GameGab Project Documentation

By Braydon Kains

GameGab is a forum designed for gamers to talk about their favourite games. It is a responsive web application built using the Laravel framework along with Vue.js and MariaDB.

Models

Thread – A thread started by user to contain discussion of a certain topic.

Post – A post that is added to a given thread.

Tag – A tag that can be attached to a thread for filtering. (not implemented)

Views

Login – Existing user can login

Register – Create an account

Password Reset – Reset an existing password

User dashboard – Carry out actions specific to the user

Threads View – View all existing threads, search for specific threads, if a user is logged in they can create a new one

Single Thread View – View all posts attached to a thread, if a user is logged in they can write new posts to the thread

Controllers

ThreadController – Create, read, update, delete, and view composing actions for threads

PostController - Create, read, update, delete, and view composing actions for posts

TagController - Create, read, update, delete, and view composing actions for tags

HomeController – Action for composing home view

Authentication controllers – a set of controllers for login, logout, registration and all related actions

How To Use

After creating a user you can do the following things:

- Create a thread

- Search for a thread

- View a thread and its posts

- Add a post

- Add a gif to your post

- Log out

The app is fully responsive and does not require refreshes for actions to take place. The views are dynamic and load components conditionally based on whether a user is logged in or not.

Questions:

What technologies have you used?

The following is the technology stack for GameGab:

Laravel – Modern PHP Web Framework

PHP Artisan – Command line tool to generate various scaffolding for Laravel applications, also to host development server

Vue.js – Modern Javascript Front-end Framework

Node.js – Package manager NPM and Node runtime used to facilitate Vue.js

MariaDB – Open source implementation of MySQL for relational databases

Apache – Open source web server for application deployment

Digital Ocean – Elastic Cloud Compute service used for web hosting

Visual Studio Code, Vim – Editors used for writing website code

Justify the usage of every technology/framework/API, specially if you are using technology/framework/API not covered in the course.

Few of the technologies learned in the class end up being used in our final project. While what we learned was a good foundation for basic knowledge of web development, I felt that exploring more modern web development practices was the best way to expand the basis we received from lectures.

Laravel is a modern PHP framework that cut down gigantically on development time. It is designed to abstract away the mundane parts of developing a web application; setting up servers, databases, routes, styling, secure communication and more are now done automatically. This completely frees the developer up to make sure their brains are on how their app works and not the boiler-plate setup that usually accompanies this work.

This was beneficial for me; my time was spent on the vision for the application's architecture and not on how it was set up. Also, I have yet to come across anything I have been unable to accomplish using the foundation and scaffolding set up by the framework.

The frameworks Object Relational Modelling is also superb. Models are designed just as simple PHP classes, but are extremely powerful. They are intrinsically linked to tables in the database, and the base model classes do loads of work for you. As a result, no actual SQL queries needed to be written for the application; everything was almost trivially accomplished with the powerful Eloquent ORM engine provided by Laravel.

HTML templating is also a breeze with Laravel's Blade Templating Engine. It allowed our HTML views to be completely compartmentalized, single responsibility, and modular. It abstracts away so many manual issues with vanilla HTML and CSS programming.

PHP Artisan is the final major benefit. PHP Artisan is a utility that Laravel provides for all actions with the app. You can use it to generate scaffolding for models, views, controllers, resource collections, default snippets, you can use it to run database migrations, seeds, and maintenance, and you can use it to serve your application in different ways.

While I could go into much more detail, it should be clear why I feel Laravel is a marvelous framework and I'm very happy to have used it.

Vue.js is the chosen front end framework. While this decision was made mostly because of how well it works with Laravel out-of-the-box, however it ended up being an excellent choice in its own right. Vue.js allows for a component-based architecture that is modular, sandboxed, and easy to get the hang of. I was able to define powerful-yet-simple components that accomplished a lot for very little code. When these components are designed, you can drop them right into any of the blade views as a custom HTML tag, and Vue.js in conjunction with Blade takes care of the rest.

I could not have imagined doing this project without view. Everything clicked together so nicely with the backend, it ended up being integral the app's architecture. I started designing the backend around how things would work with Vue.js, and everything I wanted to do was not only possible, but easy.

MariaDB is an open-source drop-in engine replacement for MySQL. Truth be told, I didn't really care about the differences between MariaDB and MySQL's engines. Laravel just works so well with MySQL that I knew I needed it, and the MySQL packages on Red Hat based distributions are frustratingly weird and different. MariaDB has a simple setup process and a much more closely maintained package on my distribution (Fedora).

Apache is an open source web server. Laravel had scaffolding for it by default, and I decided not to change it.

Digital Ocean will be used for hosting. Users are able to spin up a small Linux virtual machine, and it will be very easy to host temporarily for project marking using this as the app was developed in Linux.

Visual Studio Code and Vim were used to write and edit code. Visual Studio Code is a powerful editor with great plug-in tooling for Laravel and Vue.js. Vim is a terminal based text editor that was primarily used for small temporary changes.

Mention if you were familiar with the technology / framework. How did you come up with the specific technology chosen?

I was familiar with all the languages used by the chosen frameworks, however had never worked with any of the frameworks before. I decided to give Laravel a try just because I had been told that it makes PHP a nice language to use, and that it comes with a lot of useful utilities. This ended up being very true, and I am pleased with the choice of Laravel, as well as with the choices that stemmed from the original choice of Laravel.

If you want to extend your project by adding a feature, what would that be?

The app was built with the number one priority of being easily extensible. While I did succeed in that goal eventually, I ran out of time in the semester to actually implement all the features I wanted.
A few features were cut due to other features taking more time than planned:

- Tag filtering for threads
- Upvoting and downvoting posts
- User profile images
- A front end redesign to look less bland
- Debug the GIF functionality
- Much much more

Given as little as a few more days I probably would have been able to add these features easily, but unfortunately there was just not enough time.

Name and give the URL of one public application that resembles your project, if there is any? List the similarities and differences

GameGab very closely resembles GameFAQs, however it is significantly smaller in scale because GameGab is simply a school project, whereas GameFAQs has existed for many years. GameGab is also responsive, while GameFAQs is not.
https://gamefaqs.gamespot.com/

Which part of the project was the most time-consuming?

Figuring out Vue.js. Once things started to click, it ended up being pretty fast. For a long time though, Vue.js was primarily black magic that I just had to trust would work. Needless to say this lead to a front end that was broken a majority of the time the project was in development.

I also spent a lot of extra time making the project extensible, and less on the project's aesthetics. This meant the project is not particularly appealing to look at, and ended up lacking functionality in the end just because I had run out of time.