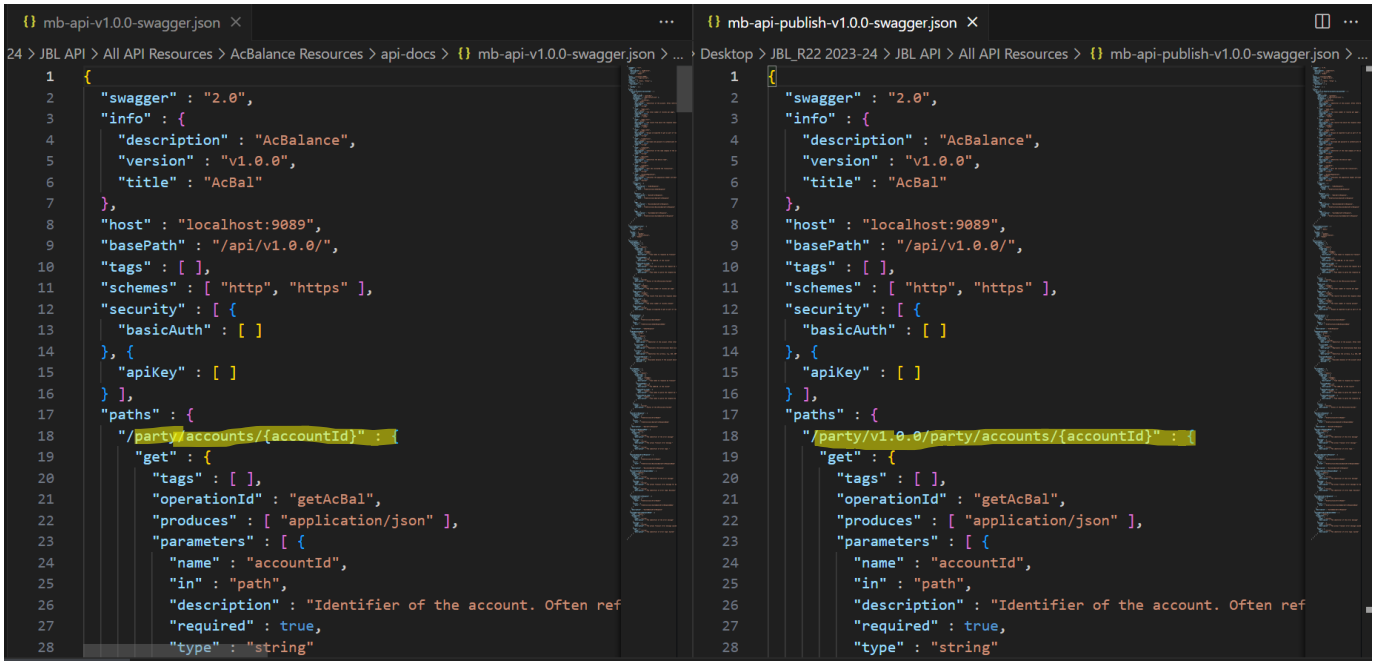
# CREATING PUBLISHER API 18/02/2024

**Publisher APIs** are built over **PROVIDER APIs** so that the **JSON** structure and field values of **PROVIDER API** request and response can be modified as per the end user requirement at **IRIS layer**.

1. From the provider APIs **api-docs**, take the **swagger JSON**, modify as per your requirement of JSON structure and **save it.**



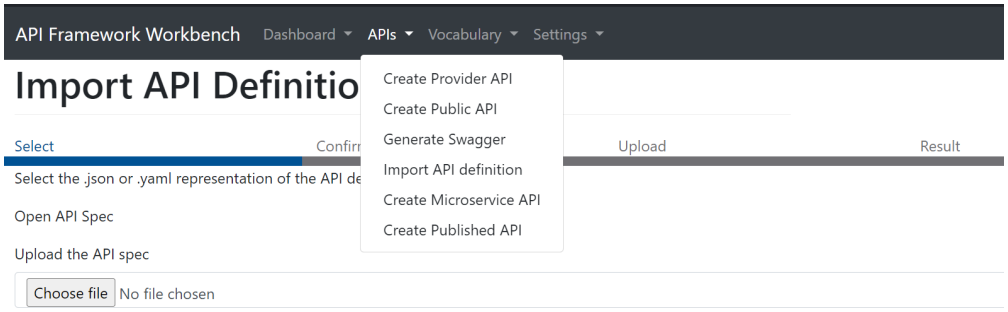

Here, the path is changed only.

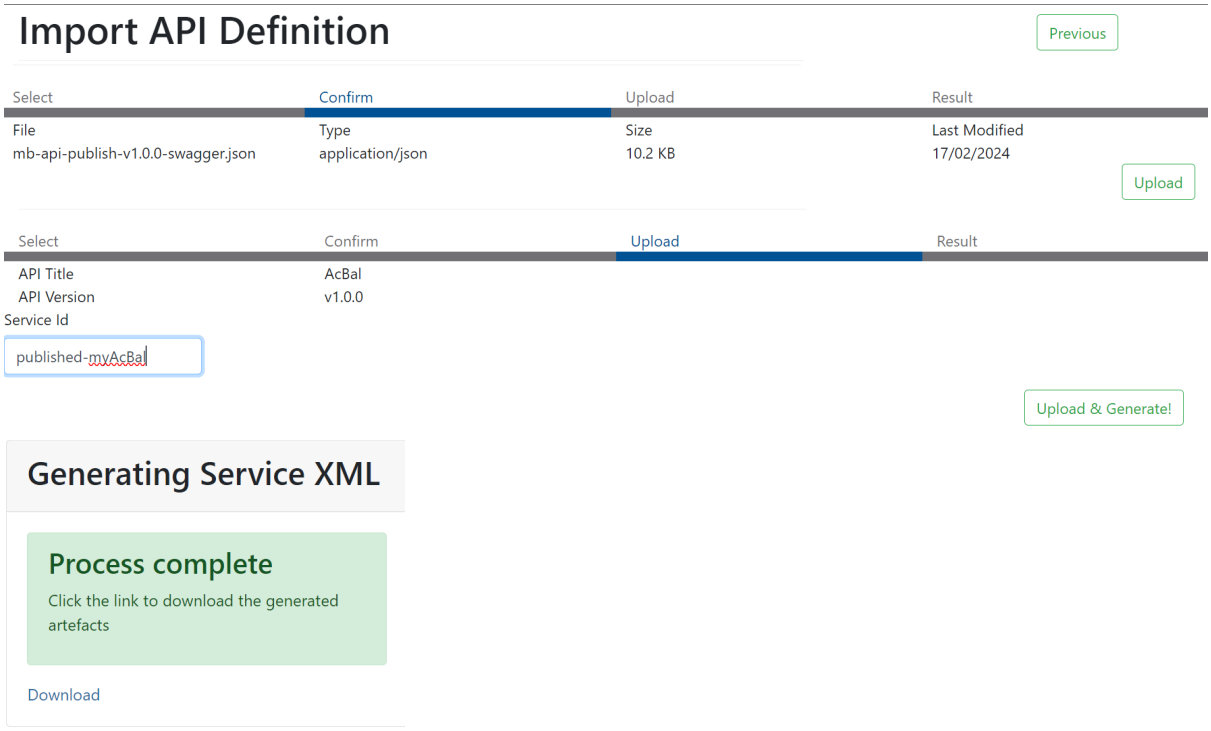
mb-api-publish-v1.0.0-swagger.json

**Modified JSON File:**

2. Check the correctness of that **swagger JSON** via the open-source swagger editor, the **URL** of which is given below. If any structural errors are thrown, resolve it.
https://editor-next.swagger.io/

3. Import this swagger definition in workbench using the **"Import API definition"** option and download the publisher zip file.



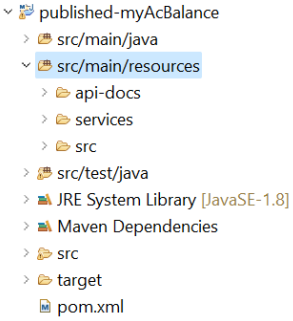






Download the zip file.

4. Create a maven project with **archetype-service catalog** and import the zip file downloaded in the previous step.

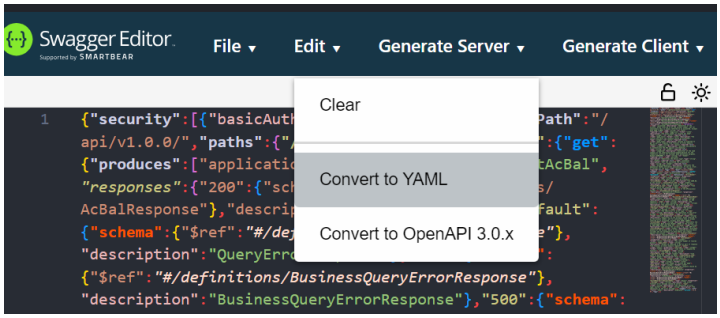


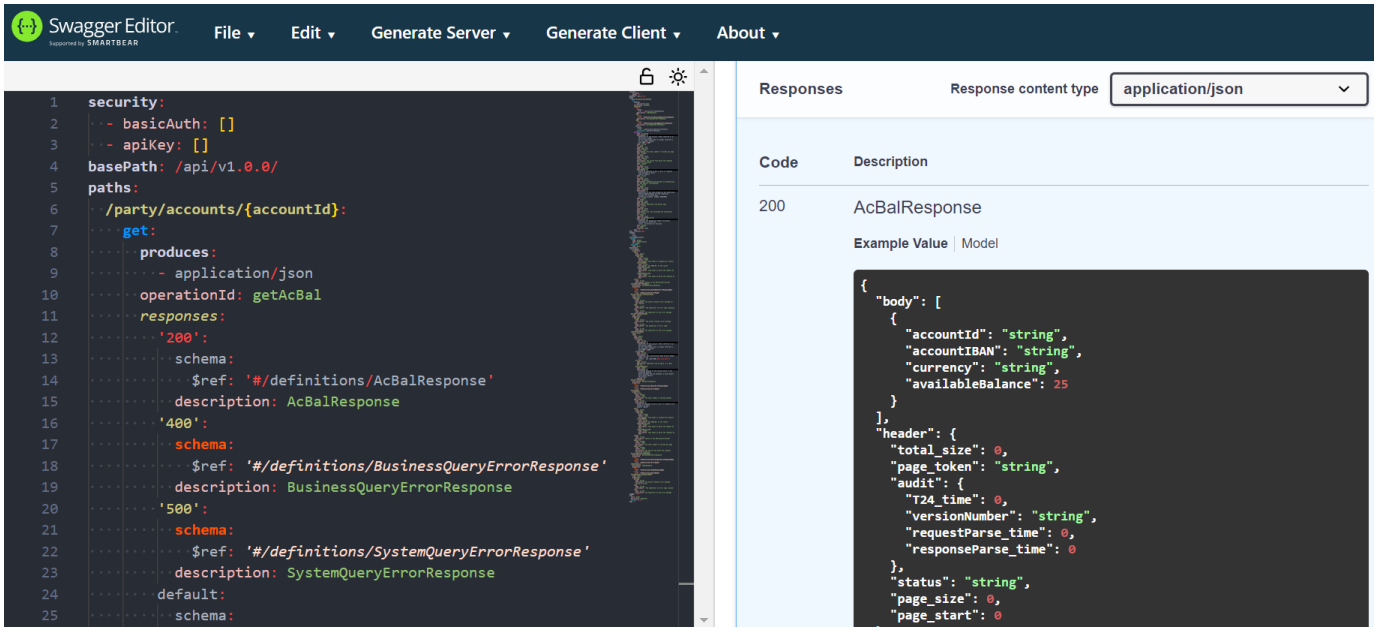

## Special Note: We can skip STEP 5,6,7

5. Generate code for the **published API swagger JSON**. To generate code, do the following steps.

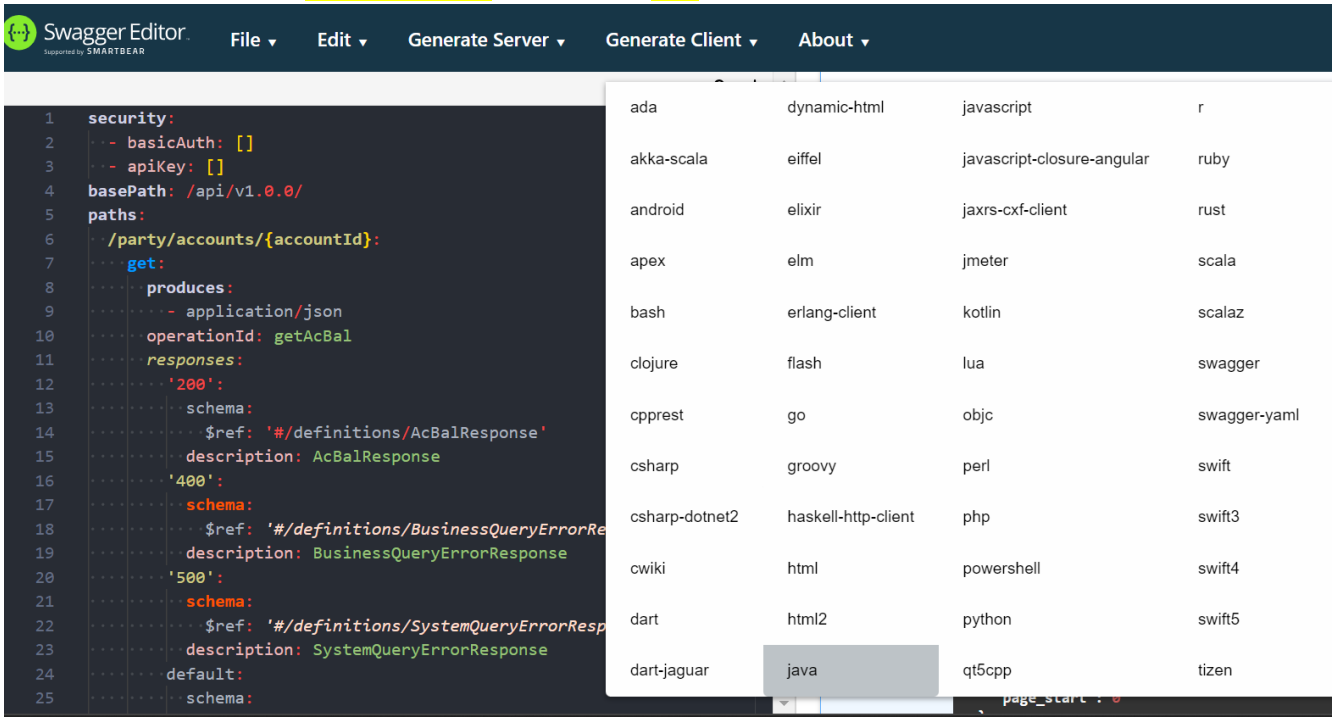
a. Launch https://editor.swagger.io/ in browser.

b. Paste the content of the published API swagger JSON file. (It will be converted to YAML). Ensure no errors are thrown.



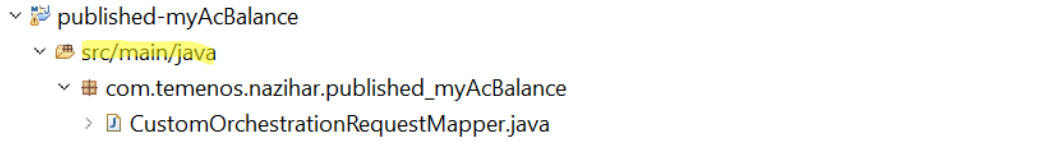


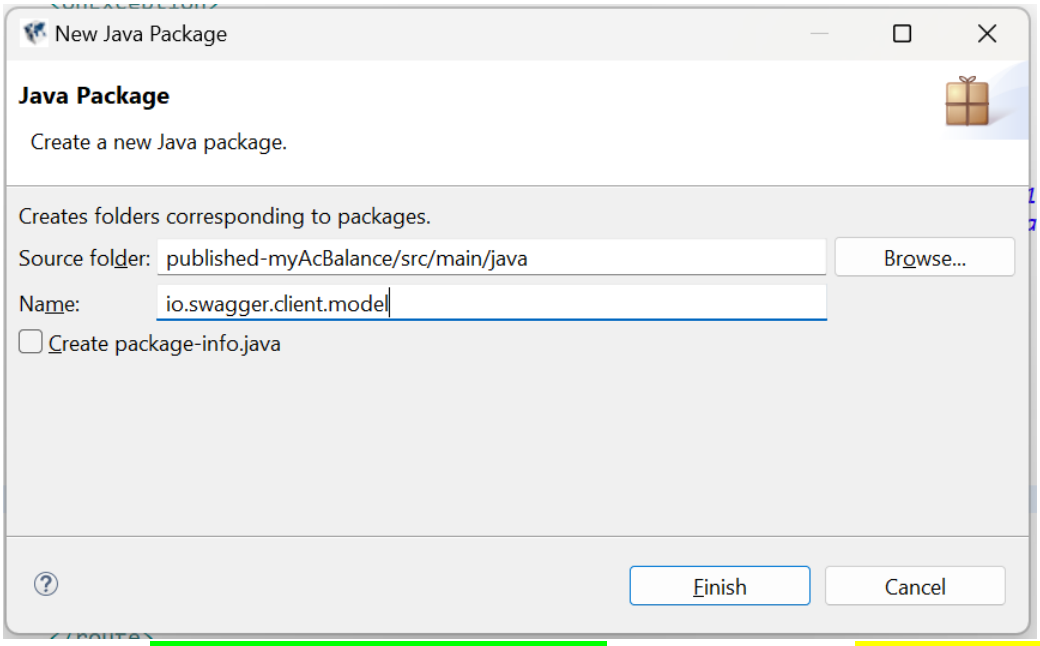

c. Click the option "Generate Client" and select "java" as shown in the below image.




d. A file named **java-client-generated.zip** will be downloaded. **Extract** the contents of this file to a folder.

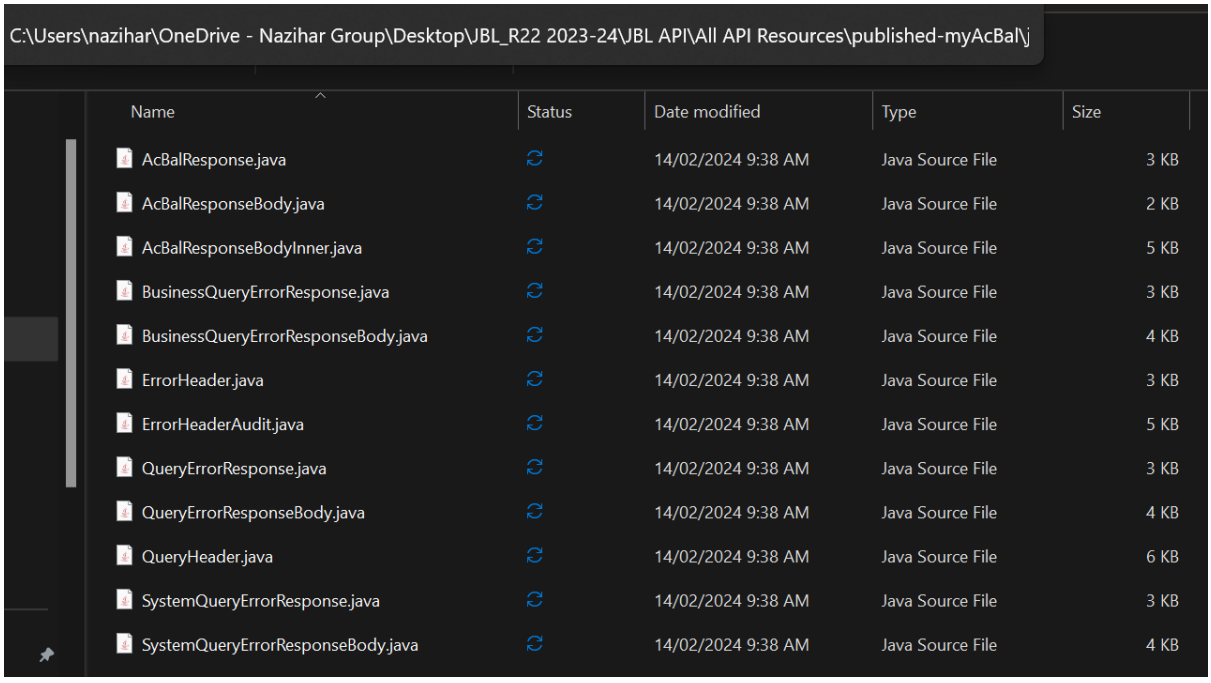
e. Under **<<PublishedApi_project>>\src\main\ java** directory, create a **java package** named io.swagger.client.model

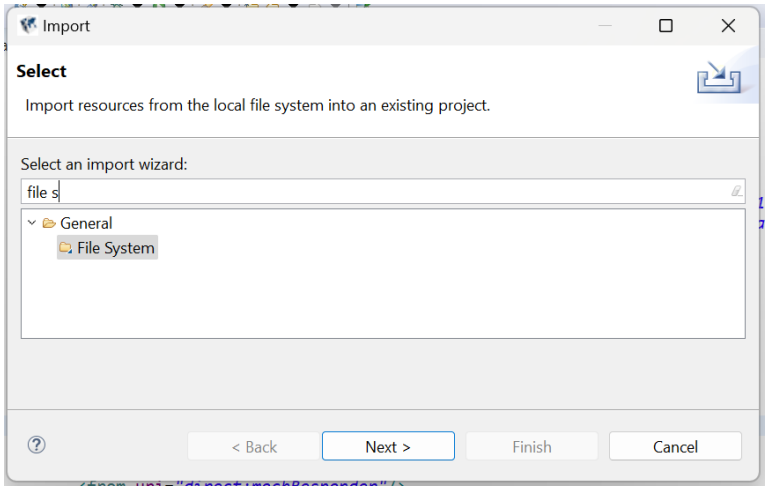


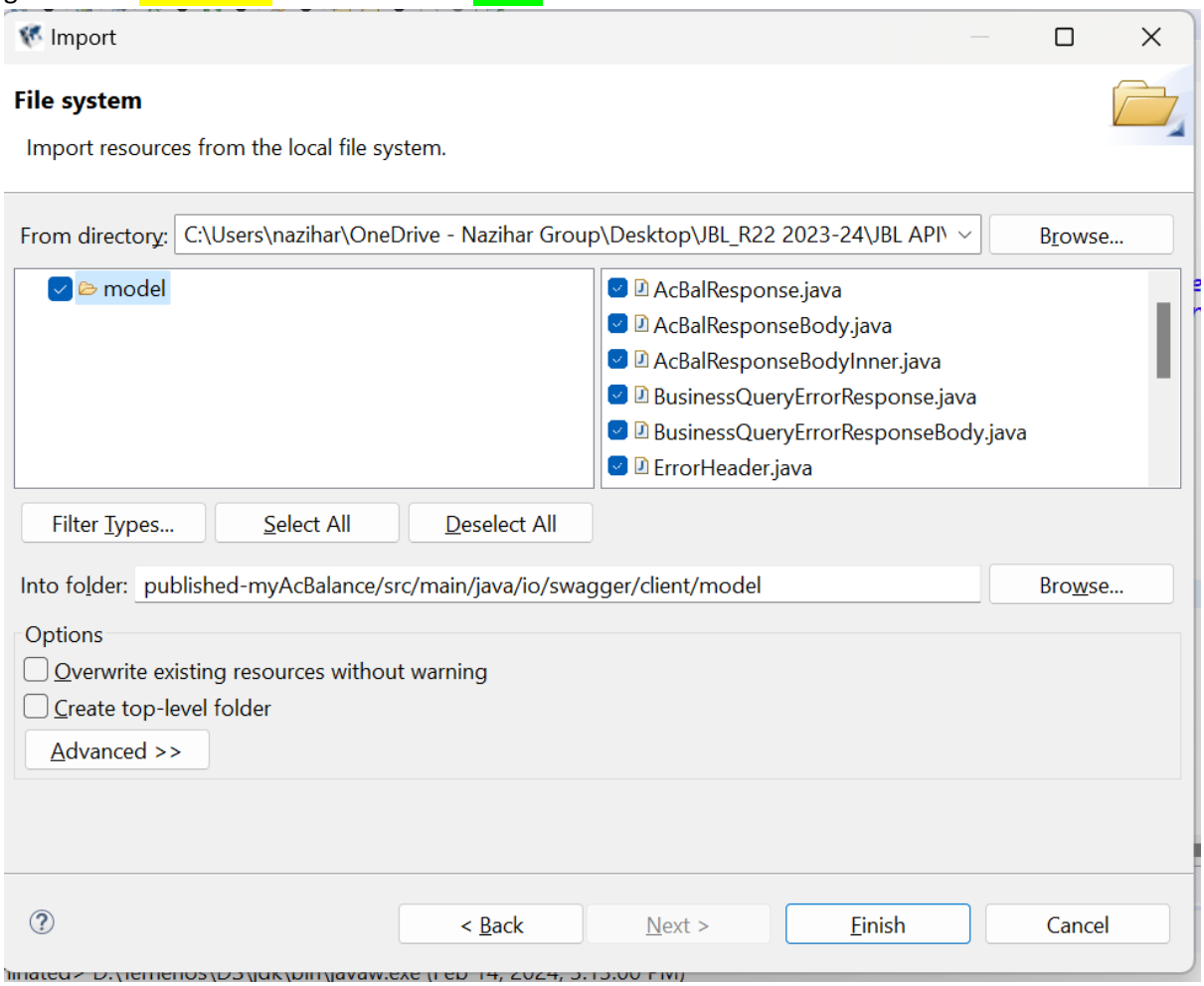

f. Right click src\main\java\io.swagger.client.model directory and select Import > File System -> navigate to the extracted zip file path\io\swagger\client\model and select this folder.

C:\Users\nazihar\OneDrive - Nazihar Group\Desktop\JBL_R22 2023-24\JBL API\All API Resources\published-myAcBal\java-client-generated\java-client\src\main\java\io\swagger\client\model

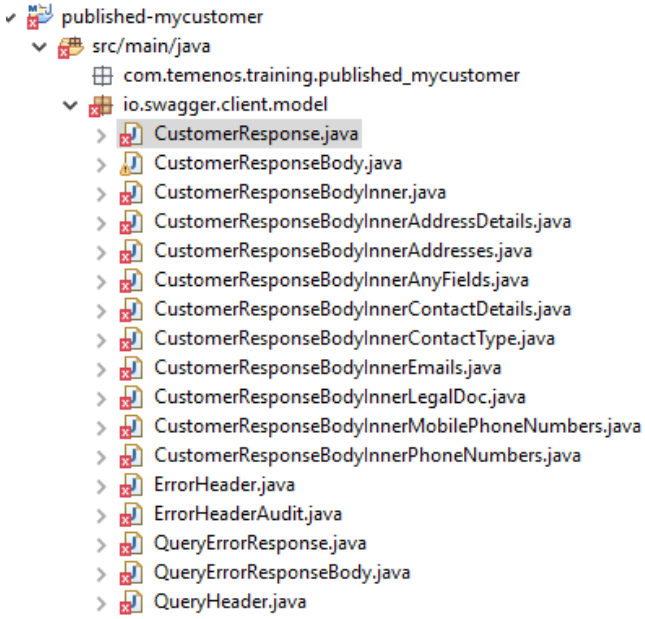




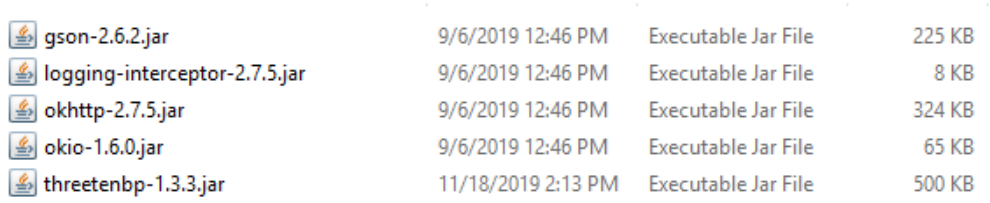

g. Click the "Select All" button and click Finish.

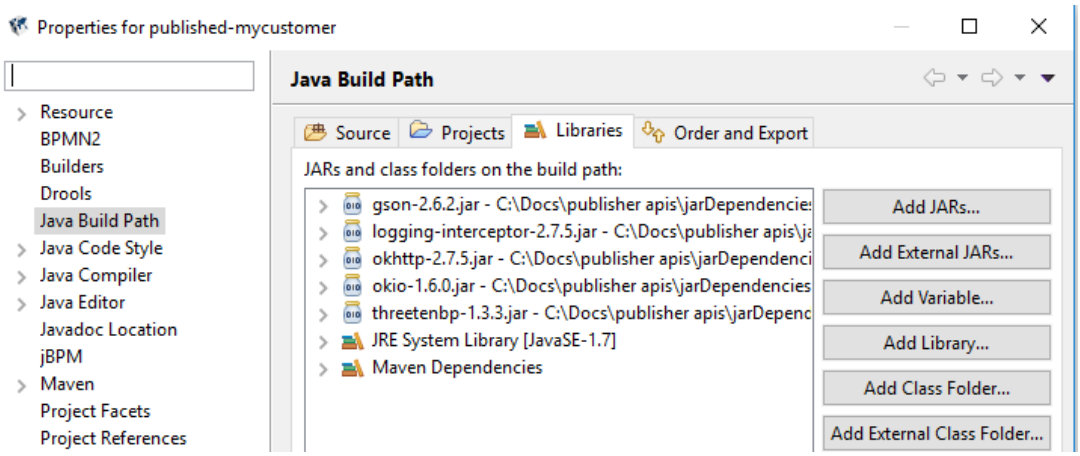



7. The imported java classes will throw error citing missing dependency jars. (DEPENDS on the SET-UP) – **May not Occur in NEW MB.**




8. To resolve these errors, download the below mentioned open-source jars (You can download it from any open-source jar download sites) and add it to the Build Path of the publisher API project (by right clicking the project > Configure Build Path > Libraries tab > Add External JARS > browse for the jars downloaded and select). **May not Occur in NEW MB.**




9. Edit the **pom.xml** of the publisher API project by adding the following dependencies.

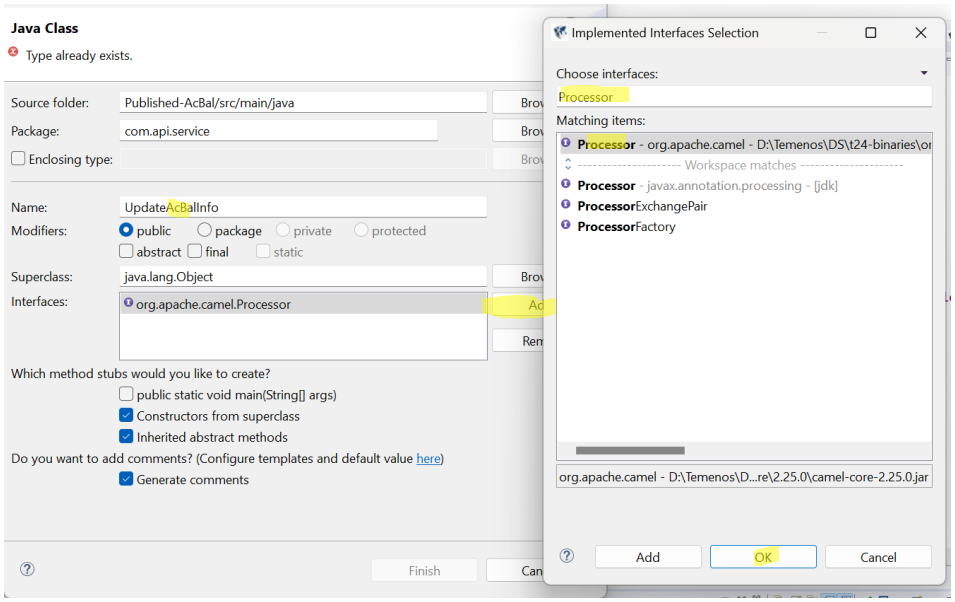
---

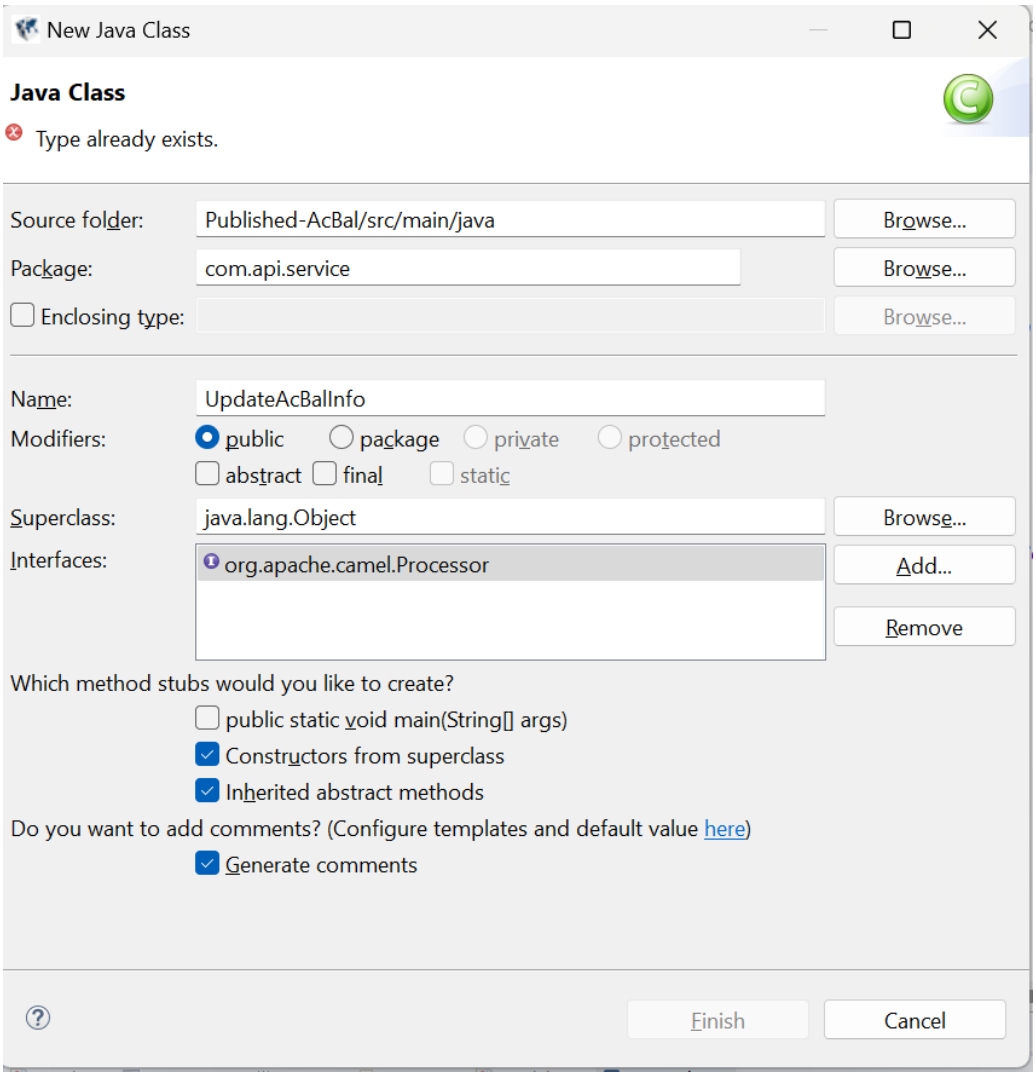
```
<dependency>
    <groupId>com.squareup.okhttp</groupId>
```

```
        <artifactId>okhttp</artifactId>
        <version>2.7.5</version>
    </dependency>
    <dependency>
        <groupId>com.squareup.okhttp</groupId>
        <artifactId>logging-interceptor</artifactId>
        <version>2.7.5</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.6.2</version>
    </dependency>
    <dependency>
        <groupId>org.threeten</groupId>
        <artifactId>threetenbp</artifactId>
        <version>1.3.3</version>
    </dependency>
```

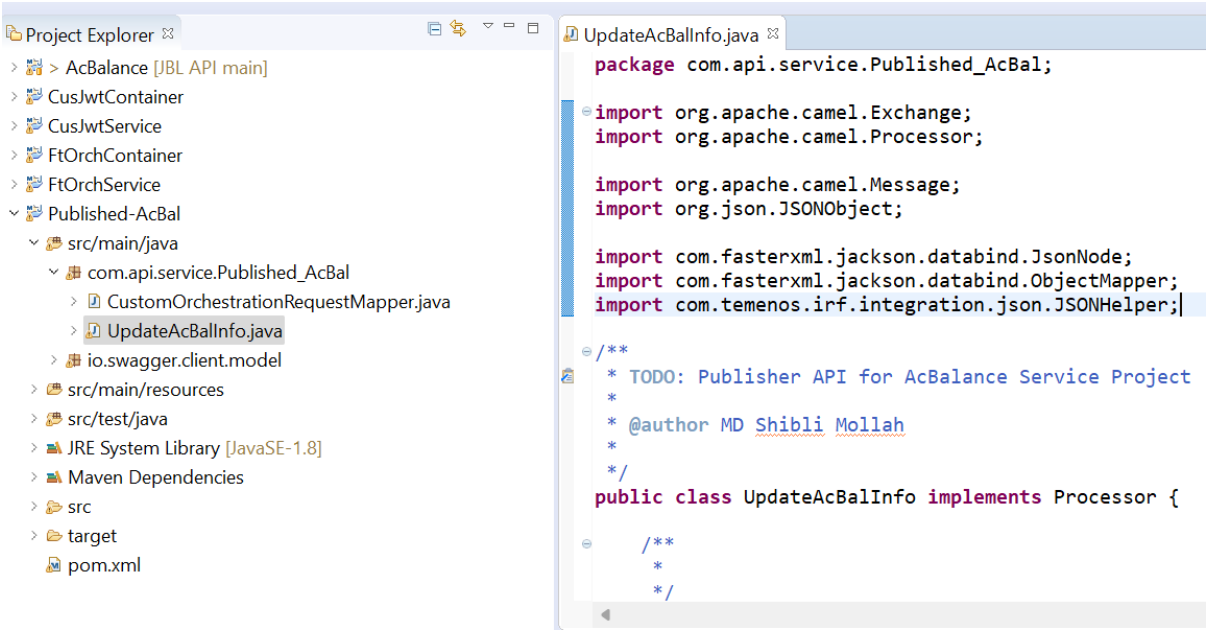
---

**10.** Navigate to **publisherProject - Published-AcBal <<publisherProject>>\src\main\java\ com.api.service** package directory, right click and select **New -> Java -> Class** and enter the class name as **<<AnyRelevantName>>Mapper** and add the interface named “Processor” under Interfaces.






11. The mapper class will be generated as shown below.



The process() method in the mapper must have the code logic to transform the provider API to publisher API and vice versa.
The code logic must be in such a way that the fields present in the **JSON response** body of the **provider** API are mapped to the fields defined in the swagger of **published** API using the **published** API swagger classes generated.

**Mapper class for ENQUIRY based API:**

For **ENQUIRY** API, we will require only a **response** mapper since we do not send a request body.

Below are the steps to create a Response Mapper for an ENQUIRY API based on ACCOUNT application.

**a.** As mentioned in **step 10**, create a **mapper class** with a relevant name (**UpdateAcBalInfo**) and Processor interface. In the generated mapper java file, comment the constructor (if not required). Only the **process** method will be used.

**b. To map the fields of the provider API**, we must first receive the header and JSON body content (containing header and body section of JSON response) of the provider API after ensuring the http response code is **200**. Below is the corresponding code line. On confirming it is a successful response, we obtain the JSON response – header and body section.

```
    public void process(Exchange exchange) throws Exception {
     // TODO Auto-generated method stub
        String HttpResponseCode =
exchange.getIn().getHeaders().get("CamelHttpResponseCode").toString();
        Message in = exchange.getIn();
        String response = (String) in.getBody(String.class);
        JSONObject = new JSONObject(response);
        JSONObject jsonHeader = jsonObject.getJSONObject("header");
        String updateResponse = null;

        if (HttpResponseCode.equals("200")) {

            JSONObject jsonBody =
jsonObject.getJSONArray("body").getJSONObject(0);
```

Receive the header and body content of provider JSON response.

```
String responseCode = null;
            String accountNo = null;
            String alternateAccountNo = null;
            String currency = null;
            String balance = null;

            try {
                responseCode = jsonBody.get("response").toString();
                accountNo = jsonBody.get("accountId").toString();
                alternateAccountNo = jsonBody.get("accountIBAN").toString();
                currency = jsonBody.get("currency").toString();
                balance = jsonBody.get("availableBalance").toString();
```

```
            } catch (Exception e) {}

            jsonBody.remove("accountId");
            jsonBody.remove("accountIBAN");
           // jsonBody.remove("currency");
            jsonBody.remove("availableBalance");

            jsonBody.put("accountNo", accountNo);
            jsonBody.put("alternateAccountNo", alternateAccountNo);
            jsonBody.put("balance", balance);

            jsonHeader.remove("audit");
            jsonHeader.remove("page_start");
            jsonHeader.remove("page_token");
            jsonHeader.remove("total_size");
            jsonHeader.remove("page_size");
            jsonHeader.remove("status");

            jsonHeader.put("statusCode", responseCode);
            updateResponse = "{\"header\":" + jsonHeader + ",\"body\":" + jsonBody
+ "}";
        }

        ObjectMapper om = JSONHelper.createObjectMapper();
        JsonNode modifiedResponse = om.readTree(updateResponse);
        in.setBody(modifiedResponse);
    }
```

**Sample Mapper Class:**

UpdateAcInfo.java

UpdateAcBalInfo.java

c. Edit the **service.xml** of the publisher API by adding this **ResponseMapper (*UpdateAcBalInfo*)** as a process after redirecting it to the provider API.

party-v1.0.0-published-myacbal-service-v1.0.0.xml

```
<camelContext xmlns="http://camel.apache.org/schema/spring" id="party-v1.0.0-published
    <onException>
        <exception>java.lang.Exception</exception>
        <handled>
            <constant>true</constant>
        </handled>
        <process ref="exceptionHandler"/>
    </onException>
    <restConfiguration component="servlet" producerApiDoc="published-myAcBal-v1.0.0-sw
    <rest path="/v1.0.0/party/v1.0.0" produces="application/json" id="party-v1.0.0.ser
        <get uri="/party/accounts/{accountId}" id="getAcBal">
            <param name="accountId" type="path" required="true"/>
            <!-- Publisher API Uri -->
            <to uri="direct-vm:party-v1.0.0.v1.0.0.getAcBal"/>
        </get>
    </rest>
    <route id="direct-vm.party-v1.0.0.v1.0.0.getAcBal">
        <from uri="direct-vm:party-v1.0.0.v1.0.0.getAcBal"/>
        <!-- Provider API Uri -->
        <to uri="direct-vm:party-accounts.v1.0.0.getAcBal"/>
        <process ref="UpdateAcBalInfo"/>
    </route>
    <route id="direct.mockResponder">
        <from uri="direct:mockResponder"/>
        <process ref="mockResponder"/>
    </route>
</camelContext>
<bean id="UpdateAcBalInfo" class="com.api.service.UpdateAcBalInfo"/>
</beans>
```

```
<get uri="/party/accounts/{accountId}" id="getAcBal">
            <param name="accountId" type="path" required="true"/>
            <!-- Publisher API Uri -->
            <to uri="direct-vm:party-v1.0.0.v1.0.0.getAcBal"/>
        </get>
    </rest>
    <route id="direct-vm.party-v1.0.0.v1.0.0.getAcBal">
        <from uri="direct-vm:party-v1.0.0.v1.0.0.getAcBal"/>
        <!-- Provider API Uri -->
        <to uri="direct-vm:party-accounts.v1.0.0.getAcBal"/>
        <process ref="UpdateAcBalInfo"/>
    </route>
    <route id="direct.mockResponder">
        <from uri="direct:mockResponder"/>
        <process ref="mockResponder"/>
    </route>
```

**Add the bean:**

```
<bean id="UpdateAcBalInfo" class="com.api.service.UpdateAcBalInfo"/>
```

party-v1.0.0-published-myacbal-service-v1.0.0.xml

**Special Note:** We must redirect the URI of the other versions of the service xml's of the Publisher API by changing like this, otherwise war file deployment will be failed.

```
<route id="direct-vm.party-v1.0.0.v1.0.getAcBal">
        <from uri="direct-vm:party-v1.0.0.v1.0.getAcBal"/>
        <!-- OWN PUBLISHER API Uri -->
        <to uri="direct-vm:party-v1.0.0.v1.0.0.getAcBal"/>
</route>
```

party-v1.0.0-published-myacbal-service-v1.0.xml

```
    xmlns:camel="http://camel.apache.org/schema/spring"
    xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://camel.apache.org/schema/spring
       http://camel.apache.org/schema/spring/camel-spring.xsd">
<camelContext xmlns="http://camel.apache.org/schema/spring" id="party-v1.0.0-published-myAcBal.service.v1.0">
    <onException>
        <exception>java.lang.Exception</exception>
        <handled>
            <constant>true</constant>
        </handled>
        <process ref="exceptionHandler"/>
    </onException>
    <restConfiguration component="servlet" producerApiDoc="published-myAcBal-v1.0-swagger.json" bindingMode="aut
    <rest path="/v1.0/party/v1.0.0" produces="application/json" id="party-v1.0.0.service.restlet">
        <get uri="/party/accounts/{accountId}" id="getAcBal">
            <param name="accountId" type="path" required="true"/>
            <to uri="direct-vm:party-v1.0.0.v1.0.getAcBal"/>
        </get>
    </rest>
    <route id="direct-vm.party-v1.0.0.v1.0.getAcBal">
        <from uri="direct-vm:party-v1.0.0.v1.0.getAcBal"/>
        <!-- OWN PUBLISHER API Uri -->
        <to uri="direct-vm:party-v1.0.0.v1.0.0.getAcBal"/>
    </route>
</camelContext>

</beans>
```

party-v1.0.0-published-myacbal-service-v1.xml

```
    xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://camel.apache.org/schema/spring
       http://camel.apache.org/schema/spring/camel-spring.xsd">
<camelContext xmlns="http://camel.apache.org/schema/spring" id="party-v1.0.0-published-myAcBal.service.v1">
    <onException>
        <exception>java.lang.Exception</exception>
        <handled>
            <constant>true</constant>
        </handled>
        <process ref="exceptionHandler"/>
    </onException>
    <restConfiguration component="servlet" producerApiDoc="published-myAcBal-v1-swagger.json" bindingMode="auto"
    <rest path="/v1/party/v1.0.0" produces="application/json" id="party-v1.0.0.service.restlet">
        <get uri="/party/accounts/{accountId}" id="getAcBal">
            <param name="accountId" type="path" required="true"/>
            <to uri="direct-vm:party-v1.0.0.v1.getAcBal"/>
        </get>
    </rest>
    <route id="direct-vm.party-v1.0.0.v1.getAcBal">
        <from uri="direct-vm:party-v1.0.0.v1.getAcBal"/>
        <!-- OWN PUBLISHER API Uri -->
        <to uri="direct-vm:party-v1.0.0.v1.0.0.getAcBal"/>
    </route>
</camelContext>

</beans>
```

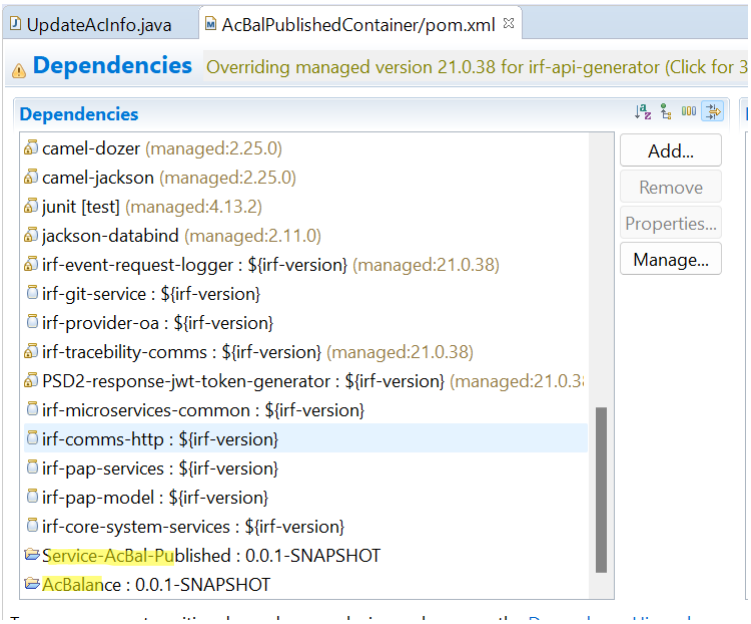
party-v1.0.0-published-myacbal-service-v1.0.xml

party-v1.0.0-published-myacbal-service-v1.xml

11. Maven Install the **publisher API** project(s).

Published-AcBal-0.0.1-SNAPSHOT.jar

12. Include the **publisher API** and **provider API** project(s) dependency in the **container** project's pom.xml and perform maven install.



```xml
        <dependency>
            <groupId>com.api.service</groupId>
            <artifactId>Service-AcBal-Published</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>AcBal</groupId>
            <artifactId>AcBalance</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
```

14. Create a new bean with any relevant name for “id” attribute in **applicationContext.xml** of the container project for the mapper class created. Enable Basic Auth/JWT required.

```xml
<!-- Response mapper added -->
      <bean id="UpdateAcBalInfo" class="com.api.service.UpdateAcBalInfo">
      </bean>
```

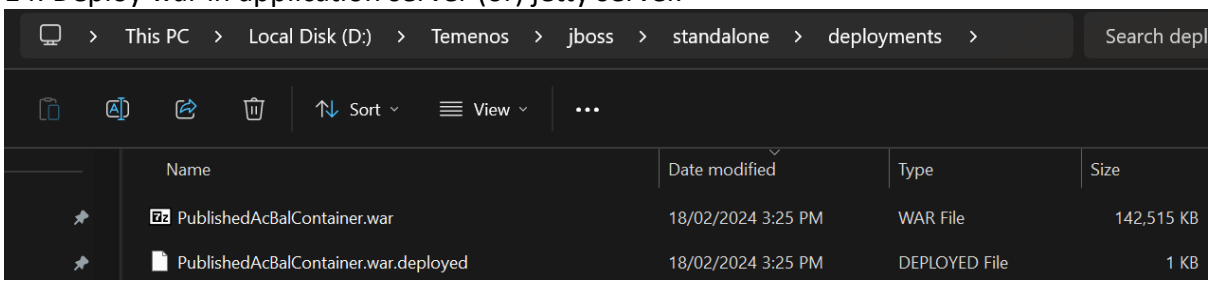
applicationContext.xml

```xml
    <!-- Response mapper added -->
    <bean id="UpdateAcBalInfo" class="com.api.service.UpdateAcBalInfo">
    </bean>

    <!-- <bean id="t24SecurityFilter" class="com.temenos.irf.comms.security.defaultimpl.NullBean" /> -->

    <!-- <bean id="t24SecurityFilter" class="com.temenos.irf.security.t24.T24SpringSecurityContextFilter" /> -->

    <!-- Comment the above bean with id t24SecurityFilter and uncomment below bean for Security Filter  -->
    <!--<bean id="t24SecurityFilter" class="com.temenos.irf.comms.security.defaultimpl.T24Security" /> -->

    <!-- Comment the above bean with id t24SecurityFilter and uncomment below bean for Basic Authentication -->
    <bean id="t24SecurityFilter" class="com.temenos.irf.comms.security.defaultimpl.T24BasicAuthenticationCheck" />

    <bean id="serviceLocatorProperties" class="com.temenos.irf.config.StandardPropertyReader">
        <property name="path" value="classpath:/irf-config/service-locator.properties" />
    </bean>

    <bean id="mockDataMgmtProcessor" class="com.temenos.irf.mock.MockDataMgmtProcessor">
        <!-- basePath should be set to the base Directory where mockFiles folder is placed.  -->
        <property name="basePath" value="classpath:/irf-config/"></property>
    </bean>
```

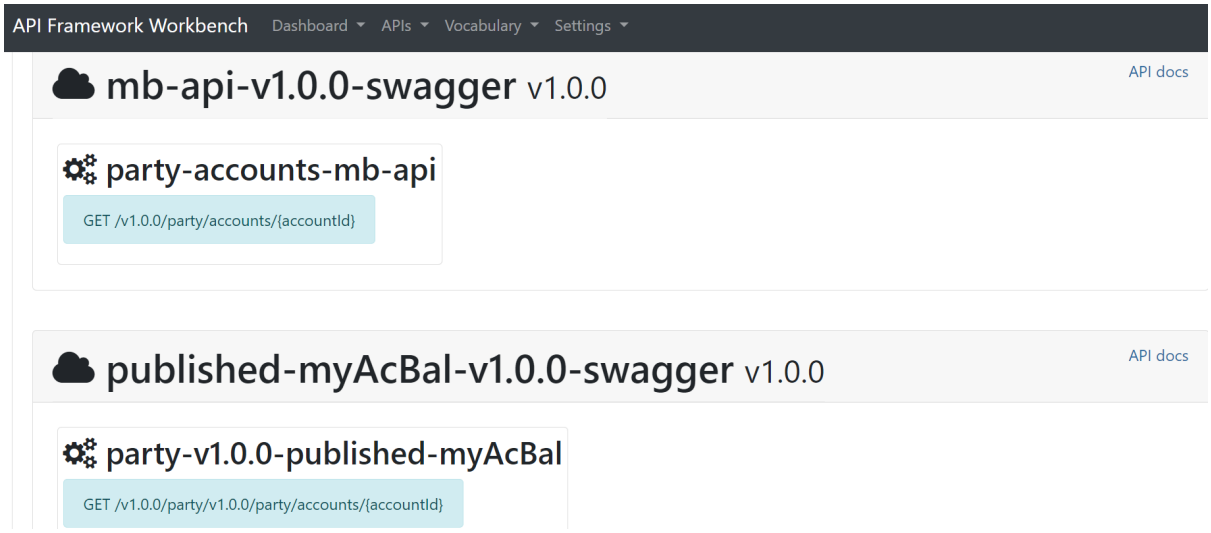
**SAVE it.**

PublishedAcBalContainer.war

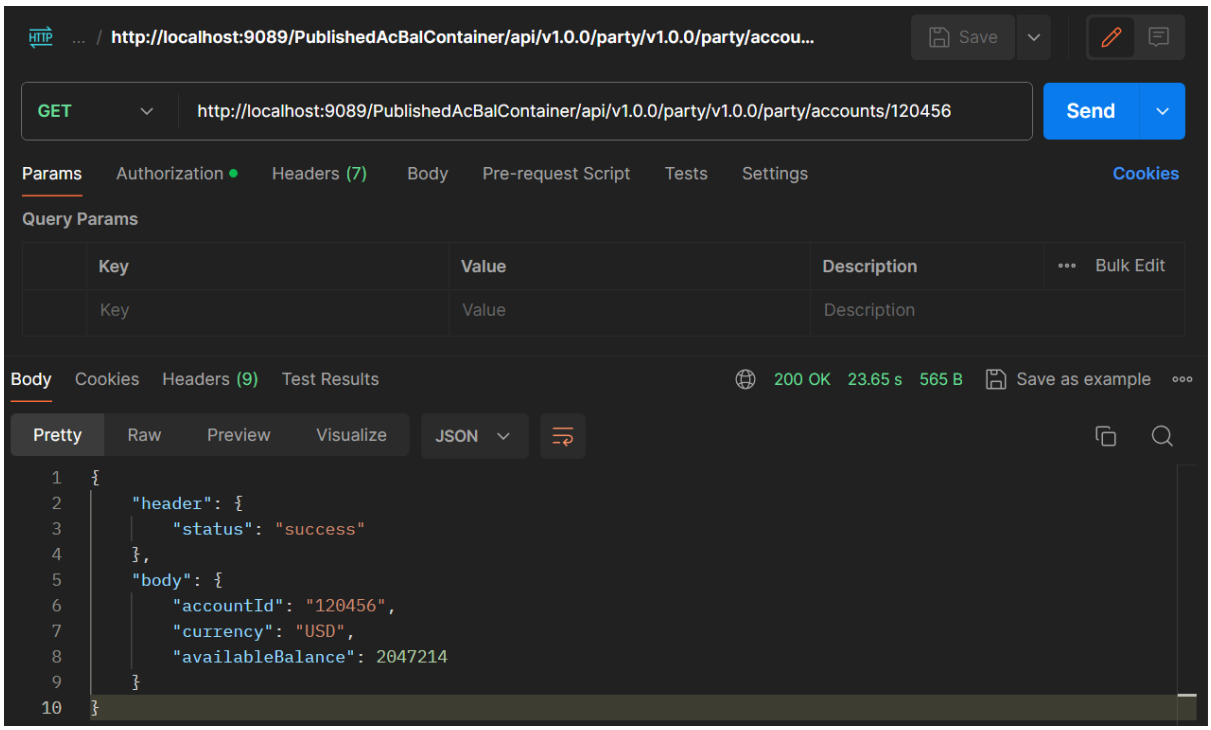
14. Deploy war in application server (or) jetty server.

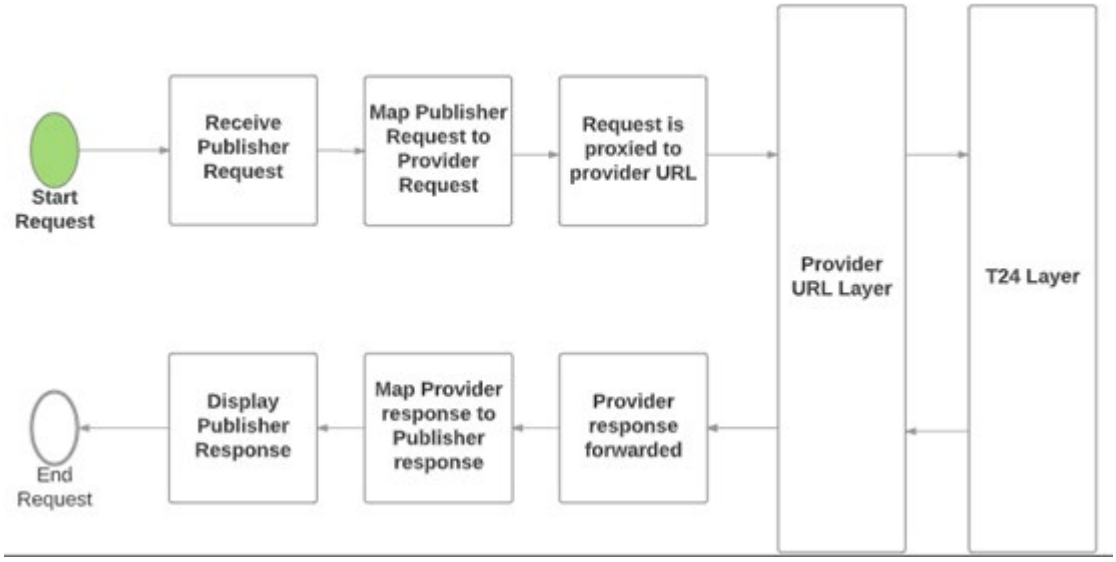



15. APIs can now be accessed with structure of published api json and the corresponding response is now formed as per requirement.




Below screenshots show the JSON requests and responses for the examples explained in this document.




Start Request
Receive Publisher Request
Map Publisher Request to Provider Request
Request is proxied to provider URL
Provider URL Layer
T24 Layer
End Request
Display Publisher Response
Map Provider response to Publisher response
Provider response forwarded