



Compt Rendu

5^{ème} année
Ingénieur Informatique & Réseaux

TP7

All API

Réalisé par : IMANE Chakrellah
YASSINE Ech-chaoui

Prof : Pr.Oussama
Classe: 5IIR-11

2025-2026

Table des matières

Objectif du TP	3
1. Architecture du projet	3
1.1. Structure du projet.....	3
1.2. Description des composants.....	4
1.3. Rôle de chaque couche	5
2. Notions importantes	5
2.1. Concepts théoriques liés au TP	5
REST	5
GraphQL.....	5
SOAP	5
gRPC.....	5
2.2. Technologies & outils utilisés	6
2.3. Principes / notions clés	6
🔧 3. Déroulement du TP	6
3.1. Étapes réalisées	6
5. Tests détaillés (avec explications + liens).....	6
TESTS GRAPHQL	7
TESTS REST	8
TESTS SOAP	8
TESTS gRPC	9
7. Outils de visualisation	9
8. Conclusion.....	10

Tout les TPs Réalisés ce trouve dans le lien suivant qui contient le projet + CR :

<https://github.com/CHAKRELLAH44/JEE-ARCHITECTURE.git>

Objectif du TP

L'objectif de ce TP est de **comprendre comment un même système peut proposer plusieurs manières différentes d'être utilisé**, selon les besoins des applications clientes.

Un même serveur expose donc **4 types d'API différents** :

Technologie	Mode de communication	Format	Usage
REST	HTTP	JSON	Web, Mobile apps
GraphQL	HTTP	JSON “sur mesure”	Applications modernes, optimisées
SOAP	HTTP	XML structuré	Systèmes bancaires, entreprises, gouvernements
gRPC	HTTP/2	Protobuf binaire	Haute performance, microservices

Le but:

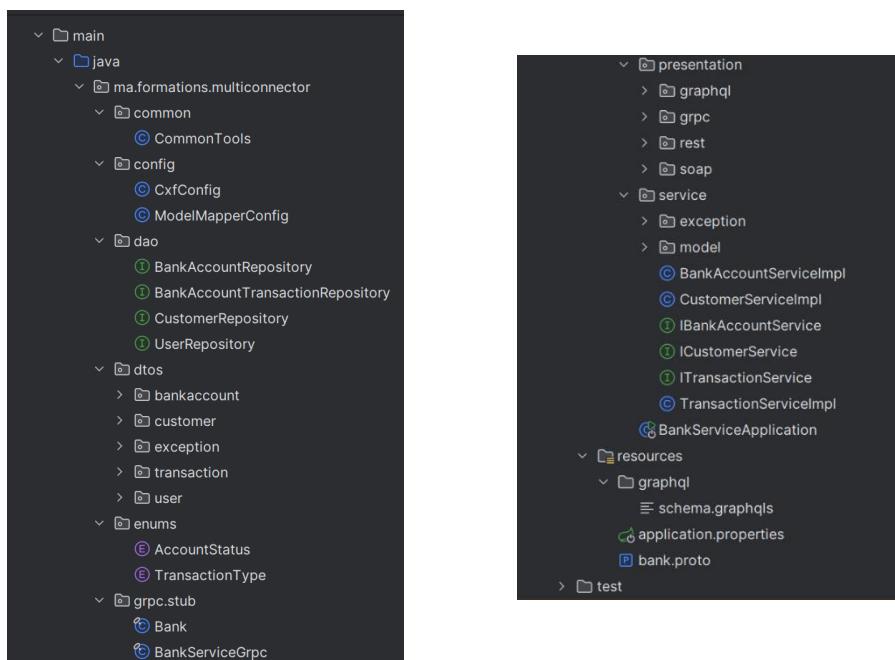
Comprendre *les forces, limites, différences et comment tester ces connecteurs*.

Pourquoi c'est important ?

Parce que dans une vraie entreprise, un même système doit souvent être utilisé par plusieurs types d'applications, anciennes et modernes.

1. Architecture du projet

1.1. Structure du projet (avec explication)



- **entities** → contient les classes qui représentent les données enregistrées en base (comme Customer, BankAccount).
- **repositories** → contient le code pour chercher, ajouter, modifier ou supprimer les données en base.
- **services** → contient la logique métier : règles, validations, traitement des opérations.
- **controllers** → c'est ici que le projet expose les API :
 - rest/ expose des endpoints REST
 - graphql/ expose des queries et mutations GraphQL
 - soap/ expose un Web Service SOAP
 - grpc/ expose le service gRPC
- **dto** → objets utilisés pour transférer les données au client sans exposer tout le modèle interne.
- **schema.graphqls** → fichier décrivant la structure des API GraphQL.
- **bank.proto** → fichier définissant les messages et services gRPC.
- **application.properties** → configuration (base de données, ports, logs...).

1.2. Description des composants

Entity

Représente un objet réel (exemple : un client).
C'est l'équivalent d'une table en base de données.

Repository

Automatise l'accès aux données : chercher par ID, sauvegarder, supprimer.

Service

Contient toute la logique métier.
C'est la partie **intelligente** du projet.

Exemples :

- vérifier si un client existe avant de le créer
- vérifier si un compte a assez d'argent
- formater les données avant de les retourner

Controller

C'est la partie qui parle avec l'extérieur (API).

1.3. Rôle de chaque couche

Imagine que le projet est un **restaurant** :

Couche	Rôle	Métaphore
Entities	Description des objets	Menu + ingrédients
Repositories	Accès aux données	Réserve du restaurant
Services	Logique métier	Cuisine (transforme, applique les règles)
Controllers	API exposée au monde	Serveur qui communique avec les clients
DTO	Données envoyées au client	Plat final servi au client

2. Notions importantes

2.1. Concepts théoriques liés au TP

REST

- Fonctionne avec des URLs
 - Utilise GET, POST, PUT, DELETE
 - Format JSON
 - Simple et universel
- ⇒ Parfait pour les applications web et mobile.

GraphQL

- Le client demande exactement ce qu'il veut.
 - Les réponses ne contiennent **jamais des données inutiles**.
 - Une seule URL, mais plusieurs types de requêtes.
- ⇒ Idéal pour optimiser les performances.

SOAP

- Fonctionne avec XML dans une enveloppe bien structurée.
 - Basé sur un contrat (WSDL).
 - Utilisé dans les banques, assurances, gouvernements.
- ⇒ Très strict, sécurisé, fiable.

gRPC

- Utilise HTTP/2 → très rapide
 - Messages binaires (Protobuf)
 - Nécessite génération automatique du code client/serveur
- ⇒ Excellent pour les microservices et les communications internes.

2.2. Technologies & outils utilisés (détailé)

Technologie	Pourquoi elle est utilisée
Spring Boot	Simplifier création d'API multi-connecteurs
Spring Data JPA	Accès aux données automatisé
GraphQL Java	Support GraphQL
JAX-WS	Support SOAP
gRPC + Protobuf	API haute performance
Swagger	Documentation interactive REST
SoapUI	Test du SOAP
BloomRPC	Test du gRPC
H2 / MySQL	Stockage des données

2.3. Principes / notions clés

- **Serialization** : transformer un objet Java en JSON/XML/Protobuf.
- **Stateless** : chaque requête est indépendante.
- **Contrat** : description d'une API (WSDL, schema.graphqls, proto).
- **Transport** : REST = HTTP1, gRPC = HTTP2.

3. Déroulement du TP

3.1. Étapes réalisées

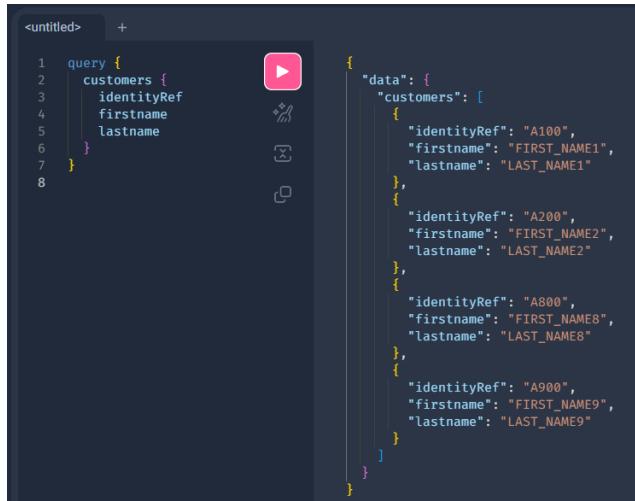
1. Création des classes de base (Customer, BankAccount...)
2. Configuration de la base de données
3. Implémentation des services métiers
4. Ajout du contrôleur REST
5. Mise en place de GraphQL (queries + mutations)
6. Mise en place du service SOAP
7. Création du fichier Proto gRPC + service gRPC
8. Test de chaque connecteur avec l'outil approprié
9. Comparaison des résultats et des performances

5. Tests détaillés (avec explications + liens)

TESTS GRAPHQL

Lien : <http://localhost:8080/graphiql?path=/graphql>

Test 1 : Récupérer tous les clients



```
<untitled> +  
1 query {  
2   customers {  
3     identityRef  
4     firstname  
5     lastname  
6   }  
7 }  
8
```

The screenshot shows a GraphQL query in the left pane:

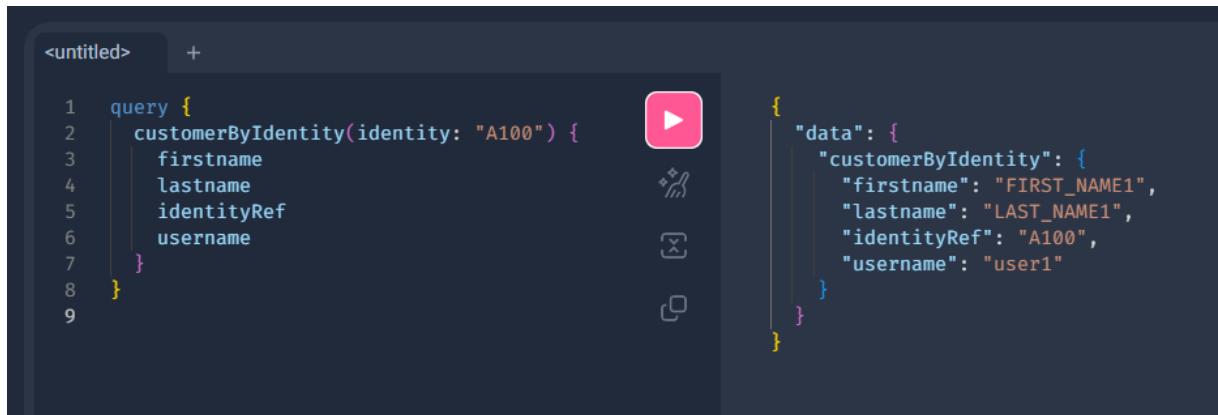
```
query {  
  customers {  
    identityRef  
    firstname  
    lastname  
  }  
}
```

The right pane displays the resulting JSON data:

```
{  
  "data": {  
    "customers": [  
      {  
        "identityRef": "A100",  
        "firstname": "FIRST_NAME1",  
        "lastname": "LAST_NAME1"  
      },  
      {  
        "identityRef": "A200",  
        "firstname": "FIRST_NAME2",  
        "lastname": "LAST_NAME2"  
      },  
      {  
        "identityRef": "A800",  
        "firstname": "FIRST_NAME8",  
        "lastname": "LAST_NAME8"  
      },  
      {  
        "identityRef": "A900",  
        "firstname": "FIRST_NAME9",  
        "lastname": "LAST_NAME9"  
      }  
    ]  
  }  
}
```

Explanation : retourne la liste des clients.

Test 2 : Récupérer un client précis



```
<untitled> +  
1 query {  
2   customerByIdentity(identity: "A100") {  
3     firstname  
4     lastname  
5     identityRef  
6     username  
7   }  
8 }  
9
```

The screenshot shows a GraphQL query in the left pane:

```
query {  
  customerByIdentity(identity: "A100") {  
    firstname  
    lastname  
    identityRef  
    username  
  }  
}
```

The right pane displays the resulting JSON data:

```
{  
  "data": {  
    "customerByIdentity": {  
      "firstname": "FIRST_NAME1",  
      "lastname": "LAST_NAME1",  
      "identityRef": "A100",  
      "username": "user1"  
    }  
  }  
}
```

Explanation : retrouve un client grâce à son identifiant unique.

TESTS REST (Swagger)

http://localhost:8080/api/rest/docs-ui

Test 1 : GET clients

Curl

```
curl -X 'GET' \
'http://localhost:8080/api/rest/customer/all' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/rest/customer/all
```

Server response

Code	Details
200	Response body

```
[  
  {  
    "id": 1,  
    "username": "user1",  
    "identityRef": "A100",  
    "firstname": "FIRST_NAME1",  
    "lastname": "LAST_NAME1"  
  },  
  {  
    "id": 2,  
    "username": "user2",  
    "identityRef": "A200",  
    "firstname": "FIRST_NAME2",  
    "lastname": "LAST_NAME2"  
  },  
  {  
    "id": 4,  
    "username": "user4",  
    "identityRef": "A800",  
    "firstname": "FIRST_NAME8",  
    "lastname": "LAST_NAME8"  
  },  
  {  
    "id": 3,  
    "username": "user3",  
    "identityRef": "A900",  
    "firstname": "FIRST_NAME9",  
    "lastname": "LAST_NAME9"
```

Test 2 : POST créer un client

Curl

```
curl -X 'POST' \
'http://localhost:8080/api/rest/customer/create' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{  
  "username": "userX",  
  "identityRef": "LH44",  
  "firstname": "Lewis",  
  "lastname": "Hamilton"  
}'
```

Request URL

```
http://localhost:8080/api/rest/customer/create
```

Server response

Code	Details
201 <small>Undocumented</small>	Response body

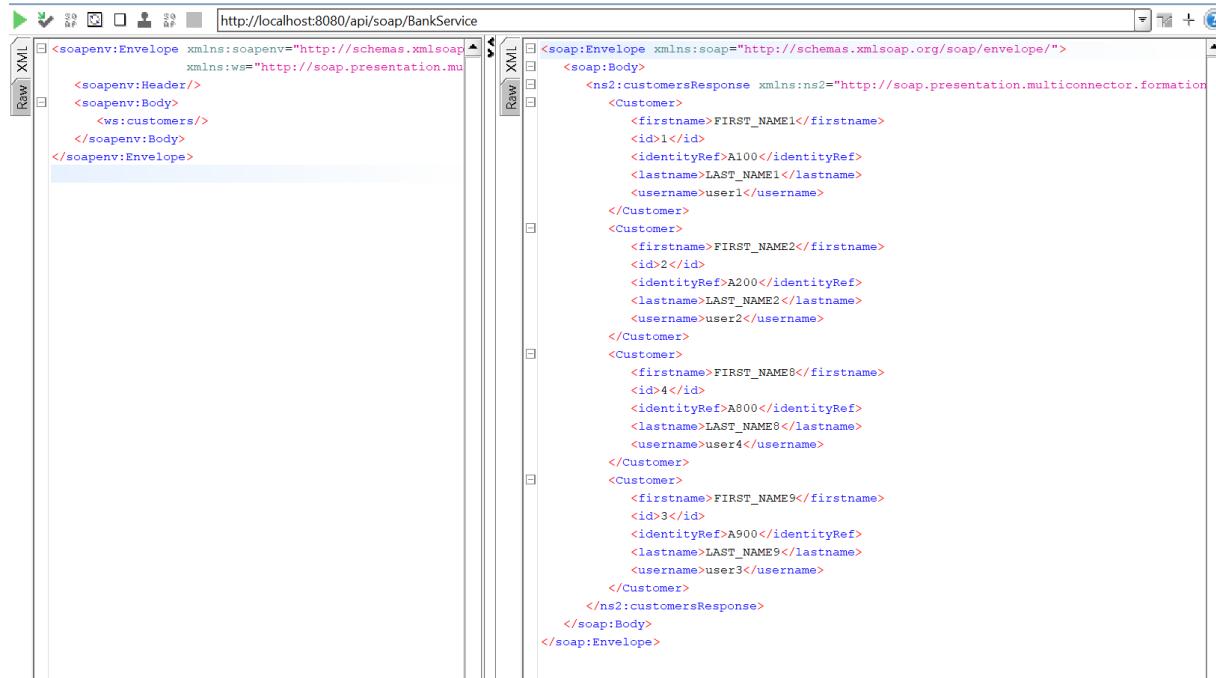
```
{  
  "message": "Customer : [identity= LH44,First Name= Lewis, Last Name= Hamilton, username= userX] was created with success"  
  "id": 5,  
  "username": "userX",  
  "identityRef": "LH44",  
  "firstname": "Lewis",  
  "lastname": "Hamilton"  
}
```

TESTS SOAP(SoapUI)

WSDL : <http://localhost:8080/api/soap/BankService?wsdl>

Dans SoapUI :

Test 1 : customers()



The screenshot shows the SoapUI interface with two panes. The left pane displays the XML request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://soap.presentation.multiconnector.formation">
```

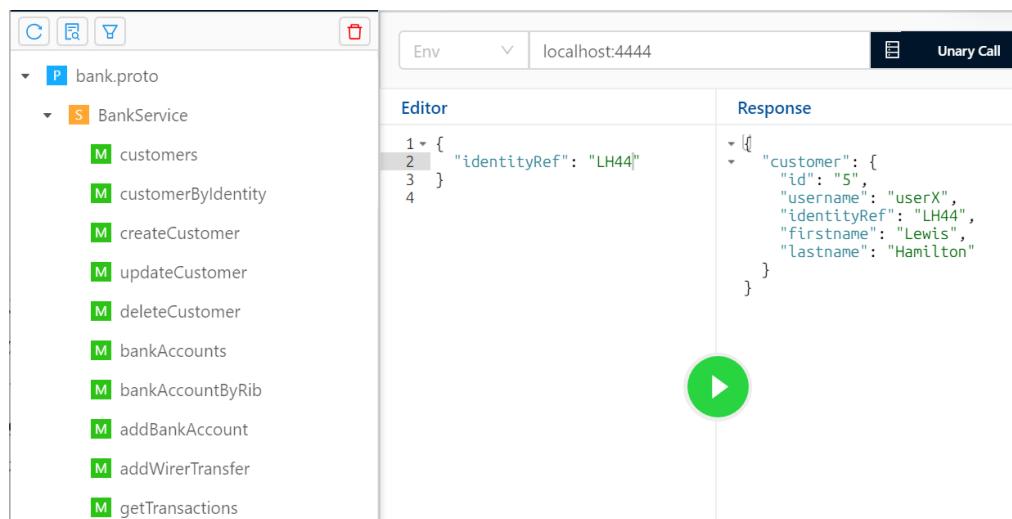
The right pane displays the XML response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">FIRST_NAME11A100LAST_NAME1user1FIRST_NAME22A200LAST_NAME2user2FIRST_NAME33A300LAST_NAME3user3FIRST_NAME44A400LAST_NAME4user4FIRST_NAME55A500LAST_NAME5user5FIRST_NAME66A600LAST_NAME6user6FIRST_NAME77A700LAST_NAME7user7FIRST_NAME88A800LAST_NAME8user8FIRST_NAME99A900LAST_NAME9user9
```

TESTS gRPC (BloomRPC)

Adresse serveur : localhost:4444

Test : customerByIdentity



The screenshot shows the BloomRPC interface with a sidebar and a main panel.

Left sidebar (File Structure):

- bank.proto
 - BankService
 - customers
 - customerByIdentity
 - createCustomer
 - updateCustomer
 - deleteCustomer
 - bankAccounts
 - bankAccountByRib
 - addBankAccount
 - addWirerTransfer
 - getTransactions

Right panel (Editor and Response):

Editor:

```
1 {  
2   "identityRef": "LH44"  
3 }  
4
```

Response:

```
{  
  "customer": {  
    "id": "5",  
    "username": "userX",  
    "identityRef": "LH44",  
    "firstname": "Lewis",  
    "lastname": "Hamilton"  
  }  
}
```

A green play button icon is located at the bottom center of the main panel.

7. Outils de visualisation

- **Swagger → tester REST**
- **GraphiQL → tester GraphQL**
- **SoapUI → tester SOAP**
- **BloomRPC → tester gRPC**
- **Console H2 → visualiser la BD**

8. Conclusion détaillée

Ce TP nous a permis de comprendre que :

- Chaque technologie répond à un besoin particulier.

il n'existe pas UNE bonne API — mais plusieurs technologies adaptées à des contextes différents.

- REST est simple mais limité.
- GraphQL optimise le trafic.

GraphQL est excellent pour les applications Web modernes, les mobiles, et les interfaces lourdes en données.

- SOAP reste utilisé dans les environnements critiques.

idéal pour les applications critiques et réglementées.

- gRPC est le plus performant et adapté aux microservices.

Limites du TP

- Pas de gestion d'erreurs avancée
- Pas d'authentification
- Pas de communication entre les connecteurs
- Peu de scénarios transactionnels complexes