DA *Distributed Algorithms*
May 2021

TD	**GraphQL**

Objective of the session :
Building a GraphQL server using Python.

1 Vehicle GraphQL Server

Consider the following example of a company that has vehicles that can be used by its employees or even rented by other people. This company would like to have information about the vehicles it has and make this information available to all employees and potential users (we will consider only some vehicles and some of their characteristics). For this purpose, a graphQL server is implemented to provide this information to the different clients.

The first table shows a subset of the available vehicles with some of their characteristics and the people who are in charge of the maintenance of the vehicles. The second table shows information about the people in charge of the maintenance of the vehicles.

ID	Manufacturer	Model	ModelYear	VehicleType	Gearbox	FuelType	Reserved	PersonID
1	Renault	Capture	2019	SUV	Automatic	Hybrid	False	1
2	Peugeot	308	2020	Car	Manual	Gasoline	False	1
3	Ford	Focus	2015	Car	Manual	Gasoline	False	2
4	Reanult	Clio	2021	Car	Manual	Gasoline	False	1
5	Mercedes Benz	EQC	2019	SUV	Automatic	Hybrid	False	2

ID	Name	LastName
1	Tony	Stark
2	Bruce	Banner

Exercises

The objective of this exercise is to create a service using a graphQL server so that the company can have an inventory of available vehicles that can be accessed by different clients. This GraphQL API allows to : Create new vehicles, List all vehicles and persons in charge of the maintenance, Mark a vehicle as reserved, Change the characteristic of a vehicle, Delete a vehicle.

1. **Download the directory graphQLServer.zip.**
2. **Go into the direcotory graphQLServer and install Flask [1], Ariadne [2] and Flask-SQLAlchemy [3].**

   ```
   pip install flask ariadne flask-sqlalchemy
   ```

1. A simple framework for building web servers in Python.
2. A library for using GraphQL applications.
3. An extension for Flask to use SQLAlchemy within a Flask application. SQLAlchemy allows to interact with SQL databases using Python.

3. **To tell Flask which is the app application instance. Set the environment variable FLASK_APP to the name of the top-level Python file that has the app, which in this case is main.py.**

   `export FLASK_APP=main.py`

   (On Windows, replace export in the command above with `set`).

   Start the Flask server by running the following command :

   `flask run`

   Visit `http://127.0.0.1:5000` in your web browser to confirm that the server is running (you should see Hello World ! on the browser).

4. **Take a moment to familiarize yourself with the project files.**

5. **The file `initdb.py` creates and initializes the vehicles.db database with the information from the previous tables. Execute it `python3 initdb.py`**

6. **The resolvers for the following queries and mutations detailed in the `schema.graphql` file must be implemented in the `queries.py` file. As an example, the resolvers for the queries `vehicles` and `vehicle` are already implemented.**

```
type Query {
  vehicles: VehiclesResult!
  vehicle(vehicleId: ID!): VehicleResult!
  persons: PersonsResult!
  person(personId: ID!): PersonResult!
}
type Mutation {
  createVehicle(manufacturer: String!, model: String!, modelyear: String!,
       vehicletype: String!, gearbox: String!, fueltype: String!,
       personid: ID!): VehicleResult!
  deleteVehicle(vehicleId: ID!): DeleteVehicleResult!
  markReserved(vehicleId: ID!): VehicleResult!
  updateVehicle(vehicleId: ID!, manufacturer: String!, model: String!,
       modelyear: String!, vehicletype: String!, gearbox: String!,
       fueltype: String!, personid: ID!): VehicleResult!
}
```

   To make queries and mutations available on the GraphQL server, do not forget to link them in the `main.py` file.

   Verify that the resolvers work properly by accessing the following link `http://127.0.0.1:5000/graphql` in your web browser and performing queries and mutations.

7. **Use the client of the previous TD to make queries and mutations on the graphQL server (you must change the link in the client so that the queries are directed to the server).**

2 Library GraphQL Server

A library wants to make available the books it owns and also information about the authors of these books. For this purpose, a graphQL server is to be implemented.

Exercises

The objective of this exercise is the creation of a graphQL server (you can use the server from the previous exercise as a guide).

1. **You must first define the graphQL schema, using the GraphQL Schema Definition Language (SDL). The book and author types should have the following form, feel free to add other characteristics and even define new types if you consider it necessary.**

```
type Book {
  id: ID!
  title: String!
  description: String!
  author: String!
}
type Author {
  id: ID!
  name: String!
  lastname: String!
  books: [Book]
}
```

2. **Define the Book and Author models to build the tables in the database and add the instances you need to test the server.**
3. **Finally, implement the queries and mutations (along with their respective resolvers) that you consider useful. Do not forget to test that the resolvers work properly.**