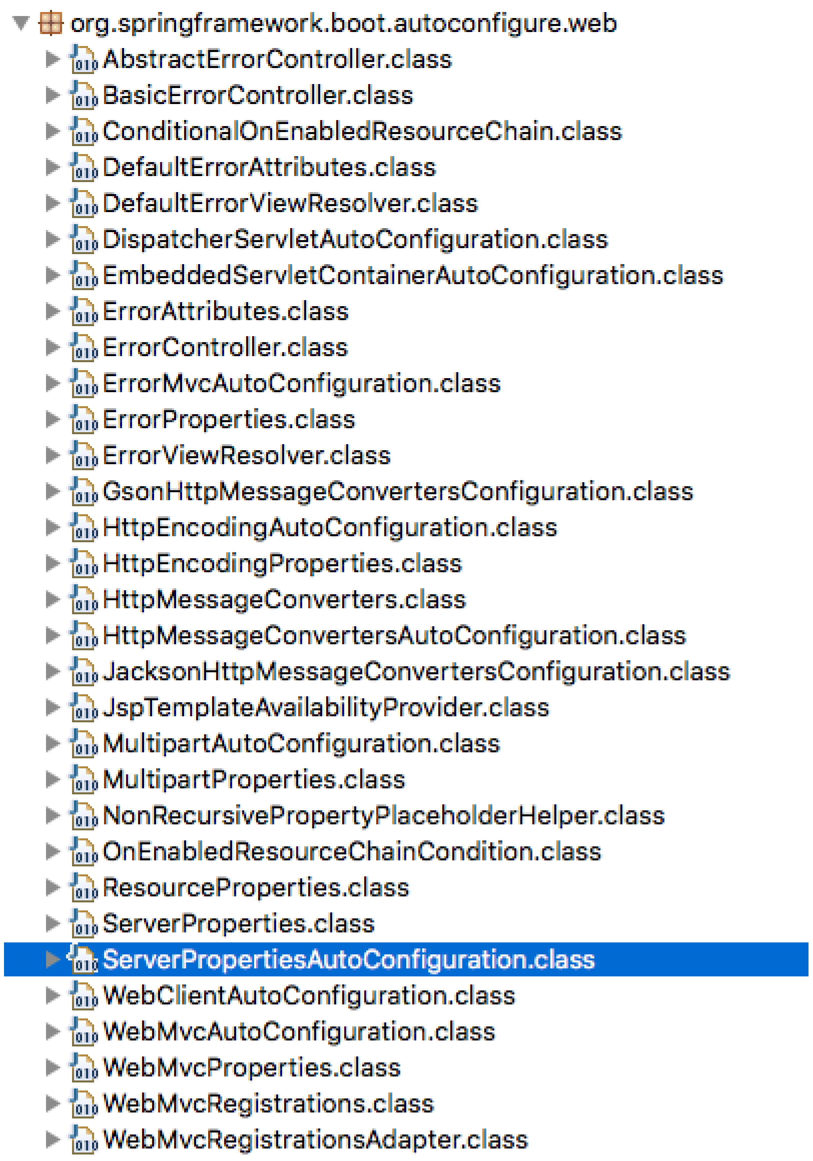
Spring Boot Starter Web auto configures the basic things that are needed. To understand the features Spring Boot Starter Web brings in, lets run StudentServicesApplication.java as a Java Application and review the log.

```
Mapping servlet: 'dispatcherServlet' to [/]

Mapped "{[/error]}" onto public
org.springframework.http.ResponseEntity<java.util.Map<java.
lang.String, java.lang.Object>>
org.springframework.boot.autoconfigure.web.BasicErrorContro
ller.error(javax.servlet.http.HttpServletRequest)

Mapped URL path [/webjars/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpReques
tHandler]
```

Spring Boot Starter Web auto-configures

- Dispatcher Servlet
- Error Page
- Web Jars to manage your static dependencies
- Embedded Servlet Container - Tomcat is the default

The image below shows the different things that might be auto configured by Spring Boot Starter Web



```
org.springframework.boot.autoconfigure.web
  AbstractErrorController.class
  BasicErrorController.class
  ConditionalOnEnabledResourceChain.class
  DefaultErrorAttributes.class
  DefaultErrorViewResolver.class
  DispatcherServletAutoConfiguration.class
  EmbeddedServletContainerAutoConfiguration.class
  ErrorAttributes.class
  ErrorController.class
  ErrorMvcAutoConfiguration.class
  ErrorProperties.class
  ErrorViewResolver.class
  GsonHttpMessageConvertersConfiguration.class
  HttpEncodingAutoConfiguration.class
  HttpEncodingProperties.class
  HttpMessageConverters.class
  HttpMessageConvertersAutoConfiguration.class
  JacksonHttpMessageConvertersConfiguration.class
  JspTemplateAvailabilityProvider.class
  MultipartAutoConfiguration.class
  MultipartProperties.class
  NonRecursivePropertyPlaceholderHelper.class
  OnEnabledResourceChainCondition.class
  ResourceProperties.class
  ServerProperties.class
  ServerPropertiesAutoConfiguration.class
  WebClientAutoConfiguration.class
  WebMvcAutoConfiguration.class
  WebMvcProperties.class
  WebMvcRegistrations.class
  WebMvcRegistrationsAdapter.class
```

Spring Boot Starter Project Options

As we see from Spring Boot Starter Web, starter projects help us in quickly getting started with developing specific types of applications.

- spring-boot-starter-web-services - SOAP Web Services
- spring-boot-starter-web - Web & RESTful applications
- spring-boot-starter-test - Unit testing and Integration Testing
- spring-boot-starter-jdbc - Traditional JDBC
- spring-boot-starter-hateoas - Add HATEOAS features to your services
- spring-boot-starter-security - Authentication and Authorization using Spring Security
- spring-boot-starter-data-jpa - Spring Data JPA with Hibernate
- spring-boot-starter-cache - Enabling Spring Framework's caching support
- spring-boot-starter-data-rest - Expose Simple REST Services using Spring Data

- REST

There are a few starters for technical stuff as well

- spring-boot-starter-actuator - To use advanced features like monitoring & tracing to your application out of the box
- spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat - To pick your specific choice of Embedded Servlet Container
- spring-boot-starter-logging - For Logging using logback
- spring-boot-starter-log4j2 - Logging using Log4j2

Spring Boot Starter Parent

You will learn

- What is Spring Boot Starter Parent?
- What are the important features of Spring Boot Starter Parent?
- When do you use Spring Boot Starter Parent?

What is Spring Boot Starter Parent?

All Spring Boot projects typically use spring-boot-starter-parent as the parent in pom.xml.

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.4.0.RELEASE</version>
    </parent>
```

Parent Poms allow you to manage the following things for multiple child projects and modules:

- Configuration - Java Version and Other Properties
- Depedency Management - Version of dependencies
- Default Plugin Configuration

What is inside Spring Boot Starter Parent?

First of all - Spring Boot Starter Parent defines spring-boot-dependencies as the parent pom. It inherits dependency management from spring-boot-dependencies. Details in the next section.

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>1.4.0.RELEASE</version>
        <relativePath>../../spring-boot-
dependencies</relativePath>
</parent>
```

Default java version is 1.6. A project can override this by specifying a property `<java.version>1.8</java.version>` in the project pom. A few other settings related to encoding and source, target version are also set in the parent pom.

```
<java.version>1.6</java.version>
<resource.delimiter>@</resource.delimiter> <!-- delimiter
that doesn't clash with Spring ${} placeholders -->
<project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
<maven.compiler.source>${java.version}
</maven.compiler.source>
<maven.compiler.target>${java.version}
</maven.compiler.target>
```

Spring Boot Starter Parent specifies the default configuration for a host of plugins including maven-failsafe-plugin, maven-jar-plugin and maven-surefire-plugin.

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-failsafe-plugin</artifactId>
        <executions>
                <execution>
                        <goals>
        <goal>integration-test</goal>
                                <goal>verify</goal>
                        </goals>
                </execution>
        </executions>
</plugin>
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <configuration>
                <archive>
                        <manifest>
                                <mainClass>${start-class}
</mainClass>

<addDefaultImplementationEntries>true</addDefaultImplementa
tionEntries>
                        </manifest>
                </archive>
        </configuration>
</plugin>
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <configuration>
                <includes>
                        <include>**/*Tests.java</include>
                        <include>**/*Test.java</include>
                </includes>
                <excludes>

<exclude>**/Abstract*.java</exclude>
                </excludes>
        </configuration>
</plugin>
```

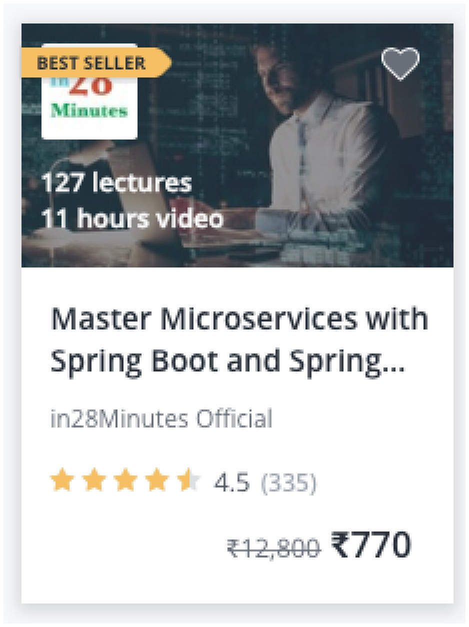
What does Spring Boot Starter Parent inherit from spring-boot-dependencies?

Spring Boot Dependencies defines the default dependency management for all Spring Boot projects. If we would want to use a new version of a specific dependency, we can override the version by specifying a new property in the project pom. The extract below shows some of the important dependencies that are managed by Spring Boot Dependencies parent pom. Since Spring Boot Starter Parent inherit from spring-boot-dependencies, it shares all these characteristics as well.

```
<properties>
        <activemq.version>5.13.4</activemq.version>
        ...
        <ehcache.version>2.10.2.2.21</ehcache.version>
        <ehcache3.version>3.1.1</ehcache3.version>
        ...
        <h2.version>1.4.192</h2.version>
        <hamcrest.version>1.3</hamcrest.version>
        <hazelcast.version>3.6.4</hazelcast.version>
        <hibernate.version>5.0.9.Final</hibernate.version>
        <hibernate-
validator.version>5.2.4.Final</hibernate-validator.version>
        <hikaricp.version>2.4.7</hikaricp.version>
        <hikaricp-java6.version>2.3.13</hikaricp-
java6.version>
        <hornetq.version>2.4.7.Final</hornetq.version>
        <hsqldb.version>2.3.3</hsqldb.version>
        <htmlunit.version>2.21</htmlunit.version>
        <httpasyncclient.version>4.1.2</httpasyncclient.ver
sion>
        <httpclient.version>4.5.2</httpclient.version>
        <httpcore.version>4.4.5</httpcore.version>
        <infinispan.version>8.2.2.Final</infinispan.version
>
        <jackson.version>2.8.1</jackson.version>
        ....
        <jersey.version>2.23.1</jersey.version>
        <jest.version>2.0.3</jest.version>
        <jetty.version>9.3.11.v20160721</jetty.version>
        <jetty-jsp.version>2.2.0.v201112011158</jetty-
jsp.version>
        <spring-security.version>4.1.1.RELEASE</spring-
security.version>
        <tomcat.version>8.5.4</tomcat.version>
        <undertow.version>1.3.23.Final</undertow.version>
        <velocity.version>1.7</velocity.version>
        <velocity-tools.version>2.0</velocity-
tools.version>
        <webjars-hal-browser.version>9f96c74</webjars-hal-
browser.version>
        <webjars-locator.version>0.32</webjars-
locator.version>
        <wsdl4j.version>1.6.3</wsdl4j.version>
        <xml-apis.version>1.4.01</xml-apis.version>
</properties>
```

Defines Maven 3.2.1 as the minimum version needed.

```
<prerequisites>
        <maven>3.2.1</maven>
</prerequisites>
```



Master Microservices with Spring Boot and Spring...

in28Minutes Official

4.5 (335)

~~₹12,800~~ ₹770

Master Microservices with Spring Boot and Spring Cloud

Checkout the Course Now!

Spring Boot Developer Tools and Live Reload

When we develop web applications with Java, we had to restart the server to pick up all changes. This kills productivity. Spring Boot Developers Tools provides solutions to automatically pick up changes without a complete server restart. Lets get productive with Spring Boot Developer Tools.

You will learn

- How to use Spring Boot Developer Tools?
- What kind of changes does Spring Boot Developer Tools pick up automatically?
- How can you use Live Reload to be more productive?

Problem with Server Restarts

When we develop our applications (Web or RESTful API), we would want to be able to test our changes quickly.

Typically, in the Java world, we need to restart the server to pick up the changes.

> *There are plugins like JRebel which help, but you need shell out $$$ for it.*

Restarting server takes about 1 to 5 minutes based on the size of the application. A typical developer does 30 - 40 restarts in a day. I leave it as an exercise to determine how much time a developer can save if the changes are automatically picked up as soon as I make a code change.

Thats where Spring Boot Developer Tools comes into picture.

Adding Spring Boot Developer Tools to Your Project

Adding Spring Boot Developer Tools to your project is very simple.

Add this dependency to your Spring Boot Project pom.xml

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

Restart the application.

You are all Set.

Go ahead and make a simple change to your controller. You would see that these changes are automatically picked up.

What kind of changes does Spring Boot Developer Tools pick up?

> *By default, any entry on the classpath that points to a folder will be monitored for changes.*

Here are few important things to note:

These folders will not trigger reload by default

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

You can configure additional folders to scan.

application.properties

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

You can also configure folders to exclude.

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

Auto refresh your browser with LiveReload

Spring Boot Developer Tools auto loads the changes to application. But if you are developing a web application, you would need to refresh the browser to pickup the change.

> *LiveReload aims to solve this problem*

LiveReload offers extensions for browsers

- Download from http://livereload.com/extensions/

Once you install the LiveReload plugin for your browser, you would see that the page auto refreshes when you make a change in application code.

> *LiveReload is Technology in progress!! So, expect a few problems!*

Logging with Spring Boot - Logback, SLF4j and LOG4j2

Default Logging Framework with Spring Boot

Spring boot provides a default starter for logging - `spring-boot-starter-logging`. It is included by default in `spring-boot-starter` which is included in all other starters.

> *This means whenever you use any starters like `spring-boot-starter-web` or `spring-boot-starter-data-jpa`, you get logging for free!*

Let's look at what is present in the Logging Starter.

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.3</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-to-slf4j</artifactId>
  <version>2.9.1</version>
  <scope>compile</scope>
</dependency> <dependency>
```

```
 <groupId>org.slf4j</groupId>
  <artifactId>jul-to-slf4j</artifactId>
  <version>1.7.25</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>log4j-over-slf4j</artifactId>
  <version>1.7.25</version>
  <scope>compile</scope>
</dependency>
```

As you can see the default logging framework is Logback with SLF4j as implementation.

By default, all logging goes to console.

Configure Logging Levels

In application.properties, we can use the "logging.level" prefix to set logging levels.

```
logging.level.some.package.path=DEBUG
logging.level.some.other.package.path=ERROR
```

Root logging level can be configured as shown below

```
logging.level.root=WARN
```

Configuring a Log File

You can configure a log file by using logging.file property in application.properties. The logging here would be in addition to the logging in console.

```
logging.file=\path_to\logfile.log
```

Custom configuration using logback.xml

Spring Boot will pick up all custom configuration using logback.xml as long as it is in the application class path.

Example code

```
LOGGER.trace("Your log - {}", value);
LOGGER.debug("debug - {}", value);
LOGGER.info("info- {}", value);
LOGGER.warn("warn - {}", value);
LOGGER.error("error - {}", value);
```

Using Log4j2 for logging with Spring Boot

We would need to exclude the dependency on `spring-boot-starter-logging` and add a dependency on `spring-boot-starter-log4j2` .

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

Let’s take a quick look at the dependencies in log4j2 starter.

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
```

```
 <artifactId>log4j-slf4j-impl</artifactId>
  <version>2.9.1</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.9.1</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.9.1</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>jul-to-slf4j</artifactId>
  <version>1.7.25</version>
  <scope>compile</scope>
</dependency>
```

Custom configuration using log4j2.xml

Spring Boot will pick up all custom configuration using log4j2.xml as long as it is in the application class path.

You also have the option of using YAML or JSON with Log4j2.

- YAML - log4j2.yaml or log4j2.yml
- JSON - log4j2.json or log4j2.jsn

However, you would need to include the appropriate dependency to handle yaml(jackson-dataformat-yaml) or json(jackson-databind).

Introduction to Spring Data

We will look at the various options that Spring Data provides and a couple of examples - Spring Data JPA and Spring Data Mongodb.

You will learn

- Basics of Spring Data
- Why is Spring Data needed?
- What are the different interfaces provided by Spring Data?
- How to get started with Spring Data?

What is Spring Data?

Think about the evolution of databases in the last few years.

When Spring Framework was created, in early 2000s, the only kind of database was relational database - Oracle, MS SQL Server, My SQL etc. In the last few years, there are a wide variety of databases that are getting popular - most of them not relational and not using SQL. Wide variety of terminology is used to refer to these databases. NoSQL, for example.

ORM frameworks (Hibernate) and specifications(JPA) were good fit for the relational databases. But, the newer databases, have different needs.

From http://projects.spring.io/spring-data/

> *Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store. It makes it easy to use data access*

technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services.

To make it simpler, Spring Data provides Abstractions (interfaces) you can use irrespective of underlying data source.

Spring Data Commons

Spring Data Commons provides all the common abstractions that enable you to connect with different data stores.

Crud Repository

The key interface in Spring Data Commons is `CrudRepository`. It provides generic CRUD operations irrespective of the underlying data store. It extends Repository which is the base class for all the repositories providing access to data stores.

All the methods in the `CrudRepository` interface are shown below

```
public interface CrudRepository<T, ID> extends
Repository<T, ID> {
        <S extends T> S save(S entity);

        <S extends T> Iterable<S> saveAll(Iterable<S>
entities);

        Optional<T> findById(ID id);

        boolean existsById(ID id);

        Iterable<T> findAll();

        Iterable<T> findAllById(Iterable<ID> ids);

        long count();

        void deleteById(ID id);

        void delete(T entity);

        void deleteAll(Iterable<? extends T> entities);

        void deleteAll(); }
```

The methods in the CrudRepository are self explanatory.

PagingAndSortingRepository

The other important interface in Spring Data is PagingAndSortingRepository. PagingAndSortingRepository provides options to

- Sort your data using `Sort` interface
- Paginate your data using `Pageable` interface, which provides methods for pagination - getPageNumber(), getPageSize(), next(), previousOrFirst() etc.

public abstract interface PagingAndSortingRepository extends CrudRepository {

public Iterable findAll(Sort sort);

public Page findAll(Pageable pageable);

}

Defining Custom Repositories

You can create a custom repository extending any of the repository classes - Repository, PagingAndSortingRepository or CrudRepository.

An example is shown below

```
interface PersonRepository extends CrudRepository<User,
Long> {
```

Defining custom queries

Spring Data also provides the feature of query creation from interface method names.

Look at the example below:

```
  List<Person> findByFirstNameAndLastname(String firstName,
String lastname);
```

Above method helps you search a data store by passing in the first name and last name of a person. This would generate the appropriate query for the data store to return the person details.

> *You can find more details in the spring data documentation - https://docs.spring.io/spring-data/commons/docs/current/reference/html/#repositories.query-methods.query-creation*

Auditing with Spring Data

Spring Data also provides auditing capabilities through simple annotations.

```
class Student {

  @CreatedBy
  private User createdUser;

  @CreatedDate
  private DateTime createdDate;

  // … further properties omitted
}
```

There are corresponding annotations for updates as well

- LastModifiedBy
- LastModifiedDate

Spring Data Implementations

There are Spring Data Modules specific to the data store you would want to use.

- Spring Data JPA - Connect to relational databases using ORM frameworks.
- Spring Data MongoDB - Repositories for MongoDB.
- Spring Data REST - Exposes HATEOAS RESTful resources around Spring Data repositories.
- Spring Data Redis - Repositories for Redis.

Spring Data JPA

Spring Data JPA helps you to connect to relational databases using ORM frameworks.

The dependency is shown below:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
  </dependency>
<dependencies>
```

The default JPA implementation used is Hibernate.

The core interface is the JpaRepository.

```
public interface JpaRepository<T, ID>
        extends PagingAndSortingRepository<T, ID>,
                QueryByExampleExecutor<T>
```

Some of the additional methods it provides (compared to PagingAndSortingRepository) are shown below. As you can see, all these methods are specific to JPA.

```
/**
  * Saves an entity and flushes changes instantly.
 *
 * @param entity
 * @return the saved entity
 */
```

```
<S extends T> S saveAndFlush(S entity);

        /**
 * Deletes the given entities in a batch which means it
will create a single {@link Query}. Assume that we will
clear
 * the {@link javax.persistence.EntityManager} after the
call.
 *
 * @param entities
 */
void deleteInBatch(Iterable<T> entities);

/**
 * Deletes all entities in a batch call.
 */
void deleteAllInBatch();
```

Recommended Reading for Spring Data JPA
- http://www.springboottutorial.com/introduction-to-jpa-with-spring-boot-data-jpa

Spring Data REST

Spring Data REST can be used to expose HATEOAS RESTful resources around Spring Data repositories.

An example using JPA is shown below

```
@RepositoryRestResource(collectionResourceRel = "todos",
path = "todos")
public interface TodoRepository
                extends PagingAndSortingRepository<Todo,
Long> {
```

```
```

A few example REST Services are shown below:

POST

- URL : http://localhost:8080/todos
- Use Header : Content-Type:application/json

Request Content

```
{
  "user": "Jill",
  "desc": "Learn Hibernate",
  "done": false
}
```

Response Content

```
{
  "user": "Jill",
  "desc": "Learn Hibernate",
  "done": false,
  "_links": {
    "self": {
      "href": "http://localhost:8080/todos/1"
    },
    "todo": {
      "href": "http://localhost:8080/todos/1"
    }
  }
}
```

The response contains the href of the newly created resource.

GET

- URI - http://localhost:8080/todos

Response

```
{
  "_embedded" : {
    "todos" : [ {
      "user" : "Jill",
      "desc" : "Learn  Hibernate",
      "done" : false,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/todos/1"
        },
        "todo" : {
          "href" : "http://localhost:8080/todos/1"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/todos"
    },
    "profile" : {
      "href" : "http://localhost:8080/profile/todos"
    },
    "search" : {
      "href" : "http://localhost:8080/todos/search"
    }
  },
  "page" : {
    "size" : 20,
    "totalElements" : 1,
    "totalPages" : 1,
    "number" :  0
  }
}
```

GET to http://localhost:8080/todos/1

```
{
  "user" : "Jill",
  "desc" : "Learn Hibernate",
  "done" :  false,
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/todos/1"
    },
    "todo" : {
      "href" : "http://localhost:8080/todos/1"
    }
  }
}
```

Spring Data Rest also supports search using column names

- Example - http://localhost:8080/todos?user=Jill

Spring Data Rest can be extended by defining custom methods in the repositories.http://localhost:8080/todos/search/findByUser?user=Jill can be used expose specific search method defined below.

```
@RepositoryRestResource(collectionResourceRel = "todos",
path = "todos")
public interface TodoRepository
                extends PagingAndSortingRepository<Todo,
Long> {

        List<Todo> findByUser(@Param("user") String user);

}
```

Spring Data REST supports

- Spring Data JPA
- Spring Data MongoDB
- Spring Data Neo4j
- Spring Data GemFire

- Spring Data Cassandra

Recommended Reading for Spring Data Rest - http://www.springboottutorial.com/introduction-to-spring-data-rest-using-spring-boot

Spring Data MongoDB

Spring Data MongoDB provides support for using MongoDB as data store.

The key interface is MongoRepository.

```
public interface MongoRepository<T, ID> extends
PagingAndSortingRepository<T, ID>,
QueryByExampleExecutor<T>
```

Some of the important methods (provided in addition to PagingAndSortingRepository) are shown below. You can see examples of search by example.

```
        /*
         * (non-Javadoc)
         * @see
org.springframework.data.repository.query.QueryByExampleExe
cutor#findAll(org.springframework.data.domain.Example)
         */
        @Override
        <S extends T> List<S> findAll(Example<S> example);

        /*
         * (non-Javadoc)
         * @see
org.springframework.data.repository.query.QueryByExampleExe
cutor#findAll(org.springframework.data.domain.Example,
org.springframework.data.domain.Sort)
         */
        @Override
        <S extends T> List<S> findAll(Example<S> example,
Sort sort);
```

Introduction to Web Services - Restful and SOAP

You will learn

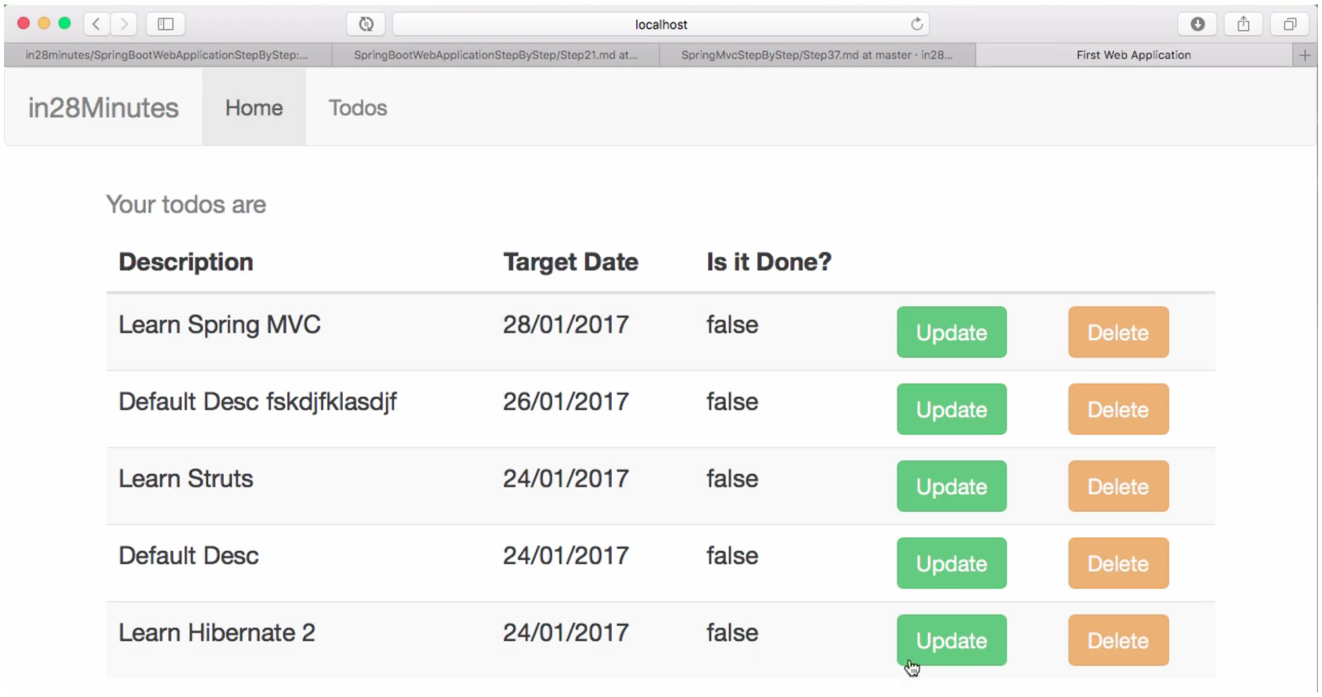
- What is a web service?
- What are the advantages of web services?
- What are the different types of web services?
- What are RESTful web services?
- What are SOAP web services?

What is a Web Service?

> *Service delivered over the web*

Is this really a complete definition. Is everything thats delivered over the web “Web Service”?

Let’s consider a web application we developed for our Spring MVC Course to manage todo’s.


localhost
in28minutes/SpringBootWebApplicationStepByStep:...
SpringBootWebApplicationStepByStep/Step21.md at...
SpringMvcStepByStep/Step37.md at master · in28...
First Web Application
in28Minutes
Home
Todos

Your todos are

Description	Target Date	Is it Done?		
Learn Spring MVC	28/01/2017	false	Update	Delete
Default Desc fskdjfklasdjf	26/01/2017	false	Update	Delete
Learn Struts	24/01/2017	false	Update	Delete
Default Desc	24/01/2017	false	Update	Delete
Learn Hibernate 2	24/01/2017	false	Update	Delete

Is this application a web service?

Nope. The answer is no. This is a web application. Not a web service.

We are back to square one.

- What is a web service?
- How is it different from a web application?

To understand this, lets consider an example.

Mark Zuckerberg likes the web application we developed to manage todo's. He thinks that our todo application is a right fit to integrate into facebook to manage todo's. Can we use the existing application to do the integration? No.

Why? Because the existing application is designed for humans - other individuals. The output of the application is html which is rendered by browser to the end user. This is not designed for other applications.

What would be the difference in my thinking if I want to design the todo application so that other applications can interact with it?

I would need to produce the output in the format that the consumers can

> *understand. Then facebook can call my web service and integrate it.*

That leads use to - W3C definition of a Web Service

> *Software system designed to support interoperable machine-to-machine interaction over a network.*

The key things to understand is

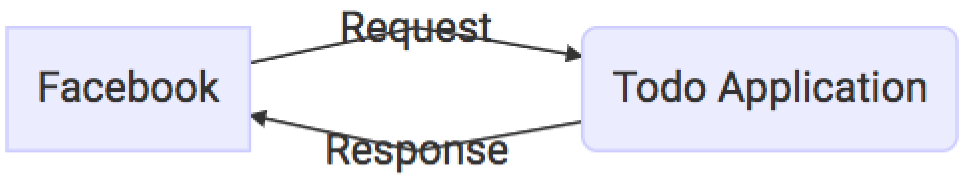
- Web services are designed for machine-to-machine (or application-to-application) interaction
- Web services should be interoperable - Not platform dependent
- Web services should allow communication over a network

Web Service Data Exchange Formats

Facebook wants to talk to Todo Application

- Facebook is built on a variety of languages - PHP is used for the front-end, Erlang is used for Chat, Java and C++ are also used.
- Todo Application is build on Java using Spring MVC

You can see that Facebook and the Todo Application use different implementation technologies. However, we would want them to talk to each other as shown in the picture below:



Both applications should be able to understand the request and response.

So, what formats should we use for the request and response?

> *They should be standard formats so that they can be used with varied kind of platforms. JSON and XML are quite popular Data Exchange formats.*

Types of Web Services

Not really types but a broad classification

- SOAP
- REST

These are not really mutually exclusive. Some SOAP services can actually be RESTful. So, the question is:

> *When does a web service become a SOAP Web service or a RESTful web service?*

SOAP

SOAP was earlier an abbreviation for Simple Object Access Protocol. In SOAP, the request and response are in XML format. However, not all types of XML are valid SOAP Requests.

SOAP defines a standard XML format. We will use WSDL (Web Service Definition Language) to define the format of request xml and the response xml.

Now lets say Facebook wants to know how to call the TODO Service? What should I give to the Facebook developer?

I will give him a WSDL of the Todo service. It will explain:

- What are the different services (operations) exposed by the server?
- How can a service (operation) be called? What url to use? (also called End Point).
- What should the structure of request xml?
- What should be the structure of response xml?

SOAP format defines a SOAP-Envelope which envelopes the entire document.

- SOAP-Header (optional) contains any information needed to identify the request. Also, part of the Header is authentication, authorization information (signatures, encrypted information etc).
- SOAP-Body contains the real xml content of request or response.
- In case of error response, server responds back with SOAP-Fault.

Isn't that cool?

REST

First of all, REST does not define a standard message exchange format. You can build REST services with both XML and JSON. However, JSON is a more popular format with REST.

So, if it does not define a standard message exchange format, what is REST then?

> *REST is a style of software architecture for distributed hypermedia systems*

REST stands for REpresentational State Transfer. The definitions for REST can be vague. So, lets understand the important concepts.

Key abstraction in REST is a Resource. There is no restriction on what can be a resource. A todo is a resource. A person on facebook is a resource.

A resource has an URI (Uniform Resource Identifier):

- /user/Ranga/todos/1
- /person/Ranga

A resource will have representations

- XML
- HTML
- JSON

A resource will have state. The representation of a resource should capture its current state.

When a resource is requested, we provide the representation of the resource.

REST and HTTP
REST builds on top of HTTP (Hypertext Transfer Protocol). HTTP is the language of the web.

HTTP has a few important verbs.

- POST - Create a new resource
- GET - Read a resource
- PUT - Update an existing resource
- DELETE - Delete a resource

HTTP also defines standard response codes.

- 200 - SUCESS
- 404 - RESOURCE NOT FOUND
- 400 - BAD REQUEST
- 201 - CREATED
- 401 - UNAUTHORIZED
- 415 - UNSUPPORTED TYPE - Representation not supported for the resource
- 500 - SERVER ERROR

Restful Service Constraints

- Client - Server : There should be a service producer and a service consumer.
- The interface (URL) is uniform and exposing resources. Interface uses nouns (not actions)
- The service is stateless. Even if the service is called 10 times, the result must be the same.
- The service result should be Cacheable. HTTP cache, for example.
- Service should assume a Layered architecture. Client should not assume direct connection to server - it might be getting info from a middle layer - cache.

Richardson Maturity Model
Richardson Maturity Model defines the maturity level of a Restful Web Service.

Following are the different levels and their characteristics.

- Level 0 : Expose SOAP web services in REST style. Expose action based services
- (http://server/getPosts, http://server/deletePosts, http://server/doThis, http://server/doThat etc) using REST.
- Level 1 : Expose Resources with proper URI's (using nouns).
  Ex: http://server/accounts, http://server/accounts/10. However, HTTP Methods are not used.
- Level 2 : Resources use proper URI's + HTTP Methods. For example, to update an account, you do a PUT to . The create an account, you do a POST to . Uri's look like posts/1/comments/5 and accounts/1/friends/1.
- Level 3 : HATEOAS (Hypermedia as the engine of application state). You will tell not only about the information being requested but also about the next possible actions that the service consumer can do. When requesting information about a facebook user, a REST service can return user details along with information about how to get his recent posts, how to get his recent comments and how to retrieve his friend's list.

Designing RESTful APIs
Following are the important things to consider when designing RESTful API's:

- While designing any API, the most important thing is to think about the api consumer i.e. the client who is going to use the service. What are his needs? Does the service uri make sense to him? Does the request, response format make sense to him?
- In Rest, we think Nouns (resources) and NOT Verbs (NOT actions). So, URI's should represent resources. URI's should be hierarchical and as self descriptive as possible. Prefer plurals.
- Always use HTTP Methods.
  - GET : Should not update anything. Should be idempotent (same result in multiple calls). Possible Return Codes 200 (OK) + 404 (NOT FOUND) +400 (BAD REQUEST)
  - POST : Should create new resource. Ideally return JSON with link to newly created resource. Same return codes as get possible. In addition : Return code 201 (CREATED) is possible.
  - PUT : Update a known resource. ex: update client details. Possible Return Codes : 200(OK)

- ◦
    - DELETE : Used to delete a resource.

REST vs SOAP

REST vs SOAP are not really comparable. REST is an architectural style. SOAP is a message exchange format.

Let’s compare the popular implementations of REST and SOAP styles.

- RESTful Sample Implementation : JSON over HTTP
- SOAP Sample Implementation : XML over SOAP over HTTP

Following are the important things to consider:

- REST is built over simple HTTP protocol. SOAP services are more complex to implement and more complex to consume.
- REST has better performance and scalability. REST reads can be cached, SOAP based reads cannot be cached.
- REST permits many different data formats (JSON is the most popular choice) where as SOAP only permits XML.
- SOAP services have well defined structure and interface (WSDL) and has a set of well defined standards (WS-Security, WS-AtomicTransaction and WS-ReliableMessaging). Documentation standards with REST are evolving(We will use Swagger in this course).

Advantages of Web Services

- Reuse : Mark Zuckerberg does not need to invest to build a todo application of his own.
- Modularity
- Language Neutral

Webservices form the building blocks of SOA and microservices architectures.

SOAP Service Examples

Request

```
<Envelope
xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Body>
        <getCourseDetailsRequest

 xmlns="http://in28minutes.com/courses">
            <id>Course1</id>
        </getCourseDetailsRequest>
    </Body>
</Envelope>
```

Response

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getCourseDetailsResponse
xmlns:ns2="http://in28minutes.com/courses">
            <ns2:course>
                <ns2:id>Course1</ns2:id>
                <ns2:name>Spring</ns2:name>
                <ns2:description>10 Steps</ns2:description>
            </ns2:course>
        </ns2:getCourseDetailsResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fault

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
```

```
 <faultcode>SOAP-ENV:Server</faultcode>
           <faultstring
xml:lang="en">java.lang.NullPointerException</faultstring>
       </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

WSDL

- view-source:http://localhost:8080/ws/courses.wsdl

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:sch="http://in28minutes.com/courses"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://in28minutes.com/courses"
targetNamespace="http://in28minutes.com/courses">
   <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="http://in28minutes.com/courses">

    <xs:element name="getCourseDetailsRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="id" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="getCourseDetailsResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="course"
type="tns:course"/>
```

```
 </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="course">
        <xs:sequence>
            <xs:element name="id"

 type="xs:string"/>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="description"
type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
  </wsdl:types>
  <wsdl:message name="getCourseDetailsRequest">
    <wsdl:part element="tns:getCourseDetailsRequest"
name="getCourseDetailsRequest">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="getCourseDetailsResponse">
    <wsdl:part element="tns:getCourseDetailsResponse"
name="getCourseDetailsResponse">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="CoursesPort">
    <wsdl:operation name="getCourseDetails">
      <wsdl:input message="tns:getCourseDetailsRequest"
name="getCourseDetailsRequest">
    </wsdl:input>
      <wsdl:output message="tns:getCourseDetailsResponse"
name="getCourseDetailsResponse">
    </wsdl:output>
    </wsdl:operation>
 </wsdl:portType>
  <wsdl:binding name="CoursesPortSoap11"
type="tns:CoursesPort">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getCourseDetails">
      <soap:operation soapAction=""/>
      <wsdl:input name="getCourseDetailsRequest">
        <soap:body

 use="literal"/>
      </wsdl:input>
      <wsdl:output name="getCourseDetailsResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CoursesPortService">
    <wsdl:port binding="tns:CoursesPortSoap11"
name="CoursesPortSoap11">
      <soap:address location="http://localhost:8080/ws"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

in28minutes

Become an expert on Spring Boot, APIs, Microservices and Full Stack Development

Checkout the Complete in28Minutes Course Guide