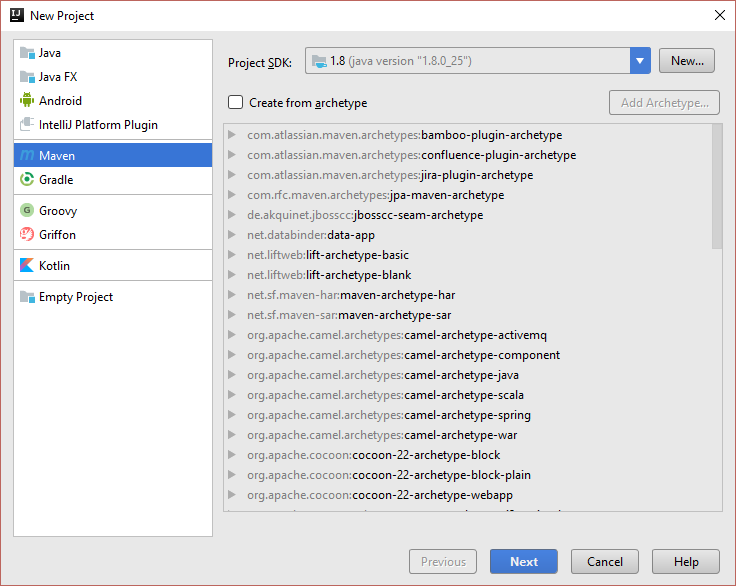
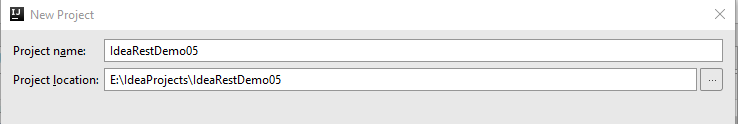
**Building, Deploying and Debugging a Basic Spring/REST Project (IntelliJ IDEA 2016.3.4)**

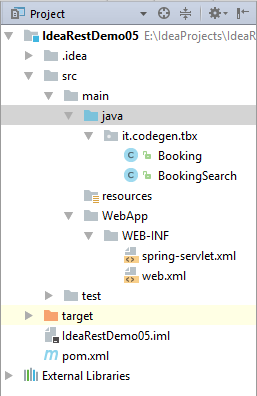
1. Create a new ‘Maven’ project







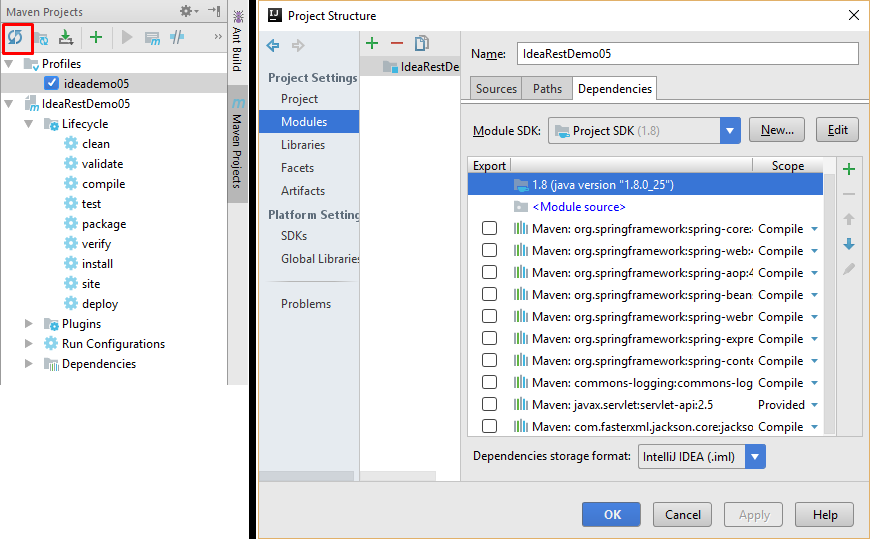
2. Create the project structure as following.



3. Update the POM.xml

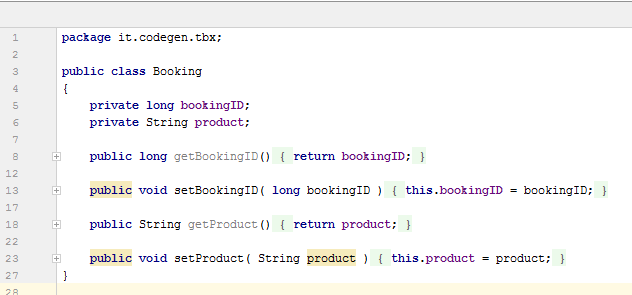
*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**groupId**>IdeaRestDemo05</**groupId**>  
 <**artifactId**>IdeaRestDemo05</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 <**packaging**>war</**packaging**>  
  
 <**properties**>  
 <**jdk.version**>1.8</**jdk.version**>  
 <**spring.version**>4.3.6.RELEASE</**spring.version**>  
 </**properties**>  
  
 <**profiles**>  
 <**profile**>  
 <**id**>ideademo05</**id**>  
 </**profile**>  
 </**profiles**>  
  
 <**dependencies**>  
  
 *<!--Spring Dependencies-->* <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-core</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-web</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-webmvc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 *<!-- Logging -->* <**dependency**>  
 <**groupId**>commons-logging</**groupId**>  
 <**artifactId**>commons-logging</**artifactId**>  
 <**version**>1.1.1</**version**>  
 </**dependency**>  
  
 *<!-- Servlet -->* <**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>servlet-api</**artifactId**>  
 <**version**>2.5</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
  
 *<!--Jackson-->* <**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-core</**artifactId**>  
 <**version**>2.8.6</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-databind</**artifactId**>  
 <**version**>2.8.6</**version**>  
 </**dependency**>  
  
 </**dependencies**>  
  
 <**build**>  
 <**finalName**>MavenWeb</**finalName**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.tomcat.maven</**groupId**>  
 <**artifactId**>tomcat7-maven-plugin</**artifactId**>  
 <**version**>2.2</**version**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
  
</**project**>

4. Refresh Maven Projects – This will update the necessary dependencies and add them to your project. You can verify this by checking the ‘Dependencies’ for the project.

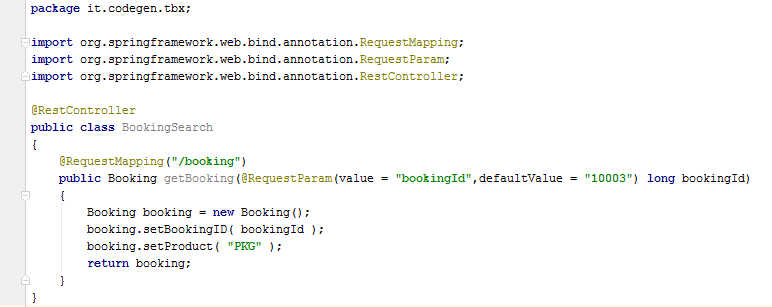


4. Create a package named ‘it.codegen.tbx’ inside ‘src/main/java’.

5. Create a class named ‘Booking’.



6. Create the Rest Controller class named ‘BookingSearch’



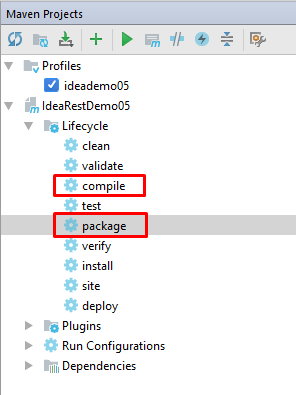
7. Create ‘web.xml’ inside ‘WebApp/WEB-INF’

<**web-app id="WebApp\_ID" version="2.4"  
 xmlns="http://java.sun.com/xml/ns/j2ee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee  
 http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"**>  
  
 <**display-name**>Spring4RestController</**display-name**>  
  
 <**servlet**>  
 <**servlet-name**>spring</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
 <**load-on-startup**>1</**load-on-startup**>  
 </**servlet**>  
  
 <**servlet-mapping**>  
 <**servlet-name**>spring</**servlet-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
 </**servlet-mapping**>  
  
</**web-app**>

8. Create ‘spring-servlet.xml’ inside ‘WebApp/WEB-INF’

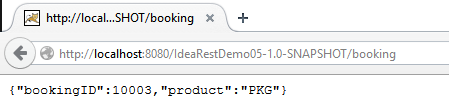
*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd  
 http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd"**>  
  
 <**context:component-scan base-package="it.codegen.tbx"** />  
  
 <**mvc:annotation-driven** />  
  
</**beans**>

9. Build the project and package using maven. Once you have done this, ‘target’ folder will be created and the .war file will be created inside the folder. As we have specified in the POM.xml ‘<packaging>war</packaging>’ the deployable will be a .war file.



10. There are two ways you can deploy the project.

* Deploy the .war manually in a, locally installed tomcat server
  + This is an easy task, which you can just deploy the war using ‘Manager-GUI’ in Tomcat and you can access the web-service using the URL.

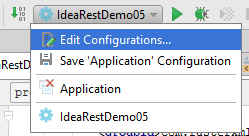


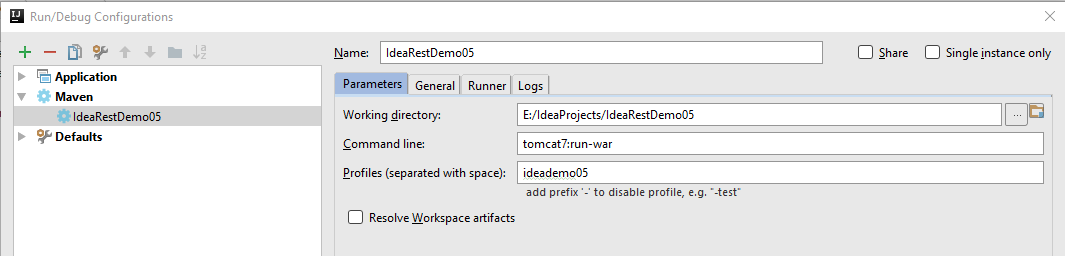
* Deploy using Idea with the purpose of remote debugging using maven tomcat plugin.
  + To do this, we need to make sure our POM.xml has the following items and the tomcat-plugin.





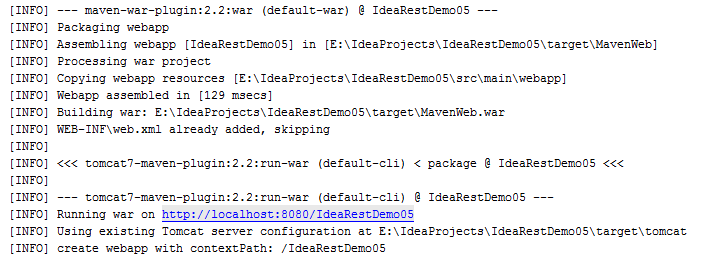
* + Once the POM.xml is validated to have the above elements, then we need to setup the idea configurations.
    - Go to Edit Configurations.

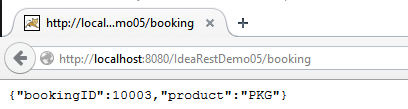


* + - Add a new Maven Configuration
    - Set the settings as following.
    - Now you can click the debug button and debug the program.



* + - You will get the following output and you can access the service with the output URL.





11. You have successfully completed your first RESTful Spring Project.

**Important Notes on Spring/REST**

What is the difference between a Request Parameter and a Path Variable?

**Answer 01:**

* @PathVariable is to obtain some placeholder from the uri (Spring call it an URI Template) — see [Spring Reference Chapter 16.3.2.2 URI Template Patterns](http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/htmlsingle/spring-framework-reference.html#mvc-ann-requestmapping-uri-templates)
* @RequestParam is to obtain an parameter — see [Spring Reference Chapter 16.3.3.3 Binding request parameters to method parameters with @RequestParam](http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/htmlsingle/spring-framework-reference.html#mvc-ann-requestparam)

If URL http://localhost:8080/MyApp/user/1234/invoices?date=12-05-2013 gets the invoices for user 1234 on December 5th, 2013, the controller method would look like:

@RequestMapping(value="/user/{userId}/invoices", method = RequestMethod.GET)

public List<Invoice> listUsersInvoices(

@PathVariable("userId") int user,

@RequestParam(value = "date", required = false) Date dateOrNull) {

...

}

Also, request parameters can be optional, but path variables cannot--if they were, it would change the URL path hierarchy and introduce request mapping conflicts. For example, would /user/invoicesprovide the invoices for user null or details about a user with ID "invoices"?

**Answer 02:**

**@RequestParam** annotation used for accessing the query parameter values from the request. Look at the following request URL:

http://localhost:8080/springmvc/hello/101?param1=10&param2=20

In the above URL request, the values for param1 and param2 can be accessed as below:

public String getDetails(

@RequestParam(value="param1", required=true) String param1,

@RequestParam(value="param2", required=false) String param2){

...

}

The following are the list of parameters supported by the @RequestParam annotation:

* **defaultValue** – This is the default value as a fallback mechanism if request is not having the value or it is empty.
* **name** – Name of the parameter to bind
* **required** – Whether the parameter is mandatory or not. If it is true, failing to send that parameter will fail.
* **value** – This is an alias for the name attribute

**@PathVariable**

@*PathVariable* identifies the pattern that is used in the URI for the incoming request. Let’s look at the below request URL:

<http://localhost:8080/springmvc/hello/101?param1=10&param2=20>

The above URL request can be written in your Spring MVC as below:

@RequestMapping("/hello/{id}") public String getDetails(@PathVariable(value="id") String id,

@RequestParam(value="param1", required=true) String param1,

@RequestParam(value="param2", required=false) String param2){

.......

}

The @**PathVariable** annotation has only one attribute value for binding the request URI template. It is allowed to use the multiple @**PathVariable** annotation in the single method. But, ensure that no more than one method has the same pattern.

Also there is one more interesting annotation: **@MatrixVariable**

<http://localhost:8080/spring_3_2/matrixvars/stocks;BT.A=276.70,+10.40,+3.91;AZN=236.00,+103.00,+3.29;SBRY=375.50,+7.60,+2.07>

And the Controller method for it

@RequestMapping(value = "/{stocks}", method = RequestMethod.GET)

public String showPortfolioValues(@MatrixVariable Map<String, List<String>> matrixVars, Model model) {

logger.info("Storing {} Values which are: {}", new Object[] { matrixVars.size(), matrixVars });

List<List<String>> outlist = map2List(matrixVars);

model.addAttribute("stocks", outlist);

return "stocks";

}

But you must enable:

<mvc:annotation-driven enableMatrixVariables="true" >