Laravel Developer Technical Test

Last modification date: June, 2020

Authors

Name	Position
Jessica Plant	WFO Lead Developer

Introduction

Thank you for taking the time to sit this technical examination. To assess your skills, we have devised an open-ended challenge that will look at your ability to plan a product and implement a prototype. There are no wrong answers, but you will be expected to discuss your work and choices should you progress to the next phase.

You are expected to spend no longer than **two hours** on this and submit whatever you come up with in that time - remember **we're not looking for a fully working system**, just an example of how you'd tackle this structurally within the below technologies:

- *Laravel*
- *Lumen*
- *PostgreSQL*
- *Docker and docker-compose*
- *RabbitMQ*
- *NodeJS for small tasks (socket servers/AMQP consumers)*

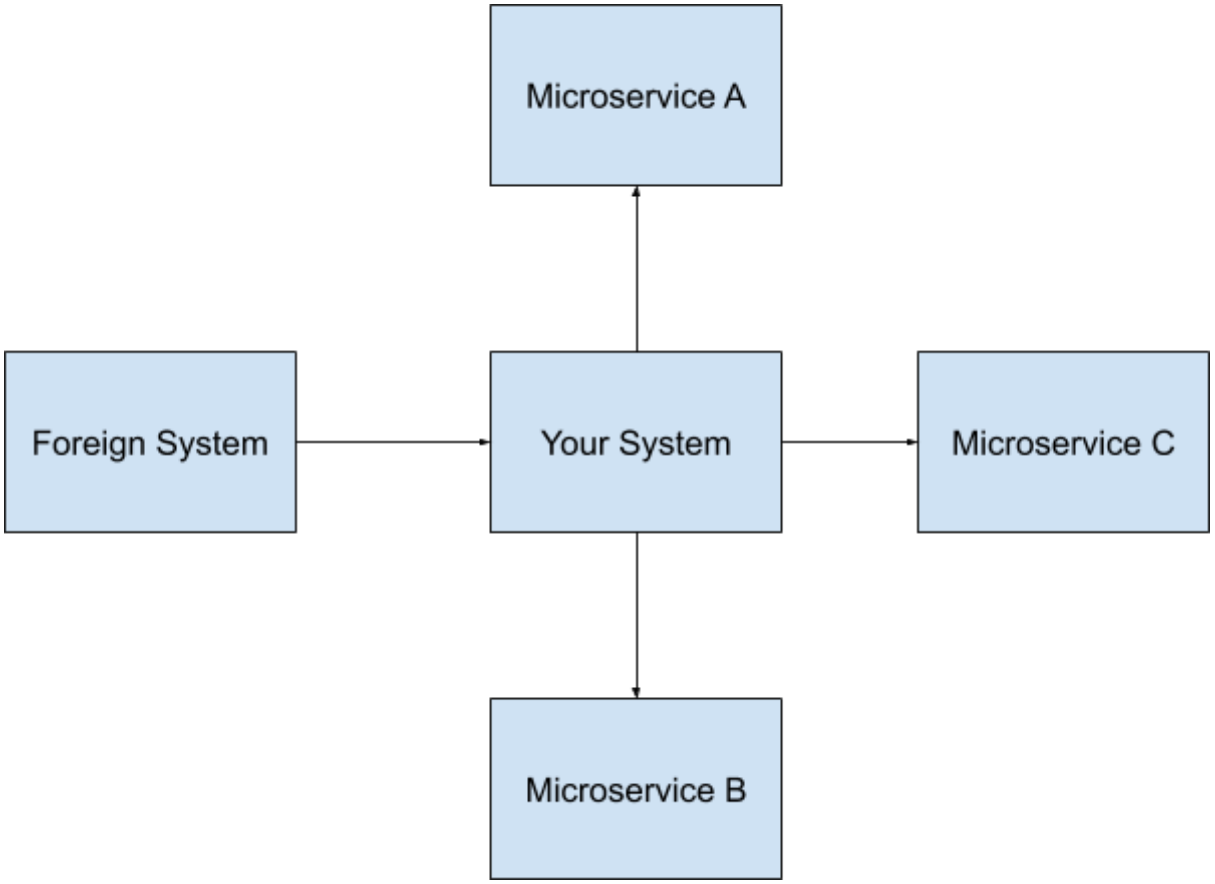
Preferred method of submission would be a git repository with a Markdown README explaining your decisions. It is up to you if you wish to use Eloquent migrations to build your database or a DB dump, but you should know we use a tool which relies on raw Postgresql so we will be looking for ability there too.

The ideal submission would consist of some quick (and legible) sketches and diagrams to show the overall solution to the whole problem, and a reasonable amount of code for us to get a feel for how you write and structure things.

Overview

Your task is to produce a basic implementation of a system that will sit between others, routing requests according to dynamic rules and permissions.

Below is a diagram of the hierarchy to work with in mind:



The foreign system will send any number of JSON payloads structured as below to an endpoint in your system of your choosing. Your system should then analyse the payload against the rule set provided and route accordingly to its three connected microservices.

Payloads

The minimum payloads the foreign system will send are below:

```
{
      "name": "Michael Collier",
      "phone": "07707000000",
      "email": "bigmike@collier.com",
      "query_type": {
            "id": 1234,
            "title": "SALE MADE"
      },
      "call_stats": {
            "id": 5678,
            "length": "01:56:34",
            "recording_url": "http://remote.system/recording/5678"
      },
      "campaign": {
            "id": 1011,
            "name": "Campaign A",
            "description": "A lovely campaign for Michael"
      }
}

{
      "name": "Jimmy Joe",
      "phone": "07707000001",
      "email": "bigjim@collier.com",
      "query_type": {
            "id": 5678,
            "title": "DECLINED OFFER"
      },
      "call_stats": {
            "id": 1213,
            "length": "00:56:43",
            "recording_url": "http://remote.system/recording/1213"
      },
      "campaign": {
            "id": 1516,
            "name": "Campaign B",
            "description": "A different campaign not for Michael"
      }
}
```

Feel free to mock other payloads to test your system but this structure should be adhered to.

Rules

The rules to follow are below:

Microservice A	Must receive ALL payloads
Microservice A	Must not receive payloads about Campaign B for security reasons
Microservice C	Must receive ALL payloads
Microservice B	Must receive payloads about sales only

Again, **we don't expect a full working example out the back of this test**, but we'd like to see a reasonable idea for how this would be achieved with code and planning to demonstrate your thought pattern.

Remember the rules are provided for this test but we are looking for them to be **user editable in nature** in what you return - imagine we are developing a UI where an end user could adjust the contents of the table above if they so desired. Your system should ideally facilitate that behaviour.

Good luck!