Rapport développement d'une Application de Réservation d'Hôtels Distribué "gRPC" (UE HAI704I à l'Université d Montpellier)

MANSOUR Malik **Génie Logiciel**

December 16, 2024

1 Introduction

Ce rapport présente le développement d'une application de réservation d'hôtels distribué en utilisant des services web springbot (langage Java) et en utilisant gRPC (Remote procedure call). L'objectif principal est de fournir une solution intuitive et performante pour permettre aux utilisateurs de rechercher et réserver des hôtels répondant à leurs besoins spécifiques.

Spécifications Fonctionnelles

1. Consulter offre des Hôtels

L'application offre une CLI permettant à l'utilisateur de saisir :

- La ville de séjour.
- La date d'arrivée et la date de départ.
- Nnombre d'étoiles.
- Le nombre de personnes à héberger.

Résultat attendu : En réponse, l'application retourne une liste d'offre correspondant aux critères de recherche avec les informations suivantes :

- Nom de l'hôtel.
- Adresse détaillée (pays, ville, rue, position GPS).
- Prix à payer.
- Nombre d'étoiles.
- Nombre de lits proposés.

2. Réservation d'un Hôtel

Après avoir sélectionné un hôtel, l'utilisateur doit fournir :

- Le nom et prénom de la personne principale à héberger.
- Les informations de la carte de crédit pour le paiement.

Ces données sont utilisées pour finaliser la réservation.

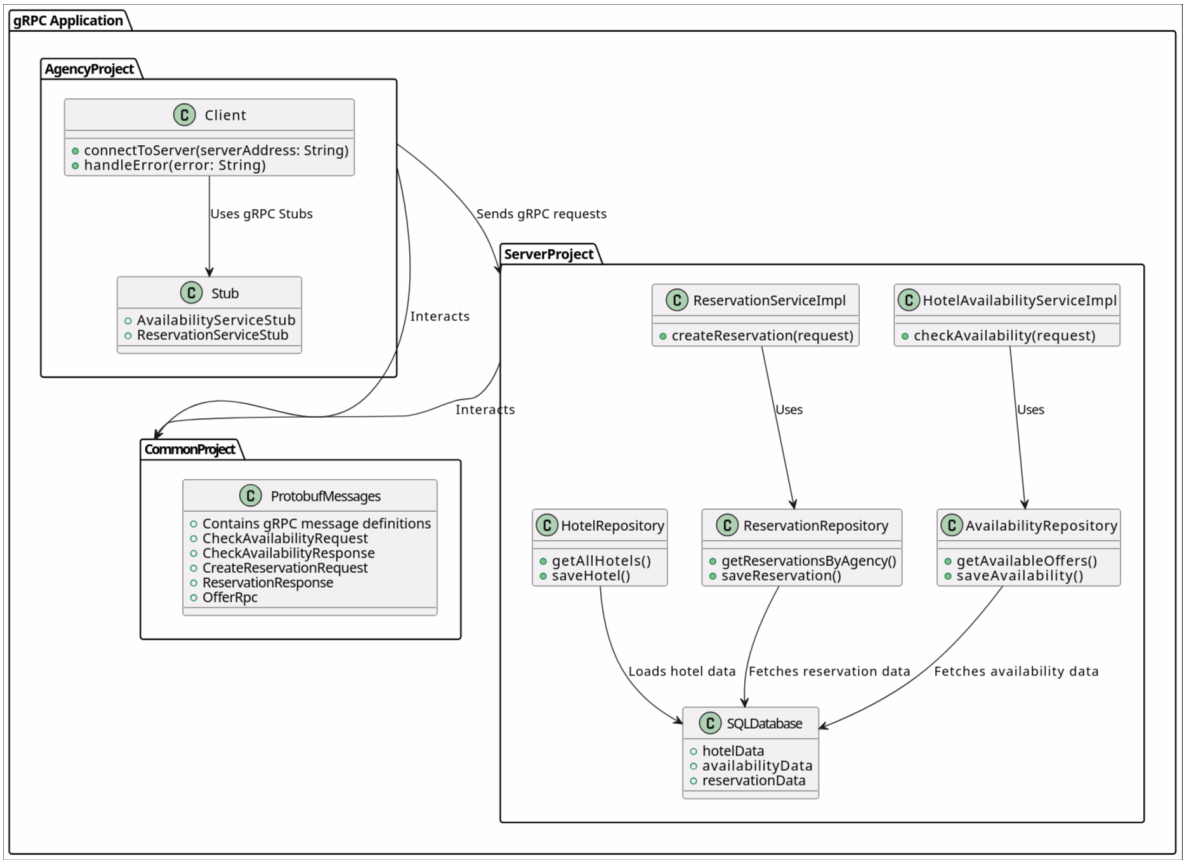

Figure 1: Conception de la solution

Architecture de l'Application

Le solution est composé de 3 projet principal: **Agency Project**, **Common Project**, **Server Project**

1. Backend : Services gRPC

- Développé en Java avec le framework **Spring Boot** pour faciliter la création des services web.
- API exposant des services gRPC pour gérer les opérations suivantes :
  - Recherche d'offres selon les critères de l'utilisateur.
  - Validation des informations utilisateur pour la réservation.
  - Enregistrement des réservations dans une base de données.

2. Implementation gRPC

1.0.1 Fichier .proto

Sur gRPC pour effectuer la recherche des offres et les réservations on doit créer des services et des messages dans vos fichiers, pour faire ceci on doit utiliser les **.proto**, ils sont des fichiers de définition utilisés par Protocol Buffers (Protobuf) et ils permettent de définir des messages (les structures de données) et des services (les RPC - Remote Procedure Calls)

AvailabilityService.proto: Contrôleur principal pour gérer les requêtes de disponibilité des offres qui permet de vérifier la disponibilité des hôtels et d'obtenir la liste complète des offres d'hébergement et il a son role de:

```
package availability;

import "google/protobuf/timestamp.proto";

/// Hotel Availability Request
message CheckAvailabilityRequest {
  int32 agence_id = 1;
  string username = 2;
  string password = 3;
  google.protobuf.Timestamp start_date = 4;
  google.protobuf.Timestamp end_date = 5;
  string city = 6;
  int32 min_stars = 7;
  int32 number_of_persons = 8;

}

/// Hotel Availability Response
message CheckAvailabilityResponse {
  repeated OfferRpc offers = 1;
}
```

Figure 2: AvailabilityService.proto

- Centralisation de la Vérification des Disponibilités : Les hôtels disponibles sont filtrés en fonction des critères fournis par l'utilisateur dans CheckAvailabilityRequest.
- Offres Complètes : La méthode GetAllOffers permet de récupérer toutes les offres d'hébergement enregistrées, utile pour des agences ou applications cherchant à afficher tout ce qui est disponible.

ReservationService.proto: permet de gérer des réservations et ce role est de définir les messages échangés :

- Les messages comme CreateReservationRequest et ReservationResponse structurent les données qui transitent entre le client et le serveur. Exemple : CreateReservationRequest contient les informations nécessaires pour créer une réservation (nom, dates, offre et agence).
- Le service ReservationService offre deux appels RPC (Remote Procedure Calls) :
  - CreateReservation : Permet de créer une réservation en recevant les informations nécessaires (requête) et en retournant les détails complets de la réservation (réponse).
  - SaveReservation : Permet de sauvegarder une réservation existante, en envoyant et en recevant un objet ReservationResponse.

```
message CreateReservationRequest {
  string name = 1;
  google.protobuf.Timestamp start_date = 2;
  google.protobuf.Timestamp end_date = 3;
  int32 offer_id = 4;
  int32 agency_id = 5;
}

message ReservationResponse {
  int32 id = 1;
  string name = 2;
  google.protobuf.Timestamp start_date = 3;
  google.protobuf.Timestamp end_date = 4;
  int32 offer_id = 5;
  int32 agency_id = 6;
  google.protobuf.Timestamp reservation_date = 7;
}

message Empty {}

service ReservationService {
  rpc CreateReservation(CreateReservationRequest) returns (ReservationResponse) {}
  rpc SaveReservation(ReservationResponse) returns (ReservationResponse) {}
```

Figure 3: ReservationService.proto:

Les définitions des services gRPC (HotelAvailabilityService) et les messages associés (CheckAvailabilityRequest, OfferRpc....) doivent être accessibles par plusieurs parties :

```
<dependency>
    <groupId>com.example</groupId>
    <artifactId>grpc.common</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Figure 4: Chemin du projet common

On doit ajouter ce chemin dans le fichier **pom.xml** pour :

- Le serveur qui implémente les services.
- Les clients qui consomment ces services.

1.0.2 Module Common

En plaçant les fichiers dans un module common, on évite la duplication et facilite la maintenance.

Cohérence entre les services : Les projets clients et serveurs utiliseront les mêmes définitions générées à partir des fichiers .proto. Cela garantit que les structures de données et les méthodes RPC sont identiques des deux côtés.

Modularité : Un projet dédié au code partagé (comme les stubs gRPC et les définitions de protocole) rend le système plus modulaire et facilite la gestion des dépendances.

Éviter les conflits : Placer les définitions et stubs dans un projet distinct limite les conflits liés à la génération des fichiers .proto si plusieurs équipes ou parties modifient le projet.

1.0.3 Fichier générés et d'autres artefacts

pour recuperer les fichiers qu'on doit utiliser dans notre implémentation on doit les générer chaque langage a sa propre methode de generation, comme on est en train d'utiliser **SpringBot** et **Java** et **maven** alors on doit utiliser**mvn clean install** pour recuper les fichiers generer

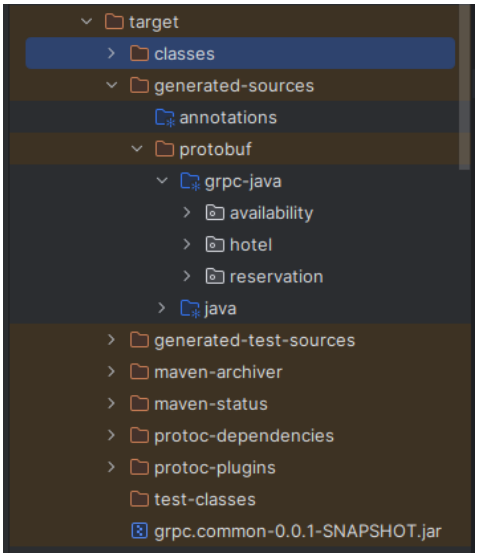

Figure 5: Target

Maintenant tout est pret pour implementer le server et le client

1.0.4 Module Server

Server est un application web SpringBot, elle expose 2 service (Les service montionné avant en parlant des fichier .proto) **AvailabilityService**, **ReservationService**

Package dans ce projet : comme dans la solution REST on a les packages **data**, **service**, **main**, **repository**, **model**, **exceptions**

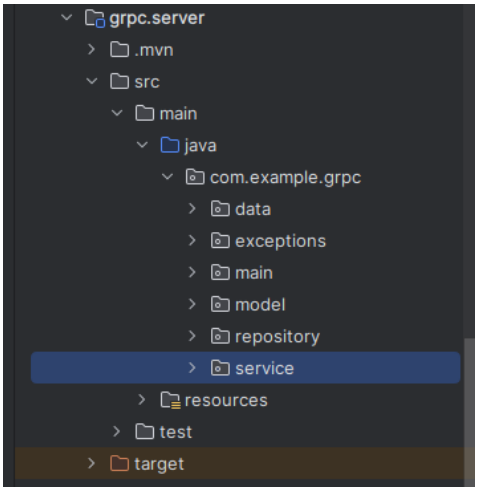

Figure 6: Package

Implementation du service Web : Comme on est sur gRPC on ne va pas utiliser les endpoints comme dans REST

- **Impemenetaion du service :** pour implementer notre web service on doit etndre la classe de base des classes generer par les fichiers proto en plus de l'utilisation de l'annotation **@GrpcService**

```
@GrpcService  2 usages
public class HotelAvailabilityServiceImpl extends HotelAvailabilityServiceGrpc.HotelAvailabilityServiceImplBase {
```

Figure 7: Implemenatation du service de base

- **Ajouter le web service :** pour que le service puisse exposer on doit utiliser un **@Bean** CommandLineRunner en passant tous les webservice en parametre avec le port du server

```
@Bean  no usages
public CommandLineRunner startGrpcServer(HotelAvailabilityServiceImpl hotelAvailabilityService,
                                         HotelServiceImpl hotelService,
                                         ReservationServiceImpl reservationService,
                                         @Value("${grpc.server.port:9095}") int grpcPort) {
    return String[] args -> {
        // Create a gRPC server with reflection enabled
        Server server = ServerBuilder.forPort(grpcPort) ServerBuilder<capture of ?>

                .addService(hotelAvailabilityService) capture of ?
                .addService(hotelService)
                .addService(reservationService)
                .addService(ProtoReflectionService.newInstance()) // Add reflection service
                .build();

        server.start();
        System.out.println("Server started on port " + grpcPort);
        server.awaitTermination();
    };
}
```

Figure 8: Main de l'application

- **Lancer Application Server :** pour lancer une application Sping on doit utiliser la commande **mvn spring-boot:run** avant il faut lancer la base de données et les fichier generés doivent etre genere par le module common sinon le server ne va pas lancer

```
, class: com.example.grpc.service.HotelAvailabilityServiceImpl
2024-12-16T14:55:25.525+01:00  INFO 64661 --- [grpc.server] [  restartedMain] n.d.b.g.s.s.AbstractGrpcServerFactory    : Registered gRPC service: hotel.HotelService, bean: hotelServiceImpl, class: com.example.grpc.servi
ce.HotelServiceImpl
2024-12-16T14:55:25.525+01:00  INFO 64661 --- [grpc.server] [  restartedMain] n.d.b.g.s.s.AbstractGrpcServerFactory    : Registered gRPC service: reservation.ReservationService, bean: reservationServiceImpl, class: com.
example.grpc.service.ReservationServiceImpl
2024-12-16T14:55:25.525+01:00  INFO 64661 --- [grpc.server] [  restartedMain] n.d.b.g.s.s.AbstractGrpcServerFactory    : Registered gRPC service: grpc.health.v1.Health, bean: grpcHealthService, class: io.grpc.protobuf.s
ervices.HealthServiceImpl
2024-12-16T14:55:25.525+01:00  INFO 64661 --- [grpc.server] [  restartedMain] n.d.b.g.s.s.AbstractGrpcServerFactory    : Registered gRPC service: grpc.reflection.v1alpha.ServerReflection, bean: protoReflectionService, c
lass: io.grpc.protobuf.services.ProtoReflectionService
2024-12-16T14:55:25.614+01:00  INFO 64661 --- [grpc.server] [  restartedMain] n.d.b.g.s.s.GrpcServerLifecycle          : gRPC Server started, listening on address: *, port: 9090
2024-12-16T14:55:25.623+01:00  INFO 64661 --- [grpc.server] [  restartedMain] com.example.grpc.main.Application       : Started Application in 3.547 seconds (process running for 3.887)
Server started on port 9095
```

Figure 9: Application Logs

Pour verifier que notre web service est bien deployé on vas voir des message de ce type **Registered gRPC service: reservation.ReservationService**

Sinon on utilise la commande **grpcul**

```
m1@m1-Precision-3581:~/Téléchargements$ grpcurl -plaintext localhost:9095 list
availability.HotelAvailabilityService
grpc.reflection.v1alpha.ServerReflection
hotel.HotelService
reservation.ReservationService
m1@m1-Precision-3581:~/Téléchargements$
```

Figure 10: Web service bien deployé

- **Teste avec Postman :** Pour tester les web services on peut utiliser **Postman**

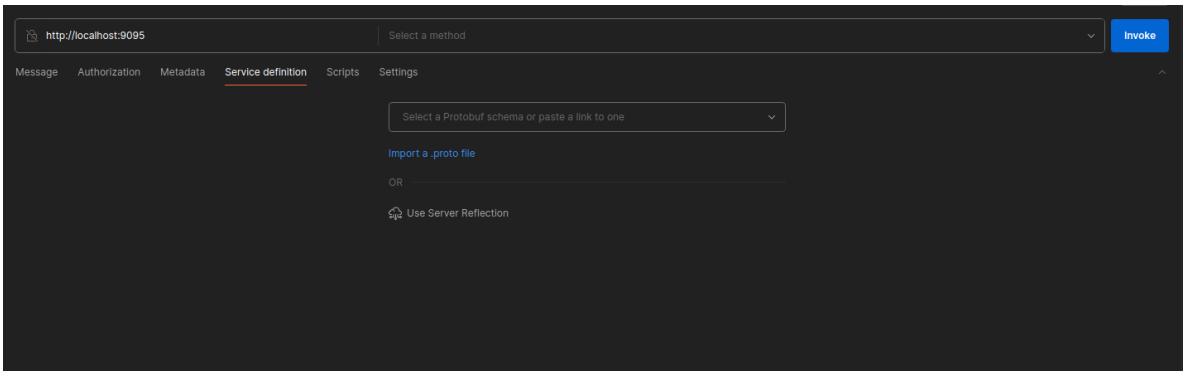

Figure 11: Interface pour les testes gRPC

Comme gRPC est basé sur les fichiers **.proto** alors on doit importer les fichier utilisé pour pouvoir lancer un test

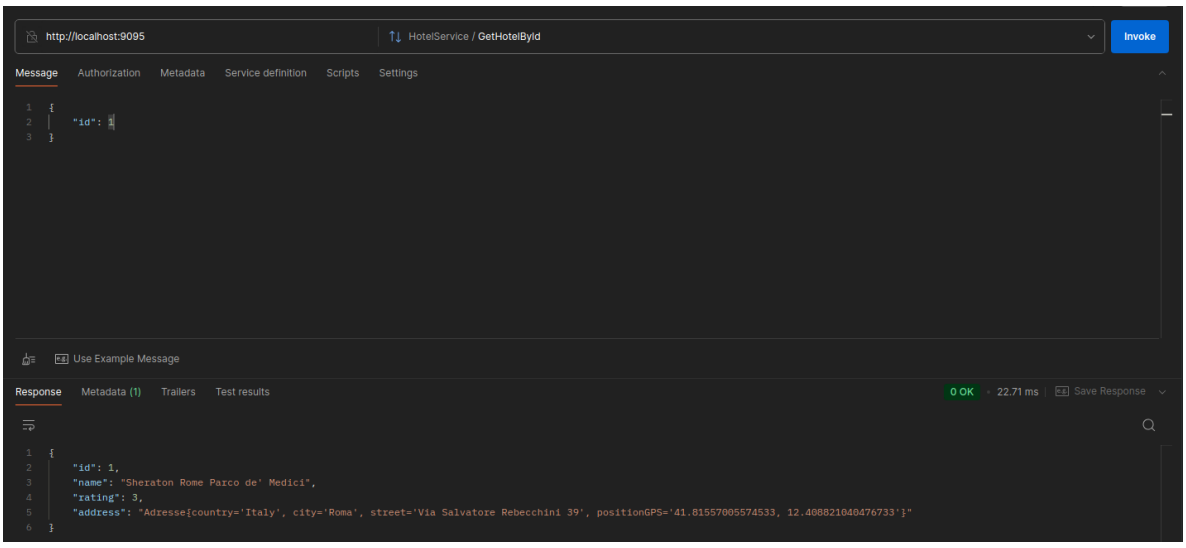

Figure 12: Succés

En cas d'erreur on peut consulter les logs du server car on a pas des status code comme dans HTTP (erreur codes du type 400,500....)

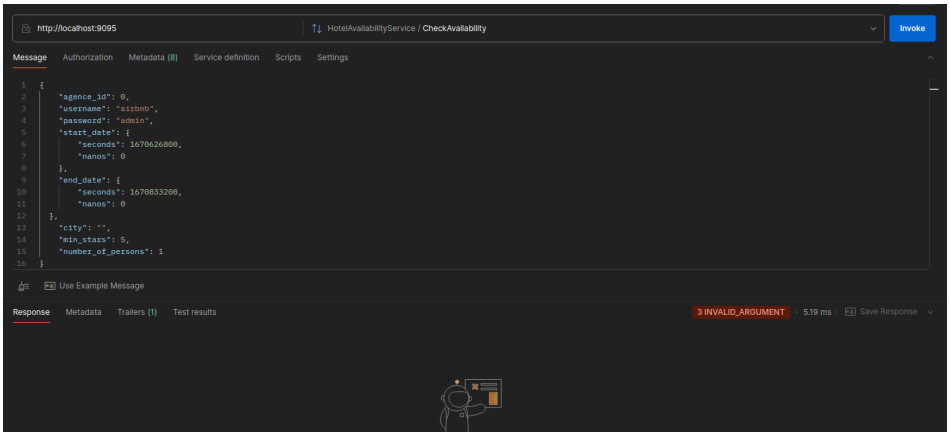

Figure 13: Une fausse requete pas de valeur d'erreur

Figure 14: Exemple d'erreur dans les Log

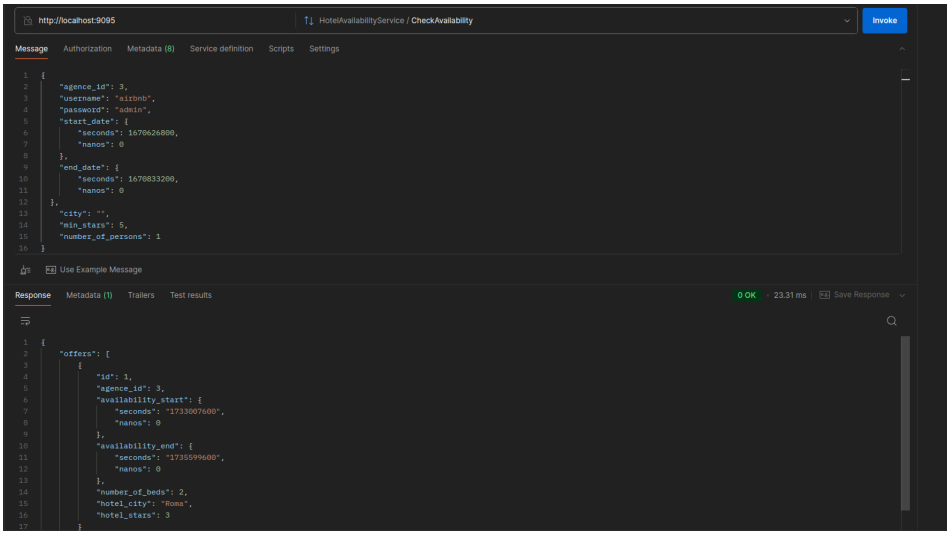

Figure 15: Succés

Pour la reservation

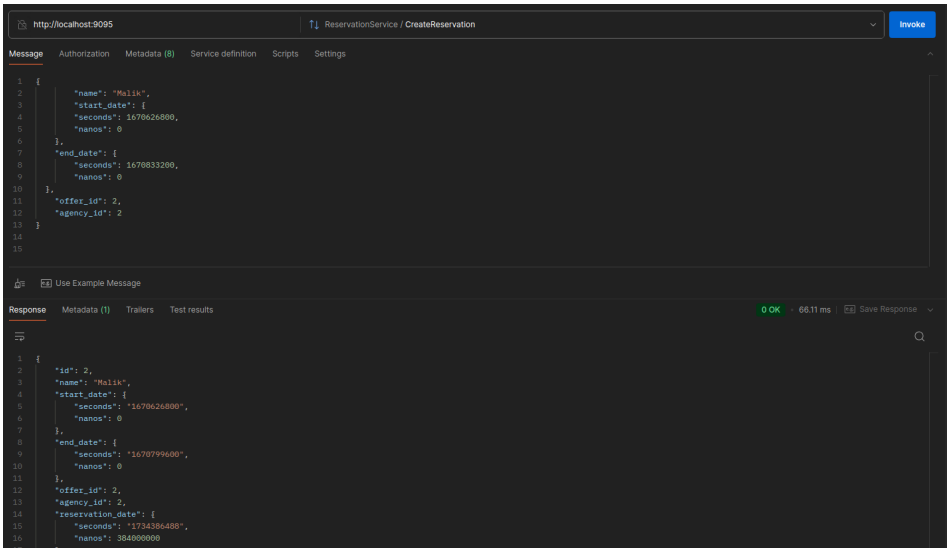

Figure 16: Succés d'une reservation

Figure 17: Reservation ajouté a la base de données

1.0.5 Module Agence

Ce module contient une CLI permet au client d'interagir et chercher des offres selon les critères mentionné avant.

Package du module : comme pour le server, le client a le meme package presque sauf un package CLI en plus

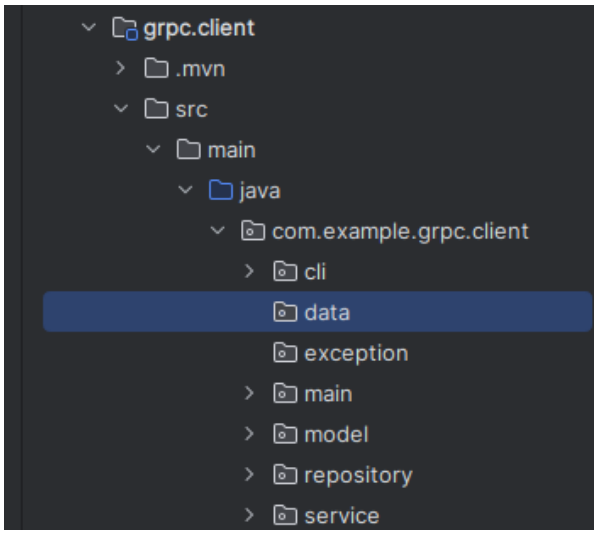

Figure 18: Package du client

Consomation des webservice : Comme dans REST et SOAP il y a la consomation des web-service a travers les proxy dans gRPC la consomation se fait d'une autre façon, on doit injecter un client gRPC dans une classe Spring pour établir une communication avec un service gRPC distant en utilisant les configurations dans **application.properties**

```
server.port=9096

grpc.client.availability-service.address=static://localhost:9095
grpc.client.availability-service.negotiation-type=plaintext
grpc.client.availability-service.enable-keep-alive=true


grpc.client.reservation-service.address=static://localhost:9095
grpc.client.reservation-service.negotiation-type=plaintext
grpc.client.reservation-service.enable-keep-alive=true
```

Figure 19: Application.properties

Address : l'adresse du serveur gRPC (ici localhost:9095 pour les deux services).
plaintext : spécifie si la communication se fait sans chiffrement (true pour le mode développement).

```
2024-12-16T21:58:04.250+01:00  INFO 220490 --- [grpc.client] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 9096 (http) with context path '/'
2024-12-16T21:58:04.261+01:00  INFO 220490 --- [grpc.client] [  restartedMain] com.example.grpc.client.Application      : Started Application in 3.673 seconds (process running for 4.009)
Welcome to the Hotel Availability CLI
Enter your email:
```

Figure 20: CLI lancé

```
2024-12-16T21:58:04.224+01:00  WARN 220490 --- [grpc.client] [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : Unable to start LiveReload server
2024-12-16T21:58:04.250+01:00  INFO 220490 --- [grpc.client] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 9096 (http) with context path '/'
2024-12-16T21:58:04.261+01:00  INFO 220490 --- [grpc.client] [  restartedMain] com.example.grpc.client.Application      : Started Application in 3.673 seconds (process running for 4.009)
Welcome to the Hotel Availability CLI
Enter your email: malik210999@gmail.com
Enter your phone number: 0657882408
Authentication successful. MANSOUR Malik Welcome!
Enter agency ID:
```

Figure 21: Authentifcation

```
Welcome to the Hotel Availability CLI
Enter your email: malik210999@gmail.com
Enter your phone number: 0657882408
Authentication successful. MANSOUR Malik Welcome!
Enter agency ID: 2
Enter check-in date (yyyy-MM-dd): 2024-12-05
Enter check-out date (yyyy-MM-dd): 2024-12-10
Enter minimum hotel stars: 5
Offer ID:2
Offer availability starts: 2024-12-01
Offer availability ends: 2024-12-31
```

Figure 22: Récuperation des offres

`https://github.com/malik-mnsr/-TP4_Archi_Dis_gRPC`
`https://drive.google.com/file/d/1Ftd32luJPy83Kz7RwrwbGlYD2rHFWWE3/view?usp=sharing`