Waifu Pics Developer Process

# Setting up

So, to start with I decided to find an API I found quite interesting. I decided I wanted to implement waifu.pics. As I am a big fan of anime and I feel although my employer will get a sense of the kind of person I am from my choice of API.

API link: <https://waifu.pics/docs>

I then started to consider what I would need to consume this piece of content. This could be done as simply as a file\_get\_contents but I thought that would come across as massively unprofessional and is not the optimal implementation.

I decided to challenge myself and do it in Laravel as that is the technology I will be using in this position.

I have never set up a Laravel project from scratch before really. So, I headed over to the installation guide for Laravel. It said there was an easy way to get started by using composer. I’ve experienced composer before it’s a useful tool, so I downloaded it.

One quick:

composer create-project laravel/laravel waifu-pics-app

And we are off to the races!

Having recently built a new computer I have absolutely no development tools installed. So I promptly thought about my tool of choice and decided to just go with VSCode for the time being.

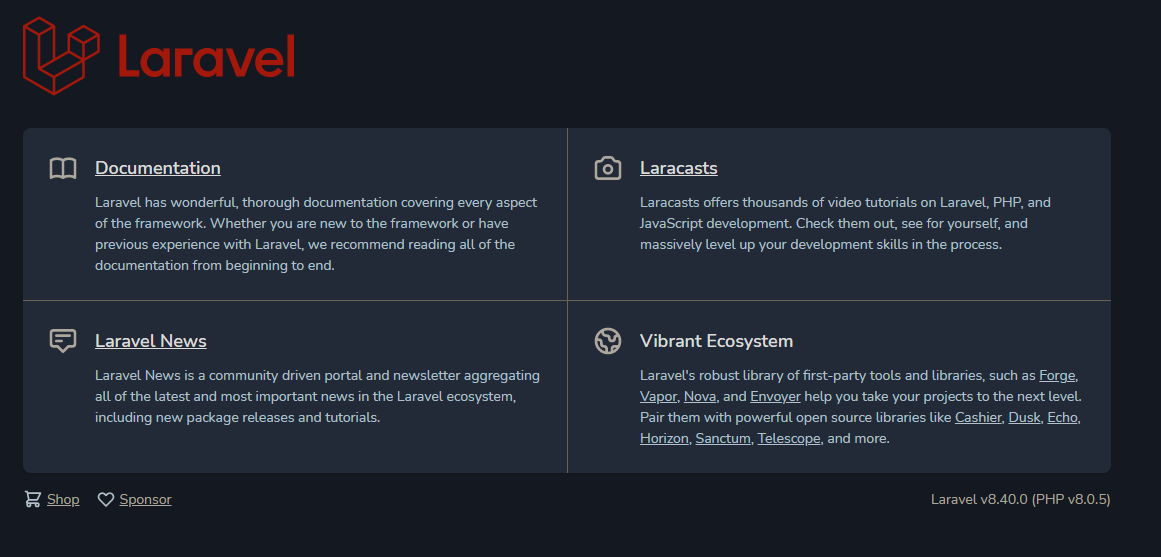
I downloaded a bunch of extensions (which I will not list) and started to piece together how things work again.

I ran:

Php artisan serve

To get my server up and running and went to <http://127.0.0.1:8000/>

# UI and Routes



Wonderful! But we obviously don’t need or want this so let’s go to the view!

I found the welcome.blade.php file. Fairly standard stuff, quick refresher to myself.

Then created a new waifuPics.blade.php file with the absolute bare minimum in it consisting of a h1 tag saying waifu pics.

I then went to the routes/web file.

Where I changed

return view(welcome);

to

return view('waifuPics');

Smashing stuff.

Refresh the page and I have the waifuPics blade view I originally implemented. It looks pretty bad let’s install bootstrap.

I could have linked to an online version but that’s inefficient and I wanted to learn how to import bootstrap into my project.

Apparently, there is an easy way to do this according to this outdated link I found on google. It says it works for Laravel 7 let’s hope it works for 8!

<https://www.techiediaries.com/laravel/how-to-install-bootstrap-in-laravel-6-7-by-example/>

I then found out that I didn’t have node.js installed when we came to run npm install.

So I quickly went and installed node.js, added the environment variable restarted CMD and ran npm install!

Wonderful. Next step was a quick npm run prod after this I noticed that there was still no bootstrap on the page. I wasn’t sure the best way to link to the bootstrap, but I ended up simply putting a link to it

<link href="{{ asset('css/app.css') }}" rel="stylesheet">

I made two select input items (filters) and

I then disappeared to sing happy birthday to my sister.

# I Need an API

Obviously in order to get the data from the API I need to actually ping the API. So I created a waifuController. Made a route that links to it. A form that links to the route and var\_dumped the results.

From there I looked at the formatting and structure of the data that is returned. Done some checks to ensure that the response was successful. Accounted for some edge cases. Passed back the parameters that were passed in and routed back to the view.

# Final Tweaks

Naturally, I needed to pretty up the front end at least a little bit so I delved into some bootstrap documentation.

This certainly is not my strong suite and my designing leaves something to be desired. But I centred a few things. Restricted the size of the Images that come back to a concise but reasonable size. I then tested it with a few images. Everything looks fine.

I added in some checks for the errors, Made it a bootstrap error message / alert. I tweaked some minor logic bits here and there. Added a placeholder image I found online into the code to show where the image will be and to fill in some space.

# Improvements and Further Development

## API

There are multiple endpoints that can be used here. I could use the getMany API endpoint to display multiple images. This would obviously require UI tweaks (carousel implementation perhaps?) , most of the API logic would we reusable in some format so that would need to be tweaked, split out and optimised. It would be a useful thing to implement but given the time restraints and my present situation wasn’t implemented.

## UI

The UI leaves much to be desired realistically you would have something a lot more formatted, a lot prettier a lot more dynamic. But It is certainly not my strongest area, and It feels like It would mostly be filler.

## Eager Loading

With the implementation of some JavaScript, I could have a listener that would wait until the page had loaded then instantly try and receive an image from the API and display it prior to the user hitting the button. This would probably take a chunk of time to implement but It would be a nice little feature to have.

## Database / Front-end Logic Tweaks

Depending on how easy you want it to be to add additional categories / whether you think they will change. You may want to have a database backend with a Model. From there you could simply pluck the ids and names of the categories. From there you could just loop over the values to display them all in the menu’s this would be done when loading and it would be dynamic, easily expandable, and just generally better.

Unfortunately, this would require some documentation reading, a little bit of database tweaking and a little more time than I have presently.