# Gousto API Test

25 September 2016

Sezer Tunca

## Table of Contents

## 1. Laravel

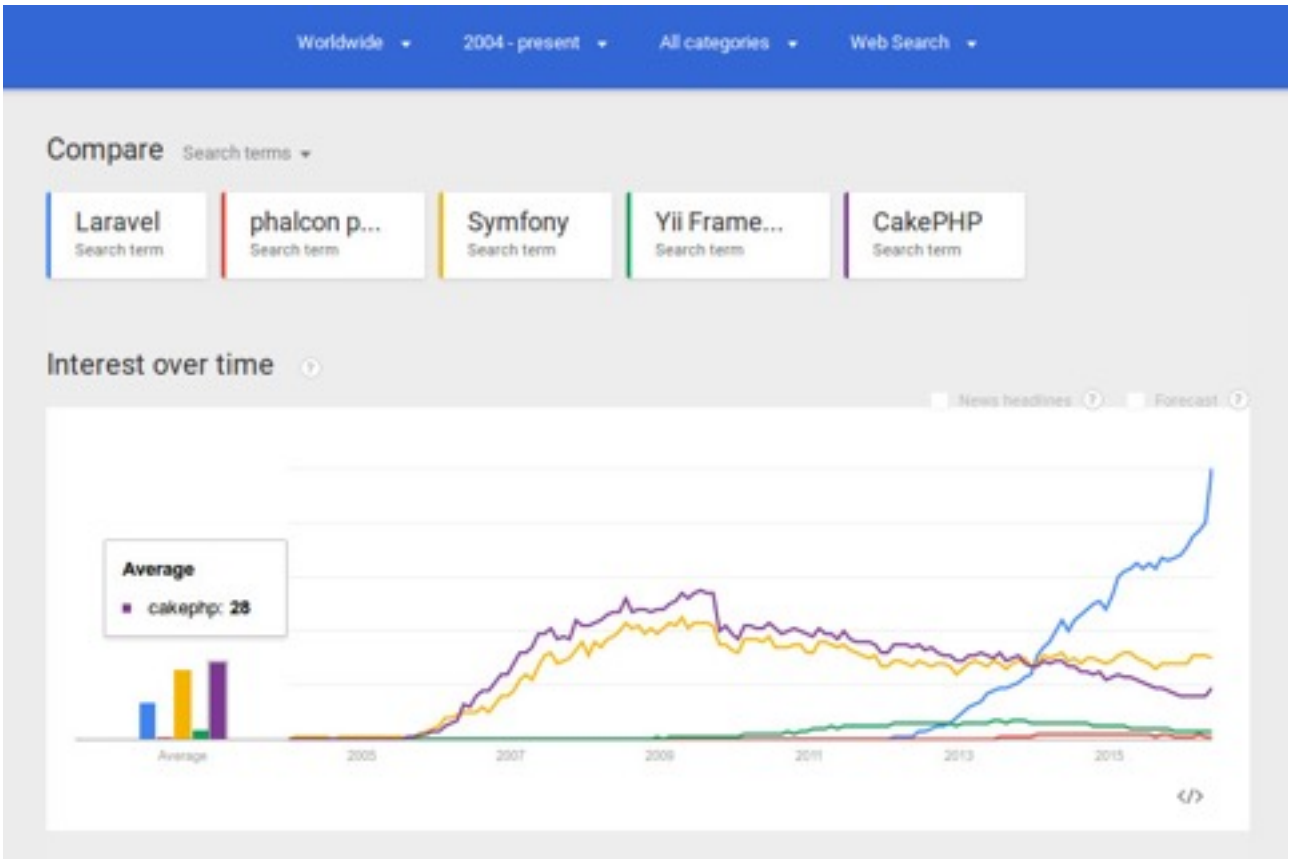
### Reasons for your choice of web application framework

I completed this project using the PHP framework Laravel with its latest release version 5.3. I chose Laravel because, firstly, I spent the last 18 months building web applications and APIs using it and really enjoy using it as a web developer. Secondly, it's the most popular PHP framework and it comes with great features that saves so much time. Features such as,

- Restful routing.
- MVC support.
- Easy authentication.
- Inherent Database version control.
- Composer packages.
- Unit testing.
- Thousands of great resources and online video tutorials.

Below images shows the popularity of Laravel over the other frameworks .

Google Trends comparison




Sitepoint's popularity of Laravel at work.


## 2. Apps and Font-End API Support

**Explain how your solution would cater for different API consumers that require different recipe data e.g. a mobile app and the front-end of a website.**

The project currently can fully support iOS and Android apps as well as any front-end of a website because it returns the data in JSON format. Almost all mobile apps consume some sort of API to work and uses JSON format. The RecipeController fulfils the
the functional requirements and it can cater to different recipe data by adding more functions to the RecipeController and by implementing additional controllers.

## 3. Evaluation

**Anything else you think is relevant to your solution**

This project is fully expandable and scaleable to become the entire back-end of Gousto system. A secure authentication system such as OAuth2 can be implement within minutes with Laravel.

To make the project expandable, I used a version system in the routing and controllers so if we wanted to update the project in the future, we can implement Version2 of the API routes and continue to support the systems that didn't get the update. I implemented another system (Transformers) to return data with custom field names so we don't show our entire database structures to the consumers and if in the future we wanted to change the fields names in our database, we can do it without breaking consumer's system.

Not being able use a database made this project a little complicated because, some of the big advantages of using Laravel from a developer and business point of view, that it comes with so many features that the developers don't have to spend time to implement same features in each project. For instance, pagination, this is a features that every project with growing data needs. Laravel assumes you use a database otherwise, it requires you to implement the most of the features manually.

I was not able to write tests on store and update functions as we write tests on real data. One way of doing this is using DatabaseTransactions in Laravel. It automatically rollbacks the changes after running the tests.

This project currently is not 'production ready' because store and update methods are not protected for authentication. Authentication comes built in with Laravel but it requires a database.