Laravel + PHP Example Case

Introduction

Dear Candidate,

At Beyn Technology, we are pleased to be sending you this document since it means you have made it to the next step of our interview process. We congratulate you on this and wish you every success with this small test case.

This test case will allow us to determine your abilities, and how much you know about Laravel and building API services with it.

General Information

1. You need to serve your code over a git service.
2. We'll run your code with "php artisan migrate && php artisan serve" so, get ready for it.

Case Story

For a corporate business engaged in car maintenance and cleaning, you will be expected to write the necessary API services for a mobile application that tracks your services with customers. Customers will be able to log in, order, and track services that they bought over the application.

Here are some requirements while you're building this application. We separated requirements with two priorities: MUST, SHOULD.

MUST

1. You must use Laravel's features and structure as much as possible.
2. Use MySQL as a database service.
3. The application must be developed using the latest version of Laravel
4. A login service must be written that allows customers to log in. Like JWT.
5. A service must be written that allows customers to add balance to their accounts.
6. A service must be written that lists the services that the customer can receive.
7. A service must be written that allows customers to place orders.
8. A service must be written that lists the current and past orders of the customer with product information and filtration options.

9. All database insert, update and delete operations must be retractable when something went wrong.
10. The customer must choose the model of the car while placing the order.
11. You must validate all requests that came over API.
12. Car models must be synced to the database frequently from https://static.novassets.com/automobile.json
13. All responses must have user balance like information as metadata.

SHOULD

1. API endpoints should have a version prefix.
2. Create a caching strategy and an engine to improve performance. Like Redis.
3. Create a naming strategy for API endpoints.
4. You should implement PSRs.