1. Qu’est-ce que ce TP cherche à nous apprendre ?

L’objectif du TP est de montrer que **le même système** (gestion de clients, comptes et transactions) peut être exposé au monde extérieur via **plusieurs types d’API** :

- **REST**
- **GraphQL**
- **SOAP**
- **gRPC**

Chaque technologie permet à des applications différentes d’utiliser ton backend selon leurs besoins.

C’est comme si tu avais **un magasin**, mais que tu ouvrais **4 portes différentes**, adaptées à 4 types de clients.

2. Pourquoi un même projet expose plusieurs connecteurs ?

Parce qu’en entreprise :

- Certains systèmes sont anciens → ils utilisent SOAP.
- Les applications mobiles modernes → utilisent REST.
- Les dashboards complexes → utilisent GraphQL.
- Les microservices internes → utilisent gRPC (très rapide).

C’est une situation réelle dans les grandes entreprises : plusieurs types d’applications consomment le même backend.

3. Comment ton système est construit ?

Ton projet contient plusieurs couches :

1. Entities

Représentent les objets métier :

- Customer
- BankAccount
- Transaction

Exemple :

```
Customer {
  identityRef = "A100",
  firstname = "Oussama",
  lastname  = "Chakrellah"
}
```

2. Repository

Permet de rechercher les objets en BD.

Ex :
`findByIdentityRef("A100")`

3. Services

Ils contiennent **la vraie logique métier** :

- créer un client
- vérifier qu'un RIB existe
- exécuter un virement
- enregistrer une transaction
- valider les données

4. Connecteurs (REST, GraphQL, SOAP, gRPC)

Ce sont juste **4 manières différentes d'appeler les mêmes services métiers.**

Ton projet ne duplique pas la logique — il change juste la manière de l'exposer.

Exemple :
Créer un client via REST ou via GraphQL → ça fait la même chose dans le service.

4. Explication des 4 types d'API de ton TP

A. REST – l'API la plus classique

Format : JSON
Transport : HTTP
Exposition : via endpoints `/api/...`

Avantages :

- simple à utiliser
- lisible
- supporté par toutes les applications modernes

Exemple :

URL :

```
GET /api/rest/customer/all
```

Réponse JSON :

```
[
  { "identityRef": "A100", "firstname": "Oussama" }
]
```

REST renvoie *toujours un ensemble fixe de données*.

B. GraphQL — API flexible et optimisée

GraphQL permet de **demander exactement ce qu'on veut**.

Exemple :

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

Réponse :

```
{
  "firstname": "Oussama",
  "lastname": "Chakrellah"
}
```

Pourquoi c'est utile ?

Si une application mobile ne veut que 2 champs → elle ne télécharge que 2 champs ✔
REST lui enverrait beaucoup plus ✘

C. SOAP — API ancienne mais très sécurisée

SOAP utilise une enveloppe XML et un contrat WSDL.

Exemple SOAP :

```
<ws:customerByIdentity>
  <identity>A100</identity>
</ws:customerByIdentity>
```

Pourquoi les banques l'utilisent ?

- messages garantis
- contrôles stricts
- sécurité avancée
- idéal pour éviter les erreurs dans les transactions

Si tu fais un virement bancaire → tu veux **ZÉRO ERREUR**, donc SOAP reste populaire.

D. gRPC — API la plus performante (Google)

Utilise :

- Protobuf (format binaire)
- HTTP/2
- Connexions persistantes

C'est très rapide, utilisé dans :

- Netflix
- Amazon
- Uber
- Google

Exemple requête BloomRPC :

```
{
  "identityRef": "A100"
}
```

Réponse gRPC :

```
{
  "firstname": "Imane",
  "lastname": "Chakrellah"
}
```

gRPC est idéal pour **microservices** car il est très rapide et consomme très peu de réseau.

5. Le cœur du TP : un système bancaire simplifié

Le TP simule un vrai système bancaire :

Tu peux :

- créer un client
- créer un compte bancaire
- faire des virements
- récupérer l'historique des transactions

Chose importante :

Peu importe si tu utilises REST, SOAP, GraphQL ou gRPC → le service métier est le même.

6. Explication claire des tests

REST

Tu testes via Swagger → tout en JSON
Exemple :

```
GET /api/rest/customer/all
```

GraphQL

Tu tests via GraphiQL

```
customerByIdentity(identity: "A100") {
  firstname
}
```

SOAP

Tu tests via SoapUI → enveloppe XML
Exemple :

```
<ws:customers/>
```

gRPC

Tu tests via BloomRPC

👉 Tous retournent *le même résultat* sous un format différent.

7. Quelle est la différence ENTRE ces connecteurs ?

Technologie	Quand l'utiliser ?	Exemple concret
REST	API Web/Mobile	App de course, réseaux sociaux
GraphQL	besoin de données optimisées	Dashboards, App React/Flutter
SOAP	secteurs critiques	Banques, assurances
gRPC	microservices internes	Netflix, Google

8. Ce que tu dois retenir ABSOLUMENT pour réussir ton examen

1. REST renvoie des données fixes en JSON.

2. GraphQL renvoie exactement ce que le client demande.

3. SOAP utilise XML et un contrat strict (WSDL).

4. gRPC est le plus rapide (format binaire + HTTP/2).

5. Le service métier est unique → seule l'exposition change.

6. Chaque technologie répond à un besoin différent.

9. Résumé final

« Ce TP montre comment exposer un même service métier via quatre technologies différentes : REST, GraphQL, SOAP et gRPC.
Chacune a son utilité : REST est simple et universel, GraphQL optimise les requêtes, SOAP garantit la sécurité dans les environnements critiques, et gRPC offre la meilleure performance pour les microservices.
Cela m'a permis de comprendre les différences, les avantages et les cas d'usage réels de chaque type d'API. »