[Tour Guide]

Besoins et conception

par
Frederic DISS

Sommaire

1. Présentation du projet

Application permettant de géolocaliser les utilisateurs, de leur fournir les attractions touristiques ainsi que les propositions de voyage personnalisées.

1.1 Objectifs du projet

Améliorer les performances de l'application notamment avec les containeurs Docker, corriger les bugs, ajouter des fonctionnalités.

1.2 Hors du champ d'application

Développement d'un client permettant d'afficher et d'interagir avec la solution logicielle actuelle.

1.3 Mesures du projet

Tests de performance (Fonctionnels), logs générés par l'application, appel des EndPoints de l'application avec l'outil PostMan.

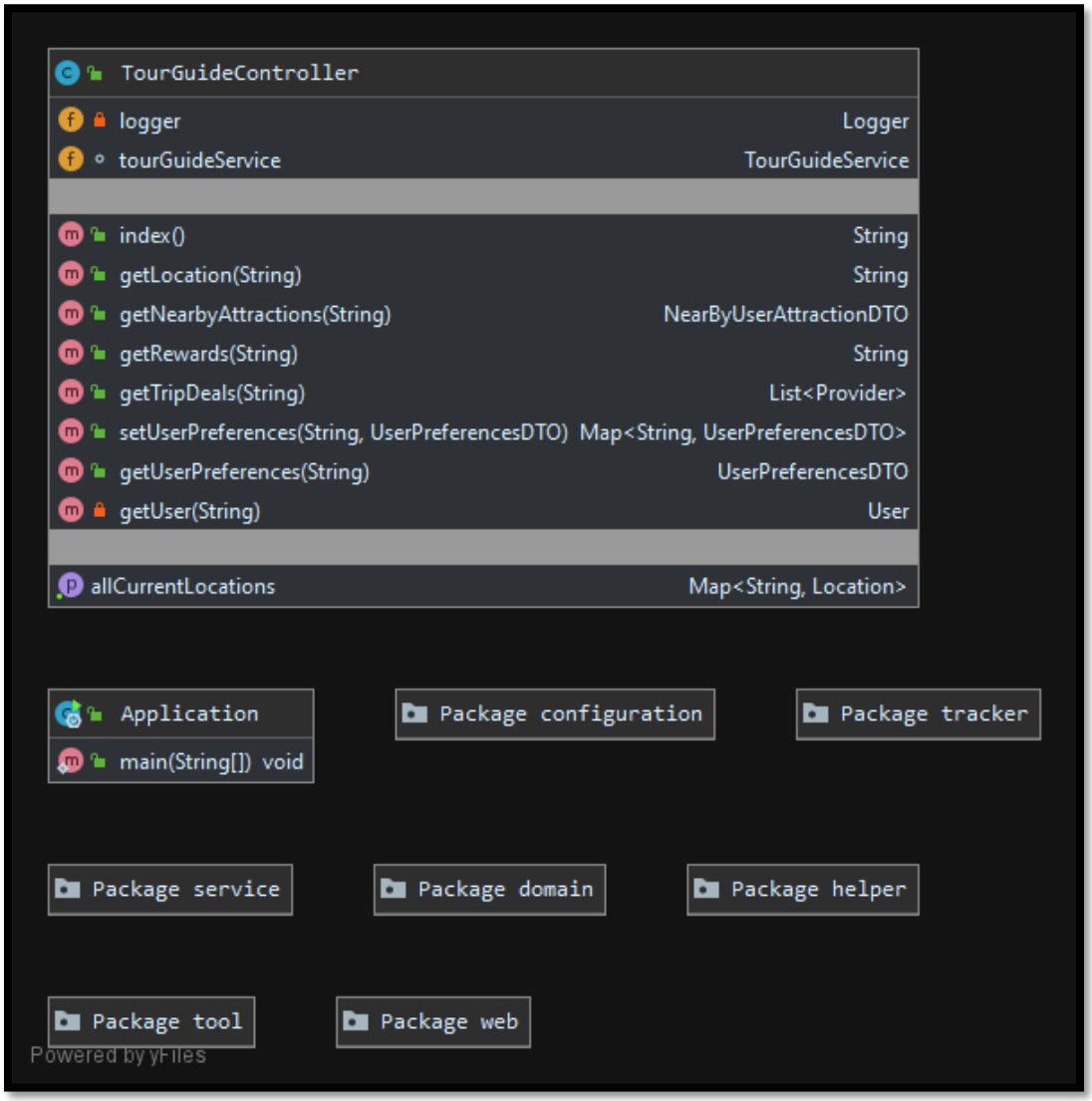
2. Caractéristiques

>> Pouvoir obtenir la localisation de l'ensemble des utilisateurs présents sur l'application. Possible si le EndPoint de l'application délivre, dans un délai raisonnable, une liste avec, pour chaque élément de la liste, l'identifiant UUID de chaque utilisateur et la localisation associée.
>> Un utilisateur doit pouvoir obtenir les 5 attractions touristiques proche de sa localisation actuelle, peu importe leur distance. Possible si l'utilisateur fourni son UUID et que l'application délivre une liste de 5 attractions quel que soit leur distance avec l'utilisateur.
>> Un utilisateur doit pouvoir obtenir des propositions de voyage en fonction des critères de préférence relatif au nombre de personne adulte, le nombres d'enfants, la durée du voyage, le prix le plus bas et le prix le plus élevé que l'utilisateur souhaite mettre. Possible si l'application permet de modifier ces préférences et délivre des propositions pertinentes au regard de ces dernières.

3. Solution proposée

3.1 Schémas de conception technique

Les Packages principaux :



Application est la classe root du projet tourguide qui permet de lancer l'application.

TourGuideController est une classe unique qui a pour fonction de communiquer avec le monde extérieur de l'application, ici sous la forme de data en Json. Du fait qu'il n'y a qu'une classe, il n'y pas eu de nécessité de créer un package spécifique pour les contrôleurs. A noter que la classe gère également les utilisateurs permettant de tester l'application.

Package Configuration va permettre de définir :

- Les métadata (properties) de l'application notamment utile pour :
    - construire les URL des Microservices (avec le port utilisé par ces derniers)
    - Definir si l'application doit fonctionner en mode docker ou local
- Les instances utilisables dans l'application SpringBoot, auxquels on peut accéder notamment grâce à l'annotation @Autowired

Package Tracker gère la géolocalisation des utilisateurs

Package Service contient l'ensemble des services de l'application :

- Permettant d'alimenter les End Points de l'application Tourguide
- Permettant d'appeler le ou les Microservices extérieur à l'application
- Nécessaire à l'accomplissement des tâches internes de l'application

Package Domain contient l'ensemble des POJO de l'application

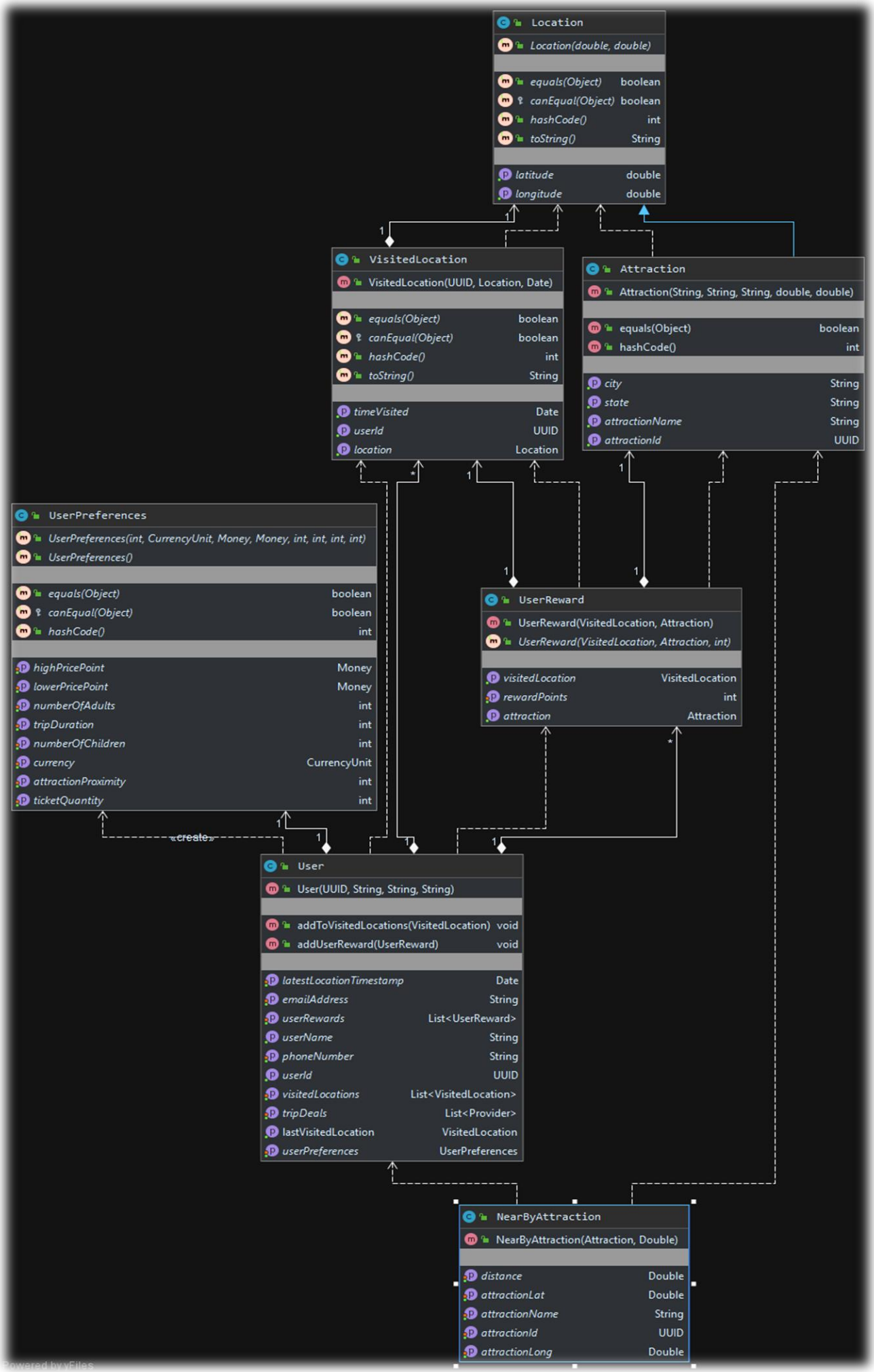
Package Tool contient l'ensemble des outils

- De gestion des listes de l'application
- Permettant le fonctionnement en mode local de l'application quand le mode Docker n'est pas activé.

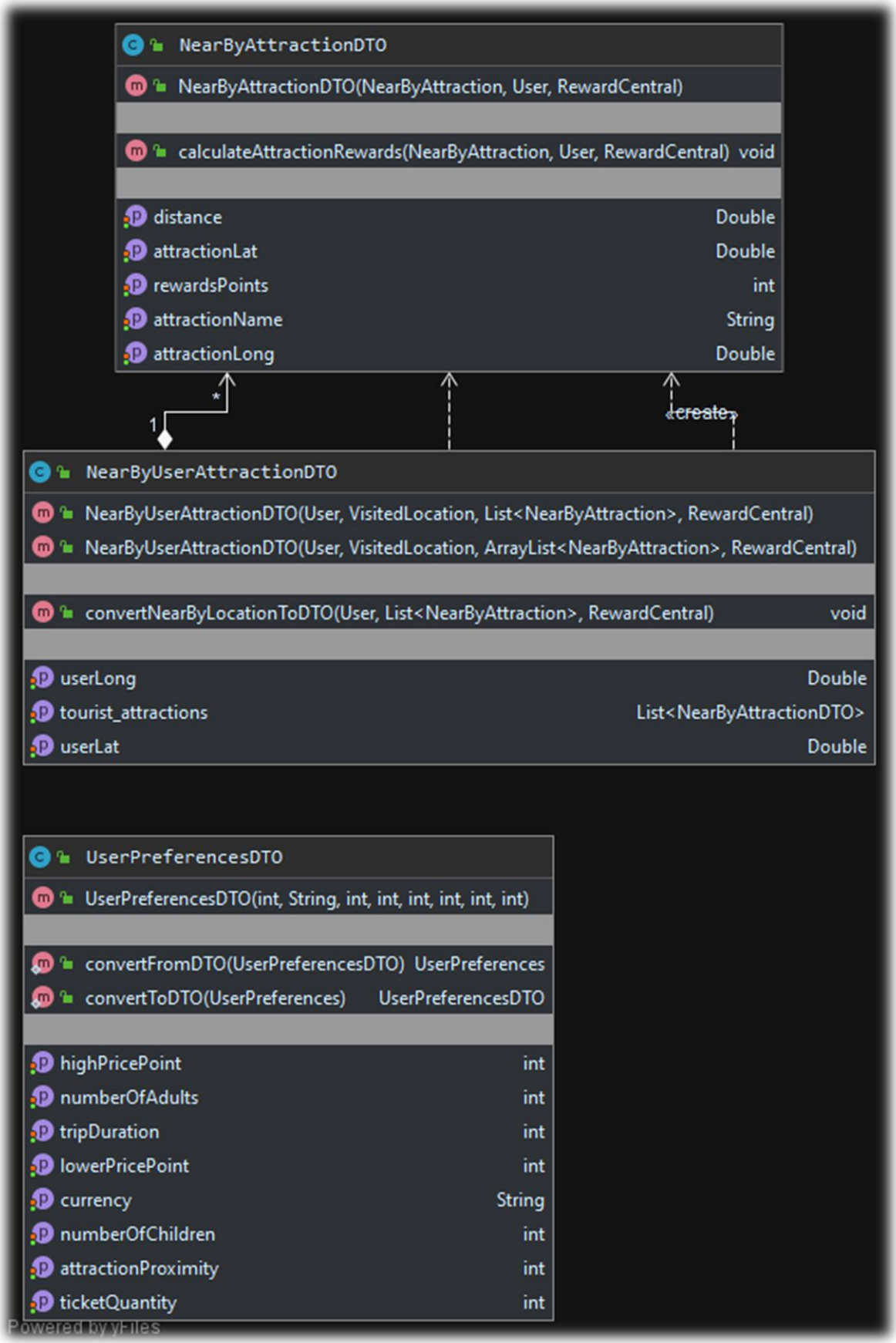
Package Web contient les DTO (Data Transfert Object) de l'application utilisés pour formater les réponses des EndPoints de l'application principale TourGuide.

Package Helper contient une fonction pouvant être utilisée par les Tests de performance de l'application.

Les Pojos de l'application Tourguide :



Les DTO de l'application TourGuide :



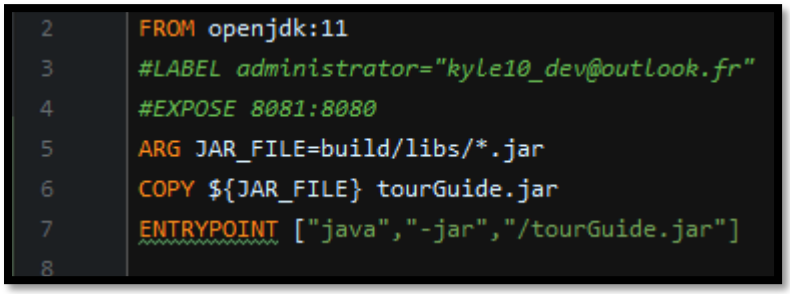
NearByAttractionDTO permet de structurer la réponse de l'EndPoint NearByAttraction.

UserPreferencesDTO permet d'échanger avec l'application Tourguide pour définir les préférences de l'utilisateurs et les visualiser : ses préférences seront utilisées pour filtrer les propositions de voyage disponible via l'Endpoint **getTripDeals**. Un filtre a été ajouté pour le prix le plus bas et le plus haut.

Certains EndPoint n'utilisent pas de DTO :
getAllCurrentLocations qui utilise une Map<UUID, lastLocation>

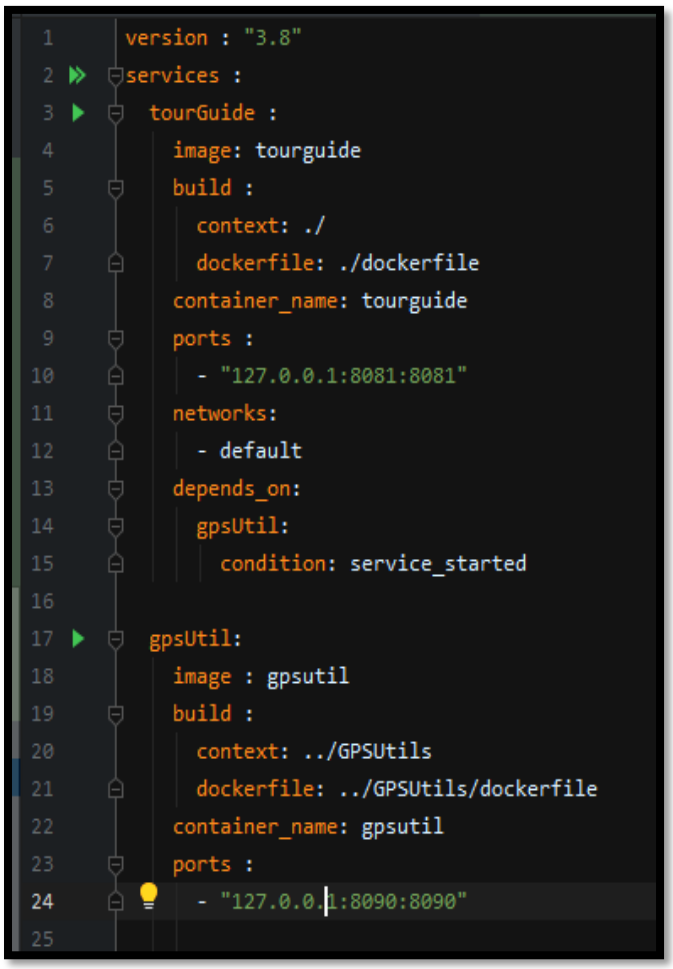
La dockerisation de TourGuide

Un fichier **DockerFile** a été ajouté dans le projet Tourguide



```
FROM openjdk:11
#LABEL administrator="kyle10_dev@outlook.fr"
#EXPOSE 8081:8080
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} tourGuide.jar
ENTRYPOINT ["java","-jar","/tourGuide.jar"]
```

Un fichier **Docker-compose.yml** permet d'interagir avec tous les DockerFile de l'application.

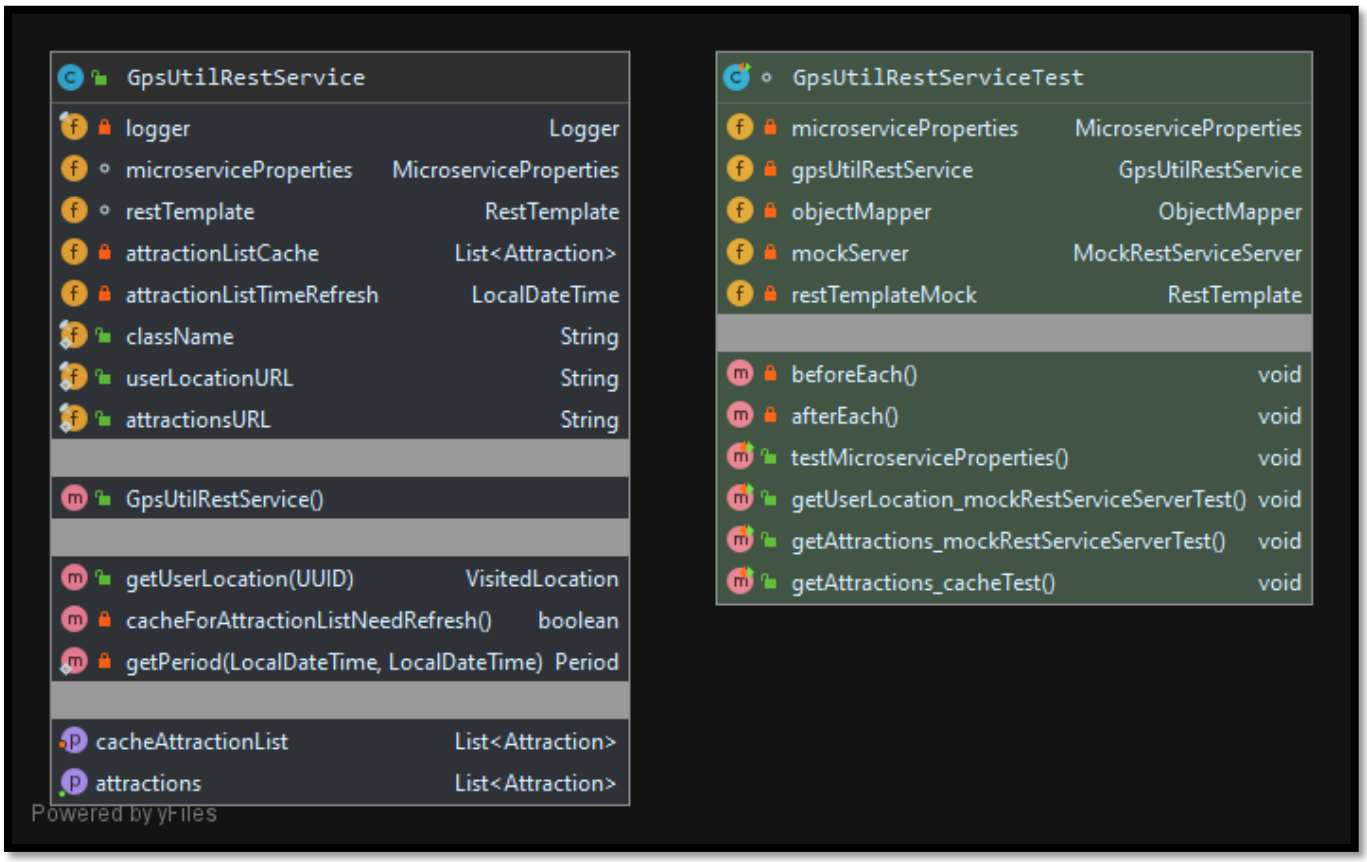


```
version : "3.8"
services :
  tourGuide :
    image: tourguide
    build :
      context: ./
      dockerfile: ./dockerfile
    container_name: tourguide
    ports :
      - "127.0.0.1:8081:8081"
    networks:
      - default
    depends_on:
      gpsUtil:
        condition: service_started

  gpsUtil:
    image : gpsutil
    build :
      context: ../GPSUtils
      dockerfile: ../GPSUtils/dockerfile
    container_name: gpsutil
    ports :
      - "127.0.0.1:8090:8090"
```

Docker-compose.yml a besoin d'avoir un build des projets (généré ans ce projet par Gradle) et permet ensuite, en une seule étape, de créer les containeurs Docker avec la ligne de commande : ***docker compose up -d*** (instructions présentent dans le fichier README lié à l'application TourGuide).

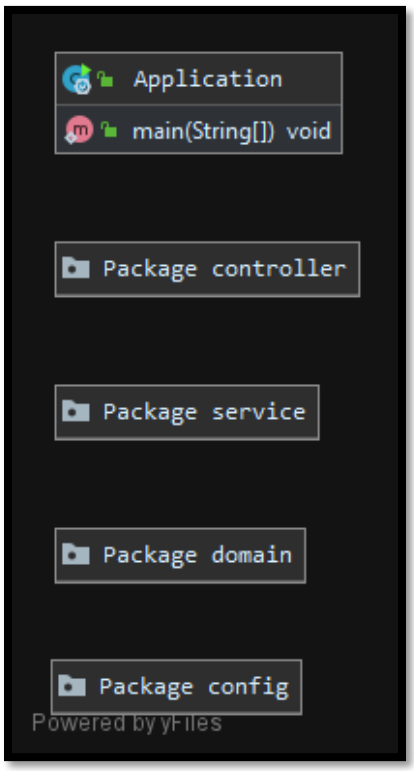
Une couche service avec **RestTemplate** a été mise en place dans TourGuide pour communiquer avec les Microservices. Un test a été mis en place avec **MockRestServiceServer** qui permet de vérifier que l'appel au microservice se déroule comme souhaité.



A noter qu'un cache permet d'éviter d'appeler le Microservice pour fournir la liste des attractions disponibles quand ce dernier a déjà été appelé dans les 10 minutes qui précèdent.

Création Microservice GpsUtils

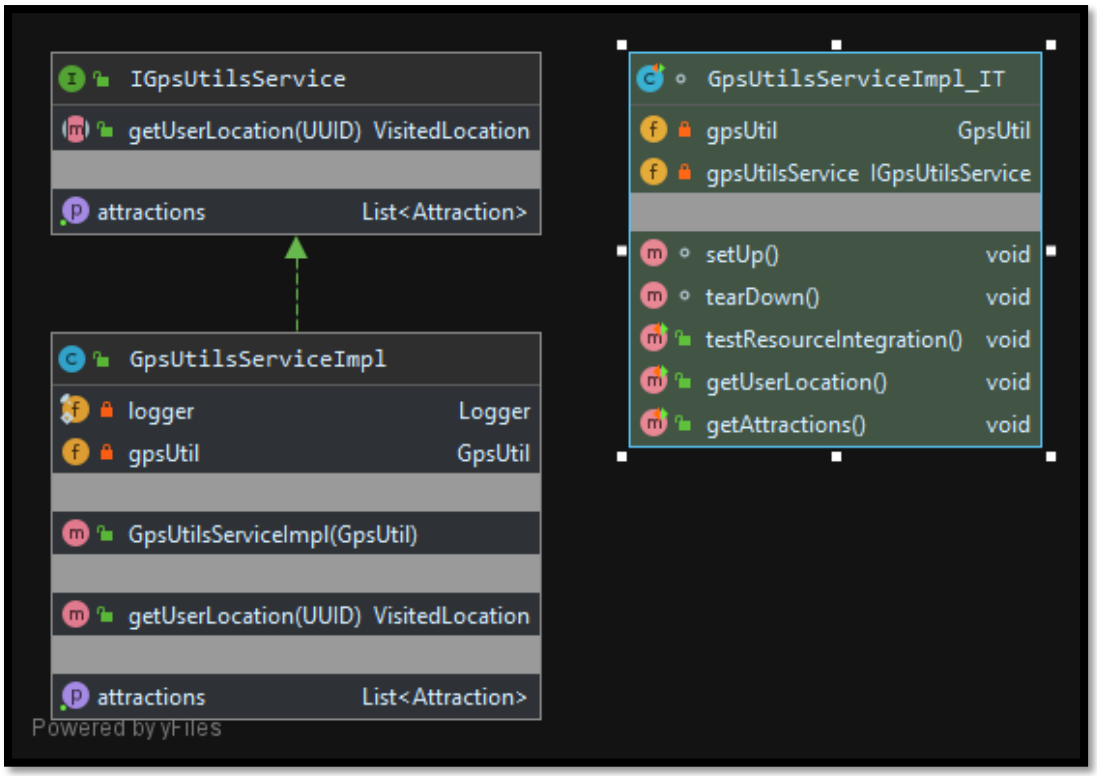
Un nouveau projet **GpsUtils** a été créée : le jar gpsUtil a été supprimé de l'application TourGuide et ajouté à GpsUtils. Les pojos relatifs ont été ajoutés à TourGuide. GpsUtils est désormais un microservice, contenant lui aussi un fichier *DockerFile* utilisé par *docker-compose* pour créer les images, les containeurs docker, et fixer le port du microservice.



Package config va permettre de définir les instances utilisables dans l'application auxquels on peut accéder notamment grâce à l'annotation @Autowired : ici nous n'avons que l'instance relative à la bibliothèque Jar gpsUtil intégré au Microservice.

Package Service permet d'avoir accès aux deux services proposés par le microservice et d'obtenir :

- la localisation de l'utilisateur
- la liste de toutes les attractions disponibles



Un Test d'intégration a été associé à la couche service, ainsi qu'a la couche contrôleur qui fait appel à cette couche service.

3.2 Glossaire

Tourguide : nom de l'application principale.

GpsUtil : nom d'une application secondaire, intégré au départ au projet principal sous forme de jar, puis externalisé sous forme de Microservice dans un conteneur Docker.

Reward Central et **Trip Pricer:** applications secondaires intégrées au projet principal sous forme de jar.

Attraction : attraction touristique que l'utilisateur peut aller visiter

Location : permet de localiser les attractions et les visited Location

Visited Location : localisation ayant été visité par l'utilisateur

Near By Attraction : attraction touristique proche de la localisation actuelle de l'utilisateur.

User : utilisateur de l'application

User Preference : les préférences de l'utilisateurs pour ses voyages : nombre d'enfants, nombre d'adultes, nombre de ticket, durée du voyage, prix minimum et maximum du voyage.

User Reward : chaque visite d'une localisation/attraction permet d'obtenir un nombre de point de récompense.

Provider : agence de voyage proposant des voyages touristiques aux utilisateurs

3.3 Spécifications techniques

***Docker* 20.10.6** *et Docker CLI* sont utilisés pour dockeriser les microservices et l'application principale.
Java 11 est utilisé comme langage de base pour le code.
Le framework Spring Boot 2.4.4 est utilisé pour structurer l'application et faire fonctionner l'application.
RestTemplate est utilisé pour interagir avec les microservices.
Gradle 5.6 est le gestionnaire de build du projet.
JUnit 5 est utilisé pour les tests unitaires. Certains tests fournis d'origine avec l'application continueront de fonctionner sur *JUnit4.*
Lombok est utilisé pour limiter le nombre ligne sur les Pojo de l'application.
Pojo-tester est utilisé pour tester les Pojo de l'application.
Jacoco est utilisé pour la couverture du code (Test unitaire, intégration, fonctionnel)
AnnotationProcessor pour la génération des MetaData (Properties data)

3.4 Solutions alternatives

3.5 Calendrier prévisionnel et exigences

Déploiements de TripPricer et RewardCentral en Microservice en respectant l'architecture docker mise en place.