Three Layer to Two layer System connector Process layer:

author @indranil banerjee

Demo Done To group Team @Lindiwe Ncala @Robin Martin @Yusuf ISAH

as discussed with Lukumon.Balogun@mtn.com for the current architecture which is having experience, aggregator and system layer should me simplified to two layer process and connector layer.

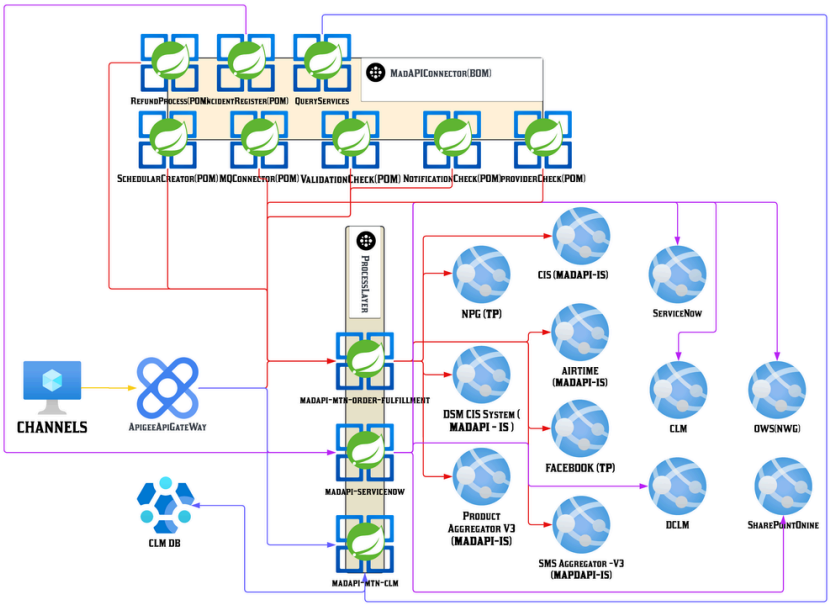
AC-22: Three Layer to Two layer System connector Process layer: DONE

Objective

- create a 2 tire architecture.
- create a template for the solution

[Architecture Diagram:] [Template:] [Creating CommonConnector:] [Step One :] [STEP2:] [mtnBackendConnectorTemplate:] [mtnvalidationConnector:] [Deployment for supporting jars:] [Source code:] [ ] [Latest code] [Creating ProcessLayer Calling CommonConnector:] [Step1:] [Step2:] [Latest Code:] [Source Code:] [Creating ProcessLayer Consuming CommonConnector:] [Solution Approach:] [Output:] [Source Code:] [Latest Code:] [Benefit for CommonConnector Architecture:]

Architecture Diagram:



this architecture is based on the detail provided in API Information - Back end system - Indranil.xlsx (sharepoint.com) .

Template:

Creating CommonConnector:

It hold 4 component

```
<module>mtnCommonBean</module>
<module>mtnBackendConnectorTemplate</module>
<module>mtnBackendConnectorSalary</module>
<module>mtnBackendConnectorUserProfile</module>
```

`mtnCommonBean` **-library**

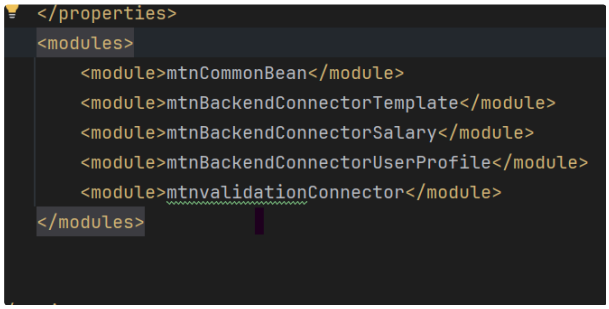
`mtnBackendConnectorTemplate` **-library**

`mtnBackendConnectorSalary` **-injectable microservice**

`mtnBackendConnectorUserProfile` **-injectable microservice**

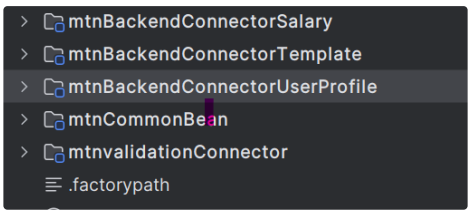
Step One :

- .



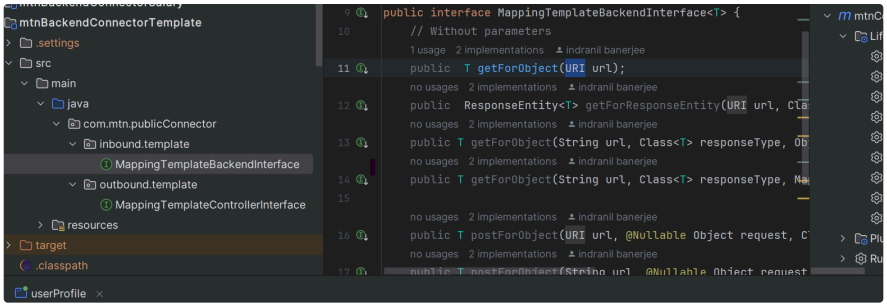
STEP2:

- create the common connector in here there is two ***injectable*** connector ***mtnBackendConnectorUserProfile*** , ***mtnBackendConnectorSalary*** supporting library is ***mtnCommonBean,*** and ***mtnBackendConnectorTemplate***



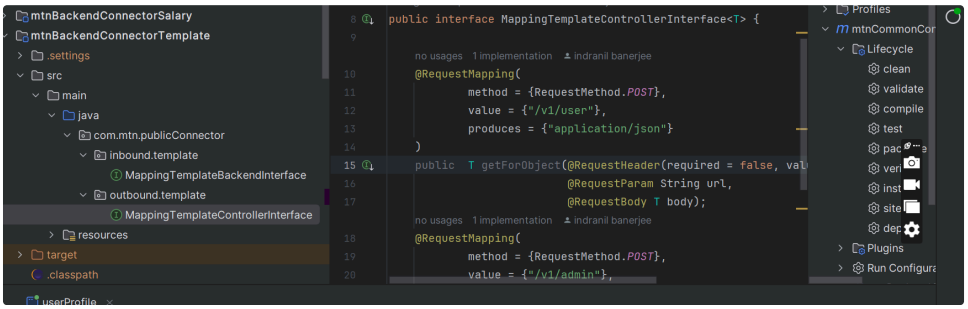
mtnBackendConnectorTemplate:

- this is suppose to connect to and back end Connection that is why generic template has been used one is to connect to the backend services.



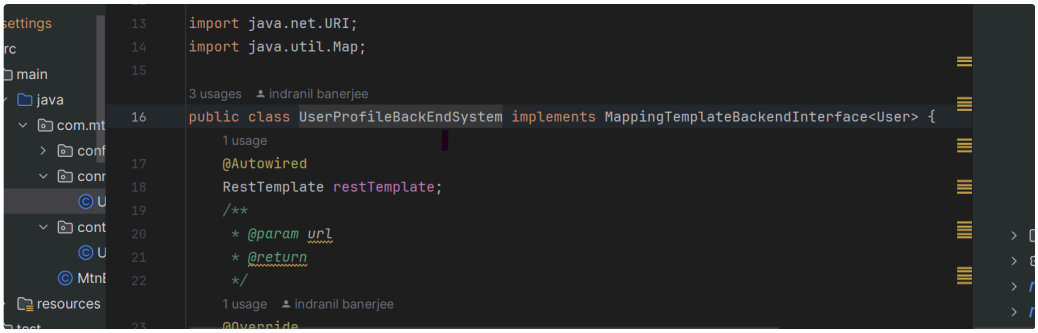
Backend connector

- Another is suppose to connect to the frontend services template.

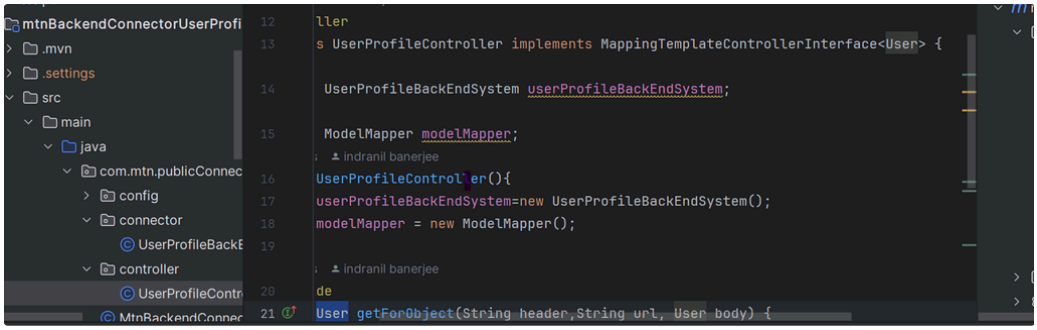


frontend template

Any **injectable** microservice trying use this two kind of template microservice trying to use the template has to **implement** ***MappingTemplateBackendInterface*** *for* creating backend connection and for frontend connector has to **implement** ***MappingTemplateControllerInterface*** for creating frontend controller.



backend connection creation



frontend controller

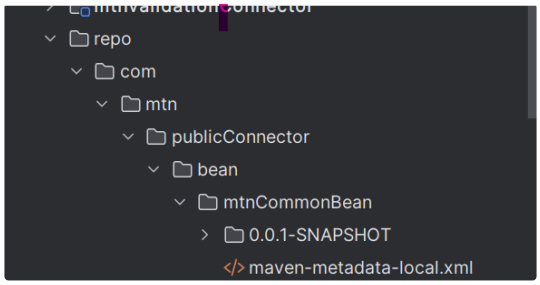
mtnvalidationConnector:

Please refer ***mtnvalidationConnector*** for better understanding.

Deployment for supporting jars:

the `artifact` should be publish in nexus it is not that is why not able to load the dependencies .

- [x] creating local repository @indranil banerjee work in progress.



Please go to link to better understanding.

- [x] Creating remote repository @indranil banerjee .

Please go to link.

Source code:

mtnCommonConnector - Copy.zip

Latest code

indranb13/mtnCommonConnector

Creating ProcessLayer Calling CommonConnector:

Step1:

properties should be injected in process layer

```
UserProfileController.java    TestAggregatorApplication.java    application.properties
spring.application.name=testAggregator
server.port=8084
logging.level.org.springframework.web: TRACE
userProfile.url.path=http://localhost:8081/v1/user?url=demoBackend%2Fsystem
custom.header.param=test
custom.header.value=test
```

Step2:

from process layer is should connect with the connector layer and give the desired output.

```
Curl
curl -X 'POST' \
  'http://localhost:8084/v1/user' \
  -H 'accept: application/json' \
  -d ''
Request URL
http://localhost:8084/v1/user
Server response
Code    Details
200     Response body
{
  "userId": 10,
  "name": "Indranil",
  "email": "indranil.banerjee@mtn.com",
  "mobile": "983670979",
  "gender": "male",
  "age": 10,
  "nationality": "Indian"
}
Download
```

Latest Code: indranb13/testProcess

Three Layer to Two layer System connector Process layer: | Latest Code:[inlineCard]

Source Code: testProcess.zip

Creating ProcessLayer Consuming CommonConnector:

As discussed with the group team @Robin Martin @Lindiwe Ncala @Valentine Nkosi @Yusuf ISAH they needed a **injectable microservice** in their process layer.

Objective : is to create a process layer which will consume injectable microservices.

Solution Approach:

please inject the dependency in the pom file.

```
<dependency>
    <groupId>com.mtn.publicConnector.backend</groupId>
    <artifactId>mtnBackendConnectorUserProfile</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
```

and then add the packages in your base package location .

```
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class},scanBasePackages
        "com.mtn.publicConnector.backend.mtnBackendConnectorUserProfile.connector",
        "com.mtn.publicConnector.backend.mtnBackendConnectorUserProfile.controller",
        "com.mtn.publicConnector.backend.mtnBackendConnectorUserProfile.config"

public class UserProcessorApplication {

    public static void main(String[] args) { SpringApplication.run(UserProcessor
```

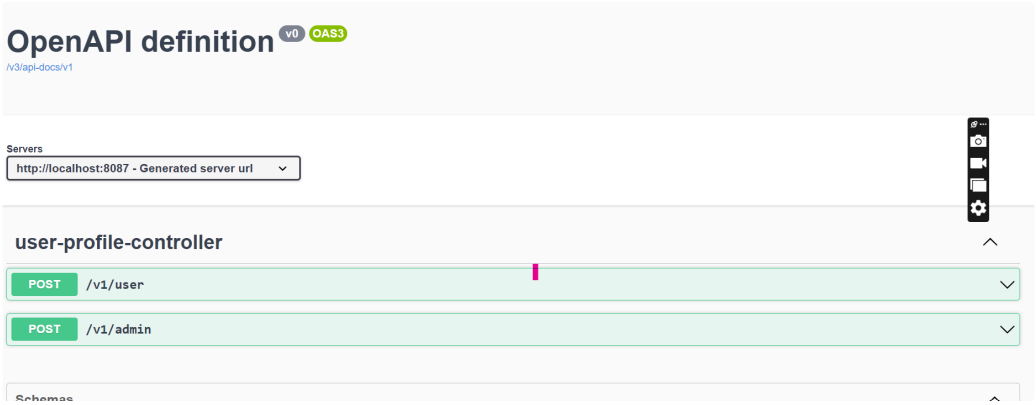
then create a bean factory to call the injectable microservices.

```
no usages
@Bean("userProfileController")
public UserProfileController createUserProfileController() { return new UserProfi
1 usage
@Bean("user")
public User getUserDetails(){
    UserProfileController userProfileController =app.getBean(UserProfileControlle
    return userProfileController.getForObject(HEADER_VALUE,USER_URL, CreateO
}
```

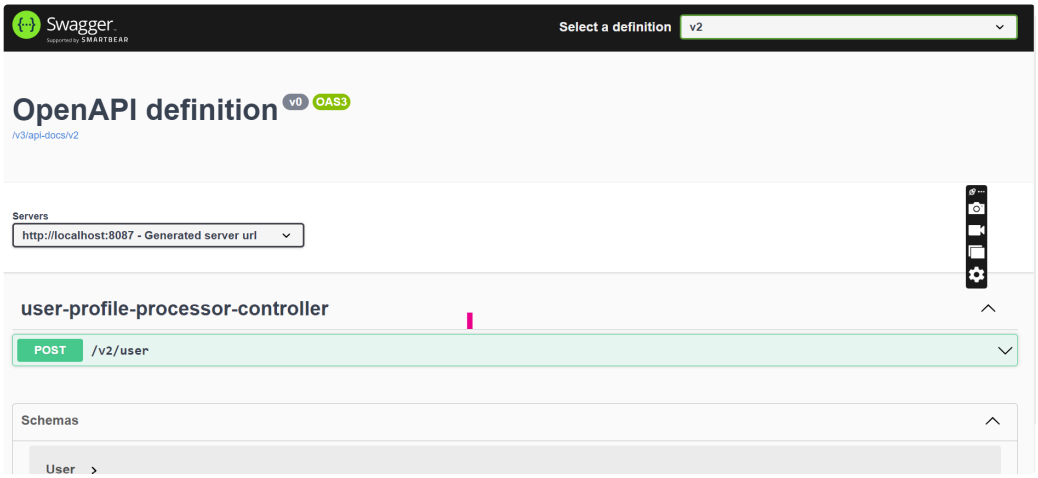
then call this microservices.

Output:

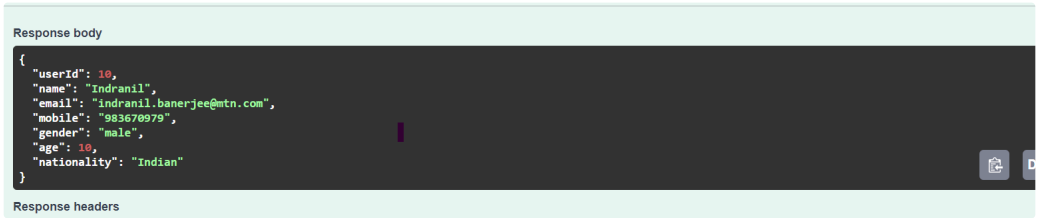
for V1 you can see the injectable microservice swagger.



for V2 you are getting the current microservice structure.



and it is working .



That is how one **Microservice in Injected in another** in this scenario **commonconnector** is injected in process layer.

Source Code: userProcessor.zip

Latest Code: indranb13/UserProcessor

indranb13/mtnCommonProcessor

Benefit for CommonConnector Architecture:

INDIVIGUAL COMPONENT	WHAY IT IS NECESSARY	BENEFIT
`mtnCommonBean` (library)	this will hold all the bean and entity and creation of those in nexus lib	so the group them will hold control of all the beans ,and entity that is there in the MADAPI and they will hold control of how the data objects are getting used and they can standardized it ,there is no duplication.

`mtnBackendConnectorTemplate` (library)	this will hold rest, soap, kafka, Mq template for backend and front end connectivity in nexus lib	so the definition for Connectivity signature is controlled by group team whether it is swagger or anything else and it is standardized it.
`mtnBackendConnectorUserProfile` (microservices)	this is a injectable microservice which is common for all user profile related business(public)	if that how we can segregate the microservices based on business.
`userProcessor`	this is the processor layer in which we will inject the common microservice and do zone specific business implementation (private)	so the zone specific business logic is expose to group.

Integration Testing processLayer:

- [x] @indranil banerjee to add integration test suit with **Cucumber and Feature File**.

please go to Link.