Technical Assignment

The problem

Mr Zorro owns a multi-storey parking lot that can hold up to **n** vehicles at any given point in time. The parking slots are numbered, beginning at 1 and increases with increasing distance from the entry point in steps of one. Mr Zorro has requested your help to design an automated ticketing system for his parking lot.

When a vehicle enters the parking lot, its vehicle registration number (i.e., number plate) and colour are noted. Then, an available parking slot is allocated. Following are the rules of parking slot ticket issuance:

Each customer should be allocated the nearest available parking slot to the **entry** point.

Upon exiting the parking lot, the customer returns the ticket which marks their previously allocated lot as now available.

The system should provide the ability to determine:

1. Registration numbers of all cars of a specific colour.
2. Slot number in which a car with a given registration number is parked.
3. Slot numbers of all slots where a car of a specific colour is parked.

The ticketing system should be operable via CLI.

Design your parking lot system to be testable, and extendable.

There is an expectation of 2 entry points to the car park (10 and 20)

There are **30 slots in the carpark**

Distance to the slots is based on the difference between the entry point no. to the slot no.

Docker Requirements

Your project should run in a docker container.

Your docker container should start the CLI at run time and be interfaceable

This means that we will start your docker container and then we will pass it inputs via CLI and it should respond

Expected Inputs

1. {new_vehicle: {vehicle_id: “ABC12345”, vehicle_colour: “red”, entry_point:10 }
2. {new_vehicle} {vehicle_id: “ABC12346”, vehicle_colour: “yellow”, entry_point: 20}
3. {new_vehicle: {vehicle_id: “ABC12347”, vehicle_colour: “blue”, entry_point: 0}
4. {new_vehicle: {vehicle_id: “ABC12348”, vehicle_colour: “red”: entry_point:20}

Expected Outputs

Parking slot numbers assigned per each request should not conflict with any others X4

Expected Inputs #2: Expected Outputs

1. {vehicle(vehicle_colour: “red”) {vehicle_id}
    a. {“data”: {“vehicle”:[{“vehicle_id”:”ABC12345”}, {“vehicle_id”:”ABC12348”}] }}

2. {slot(slot_number: 2) {vehicle_id
   vehicle_colour}
    a. {“data”: {“slot”:{“vehicle_id”:”ABC12345”, “vehicle_colour”: red} }}
    b. Note that the slot number 2 might not have a vehicle depending on the inputs because 2 is far away, and then it should return empty.
3. {vehicle(vehicle_id: “ABC12345”) {slot_number}
    a. {“data”: {“ vehicle”:{“slot_number”:”2”, } }}
    b. Should return whatever the slot number this vehicle is in
4. {vehicle(vehicle_colour: “red”) {slot_number}
    a. {“data”: {“vehicle”:[{“ slot_number”:”2”}, slot_number”:”5”}] }}
    b. This assumes that there are 2 red cars. So we have 2 outputs. The cars are parked at #2 and number #5 respectively.

Assessment Guide

What we are looking for

- A clear thought process on how you approach the problem that bring value to Mr Zorro as early as possible.
- A testable solution, with unit tests. How to make the code easily understood.
- A clean and simple solution to tackle the problem.

What we are not looking for

- System that is using ML
- Solving issues that are not listed above.
- The system with the most functionalities.

What we will ask / want to see

- How you explain and run through the design of your code.
- How you test each piece of code you write.
- Readability of code over being “clever” or “fast”