Backend Technical Challenge

Parking Lot

Summary

Confronted with the ongoing challenge of locating parking spaces within our parking structures in congested city streets, we embarked on a mission to rethink the parking experience using object-oriented design principles. Create the necessary APIs to satisfy the requirements below (no frontend work required).

Feel free to use your imagination when making product decisions and determining functionality, while keeping outcomes simple. We won't be running the code (or deploying it to prod), so it doesn't need to be perfect.

Implementation

Functional Requirements

1. **Real-time Capacity Monitoring**: Users should have access to accurate information about the number of available parking spots and the total capacity of the parking lot.

2. **Spot Type Status**: The system should track when certain types of spots, such as motorcycle spots or van spots are taken up.

Non-Functional Requirements

- **Multi-vehicle Accommodation**: The parking lot accommodates motorcycles, cars, and vans
  - Motorcycles can park in any available spot.
  - Cars can take any regular spot within the parking lot.
  - Vans are also permitted to park, but they need a space equivalent to three regular spots.

- **Vehicle Information Availability**: Vehicle information, such as the type of vehicle, will be provided to the system via an AI-integrated camera system, which sends API requests with the necessary information.

API Endpoints

Below are the proposed endpoints we'll need (feel free to modify as needed). You can assume that all requests originate from a trusted source.

1. *POST /api/parking-spot/$id/park*
   a. This endpoint will be called every time a vehicle parks, including the information about the vehicle type.
2. *POST /api/parking-spot/$id/unpark*
   a. This endpoint will be called every time a parking spot becomes free.
3. *GET /api/parking-lot*
   a. This endpoint will be called to visualize the current parking lot availability.

Instructions

1. We ask that you **allocate an hour** to complete as much as possible. Don't worry if you do not finish everything, or feel free to be creative and add functionality.

2. Clone the specified technical repository and follow the steps in the README to set up your local environment. While we've included what we consider essential packages for the task (Laravel, PostgreSQL and Redis) you're free to modify as needed, but it's not required.

3. Upload your solution to a private GitHub repository under your personal account. Please add jcart, rorydcampbell and fabriciobrito as collaborators.

4. Reach out to your talent representative indicating that you have completed your challenge, providing the repository URL.

Assessment

1. Code should be clean, well structured, and testable.
2. Be prepared to discuss your solution in a follow up code review with one of our peers.
3. Tests are nice, but optional. We'll discuss testing strategy, so keep this in mind.