Assignment: Remote Procedure Calls with gRPC

This assignment introduces the concept of Remote Procedure Calls (RPC) often used in building distributed systems. The goal is to become familiar with gRPC, a popular open source RPC framework, through a hands-on programming exercise. You may read more about gRPC here.

Getting started

Follow the tutorial on MyCourses, which provides the necessary instructions to set up gRPC and links to standard tutorials that describe the core concepts required for this assignment. This assignment is coded in Python (**version 3.6** or greater).

Task

The assignment is to create a gRPC server that functions as a restaurant. Requests with an order ID and a list of items will be sent to the server and it will respond with the order ID and a **ACCEPTED** or **REJECTED** status. You'll be provided with a scaffolding application, study the `restaurant.proto` file and use what you learned from the MyCourses tutorial to fill out `restaurant_server.py`

Code structure

Note

To get started download the scaffolding application **`here`**.

The files in the scaffolding application are as follows:

- `restaurant.proto` contains the message definitions and the following service definition:
    - `FoodOrder` which will only receive orders that contain food items.
    - `DrinkOrder` which will only receive orders that contain drink items.
    - `DessertOrder` which will only receive orders that contain dessert items.
- `restaurant_pb2.py` generated automatically from `restaurant.proto`.
- `restaurant_pb2_grpc.py` generated automatically from `restaurant.proto`.
- `restaurant_server.py` which **you** will need to fill out.

Requirements

Your server should fulfil the following requirements:

- Initializing a gRPC server to `localhost` with the port set by the first command line argument
- Implement the functions defined in `restaurant.proto`
- When a request is received to one of the functions, the items in the request should be checked against the arrays defined in `restaurant_server.py`.
- If all the items from the request are in the restaurant menu, respond with an **ACCEPTED** status and the original **order ID**.

- If one or more items from the request is **NOT** in the restaurant menu, respond with a **REJECTED** status and the original **order ID**.
- `FoodOrder`, `DrinkOrder` and `DessertOrder` will only receive items from their respective categories.

For example:

When the function `DrinkOrder` receives the following request:

```
orderID="12345abc"
items=[ "fizzy drink", "water", "water" ]
```

It should return:

```
orderID="12345abc"
status=ACCEPTED
```

The status would be **REJECTED** if **one or more** of the items did not exist in the arrays defined in `restaurant_server.py`.

Warning

More variables and methods can be added to the classes, but keep the existing ones.

DO NOT change existing method names OR signatures.

DO NOT alter the method names / signatures for the provided scaffolding as these interfaces are used to test your submission.

DO NOT alter the `restaurant.proto` file as the one given is the one used in the grader.

Testing your code

You can test your code locally before submitting by constructing a simple client similar to the one found in the gRPC quick start guide

Grading

You only need to submit **restaurant_server.py**

Our automated grading system will test your server against the requirements specified above.

Note

Test	Points
Request with **correct** items sent to function `FoodOrder` responds correctly	10
Request with **correct** items sent to function `DrinkOrder` responds correctly	10
Request with **correct** items sent to function `DessertOrder` responds correctly	10
Request with **incorrect** items sent to function `FoodOrder` responds correctly	10
Request with **incorrect** items sent to function `DrinkOrder` responds correctly	10
Request with **incorrect** items sent to function `DessertOrder` responds correctly	10