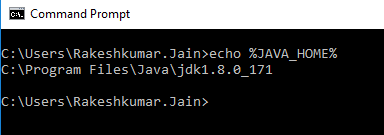
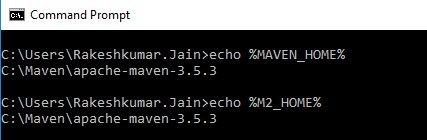
**Starting with Spring Boot Application**

**Prerequisite:**

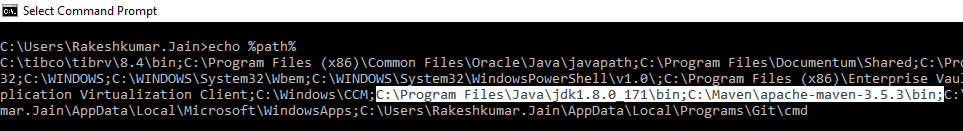
* Install Java 1.8 or above and create the environment variable JAVA\_HOME and set the path for the same. Same can be verified as shown below:



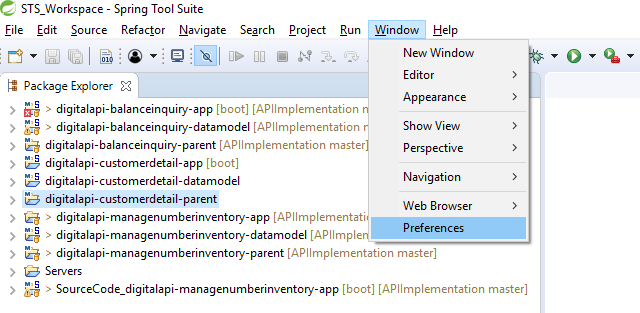
* Download STS 3.9.4 or above and Unzip the same.
* Download Maven 3.5.3 or above and Unzip the same.
* Create the environment variable MAVEN\_HOME and M2\_HOME and set the path for the same. Same can be verified as shown below:

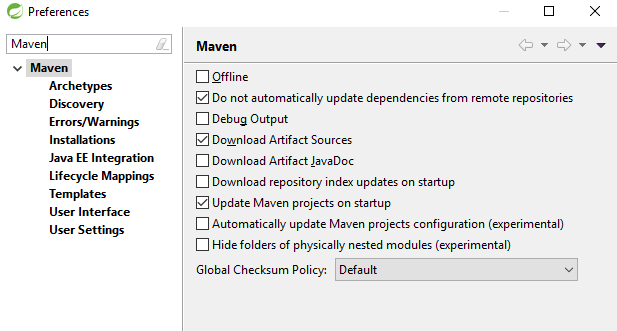


* Edit the path variable and add the Bin Path for JAVA and Maven. Same can be verified as shown below:

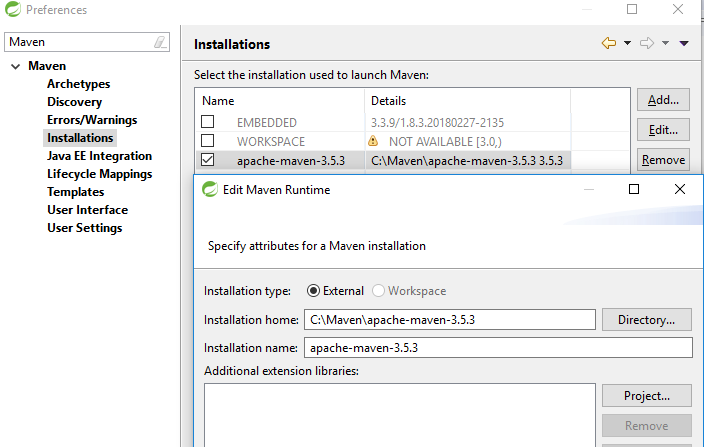


* Download Soap-UI and Unzip the same.
* Download Postman.
* Open STS and create a default workspace. Go to Windows🡪Preference as shown and search for Maven:





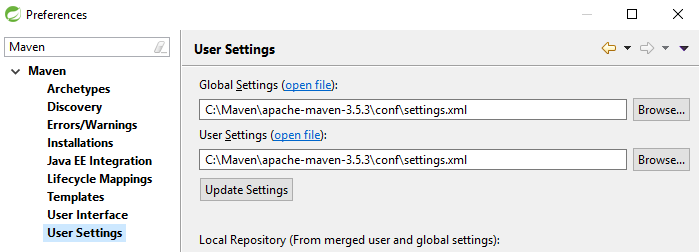
* Add Maven Path under Maven🡪Installations.



* Take a backup of existing Setting.xml and then Replace Setting.xml in your Maven/conf folder (C:\Maven\apache-maven-3.5.3\conf) with the Setting.xml given below.

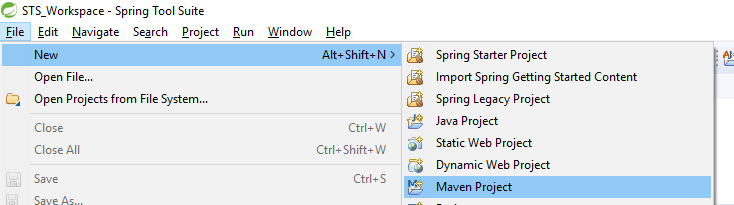


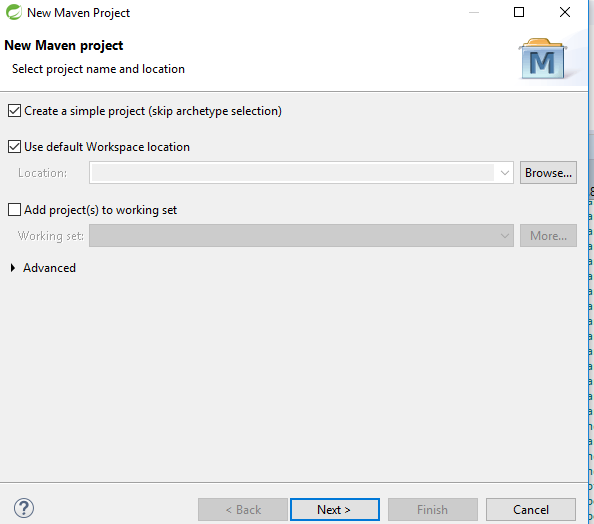
* Add the path for the Setting.xml under User Setting🡪Global Settings and User Settings and click on Update Settings and then Apply.
* This setting.xml file contains path for the Repository from where it will download all the dependent jars, the Server ID, Username and Password and the location where your local Repository is going to be.



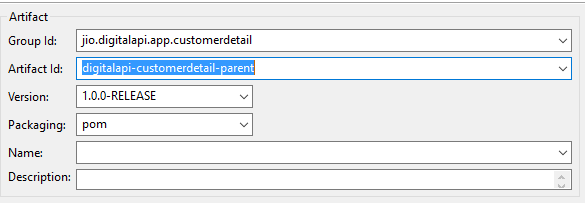
**Creating First Maven Project(digitalapi-customerdetail-parent):**

* This Project is all about adding dependency and creating a pom file which will act as a Parent for the other two Maven Projects that are about to be created.
* Create Your first Maven Project by selecting File🡪New🡪Maven Project and click on Create a Simple Project Checkbox and click next.





* Provide values for Group Id (Package Name), Artifact Id (Project Name), Version and select Packaging as pom from the dropdown and click Finish. Conventions used for Group Id and Artifact ID are given below, where customerdetail is the name of the project.
* Group Id: jio.digitalapi.app.customerdetail
* Artifact Id: digitalapi-customerdetail-parent



* In pom.xml created above add parent, plugin, properties and dependency as specified in the below text file under <project></project>. Read all the comments carefully provided in all the below files:

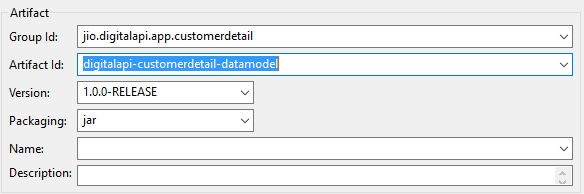


* The directory structure for the above project is as shown below:



**Creating Second Maven Project(digitalapi-customerdetail-datamodel):**

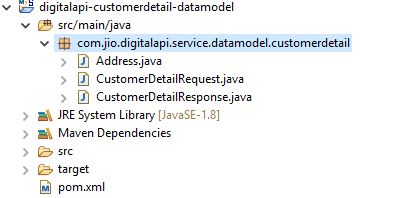
* This project is all about creating POJO’s as per the FM Document you received and adding Getter’s and Setter’s along with Default and Parameterized Constructors.
* Again create a Simple Maven Project and provide all the parameters as discussed above. This time select Packaging as jar from the dropdown and click finish. Provide Group Id and Version exactly same as that of the Parent Project we created above.



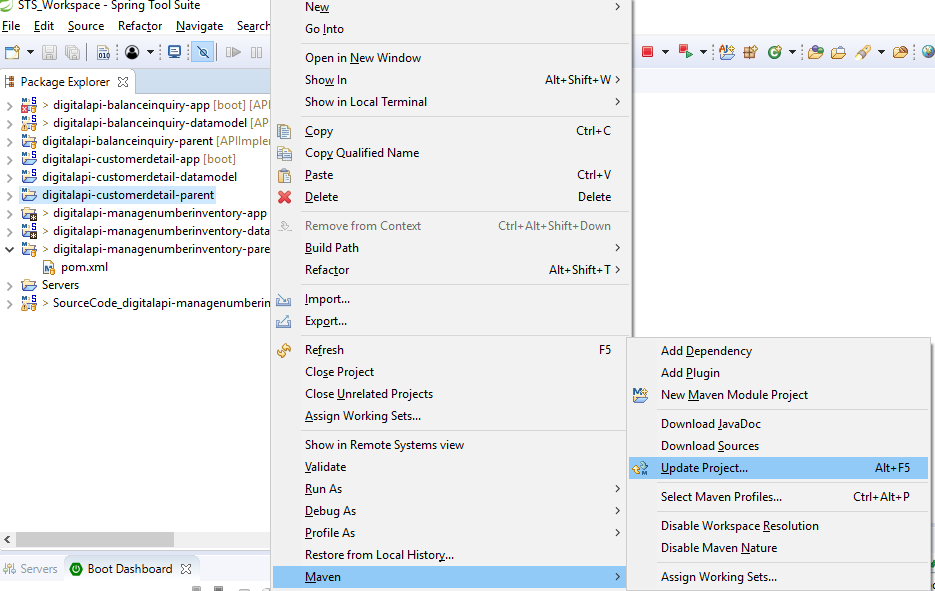
* In pom.xml we add parent and dependency as specified in the below text file under <project></project>:

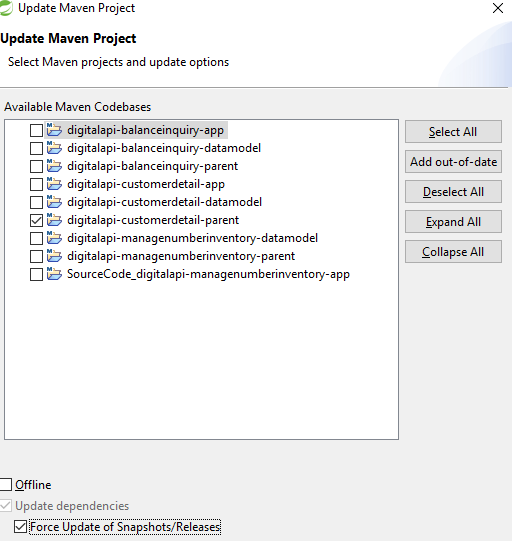


* Create a Package under src/main/java and convention for creating the same is as shown below:
* Package name: com.jio.digitalapi.service.datamodel.customerdetail
* Under this package create all the POJO’s, i.e., Java Classes for the with respect to your Functional Mapping Document.
* The directory structure for the above project is as shown below:

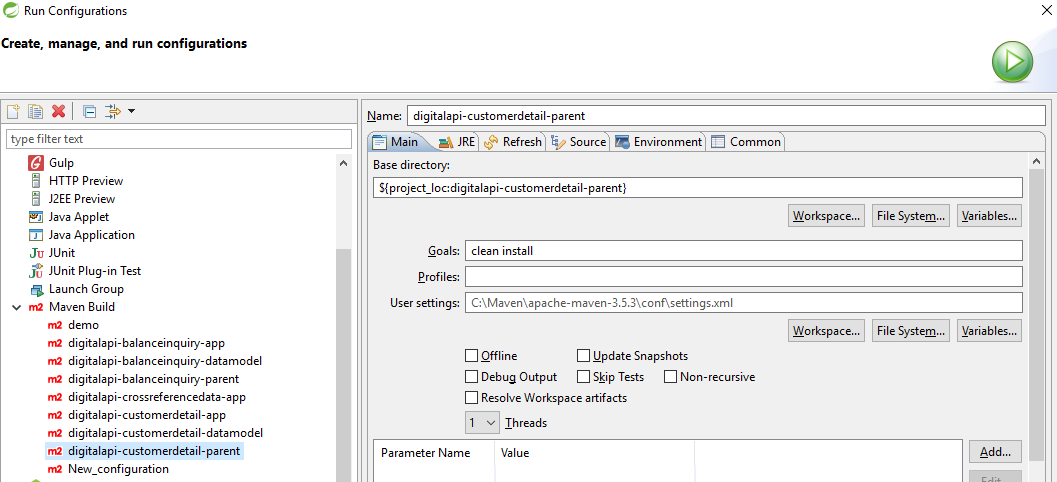


* **Step 1:**Once all the above steps are done go to the Parent project, Right click on project, Maven🡪Update Project and select your Project and enable the check box “Force Update of Snapshots/Releases” and click ok.





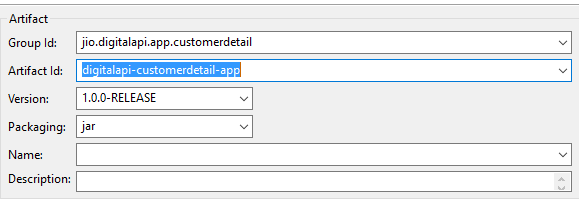
* **Step 2:**After the above steps are done, Right click on Parent Project Run As🡪Run Configurations🡪Maven Build and write **clean install** in Goals and click on Run.



* After this repeat only Step 1 for the current Project we have created(digitalapi-customerdetail-datamodel).

**Creating Third Maven Project(digitalapi-customerdetail-app):**

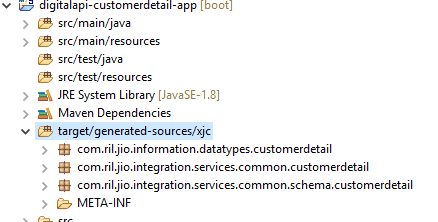
* This Project is all about where your business logic is implemented, Validations are performed, Mapping is done and the target service is invoked and the response is returned back to the calling service.
* Again create a Simple Maven Project and provide all the parameters as discussed above. This time select Packaging as jar from the dropdown and click finish. Provide Group Id and Version exactly same as that of the Parent Project we created above.



* In pom.xml created add parent, plugin, properties and dependency as specified in the below text file under <project></project>:



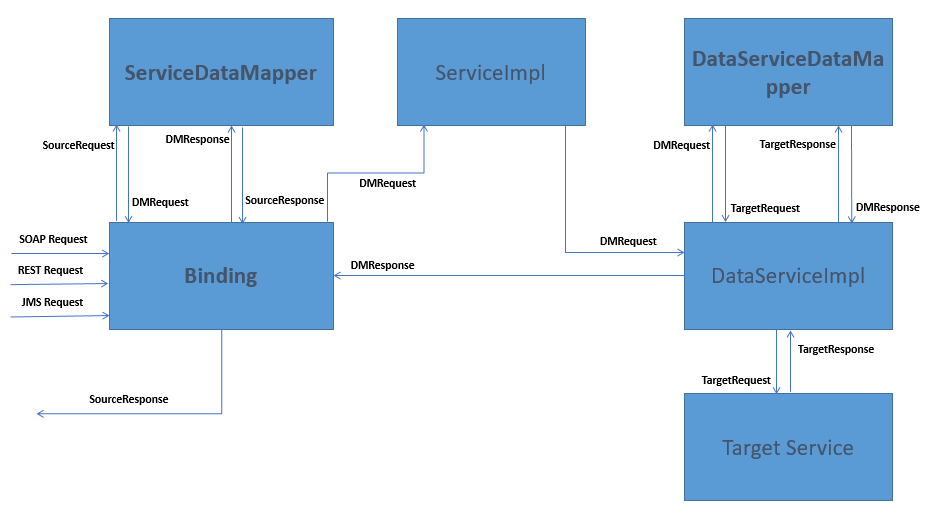
* Add a wsdl folder under src/main/resource and Copy the .wsdl file to this folder.
* After you update your Maven Project as shown in Second Project(digitalapi-customerdetail-datamodel), you will be able to see the Classes are generated related to WSDL file as shown below.



* Create application.properties and commonlogconfig.properties file under src/main/resource and copy the contents from the file give below. Also go through the comments given in the below files:

* Now we will start writing the java classes which are responsible for receiving the request, mapping it to Datamodel, mapping from Datamodel to the request format required by the target service and finally invoking the service. This entire process is divided in 6 packages we are going to create, the detailed explanation for the same is given below.
* Let’s first understand the framework flow and then we will jump to detailed explanation.



* SOAP or REST or JMS Request Received. (**Binding**)
* The above request is passed to ServiceDataMapper(DataMapper) where the above requested is mapped to the DataModel objects created in Second Project and this request called as DMRequest is returned back to Binding class. (**DataMapper**)
* The DMRequest is then passed to ServiceImpl where all the validations if any are performed. After all validations are done, the same Request is passed to DataServiceImpl. (**ServiceImpl**)
* The above Request is than passed to DataServiceDataMapper(DataMapper) for converting the DMRequest in the format Required by the Target Service and do the additional mappings if any required by the target service and finally target service is invoked and the response is captured. (**DataServiceImpl**)
* The above Response which is in the format of the Target service is passed to the DataServiceDataMapper and this response is mapped to DataModel objects and DMResponse is created. (**DataMapper**)
* The above Response is returned to the Binding class. (**DataServiceImpl**)
* This response is than passed to ServiceDataMapper(DataMapper). (**Binding**)
* The above DMResponse is converted back to the response required by the calling service and the result is mapped to the respective elements of the calling service, and this response is returned back to the Binding class. (**DataMapper**)
* Finally, this response is returned back to the calling service. (**Binding**)

**Detailed Explanation of each Package used in Framework**

**Binding**

* Create a Package under src/main/java with the name as com.jio.digitalapi.app.customerdetail, where customerdetail is your application name. This package will contain your main class DigitalapiCustomerDetailApplication.java, where you will right the following lines in your main method as shown below:

init (DigitalapiCustomerDetailApplication.class, args);

This Main class extends DigitalApiPlatformApp and is annotated with @DigitalApiPlatform as per the framework.

* Now create a package under src/main/java with the name as com.jio.digitalapi.app.customerdetail.bindings. This package will contain 3 classes one will act as entry point for SOAP call, one will be for Rest call and one will be for JMS call.
* For **Soap** call, we will be using annotation as **@DigitalApiWebService** for the class, **@ResponsePayload** for the method and the class extends **DigitalApiSoapServiceBinding**. Also we will be using annotation **@PayloadRoot(namespace=NAMESPACE\_URL,localPart= getCustomerDetail"),** here NAMESPACE\_URL is the target namespace(tns) which you can retrieve from the .wsdl file and localPart is SoapAction.
* For **REST** call, we will be using annotation **@DigitalOpenApiController ()** and the class extends from **DigitalApiRestServiceBinding.**  Also we need to create an interface which will be implemented by the above Rest class. This interface will contain annotation at method level as: **@ApiOperation(value="${digital.api.platform.appimpl.getcustomerdetail.desc}”, notes="${digital.api.platform.appimpl.getcustomerdetail.notes}")**, where value and notes get the Values from the variables that are defined in application.properties and are displayed on swagger. Also it contains annotation **@PostDigitalOpenApi(value = "/getCustomerDetail")**, where /getCustomerDetail is the extended path for the Rest Service. Other annotations used are **@RequestHeader and @RequestBody.**
* For **JMS** call, depending upon whether you are using **JMS Sender** or **JMS Receiver** and depending upon whether you are using **JMS Request** or **JSON Request**, different annotations used are: **@DigitalApiJmsSender** and **@DigitalApiJmsReceiver.** Also the class you create for Sender or Receiver, should extends from **DigitalApiJmsXmlMessageReceiver** or **DigitalApiJmsXmlMessageSender** or **DigitalApiJmsJsonMessageReceiver** or **DigitalApiJmsJsonMessageSender** depending on your requirement. All the classes above that you extend above achieves Synchronization. For Async Request or Response the class should extends from **DigitalApiAsyncJmsJsonMessageReceiver** or **DigitalApiAsyncJmsXmlMessageReceiver.**

**DataMapper**

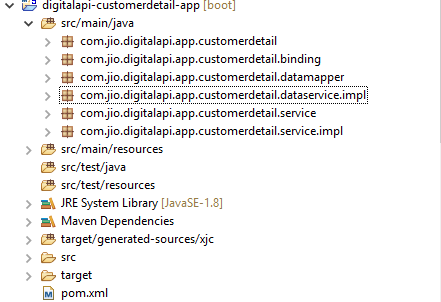
* Create a Package under src/main/java with the name as com.jio.digitalapi.app.customerdetail.datamapper.
* This package will contain 2 Java classes, one for mapping input request to DataModel request and mapping the DataModel response to the response required by the calling service and other Java class for mapping DataModel Request to the request format required by the target service and mapping the response from the target service back to DataModel Response.
* This package may also contain Java classes that extends from **DigitalApiExceptionMapper<JAXBElement<InternalExceptionCollection>>** related to the exceptions, that you might throw to handle business exceptions. Creating this class, we will set the errorCode, reason and details.
* The classes we create are annotated with **@DigitalApiDataMapper.**

**ServiceImpl**

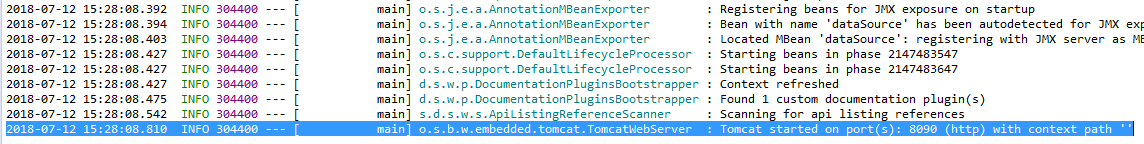
* Create a Package under src/main/java with the name as com.jio.digitalapi.app.customerdetail.service.impl.
* This Package will contain a Java class, which is responsible for doing all the Business validations if any. So this class will receive DataModel Request, will perform some validations and will return back the DataModel Response.
* This class is annotated with **@Service.**

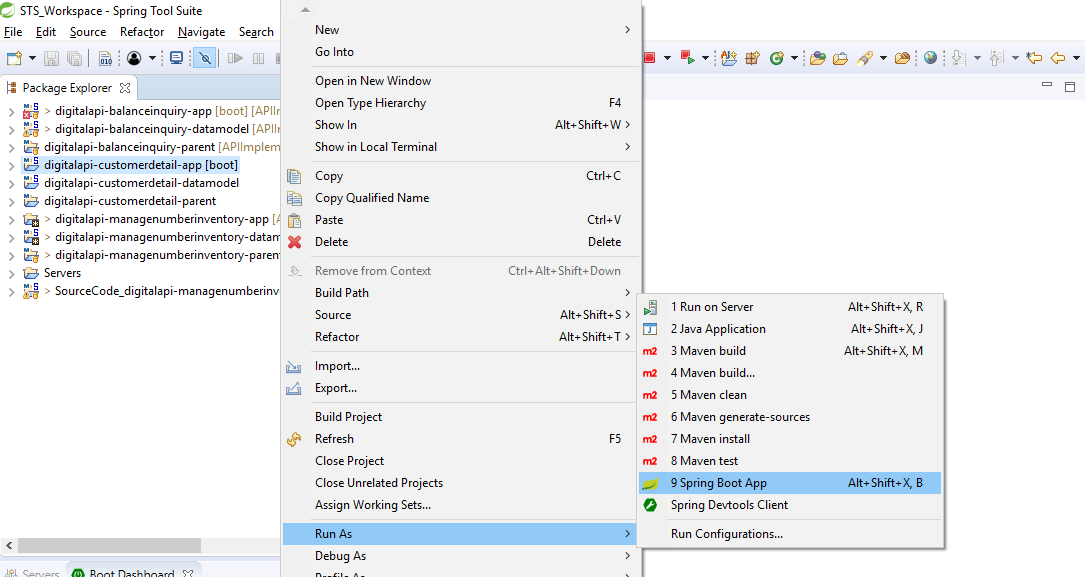
**DataServiceImpl**

* Create a Package under src/main/java with the name as com.jio.digitalapi.app.customerdetail.dataservice.impl.
* This Package will contain a Java class, which is responsible for setting additional request parameters if any and finally invoking the target service using **invokeService(serviceRequestContext, requestMsg)**, where we pass the ServiceRequestContext and Request Message required by the target service.
* This class is annotated with **@DigitalApiDataService() and @ConfigurationProperties(prefix="digital.api.platform.app.serviceclient.getCustomerDetail")** and extends **DigitalApiWebServiceClient** or **DigitalApiRestServiceClient.** Hereprefix in ConfigurationProperties is the prefix we will be using in application.properties file, to declare parameters related to target service.
* After all the required classes are created and finishing the above steps, Repeat Step 1 and Step 2 as explained in Second Project(digitalapi-customerdetail-datamodel).
* The directory structure for the above Project is as shown below:



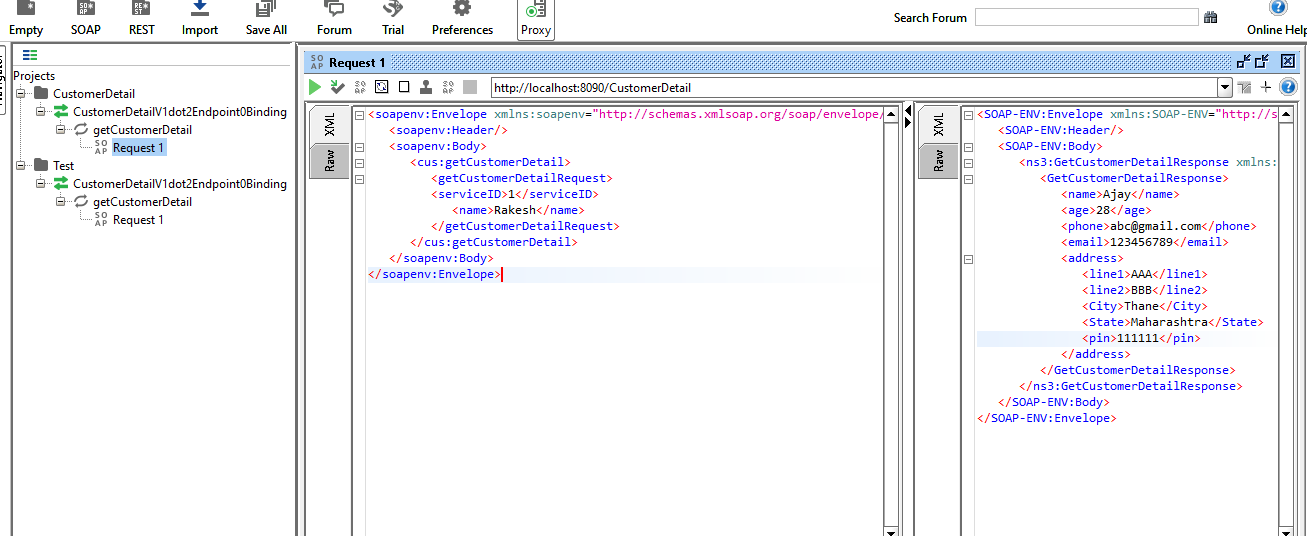
* Now after all the above steps are completed and when there are no more errors in the above projects, it’s time to test the application. Right click on your app project Run As🡪Spring Boot App or also you can run the Spring Boot Project from the command prompt using command: **mvn spring-boot:run.**
* After your application is successfully started you will see the last line on console as (Tomcat started on port(s): 8090 (http) with context path ''):





**Testing your Application with SOAP-UI:**

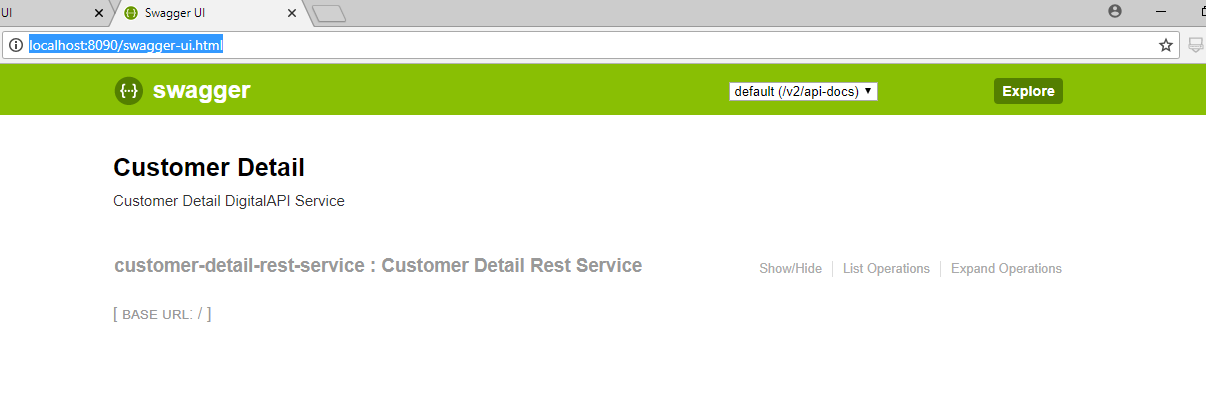
* Import the WSDL in Soap-UI, provide the required values and test.
* The URL used in this Project is: [**http://localhost:8090/CustomerDetail**](http://localhost:8090/CustomerDetail)



**Testing your Application with REST(Swagger):**

* For testing REST Service, go to your Web Browser and paste the below URL:

[**http://localhost:8090/swagger-ui.html**](http://localhost:8090/swagger-ui.html)



* Click on Expand Operations and you will see list of all the Operations applicable for your service.
* The Base URL for in this case would be:

**http://localhost:8090/customerDetail/getCustomerDetail**

* The only thing you need to do is select Response Content Type as: application/json and provide the values in the Text Box and hit Button **Try it out.** With this you will see the Curl command, Request URL and Request Headers are generated automatically and will find the required response.

