1) Chno howa gRPC

Imagine 3 applications :

- Application Java
- Application Python
- Application JavaScript

Baghin yhdrro m3a b3adhiyathom bzraaab…
REST kaytsst3ml JSON : kbir, kayakhod place, kaychri CPU.
GraphQL kaymchi HTTP1, mais mazal kyzid overhead.

gRPC = service super rapide, super léger, kaytsst3ml **HTTP/2** + **Protocol Buffers** (format binaire).

➡ **Resultat :**

- communication 10x أسرع
- messages sghr 60%
- streaming (communication bidirectionnelle live)
- multi-langage (Java ↔ Python ↔ C++ ↔ JS ...)

Daba, f TP drna **4 modèles de communication** :

Modèle	Darija	Chno kaydir
Unary	chi demande + chi réponse	exactement b7al REST
Server Streaming	demande wa7da, réponses bzaaaf	b7al live feed mn serveur
Client Streaming	bzaaaf dyal requêtes, réponse wa7da	client kay-sift data bechwiya
Bidirectional Streaming	les deux kaytsiftou data	b7al WhatsApp call (real-time)

2) Workflow f gRPC

Client → envoie message ProtoBuf → Server → ycalculi → yjawb → Client

REST : kaymchi GET /articles
GraphQL : kaymchi POST /graphql
gRPC : kaymchi **Appel direct** d'une méthode du service.

Daba, f gRPC kanktbou **contract** f fichier `.proto`
= b7al **WSDL** fi SOAP
= b7al **schema** f GraphQL

Hada fichier `.proto` kaygeneri code automatiquement (STUB).

3) Explication du TP, étape par étape

Étape 1 — Création du fichier .proto

F fichier `calculator.proto` :

- nom dyal service
- les méthodes
- structure dyal message li ghadi ytsifto

Darija :

fichiers .proto = contrat li kaygoul :
"had service kaykhdem b had les méthodes, o had les messages."

Example :

```
rpc sum(UnaryRequest) returns (UnaryResponse);
```

= chi demande (UnaryRequest) + chi réponse (UnaryResponse)

Étape 2 — Ajouter dépendances + plugin ProtoBuf

Had plugin :

- kay9ra fichier .proto
- kaygeneri les classes Java li katmchi m3a gRPC

Darija :

Proto compiler = machine li kt7awel .proto → Java code automatiquement.

Étape 3 — Génération du STUB

Kandiro :

```
mvn clean install
```

W kaytsn3o :

- CalculatorServiceGrpc.java
- Calculator.java

Hado howa **le code généré automatiquement**.

STUB = l-interpréteur li kay5lli client o serveur yfhmo b3adhiyathom.

Étape 4 — Développement du service Unary

Kankhdmo b modèle standard :
client kaytsift → serveur y7ssb → yjawb be réponse wa7da.

Code:

```
double result = a + b;
responseObserver.onNext(response);
responseObserver.onCompleted();
```

onNext = jib réponse
onCompleted = safi, sdat connection

Étape 5 — Création du serveur gRPC

```
ServerBuilder.forPort(9999)
 .addService(new CalculatorService())
```

hadi service li kayjm3 : port + service li bghiti t3ti.

Étape 6 — Test b BloomRPC

BloomRPC = Postman dyal gRPC.

Kaddir :

- load `.proto`
- tchoisi method
- tsift request

- tchof réponse

Étape 7 — Développement du client gRPC (Unary)

Client kaydir :

```
newBlockingStub(channel);
```

blocking = client kaytsna serveur sakn.

Étape 8 — Server Streaming

Server kayjawb b plusieurs réponses :

- 1ère seconde → addition
- 2ème → soustraction
- 3ème → multiplication
- …

nta katsift demande wa7da, serveur kayb9a ysiftek résultats b 1 par seconde.

Étape 9 — Client Streaming

Client kaytsift bzaaaf requests :

```
onNext()
```

o serveur kayjawb **gha f l'akhir** :

```
onCompleted()
```

nta katb3at 10 messages, w serveur kayjme3hom w yjawb mra wa7da.

Étape 10 — Bidirectional Streaming

Hadi **live communication** :

- client katsift number
- serveur langsung kayrja3 carré
- ma kaytsallich 7tta nta tgoul

b7al WhatsApp live : nti katsift data, serveur kayrja3 direct.

4) Fark kbiiiir ben gRPC o REST o GraphQL

Technologie	Darija	Chno kayn
REST	bgharraf (kbir)	JSON, HTTP1, simple mais slow
GraphQL	service m3a server	client kaytsift chno bgha
gRPC	turbo engine	HTTP2, binary, fast, streaming

5) Workflow global (Résumé final)

🟢 Client —Unary→ Serveur

→ Réponse wa7da

🟢 Client —Request→ Serveur —Stream→ bzzaf dyal réponses

🟢 Client —Stream→ Server —Response wa7da

🟢 Client —Stream→ Server —Stream→ Real-time

6) daba bssa7 t-fham TP kaml

Hadchi howa :

- gRPC = technologie dyal real-time & performance
- .proto = contract
- Stub = code auto généré
- 4 modèles : Unary / Server-Stream / Client-Stream / Bi-Stream
- BloomRPC = Postman dyal gRPC
- T9dr tdir server + client b Java wela technologies okhra