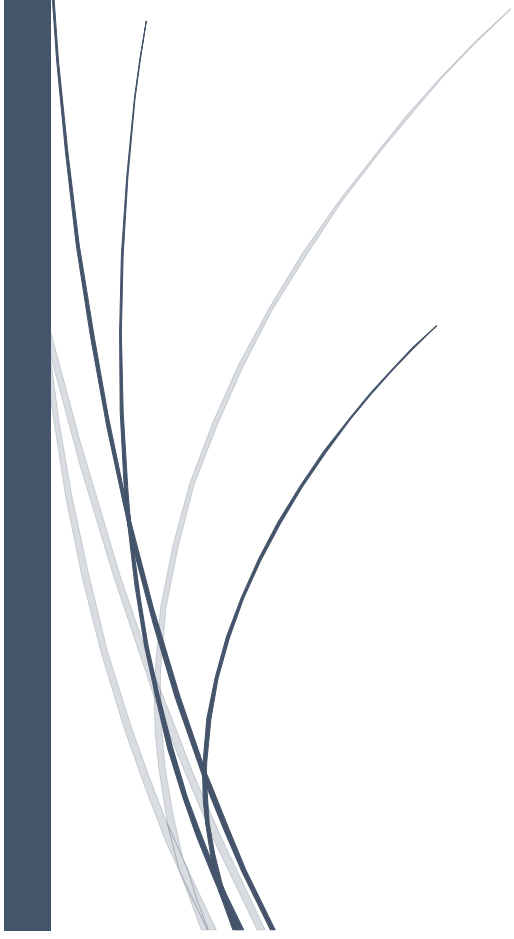
TP : Développer un service web multi connecteurs avec Rest API, GraphQL API, SOAP API et gRPC API



Architecture des composants d’entreprise

Table des matières

I. Objectif du TP

Mettre en œuvre un micro service avec Spring Boot avec les trois couches :

- La couche DAO (Data Access Layer) : avec Spring Data JPA.
- La couche métier (Business Layer).
- La couche présentation (Web Layer) avec les APIs suivantes :
  - ✓ GraphQL API.
  - ✓ REST API.
  - ✓ SOAP API.
  - ✓ gRPC API.

II. Prérequis

- IntelliJ IDEA ;
- JDK version 17 ;
- Une connexion Internet pour permettre à Maven de télécharger les librairies.
- Le TP relatif à GraphQL.

NB : *Ce TP a été réalisé avec IntelliJ IDEA 2023.2.3 (Ultimate Edition).*

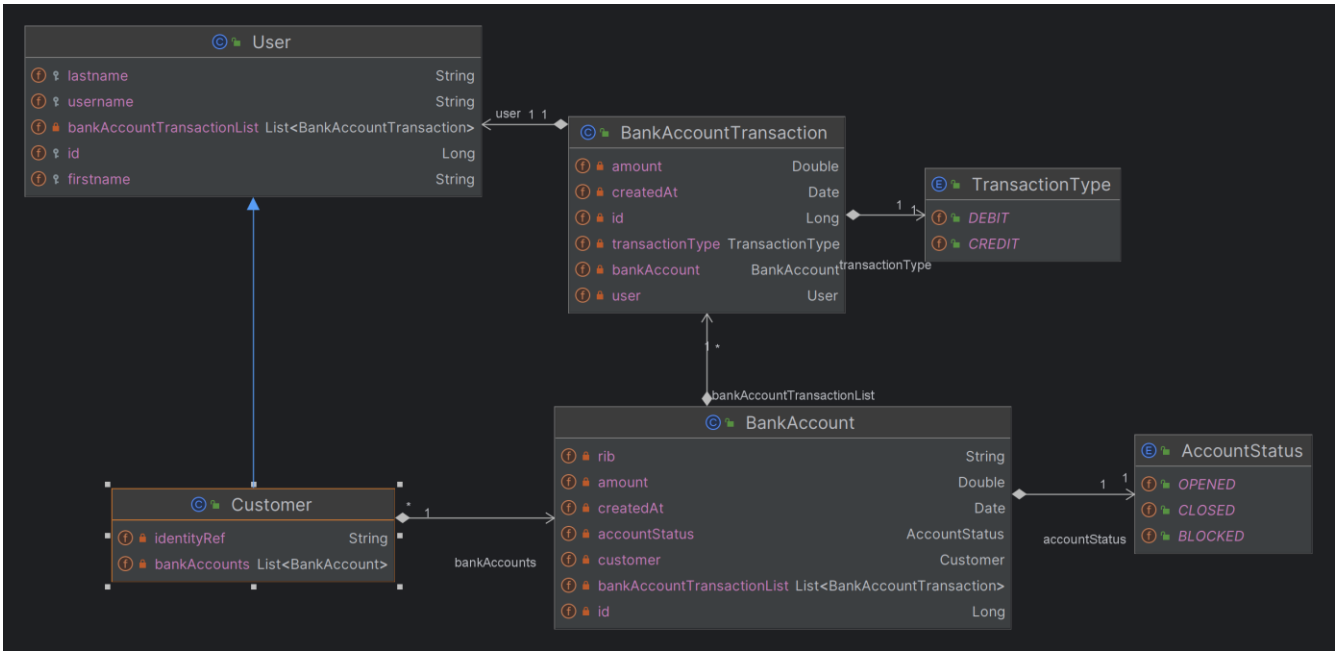
III. Développement de l'application

a. Le diagramme de classe

Nous allons prendre le même diagramme de classe que nous avons implémenté dans l'atelier relatif à GraphQL (https://github.com/abbouformations/bank-service-graphql.git) dans lequel nous avons utilisé GraphQL. Pour rappel, le service web offre les services suivants :

- Consulter la liste des clients de la banque.
- Consulter un client par son numéro d'identité.
- Modifier un client par son numéro d'identité.
- Supprimer un client par son numéro d'identité.
- Consulter la liste des comptes bancaires.
- Consulter un compte bancaire par son RIB.
- Effecteur des virements d'un compte vers un autre compte.

Le diagramme de classe est le suivant :



- Un client peut avoir un ou plieurs comptes bancaires.
- Sur un compte bancaire, le client peut effectuer une ou plusieurs transactions.
- Un client est un utilisateur.
- Un utilisateur (exemple : agent guichet) peut effectuer une ou plusieurs transactions.
- Un compte bancaire peut avoir les statuts suivants : OPENED, CLOSED ou bien BLOCKED.
- Une transaction est de deux types : DEBIT ou CREDIT.
- Aucune transaction ne peut être effectuée sur un compte « CLOSED » ou bien « BLOCKED ».
- Pour effectuer un virement, le solde du compte bancaire de l'émetteur doit être supérieur au montant du virement.
- L'identité du client est unique.
- Le nom d'utilisateur (*username*) est unique.

b. Clone de l'application

- Ouvrir un terminal dans Intellig.
- Se positionner par exemple dans le dossier c:\workspace\tp7 et lancer la commande suivante :

  git clone https://github.com/abbouformations/bank-service-graphql.git

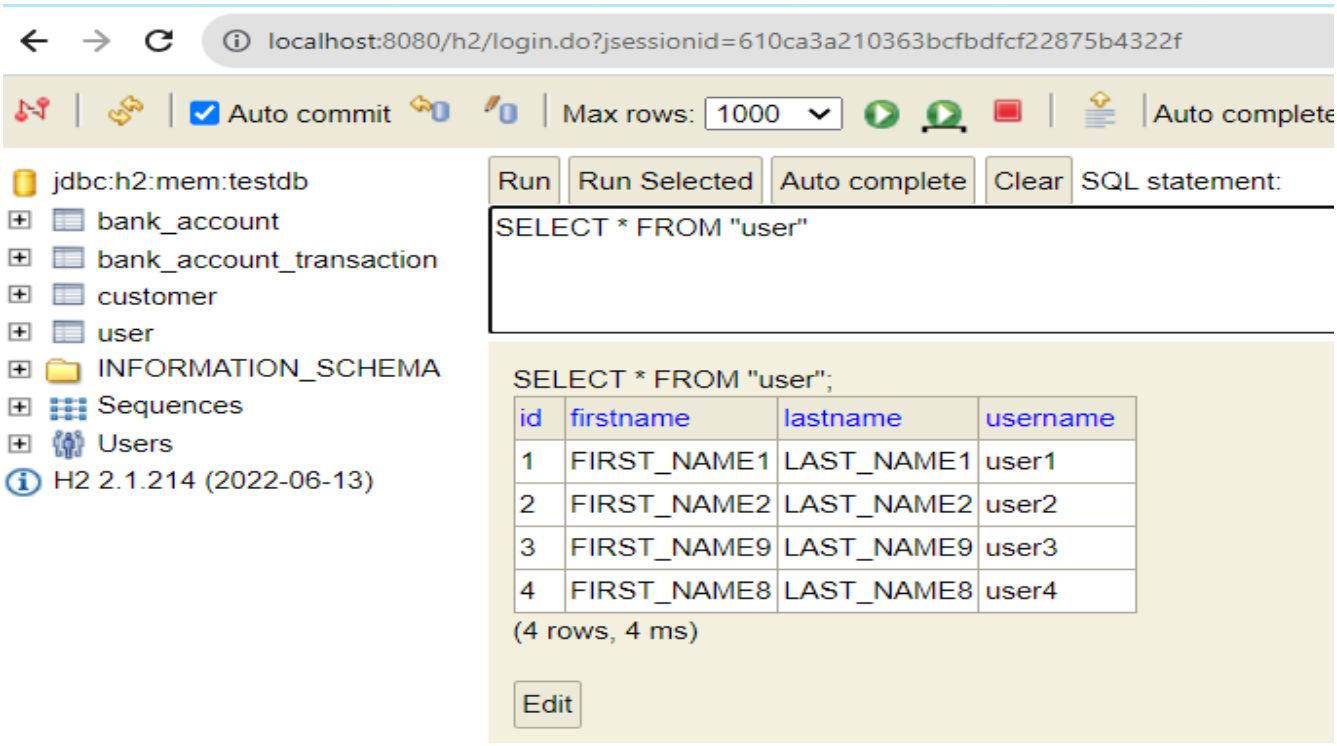
```
PS C:\workspace\emsi\2022-2023\tp7> git clone https://github.com/abbouformations/bank-service-graphql.git
```

- Vérifier que le projet nommé bank-service-graphql a été bien crée.
- Renommer le projet par exemple ***bank-service-multi-connector***.
- Modifier également le nom de l'ArtifactID dans pom.xml et ouvrir le projet avec Intellig.
- Renommer le package ma.formations.graphql par ***ma.formations.multiconnector.***

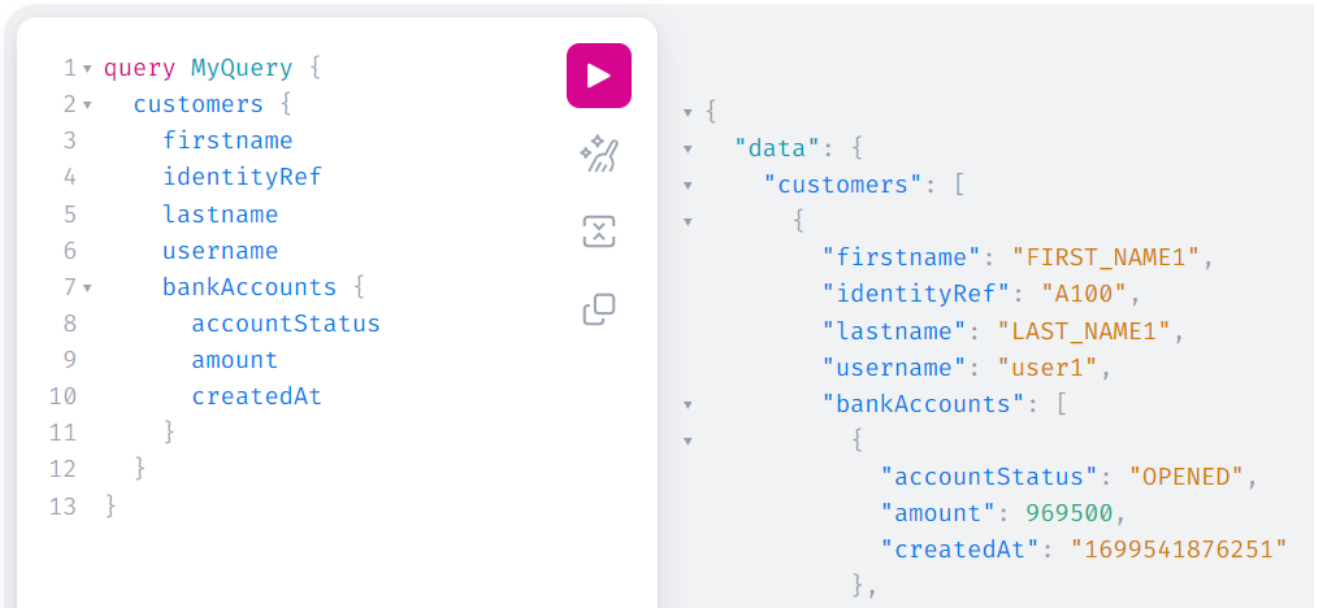
- Créer le package ***ma.formations.multiconnector.presentation.graphql*** et déplacer dans ce dernier les classes contrôleurs suivantes ***:*** *BankAccountGraphqlController, CustomerGraphqlController, TransactionGraphqlController.*

c. Tester l'api GRAPHQL

- Lancer la méthode main de la classe de démarrage.
- Accéder à la console h2 : http://localhost:8080/h2 et vérifier que les tables USER, CUSTOMER, BANK_ACCOUNT et BANK_ACCOUNT_TRANSACTION ont été bien créées et initialisées :



- Aller au lien : http://localhost:8080/graphiql?path=/graphql et tester les services en utilisant l'explorateur de GRAPHIQL :



Remarque :

Remarquez que grâce à l'interface de GraphiQL, vous avez pu connaître la liste des Endpoint fournis par le SW et également comment les consommer : **vous venez de développer une API GraphQL**.

NB : Pour plus de détails par rapport au développement du service web avec GraphQL, veuillez se référer à l'atelier n°4 dont le code source est disponible sur GITHUB : https://github.com/abbouformations/bank-service-graphql.git.

d. Développement du connecteur Rest (Rest API)

1. Développement du contrôleur Rest

- Ajouter la dépendance suivante au niveau du fichier pom.xml :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Explications :

- Pour la validation des données envoyées dans la requête. Par défaut, Spring utilise comme implémentation de l'API *Bean Validation* le Framework *Hibernate Validator*. L'api propose plusieurs annotations comme par exemple : @NotNull, @NotEmpty, @Max, ….
- Avec l'api Bean Validation, vous pouvez implémenter vos propres règles de gestion.

- Créer le package ***ma.formations.multiconnector.dtos.exception*** et ensuite créer la classe ***ErrorResponse*** suivante :

```
package ma.formations.multiconnector.dtos.exception;


import lombok.Getter;
import lombok.Setter;

import java.util.List;

@Getter
@Setter
public class ErrorResponse {
   private String message;
   private List<String> details;

   public ErrorResponse(String message, List<String> details) {
      super();
      this.message = message;
      this.details = details;
   }
}
```

Explications :

- Cette classe sera utilisée pour l'envoi des messages d'erreurs dans la réponse Rest.
- Au niveau de la couche Service, nous allons vérifier les règles de gestion métier si elles ont été bien respectées, le cas échéant nous allons lever l'exception ***BusinessException***. Cette même exception sera interceptée par la suite par la classe ***ExceptionHandlerController***

- Créer le package ***ma.formations.multiconnector.presentation.rest*** et ensuite créer les classes suivantes : *CustomerRestController, BankAccountRestController, TransactionRestController* et *ExceptionHandlerController :*

❖ La classe **CustomerRestController :**

```
package ma.formations.multiconnector.presentation.rest;

import jakarta.validation.Valid;
import ma.formations.multiconnector.dtos.customer.*;
import ma.formations.multiconnector.service.ICustomerService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/rest/customer")
@CrossOrigin("http://localhost:3000/")
```

```java
public class CustomerRestController {
    private final ICustomerService customerService;

    public CustomerRestController(ICustomerService customerService) {
        this.customerService = customerService;
    }

    @GetMapping("/all")
    List<CustomerDto> customers() {
        return customerService.getAllCustomers();
    }

    @GetMapping
    CustomerDto customerByIdentity(@RequestParam(value = "identity") String identity) {
        return customerService.getCustomByIdentity(identity);
    }

    @PostMapping("/create")
    public ResponseEntity<AddCustomerResponse> createCustomer(@RequestBody @Valid AddCustomerRequest dto) {
        return new ResponseEntity<>(customerService.createCustomer(dto), HttpStatus.CREATED);
    }

    @PutMapping("/update/{identityRef}")
    public ResponseEntity<UpdateCustomerResponse> updateCustomer(@PathVariable String identityRef, @RequestBody
@Valid UpdateCustomerRequest dto) {
        return new ResponseEntity<>(customerService.updateCustomer(identityRef, dto), HttpStatus.OK);
    }

    @DeleteMapping("/delete/{identityRef}")
    public ResponseEntity<String> deleteCustomer(@PathVariable String identityRef) {
        customerService.deleteCustomerByIdentityRef(identityRef);
        return new ResponseEntity<>(String.format("Customer with identity %s is removed", identityRef), HttpStatus.OK);
    }
}
```

Explications :

- Remarquer que nous avons annoté la classe par ***@CrossOrigin("http://localhost:3000/")*** . En fait, l'objectif est de permettre à l'application Javascript (que nous allons développer dans un autre atelier en utilisant React) de consommer le service web Rest de notre couche Back-end.

❖ La classe **BankAccountRestController :**

```java
package ma.formations.multiconnector.presentation.rest;

import jakarta.validation.Valid;
import ma.formations.multiconnector.dtos.bankaccount.AddBankAccountRequest;
import ma.formations.multiconnector.dtos.bankaccount.AddBankAccountResponse;
import ma.formations.multiconnector.dtos.bankaccount.BankAccountDto;
import ma.formations.multiconnector.service.IBankAccountService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
```

```java
import java.util.List;

@RestController
@RequestMapping("/api/rest/bank")
@CrossOrigin("http://localhost:3000/")
public class BankAccountRestController {
    private final IBankAccountService bankAccountService;

    public BankAccountRestController(IBankAccountService bankAccountService) {
        this.bankAccountService = bankAccountService;
    }

    @GetMapping("/all")
    List<BankAccountDto> bankAccounts() {
        return bankAccountService.getAllBankAccounts();
    }

    @GetMapping
    BankAccountDto bankAccountByRib(@RequestParam(value = "rib") String rib) {
        return bankAccountService.getBankAccountByRib(rib);
    }


    @PostMapping("/create")
    public ResponseEntity<AddBankAccountResponse> addBankAccount(@Valid @RequestBody AddBankAccountRequest
dto) {
        return new ResponseEntity<>(bankAccountService.saveBankAccount(dto), HttpStatus.CREATED);
    }
}
```

- La classe **TransactionRestController :**

```java
package ma.formations.multiconnector.presentation.rest;

import jakarta.validation.Valid;
import lombok.AllArgsConstructor;
import ma.formations.multiconnector.common.CommonTools;
import ma.formations.multiconnector.dtos.transaction.AddWirerTransferRequest;
import ma.formations.multiconnector.dtos.transaction.AddWirerTransferResponse;
import ma.formations.multiconnector.dtos.transaction.GetTransactionListRequest;
import ma.formations.multiconnector.dtos.transaction.TransactionDto;
import ma.formations.multiconnector.service.ITransactionService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;


@AllArgsConstructor
@RestController
@RequestMapping("/api/rest/transaction")
@CrossOrigin("http://localhost:3000/")
public class TransactionRestController {
```

```
    private ITransactionService transactionService;
    private CommonTools commonTools;

    @PostMapping("/create")
    public ResponseEntity<AddWirerTransferResponse> addWirerTransfer(@Valid @RequestBody
AddWirerTransferRequest dto) {
        return new ResponseEntity<>(transactionService.wiredTransfer(dto), HttpStatus.CREATED);
    }

    @GetMapping
    public List<TransactionDto> getTransactions(GetTransactionListRequest dto) {
        return transactionService.getTransactions(dto);
    }
}
```

- La classe **ExceptionHandlerController :**

```
package ma.formations.multiconnector.presentation.rest;


import ma.formations.multiconnector.dtos.exception.ErrorResponse;
import ma.formations.multiconnector.service.exception.BusinessException;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.HttpStatusCode;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

import java.util.ArrayList;
import java.util.List;
@ControllerAdvice
public class ExceptionHandlerController extends ResponseEntityExceptionHandler {
    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
HttpHeaders headers, HttpStatusCode status, WebRequest request) {
        List<String> details = new ArrayList<>();
        for (ObjectError error : ex.getBindingResult().getAllErrors()) {
            details.add(error.getDefaultMessage());
        }
        ErrorResponse error = new ErrorResponse("Validation Failed", details);
        return new ResponseEntity(error, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(value = BusinessException.class)
    public final ResponseEntity<Object> handleBusinessException(BusinessException ex, WebRequest request) {
        List<String> details = new ArrayList<>();
        details.add(ex.getLocalizedMessage());
        ErrorResponse error = new ErrorResponse("Functional errors", details);
        return new ResponseEntity(error, HttpStatus.BAD_REQUEST);
    }
```

```
  @ExceptionHandler(Exception.class)
  public final ResponseEntity<Object> handleOtherExceptions(Exception ex, WebRequest request) {
    List<String> details = new ArrayList<>();
    details.add(ex.getMessage());
    ErrorResponse error = new ErrorResponse("Technical error, please consult your administrator", details);
    return new ResponseEntity(error, HttpStatus.INTERNAL_SERVER_ERROR);
  }
}
```

Explications :

- Avec cette classe, nous avons implémenté le Design Pattern ***AOP*** (Aspect Oriented Programming). En effet, nous avons externalisé le code ayant pour vocation le traitement des exceptions de notre couche métier. Remarquer que cette classe intercepte et traite 03 types d'exceptions :
  - ✓ Les exceptions levées par l'api Bean Validation (il s'agit de la classe exception : ***MethodArgumentNotValidException***) qui sont traitées par la méthode *handleMethodArgumentNotValid*. Remarquer que la classe *ExceptionHandlerController* hérite de la classe ***ResponseEntityExceptionHandler*** de Spring et ceci afin que la JVM exécute la méthode redéfinie au lieu d'exécuter la méthode de la classe ***ResponseEntityExceptionHandler*** (le principe de Polymorphisme).
  - ✓ Les exceptions de type ***BusinessException*** levées par la couche Service.
  - ✓ Les autres exceptions (par exemple : panne de réseau, …).

2. Configuration de l'api OpenAPI et SWAGGER

- Ajouter la dépendance suivante au niveau du fichier pom.xml :

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.2.0</version>
</dependency>
```

- Ajouter les 03 lignes suivantes au niveau du fichier ***application.properties*** :

```
springdoc.api-docs.path=/api/rest/docs
springdoc.swagger-ui.path=/api/rest/docs-ui
springdoc.swagger-ui.operationsSorter=method
```

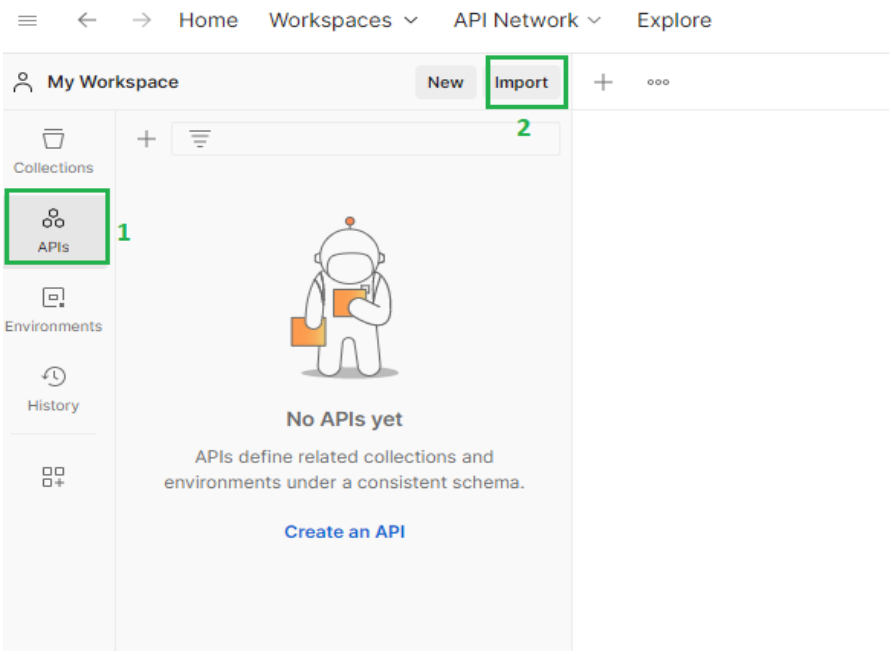
Explications :

- La clé ***springdoc.api-docs.path*** permet de personnaliser le lien pour accéder à l'interface d'OpenAPI.

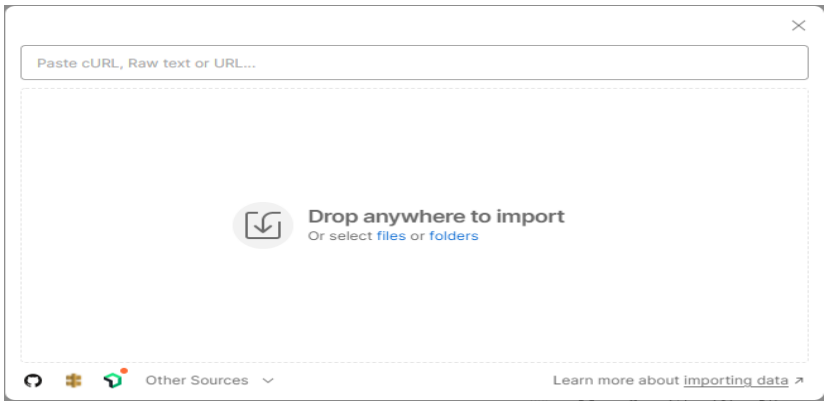
- La clé ***springdoc.swagger-ui.path*** permet de personnaliser le lien pour accéder à l'interface de Swagger afin de tester les EndPoints de notre SW Rest.
- La clé ***springdoc.swagger-ui.operationsSorter*** permet de trier par ordre alphabétique les méthodes au niveau de l'interface de Swagger.

3. Importer l'interface du SW au niveau de Postman

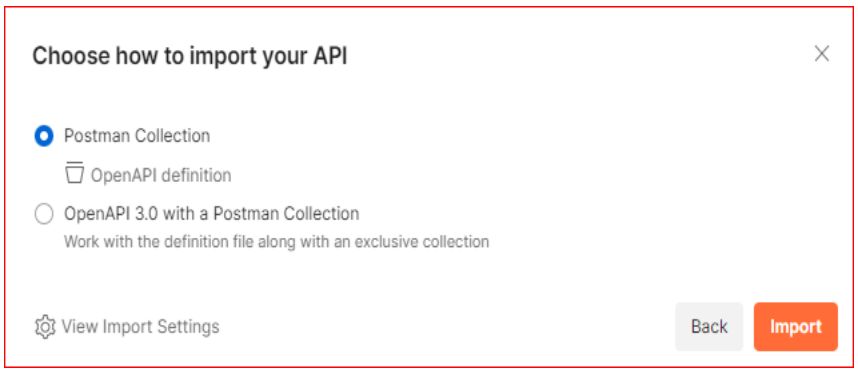
- Lancer Postman et suivre les étapes suivantes :



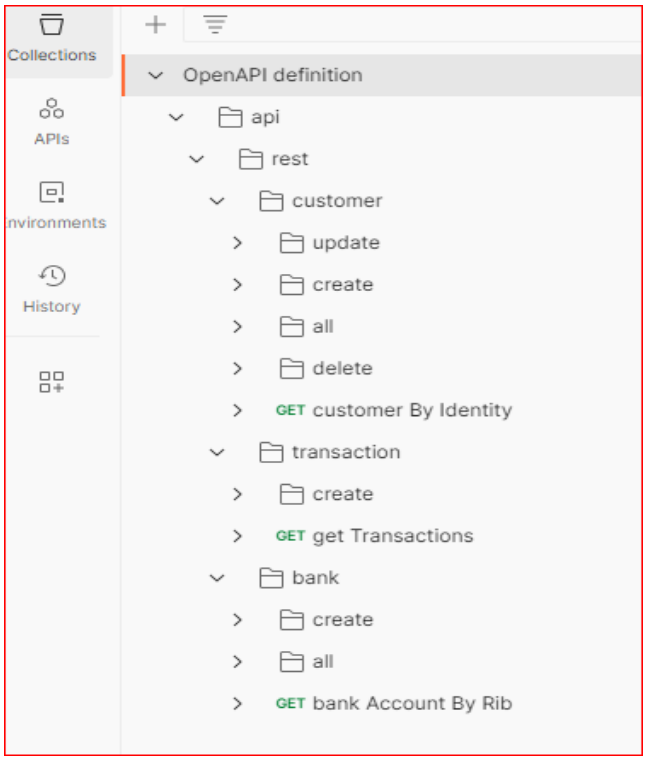
- Cliquer sur APIs.
- Cliquer sur Import. La fenêtre suivante s'affiche :



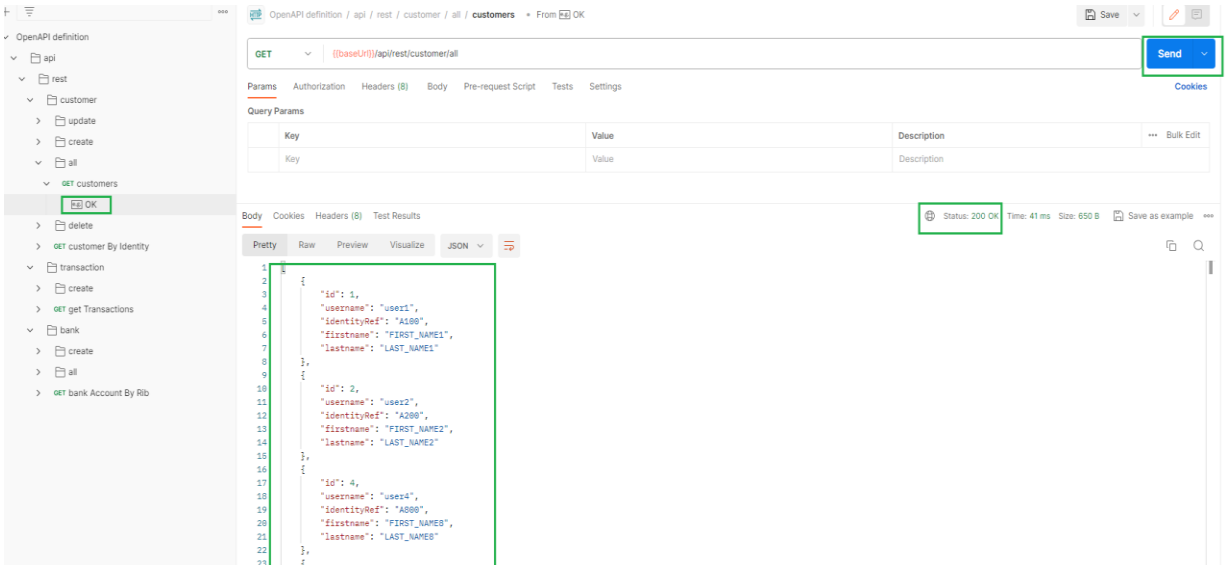
- Entrer le lien de l'interface JSON généré par OpenAPI : http://localhost:8080/api/rest/docs, la fenêtre suivante s'affiche :



- Cocher « ***Postman Collection*** » et cliquer sur ***Import***. Vérifier que le lien de l'interface de votre WS a été bien exploité par Postman et que les Endpoints ont été toutes affichées comme montré ci-après :



- Vous pouvez tester par exemple l'EndPoint /api/rest/customer/all en cliquant tout simplement sur l'Endpoint comme montré ci-après :



Remarquer que grâce à l'interface générée par OpenAPI, vous pouvez tester les EndPoint de votre WS très facilement.

Remarque :

Remarquez que grâce à l'api OpenAPI V3, vous avez pu connaître la liste des Endpoint fournis par le SW et également comment les consommer : **vous venez de développer une API REST**.

e. Développement du connecteur SOAP (SOAP API)

1. Le fichier pom.xml

Ajouter les dépendances suivantes au niveau du fichier pom.xml :

```
<dependency>
   <groupId>com.sun.xml.ws</groupId>
   <artifactId>jaxws-ri</artifactId>
   <version>4.0.2</version>
   <type>pom</type>
</dependency>

<dependency>
   <groupId>org.apache.cxf</groupId>
   <artifactId>cxf-spring-boot-starter-jaxws</artifactId>
   <version>4.0.3</version>
</dependency>
```

Explications :

- La dépendance ***jaxws-ri*** est l'implémentation de référence de l'api JAX WS (Java Architecture based XML for Web Service), il s'agit du projet ***Metro*** de la communauté Jakarta (le site officiel de Metro est https://javaee.github.io/metro).

- La dépendance **cxf-spring-boot-starter-jaxws** (https://cxf.apache.org/docs/springboot.html) permet d'enregistrer la servlet fournie par Apache CXF (***CXFServlet***) avec l'URL « /services/* » au niveau du conteneur de Spring pour accéder aux Endpoints de JAX-WS. Apache CXF est une implémentation de l'api JAX WS.

2. Le contrôleur SOAP

- Créer le package ***ma.formations.multiconnector.presentation.soap***
- Créer ensuite la classe BankSoapController suivante :

```
package ma.formations.multiconnector.presentation.soap;

import jakarta.jws.WebMethod;
import jakarta.jws.WebParam;
import jakarta.jws.WebResult;
import jakarta.jws.WebService;
import jakarta.jws.soap.SOAPBinding;
import lombok.AllArgsConstructor;
import ma.formations.multiconnector.common.CommonTools;
import ma.formations.multiconnector.dtos.bankaccount.AddBankAccountRequest;
import ma.formations.multiconnector.dtos.bankaccount.AddBankAccountResponse;
import ma.formations.multiconnector.dtos.bankaccount.BankAccountDto;
import ma.formations.multiconnector.dtos.customer.*;
import ma.formations.multiconnector.dtos.transaction.AddWirerTransferRequest;
import ma.formations.multiconnector.dtos.transaction.AddWirerTransferResponse;
import ma.formations.multiconnector.dtos.transaction.GetTransactionListRequest;
import ma.formations.multiconnector.dtos.transaction.TransactionDto;
import ma.formations.multiconnector.service.IBankAccountService;
import ma.formations.multiconnector.service.ICustomerService;
import ma.formations.multiconnector.service.ITransactionService;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
@WebService(serviceName = "BankWS")
@SOAPBinding
@AllArgsConstructor
public class BankSoapController {

   private final IBankAccountService bankAccountService;
   private final ICustomerService customerService;
   private ITransactionService transactionService;
   private CommonTools commonTools;

   @WebMethod

   /**
    *
    @WebResult was user in order to replace return balise
    by Customer balise in SOAP Response.
    */
   @WebResult(name = "Customer")
```

```
    public List<CustomerDto> customers() {
        return customerService.getAllCustomers();
    }


    @WebMethod
    /**
     * @WebResult was user in order to replace return balise
     * by Customer balise in SOAP Response.
     */
    @WebResult(name = "Customer")
    public CustomerDto customerByIdentity(@WebParam(name = "identity") String identity) {
        return customerService.getCustomByIdentity(identity);
    }

    @WebMethod
    /**
     * @WebResult was user in order to replace return balise
     * by Customer balise in SOAP Response.
     */
    @WebResult(name = "Customer")
    public AddCustomerResponse createCustomer(@WebParam(name = "Customer") AddCustomerRequest dto) {
        return customerService.createCustomer(dto);
    }

    /**
     * @WebResult was user in order to replace return balise
     * by BankAccount balise in SOAP Response.
     */
    @WebResult(name = "BankAccount")
    @WebMethod
    public List<BankAccountDto> bankAccounts() {
        return bankAccountService.getAllBankAccounts();
    }

    @WebMethod
    /**
     * @WebResult was user in order to replace return balise
     * by BankAccount balise in SOAP Response.
     */
    @WebResult(name = "BankAccount")
    public BankAccountDto bankAccountByRib(@WebParam(name = "rib") String rib) {
        return bankAccountService.getBankAccountByRib(rib);
    }


    /**
     * @WebResult was user in order to replace return balise
     * by BankAccount balise in SOAP Response.
     */
    @WebResult(name = "BankAccount")
    @WebMethod
    public AddBankAccountResponse createBankAccount(@WebParam(name = "bankAccountRequest")
AddBankAccountRequest dto) {
        return bankAccountService.saveBankAccount(dto);
    }
```

```
    /**
     * @WebResult was user in order to replace return balise
     * by Transaction balise in SOAP Response.
     */
    @WebResult(name = "Transaction")
    @WebMethod
    public AddWirerTransferResponse createWirerTransfer(@WebParam(name = "wirerTransferRequest")
AddWirerTransferRequest dto) {
        return transactionService.wiredTransfer(dto);
    }

    /**
     * @WebResult was user in order to replace return balise
     * by Transaction balise in SOAP Response.
     */
    @WebResult(name = "Transaction")
    @WebMethod
    public List<TransactionDto> getTransactions(@WebParam(name = "dto") GetTransactionListRequest dto) {
        return transactionService.getTransactions(dto);
    }

    @WebResult(name = "Customer")
    @WebMethod
    public UpdateCustomerResponse changeCustomer(@WebParam(name = "identityRef") String identityRef,
@WebParam(name = "dto") UpdateCustomerRequest dto) {
        return customerService.updateCustomer(identityRef, dto);
    }

    @WebMethod
    public String deleteCustomer(@WebParam(name = "identityRef") String identityRef) {
        return customerService.deleteCustomerByIdentityRef(identityRef);
    }
}
```

3. Le serveur SOAP

- Créer la classe **CxfConfig** suivante :

```
package ma.formations.multiconnector.config;

import lombok.AllArgsConstructor;
import ma.formations.multiconnector.presentation.soap.BankSoapController;
import org.apache.cxf.Bus;
import org.apache.cxf.jaxws.EndpointImpl;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
@AllArgsConstructor

public class CxfConfig {

    private Bus bus;
    private BankSoapController bankSoapController;

    @Bean
    public EndpointImpl bankWSEndpoint() {
```

```
    EndpointImpl endpoint = new EndpointImpl(bus, bankSoapController);
    endpoint.publish("/BankService");
    return endpoint;
  }
}
```

Explications :

- La méthode bankWSEndpoint() ci-dessus permet d'enregistrer le Servlet de CXF au niveau du conteneur. Le SW SOAP nommé « /bankService » est accessible via lien http://localhost:8080/services/bankService.

4. Personnaliser le lien de votre SW SOAP

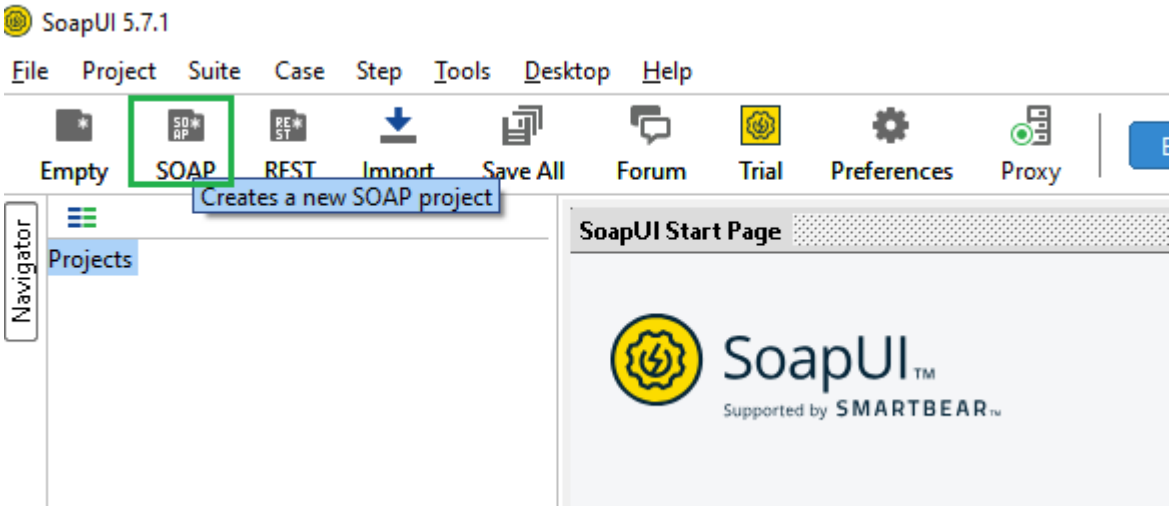
- Pour changer le lien de votre WS SOAP, il suffit d'ajouter la ligne suivante au niveau du fichier ***application.properties*** :

```
cxf.path=/api/soap
```

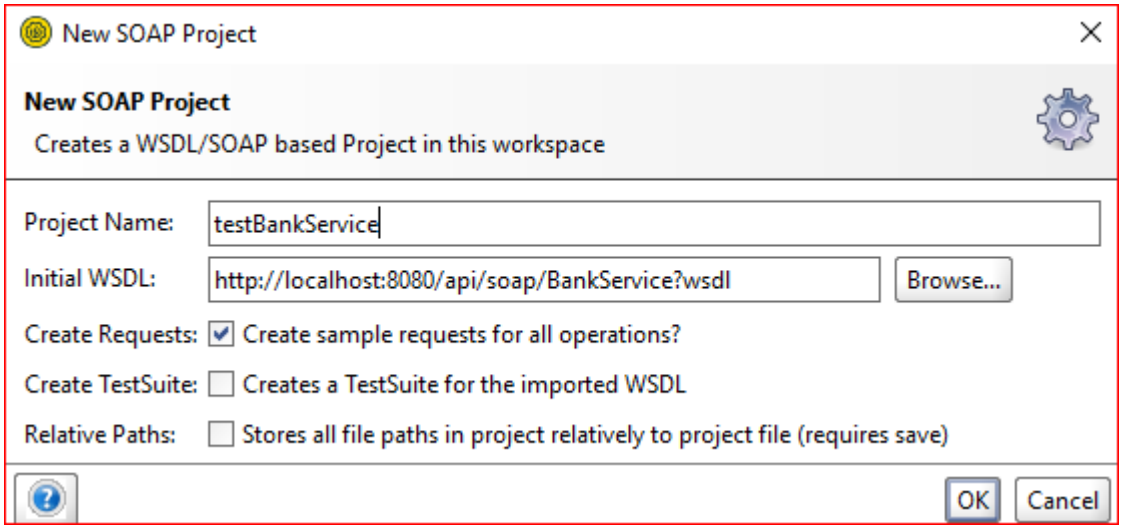
5. Tester avec SoapUI

- Lancer l'application.
- Lancer le lien suivant : http://localhost:8080/api/soap/BankService?wsdl et vérifier que le fichier WSDL a été bien généré :

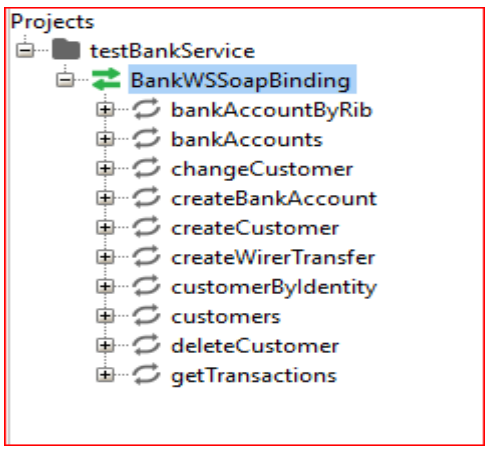
- Lancer SoapUI :



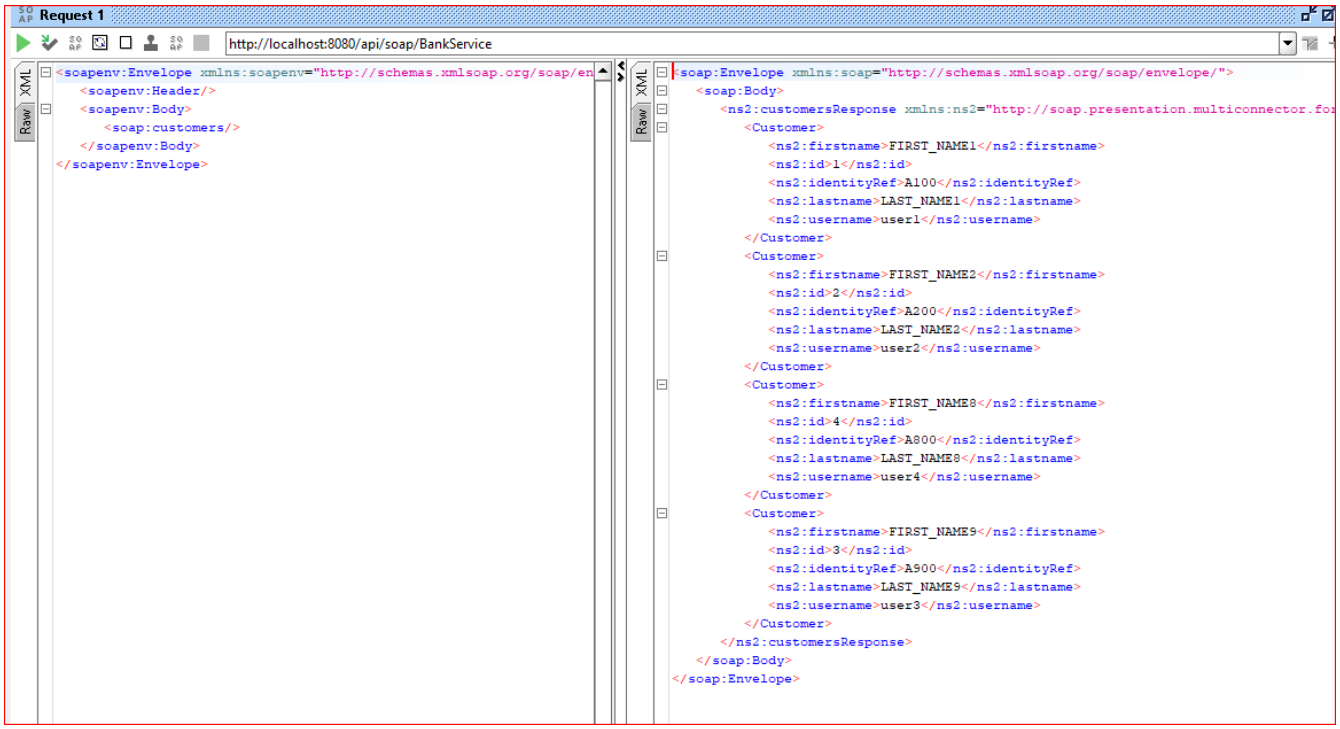
- Cliquer sur SOAP comme montré ci-dessus, la fenêtre suivante s'affiche :



- Entrer l'URL du WS, le nom de votre projet de test et cliquer sur le bouton OK. Vérifier que tous les Endpoints de votre WS sont listés :



- Pour tester par exemple customers, cliquer sur ce dernier et cliquer sur la flèche au milieu comme montré ci-après :



Remarque :

Remarquez que grâce au fichier WSDL, vous avez pu connaître la liste des Endpoint fournis par le SW et également comment les consommer : **vous venez de développer une API SOAP**.

f. Développement du connecteur gRPC

1. Le fichier pom.xml

- Dans la balise <properties> du fichier pom.xml, ajouter les lignes suivantes :

```
<protobuf.version>3.25.0</protobuf.version>
<grpc.version>1.59.0</grpc.version>
<grpc.server.spring.boot.starter>2.15.0.RELEASE</grpc.server.spring.boot.starter>
<grpc.server.spring.boot.autoconfigure>2.15.0.RELEASE</grpc.server.spring.boot.autoconfigure>
```

- Ajouter les dépendances suivantes au niveau du fichier pom.xml :

```
<!-- Needed for gRPC WS -->
<dependency>
  <groupId>io.grpc</groupId>
  <artifactId>grpc-stub</artifactId>
  <version>${grpc.version}</version>
</dependency>
<dependency>
  <groupId>io.grpc</groupId>
  <artifactId>grpc-protobuf</artifactId>
  <version>${grpc.version}</version>
```

```
</dependency>

<dependency>
  <groupId>com.google.protobuf</groupId>
  <artifactId>protobuf-java</artifactId>
  <version>${protobuf.version}</version>
</dependency>
<dependency>
  <!-- Java 9+ compatibility - Do NOT update to 2.0.0 -->
  <groupId>jakarta.annotation</groupId>
  <artifactId>jakarta.annotation-api</artifactId>
  <version>2.1.1</version>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>net.devh</groupId>
  <artifactId>grpc-server-spring-boot-starter</artifactId>
  <version>${grpc.server.spring.boot.starter}</version>
</dependency>
<dependency>
  <groupId>net.devh</groupId>
  <artifactId>grpc-server-spring-boot-autoconfigure</artifactId>
  <version>${grpc.server.spring.boot.autoconfigure}</version>
</dependency>
<dependency>
  <groupId>javax.annotation</groupId>
  <artifactId>javax.annotation-api</artifactId>
  <version>1.3.2</version>
</dependency>
```

- Ajouter ensuite le plugin ci-dessous :

```
<plugin>
  <groupId>com.github.os72</groupId>
  <artifactId>protoc-jar-maven-plugin</artifactId>
  <version>3.11.4</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals>
        <goal>run</goal>
      </goals>
      <configuration>
        <includeMavenTypes>direct</includeMavenTypes>
        <inputDirectories>
          <include>src/main/resources</include>
        </inputDirectories>
        <outputTargets>
          <outputTarget>
            <type>java</type>
            <outputDirectory>src/main/java</outputDirectory>
          </outputTarget>
          <outputTarget>
            <type>grpc-java</type>
            <pluginArtifact>io.grpc:protoc-gen-grpc-java:1.15.0</pluginArtifact>
            <outputDirectory>src/main/java</outputDirectory>
```

```
            </outputTarget>
          </outputTargets>
        </configuration>
      </execution>
    </executions>
</plugin>
```

2. Le fichier bank.proto

Créer le fichier ***src/resources/bank.proto*** suivant :

```
syntax = "proto3";
option java_package = "ma.formations.multiconnector.grpc.stub";
service BankService{
  rpc customers(CustomersRequest) returns (CustomersResponse);
  rpc customerByIdentity(CustomerByIdentityRequest) returns (CustomerByIdentityResponse);
  rpc createCustomer(CreateCustomerRequest) returns (CreateCustomerResponse);
  rpc updateCustomer(UpdateCustomerRequest) returns (UpdateCustomerResponse);
  rpc deleteCustomer(DeleteCustomerRequest) returns (DeleteCustomerResponse);

  rpc bankAccounts(BankAccountsRequest) returns (BankAccountsResponse);
  rpc bankAccountByRib(BankAccountByRibRequest) returns (BankAccountByRibResponse);
  rpc addBankAccount(AddBankAccountRequest) returns (AddBankAccountResponse);


  rpc addWirerTransfer(AddWirerTransferRequest) returns (AddWirerTransferResponse);
  rpc getTransactions(GetTransactionsRequest) returns (GetTransactionsResponse);

}

message CustomersRequest {
}

message CustomersResponse {
  repeated CustomerDTO customers = 1;
}
message CustomerDTO {
  int64 id = 1;
  string username = 2;
  string identityRef = 3;
  string firstname = 4;
  string lastname = 5;
}

message CustomerByIdentityRequest {
  string identityRef = 1;
}

message CustomerByIdentityResponse {
  CustomerDTO customer = 1;
}

message CreateCustomerRequest {
  string username = 1;
  string identityRef = 2;
  string firstname = 3;
  string lastname = 4;
```

```
}

message CreateCustomerResponse {
  string message = 1;
  CustomerDTO customer = 2;

}

message UpdateCustomerRequest {
  string identityRef = 1;
  UpdatedCustomerDTO updatedCustomer = 2;
}

message UpdatedCustomerDTO {
  string username = 2;
  string firstname = 4;
  string lastname = 5;
}


message UpdateCustomerResponse {
  string message = 1;
  CustomerDTO customer = 2;
}

message DeleteCustomerRequest {
  string identityRef = 1;
}

message DeleteCustomerResponse {
  string message = 1;
}

message BankAccountDto {
  int64 id = 1;
  string rib = 2;
  double amount = 3;
  string createdAt = 4;
  string accountStatus = 5;
  CustomerDTO customer = 6;
}

message BankAccountsRequest {

}

message  BankAccountsResponse {
  repeated BankAccountDto bankAccount = 1;
}


message BankAccountByRibRequest {
  string rib = 1;
}

message BankAccountByRibResponse {
  BankAccountDto bankAccount = 1;
```

```
}

message AddBankAccountRequest {
  string rib = 1;
  double amount = 2;
  string customerIdentityRef = 3;
}

message AddBankAccountResponse {
  string message = 1;
  BankAccountDto bankAccount = 2;
}

message AddWirerTransferRequest {
  string ribFrom = 1;
  string ribTo = 2;
  double amount = 3;
  string username = 4;
}

message AddWirerTransferResponse {
  string message = 1;
  TransactionDto transactionFrom = 2;
  TransactionDto transactionTo = 3;
}

message GetTransactionsRequest {
  string rib = 1;
  string dateTo = 2;
  string dateFrom = 3;
}

message GetTransactionsResponse {
  repeated  TransactionDto transaction = 1;
}

message UserDto {
  string username = 1;
  string firstname = 2;
  string lastname = 3;
}

message TransactionDto {
  string createdAt = 1;
  string transactionType = 2;
  double amount = 3;
  BankAccountDto bankAccount = 4;
  UserDto user = 5;
}
```

3. Build du projet

- Lancer la commande ***clean install*** de Maven pour que le plugin gRPC puisse générer le stub. Vérifier que les classes du stub ont été bien générées dans le package ***ma.formations.multiconnector.grpc.stub.***

4. Configuration du port du serveur gRPC

- Le serveur gRPC sera démarré par défaut au port 9999. Pour changer ce dernier, il suffit d'ajouter la ligne suivante au niveau du fichier *application.properties*. dans ce cas le serveur gRPC sera démarré au port 4444.

```
grpc.server.port=4444
```

5. Le contrôleur

- Créer le package ***ma.formations.multiconnector.presentation.grpc*** et ensuite créer la classe ***BankGrpcController*** suivante :

```
package ma.formations.multiconnector.presentation.grpc;

import io.grpc.stub.StreamObserver;
import lombok.AllArgsConstructor;
import ma.formations.multiconnector.dtos.bankaccount.AddBankAccountRequest;
import ma.formations.multiconnector.dtos.bankaccount.AddBankAccountResponse;
import ma.formations.multiconnector.dtos.bankaccount.BankAccountDto;
import ma.formations.multiconnector.dtos.customer.*;
import ma.formations.multiconnector.dtos.transaction.AddWirerTransferRequest;
import ma.formations.multiconnector.dtos.transaction.AddWirerTransferResponse;
import ma.formations.multiconnector.dtos.transaction.GetTransactionListRequest;
import ma.formations.multiconnector.dtos.transaction.TransactionDto;
import ma.formations.multiconnector.grpc.stub.Bank;
import ma.formations.multiconnector.grpc.stub.BankServiceGrpc;
import ma.formations.multiconnector.service.IBankAccountService;
import ma.formations.multiconnector.service.ICustomerService;
import ma.formations.multiconnector.service.ITransactionService;
import net.devh.boot.grpc.server.service.GrpcService;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.stream.Collectors;

@Component
@AllArgsConstructor
@GrpcService
public class BankGrpcController extends BankServiceGrpc.BankServiceImplBase {
    private ICustomerService customerService;
    private IBankAccountService bankAccountService;
    private ModelMapper modelMapper;
```

```
    private ITransactionService transactionService;

    @Override
    public void customers(Bank.CustomersRequest request, StreamObserver<Bank.CustomersResponse>
responseObserver) {
        List<CustomerDto> customers = customerService.getAllCustomers();
        Bank.CustomersResponse response = Bank.CustomersResponse.newBuilder().
                addAllCustomers(
                        customers.stream().map(customerDto ->
                                        modelMapper.map(customerDto, Bank.CustomerDTO.Builder.class).build()).
                                collect(Collectors.toList())).
                build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void customerByIdentity(Bank.CustomerByIdentityRequest request,
StreamObserver<Bank.CustomerByIdentityResponse> responseObserver) {
        CustomerDto customerDto = customerService.getCustomByIdentity(request.getIdentityRef());
        Bank.CustomerByIdentityResponse response = Bank.CustomerByIdentityResponse.newBuilder().
                setCustomer(modelMapper.map(customerDto, Bank.CustomerDTO.Builder.class).build()).
                build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void createCustomer(Bank.CreateCustomerRequest request, StreamObserver<Bank.CreateCustomerResponse>
responseObserver) {
        AddCustomerResponse addCustomerResponse = customerService.createCustomer(modelMapper.map(request,
AddCustomerRequest.class));
        Bank.CreateCustomerResponse response = Bank.CreateCustomerResponse.newBuilder().
                setMessage(addCustomerResponse.getMessage()).
                setCustomer(modelMapper.map(addCustomerResponse, Bank.CustomerDTO.Builder.class).build()).
                build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void updateCustomer(Bank.UpdateCustomerRequest request, StreamObserver<Bank.UpdateCustomerResponse>
responseObserver) {

        UpdateCustomerResponse updateCustomerResponse = customerService.updateCustomer(
                request.getIdentityRef(), modelMapper.map(request.getUpdatedCustomer(), UpdateCustomerRequest.class));
        Bank.UpdateCustomerResponse response = Bank.UpdateCustomerResponse.newBuilder().
                setMessage(updateCustomerResponse.getMessage()).
                setCustomer(modelMapper.map(updateCustomerResponse, Bank.CustomerDTO.Builder.class).build()).
                build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void deleteCustomer(Bank.DeleteCustomerRequest request, StreamObserver<Bank.DeleteCustomerResponse>
responseObserver) {
```

```
        String message = customerService.deleteCustomerByIdentityRef(request.getIdentityRef());
        Bank.DeleteCustomerResponse response = Bank.DeleteCustomerResponse.newBuilder().
                setMessage(message).
                build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }


    @Override
    public void bankAccounts(Bank.BankAccountsRequest request, StreamObserver<Bank.BankAccountsResponse>
responseObserver) {
        List<BankAccountDto> bankAccounts = bankAccountService.getAllBankAccounts();
        Bank.BankAccountsResponse response = Bank.BankAccountsResponse.newBuilder().
                addAllBankAccount(
                        bankAccounts.stream().
                                map(bankAccount -> modelMapper.map(bankAccount, Bank.BankAccountDto.Builder.class).build()).
                                collect(Collectors.toList())).
                build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void bankAccountByRib(Bank.BankAccountByRibRequest request,
StreamObserver<Bank.BankAccountByRibResponse> responseObserver) {
        BankAccountDto bankAccount = bankAccountService.getBankAccountByRib(request.getRib());

        Bank.BankAccountByRibResponse response = Bank.BankAccountByRibResponse.newBuilder().
                setBankAccount(modelMapper.map(bankAccount, Bank.BankAccountDto.Builder.class).build()).
                build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void addBankAccount(Bank.AddBankAccountRequest request, StreamObserver<Bank.AddBankAccountResponse>
responseObserver) {
        AddBankAccountResponse addBankAccountResponse = bankAccountService.saveBankAccount(
                modelMapper.map(request, AddBankAccountRequest.class));
        Bank.AddBankAccountResponse response = Bank.AddBankAccountResponse.newBuilder().
                setMessage(addBankAccountResponse.getMessage()).
                setBankAccount(modelMapper.map(addBankAccountResponse, Bank.BankAccountDto.Builder.class).build()).
                build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void addWirerTransfer(Bank.AddWirerTransferRequest request,
StreamObserver<Bank.AddWirerTransferResponse> responseObserver) {
        AddWirerTransferResponse addWirerTransferResponse =
transactionService.wiredTransfer(modelMapper.map(request, AddWirerTransferRequest.class));
        Bank.AddWirerTransferResponse response = Bank.AddWirerTransferResponse.newBuilder().
                setMessage(addWirerTransferResponse.getMessage()).
                setTransactionFrom(modelMapper.map(addWirerTransferResponse.getTransactionFrom(),
Bank.TransactionDto.Builder.class).build()).
```

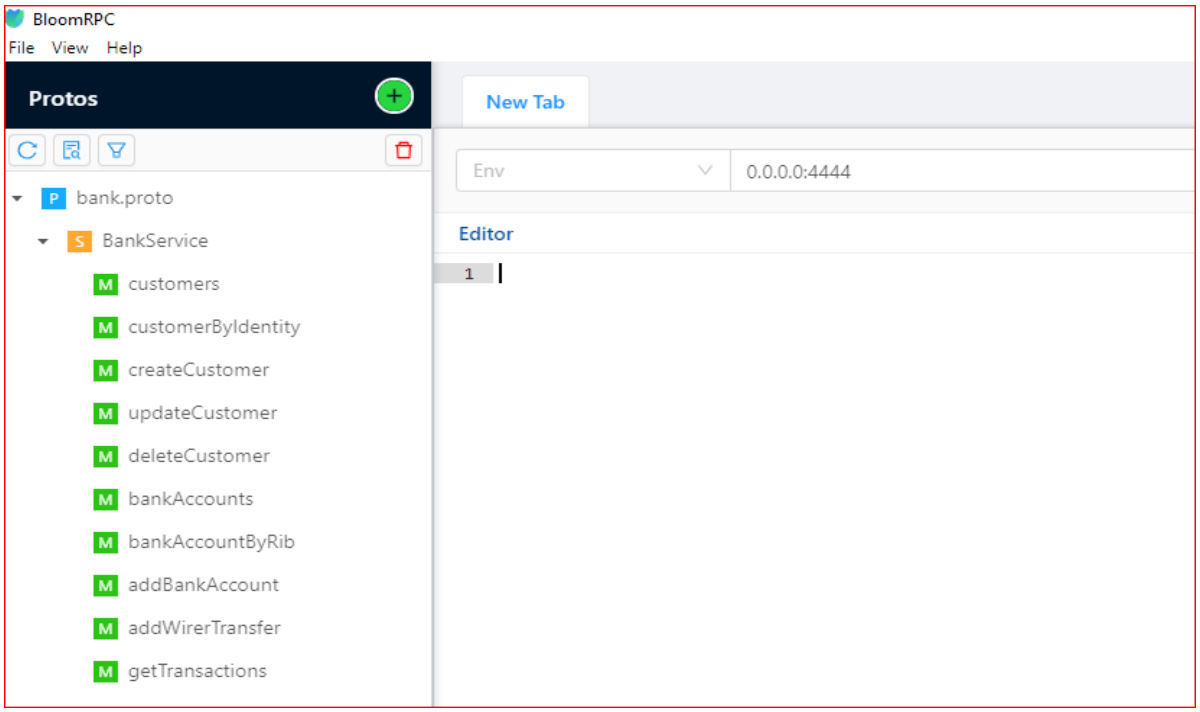
```
            setTransactionTo(modelMapper.map(addWirerTransferResponse.getTransactionTo(),
Bank.TransactionDto.Builder.class).build()).
            build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void getTransactions(Bank.GetTransactionsRequest request, StreamObserver<Bank.GetTransactionsResponse>
responseObserver) {
        List<TransactionDto> transactions = transactionService.getTransactions(modelMapper.map(request,
GetTransactionListRequest.class));
        Bank.GetTransactionsResponse response = Bank.GetTransactionsResponse.newBuilder().
            addAllTransaction(transactions.stream().
                map(transactionDto -> modelMapper.map(transactionDto, Bank.TransactionDto.Builder.class).build()).
                collect(Collectors.toList())).
            build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();

    }
}
```

6. Test de l'API gRPC

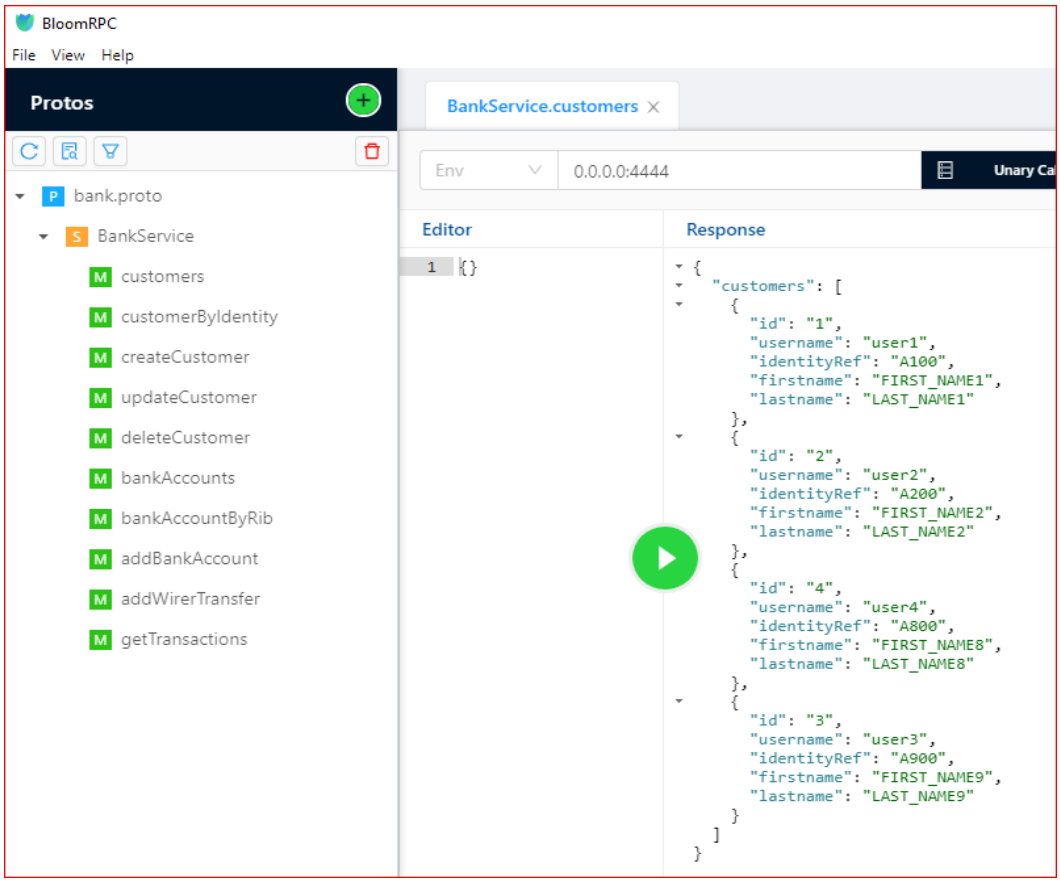
- Lancer votre application et vérifier que le serveur gRPC est bien démarré comme montré ci-dessous :

```
2023-11-23T09:01:13.271+01:00  INFO 2112 --- [  restartedMain] n.d.b.g.s.s.GrpcServerLifecycle          : gRPC Server started, listening on address: *, port: 4444
2023-11-23T09:01:13.284+01:00  INFO 2112 --- [  restartedMain] m.f.m.BankServiceApplication             : Started BankServiceApplication in 8.144 seconds (process running
```

- Lancer BloomRPC, configurer l'URL du serveur (0.0.0.0:4444), ajouter le fichier *bank.proto* et vérifier que les méthodes fournies par le service gRPC sont bien listées au niveau De l'interface de BloomRPC :



- Pour tester par exemple le service ***customers***, cliquer sur ce dernier ensuite cliquer sur la flèche au centre et vérifier que le serveur a bien envoyé la liste des clients :



Remarque :

Remarquez que grâce au fichier PROTO, vous avez pu connaître la liste des Endpoint fournis par le SW et également comment les consommer : **vous venez de développer une API gRPC**.

Conclusion

Le code source de cet atelier est disponible sur GITHUB :

https://github.com/abbouformations/bank-service-multi-connecteur.git