Note technique – Projet TransConnect

# Structure de la solution

Le projet est composé de 5 blocs distincts : TransLib, TAPI, BDD, TranscoTests & l’interface web.

## TransLib

Le projet TransLib regroupe toutes les briques logiques de la solution. Il comporte notamment les modules suivants (non exhaustif) :

* Auth

Le module Auth ou Authentification couvre la gestion des utilisateurs : création, suppression, login, …

* Collections

Le module collections comprend une série de Collections ‘custom’ conformes à l’interface IListAlike. La collection List par défaut est réécrite. Les collections BinaryTree, MultiNodeTree, ChainedList sont également supportées.

* Vehicle, Person & Order

Ces 3 modules servent d’interface et contienne un palette de classes et de fonctions pour récupérer ou écrire des informations dans la base de données mySQL.

Le module Order permet la gestion des commandes, l’attribution automatique de chauffeur, de véhicule.

* Itinerary

Le module Itinerary propose un calcul d’itinéraire reposant sur l’algorithme [A\* (ou A star)](https://fr.wikipedia.org/wiki/Algorithme_A*). Plusieurs heuristiques sont proposées ([Manhattan distance](https://fr.wikipedia.org/wiki/Distance_de_Manhattan) & [Euclidean distance](https://en.wikipedia.org/wiki/Euclidean_distance)), et plusieurs méthodes de calcul de coût (Distance, durée, péage)

*Figure 1 : Carte de debug des nœuds pour l'algorithme A\**

Ce module repose sur un set de données construites à partir de ce dataset public.  
<https://www.data.gouv.fr/en/datasets/routes-europeennes/>

Les données & scripts de construction sont disponible dans Routing\_maps, la sortie est ‘nodes.json’.

## TAPI

Le projet TAPI (TransConnect API) est une API utilisant le framework ASP.NET. Il fait l’interface entre le frontend (ou interface utilisateur) et la partie logique. Ses fonctions ou ‘endpoints’ sont appelées via des requêtes http ou https.

Nous utilisons Swagger (renommé en OpenAPI) pour tester individuellement ces fonctions. L’interface est automatiquement chargée au lancement du projet.

Figure 2 : Interface de Swagger

Figure 3 : Swagger - test de l'endpoint ListVehicles

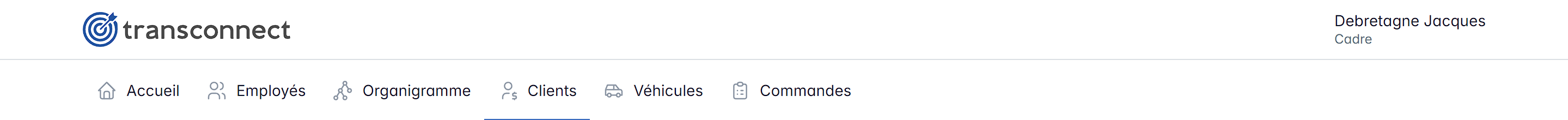
Pour chaque endpoint, son nom décrit sa fonction.

## TranscoTests

Projet de tests unitaires de la solution, il comporte principalement des tests sur la classe List.

## Interface Web

L’interface web est le point d’entrée de l’utilisateur. On retrouve une page de login, et des pages d’administration pour la gestion de l’entreprise



# Lancement du projet

Pour la connexion, utiliser les identifiants

User : **pierre.dupont@tmail.com** mot de passe : **password**

Pour tester le projet, plusieurs solutions sont disponibles. La première est d’accéder au site

(Interface) <https://transconnect.merll.eu/sign-in>

(Swagger) <https://transconnect.merll.eu/swagger/index.html>

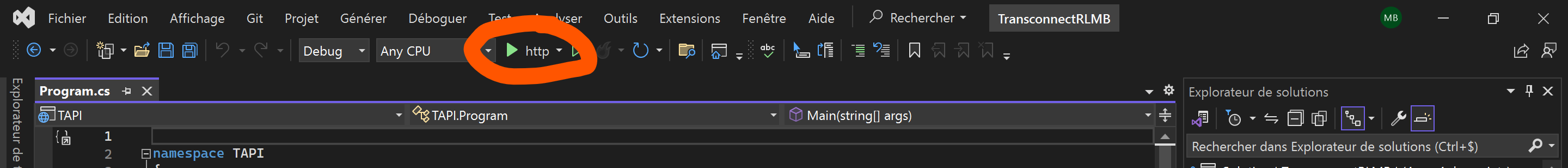
La seconde option est le lancement manuel à partir des sources.

1. Lancer la BDD

Utiliser le script de génération présent dans /data/transcodb.sql et le lancer dans une base de données mySQL

1. Lancer l’API

Lancer la solution en indiquant TAPI comme projet de lancement



1. Lancer le front-end

Ouvrir une invite de commandes dans "./frontend" et lancer test\_server.py à l’aide de la commande "python test\_server.py"

1. Accéder aux pages

(Interface) <http://localhost:8000>

(Swagger) Lancé automatiquement http://localhost: 5132/swagger/index.html

# Répartition du travail

Pour organiser notre travail, nous avons utilisé le logiciel git, via un repository GitHub.

Annexe

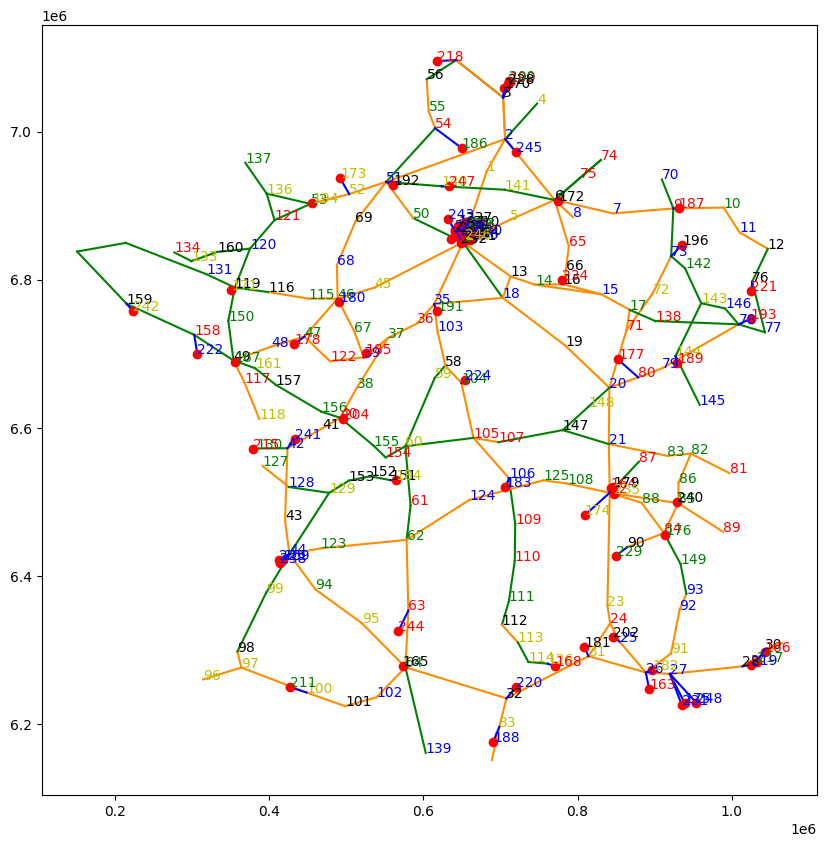


Figure 1 : Carte de debug des nœuds pour l'algorithme A\*

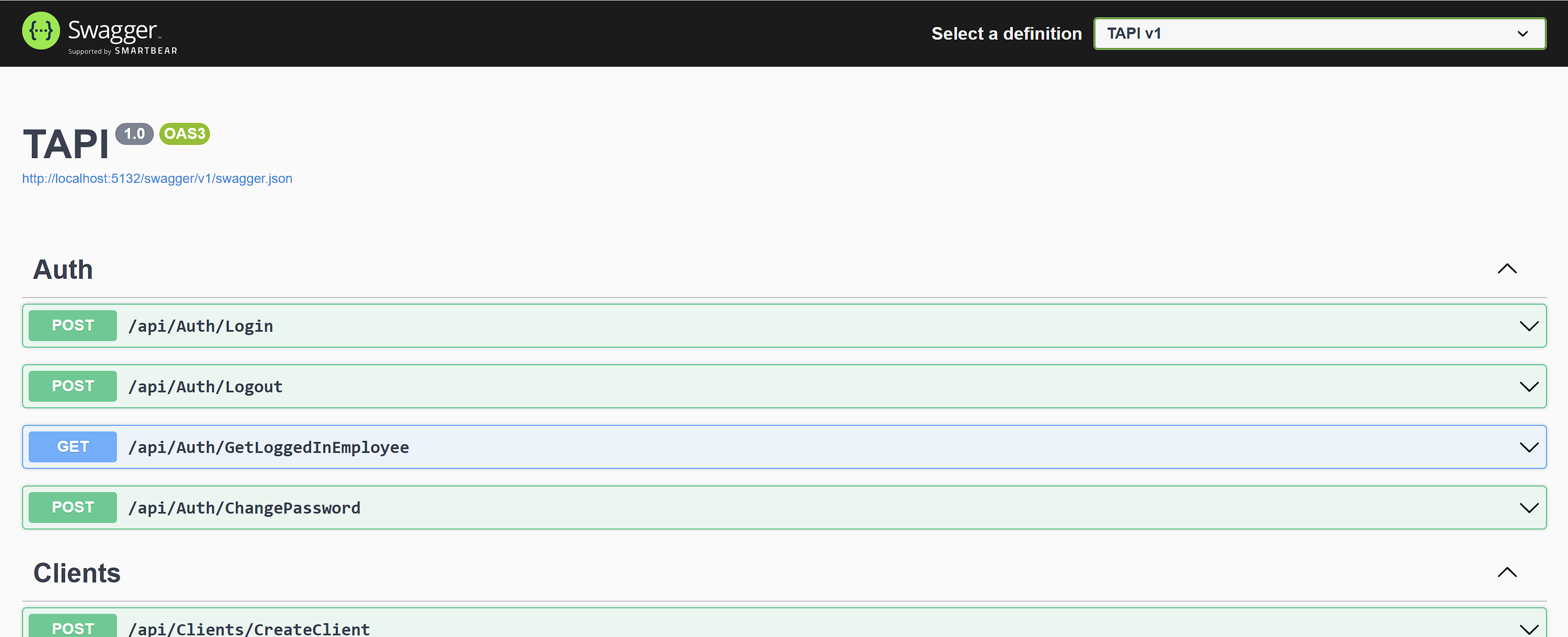


Figure 2 : Interface de Swagger

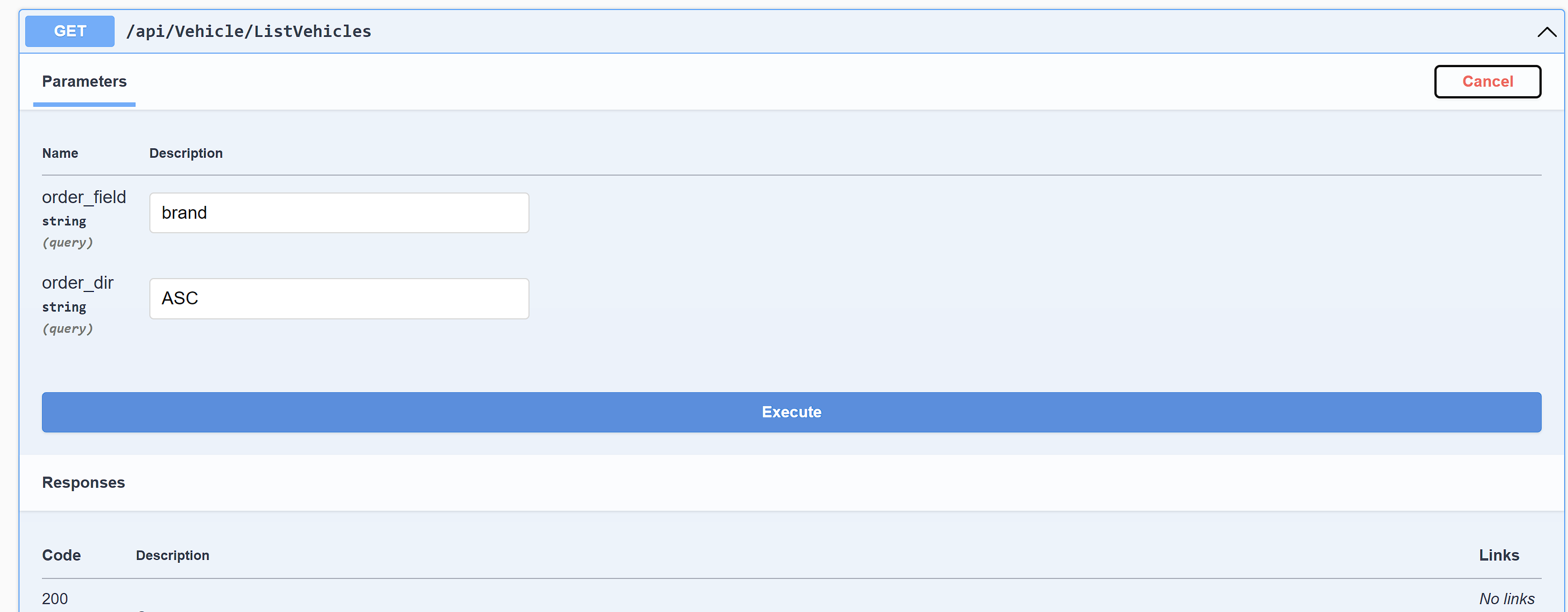


Figure 3 : Swagger - test de l'endpoint ListVehicles