***Min Spring Boot application***

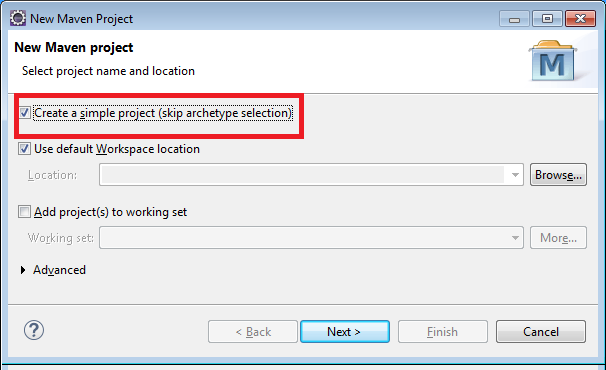
***Spring Boot Creation Web Site***

***Application Properties***

***Autowiring (Under Spring Boot)***

*Min Spring Boot application*

1. ***Create a Maven Project***
2. Menu File 🡪 New 🡪 Maven Project
3. Check item ‘*Create a simple project*’



1. Enter a

Group ID (Group of application normally),

Artifact Id (Your application name)

1. Click Finish
2. Update the ***pom.xml***

<parent>

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

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

<version>1.4.2.RELEASE</version>

</parent>

<dependencies>

<dependency>

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

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

</dependency>

<!-- Database Connection Dependencies:

Not mandatory but very handy! -->

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.4</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.46</version>

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

1. ***Add Web Controller***
2. The handling of web request done be Spring Controller,

(Classes marked with annotation ***@RestController***)

1. Each partial URL to be handled by Request method

(Method marked with annontation ***@RequestMapping("/urlToHandle")***

@RequestMapping("/topics")

**public** List<Topic> getTopics(){ …

Parameters to be passed to handler method marked and use Spring Expression language

(Variable name in curly bracket-Expression language- and Annotation ***@PathVariable***)

@RequestMapping("/topics/{id}")

**public** Topic getTopic(@PathVariable String id){

***Various Types of Methods and Parameters Example***

@RestController

**public** **class** TopicController {

@Autowired

**private** TopicService topicService;

//No Parameters passed

@RequestMapping("/topics")

**public** List<Topic> getTopics(){

}

//One param - id

@RequestMapping("/topics/{id}")

**public** Topic getTopic(@PathVariable String id){

}

//POST version of no parameter

@RequestMapping(method=RequestMethod.***POST***, value= "/topics")

**public** **void** addTopic(@RequestBody Topic topic){

}

//Post version of one parameter

@RequestMapping(method=RequestMethod.***PUT***, value= "/topics/{id}")

**public** **void** updateTopic(@RequestBody Topic topic , @PathVariable String id){

}

Alternate Syntax: @***GetMapping***( url ) 🡪 for Get HTTP method handling

@**PostMapping**( url ) 🡪 for POST HTTP method handling

1. ***Create an Application Class (Standard J2SE class)***

Spring Boot runs from a simple Java executable class (with a main method)

(Main Application Class is marked with annotation ***@SpringBootApplication***)

**package** ca.gc.cbsa.traveller;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

//This annotation marks the class as the base of the Web Application

@SpringBootApplication

**public** **class** Application {

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

//An utility class SpringApplication handle the task of

//starting a local Tomcat server

SpringApplication.*run*(Application.**class**, args);

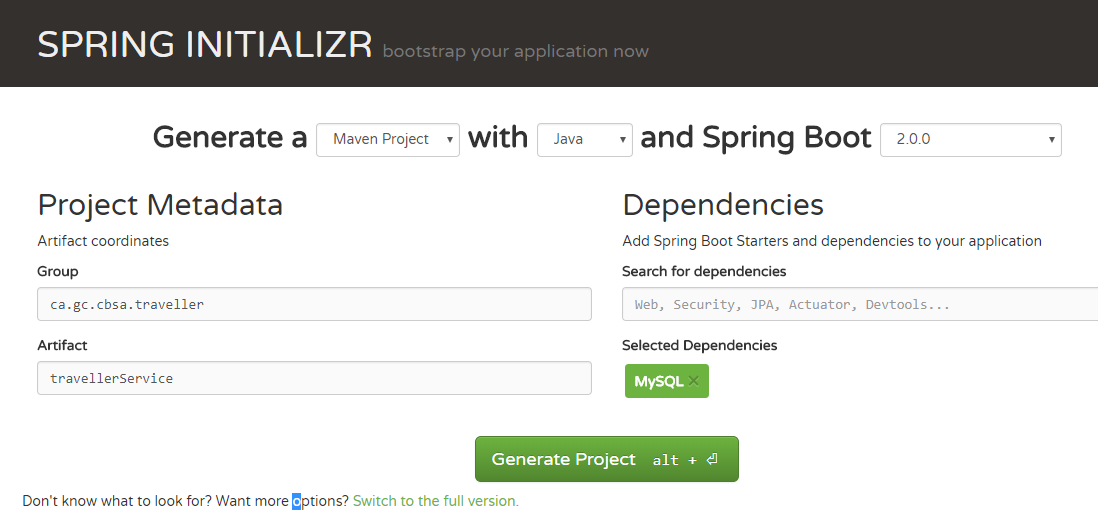
}

}

***Spring Boot Creation Web Site***

A part of the configuration and effort to create a Spring Boot project can be saved by using the following web site

URL: <https://start.spring.io/>



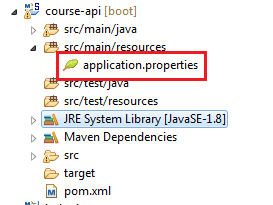
***Note:*** The final artifacts has a batch file (mvnw.bat) that has to be run. This downloads a significant number of jar from the web (if the jar are not already installed on your hard drive).

***Conclusion*** (March 19, 2018): May require more research because not working!

***Application Properties***

You can use a central property file in your Spring Boot application

1. Add a file ***application.properties*** inside the resources folder of your application



**Spring Document on available properties**

All the possible properties that can be set on Spring Boot are documented here:

<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

//Post version of one parameter

***Useful properties***

logging.level.org.springframework.web=DEBUG

***Autowiring (Under Spring Boot)***

Spring Boot handle the heavy lifting of the Autowiring (Dependency injection)

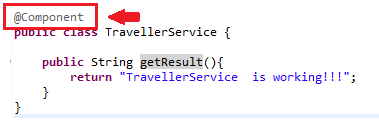
1. Package Scan

A package scan of the package where the main application is (with the annotation @***SpringBootApplication***) is done, along with all the sub component

*Note*: this is the “***context:component-scan***” configuration inside the Spring Defintion file of a regular Spring application

1. Mark the Beans

As regular Spring application, every injectable beans have to annoted with ***@Component***



1. Invoke the application context in the main Application

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

ApplicationContext applicationContext = SpringApplication.*run*(Application.**class**, args);

TravellerService service = applicationContext.getBean(TravellerService.**class**);

System.***out***.println(service.getResult());

}

iiib) Or mark a field with annotation ***@Autowired***

@RestController

**public** **class** UserResource {

@Autowired

**private** UserDaoService service;