PHP technical challenge documentation

Introduction

This document will serve as a guide to explain how to get the Laravel application up and running on your local environment and how to use it.

Requirements.

Docker
MacOS/Windows

How to install and run

The app uses Laravel Sail https://laravel.com/docs/8.x/sail#introduction which uses docker to generate all necessary things to run like PHP 8, Laravel 8, MySQL and Node.

Laravel documentation recommends that you add the following line into you bash or zshrc file

alias sail="bash vendor/bin/sail"

You need to run this command in order to install a docker container that will provide the required PHP version in order to install sail in the vendor folder

```
$ docker run --rm \
   -u "$(id -u):$(id -g)" \
   -v $(pwd):/opt \
   -w /opt \
   laravelsail/php80-composer:latest \
   composer install --ignore-platform-reqs
```

In case this is not added you can run the following command to start the sail docker environment and install the app within docker.

```
$./vendor/bin/sail up -d
```

It may take a while to download all needed images and containers but eventually you will have the prompt back

Once this installation has completed run the following command

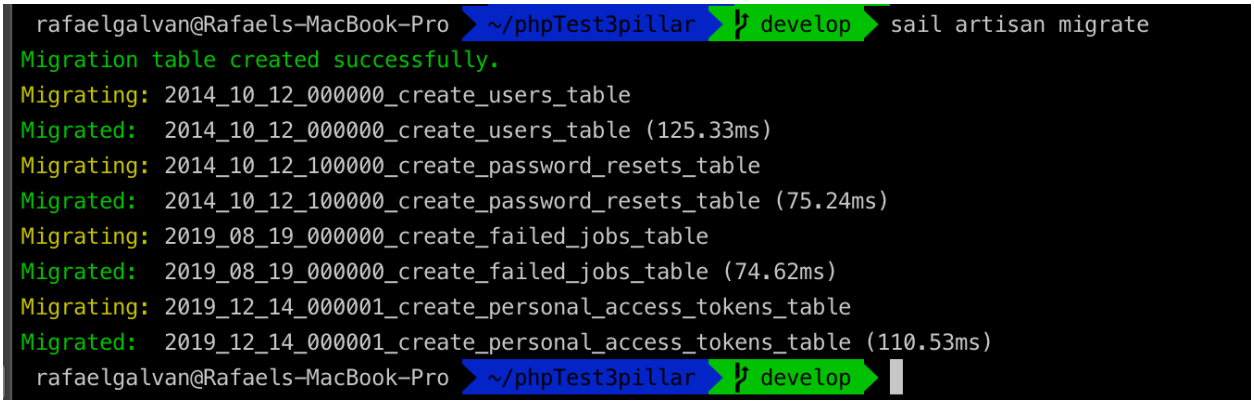
$ sail artisan migrate

Or

$ ./vendor/bin/sail artisan migrate *in case you did not add alias*

This will create the users table and all needed tables for authentication. The app uses Laravel Sanctum to provide JWT that are used eventually to authenticate other endpoints. Sanctum was selected to provide a quick authentication solution and token to use the API.

The result should be something like this:

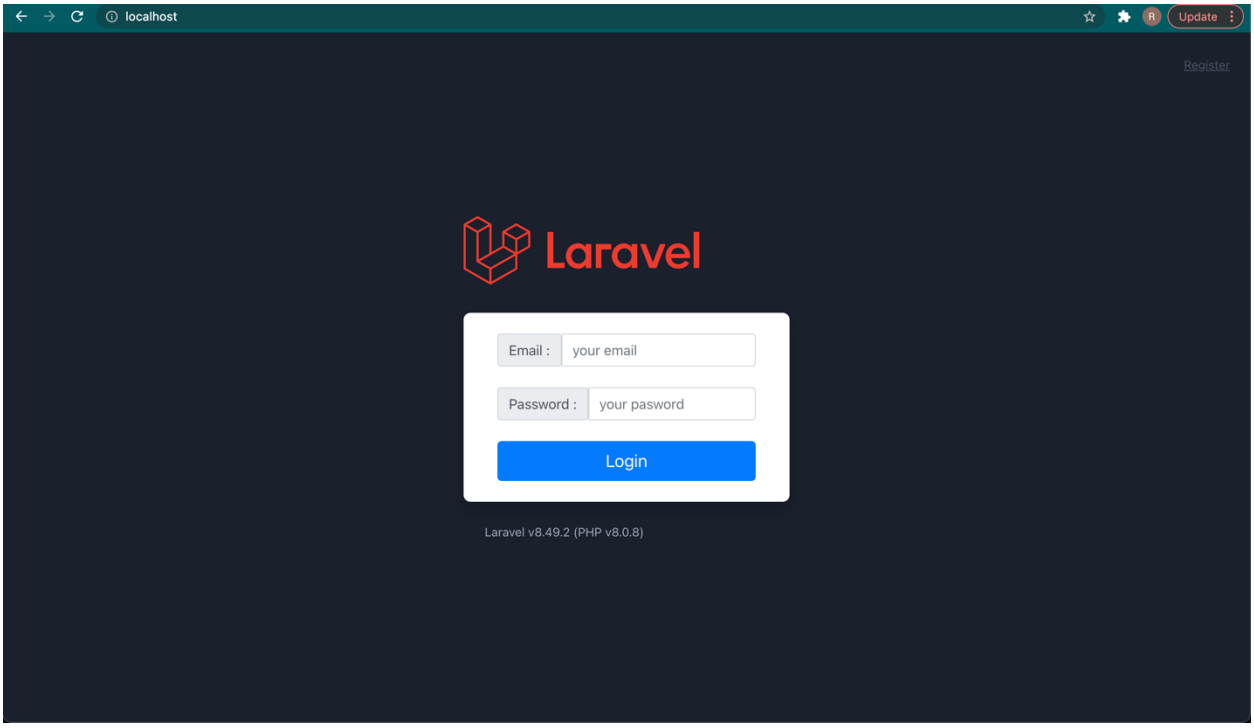


Then in order to install react and its dependencies you need to run:

$sail npm install

After its completion, on your browser go to the following URL

http://localhost/ it should take you to a page like this:

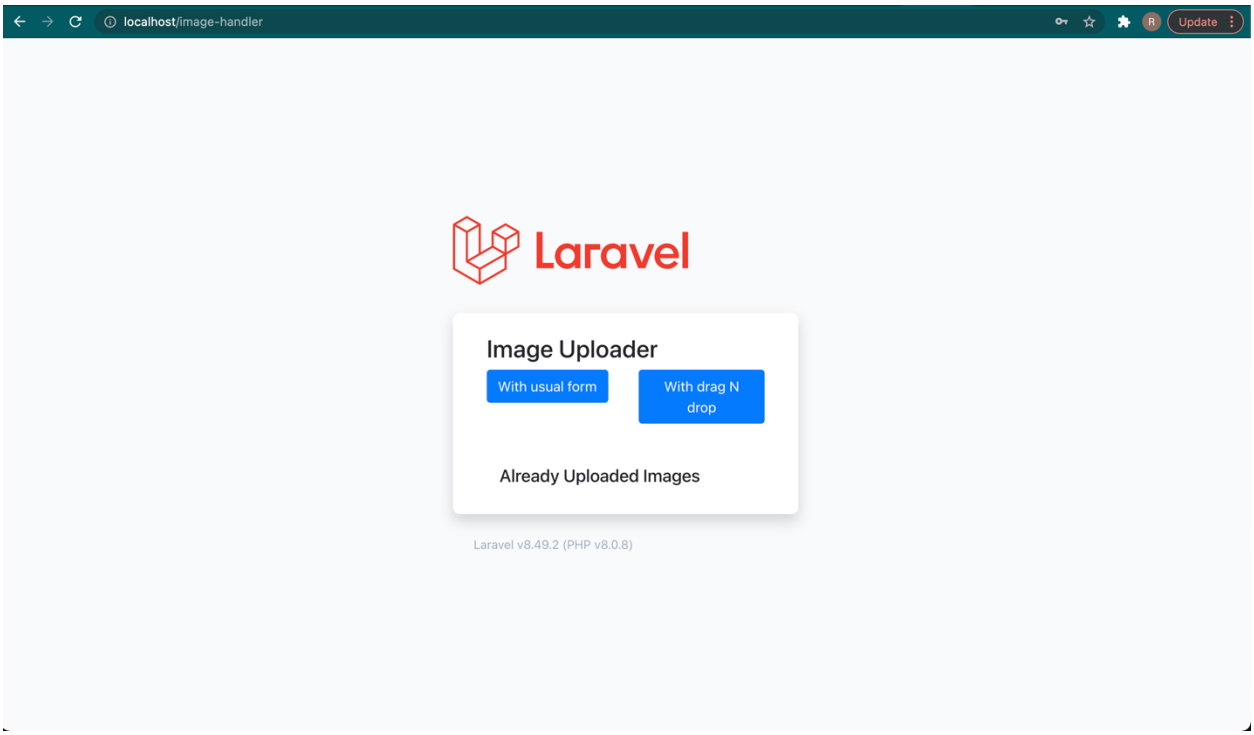


The page uses a blade view that eventually renders react components. Laravel was configured to use react instead of Vue. The component is called ../resources/js/components/Login.js

In the top right corner, you have a link to register a user. This registration form is based on Laravel Breeze. Click on it to create a user; Use your recently created credentials to login and get your token.

The JWT is saved into local storage as well as the uploaded images.

When you have logged in you will be redirected to the following page.



This is another blade view, but it uses react components to do all the SPA, this component is called ../resources/js/components/ImageManager.js

You have two options to upload images.

1) Usual Form. This form uses the classic input type file that will allow you to select multiple png images

2) Using the Drag And Drop third party library called react-dropzone.

When clicking on the first button you should see a form like this:

Image Uploader

With usual form

With drag N drop

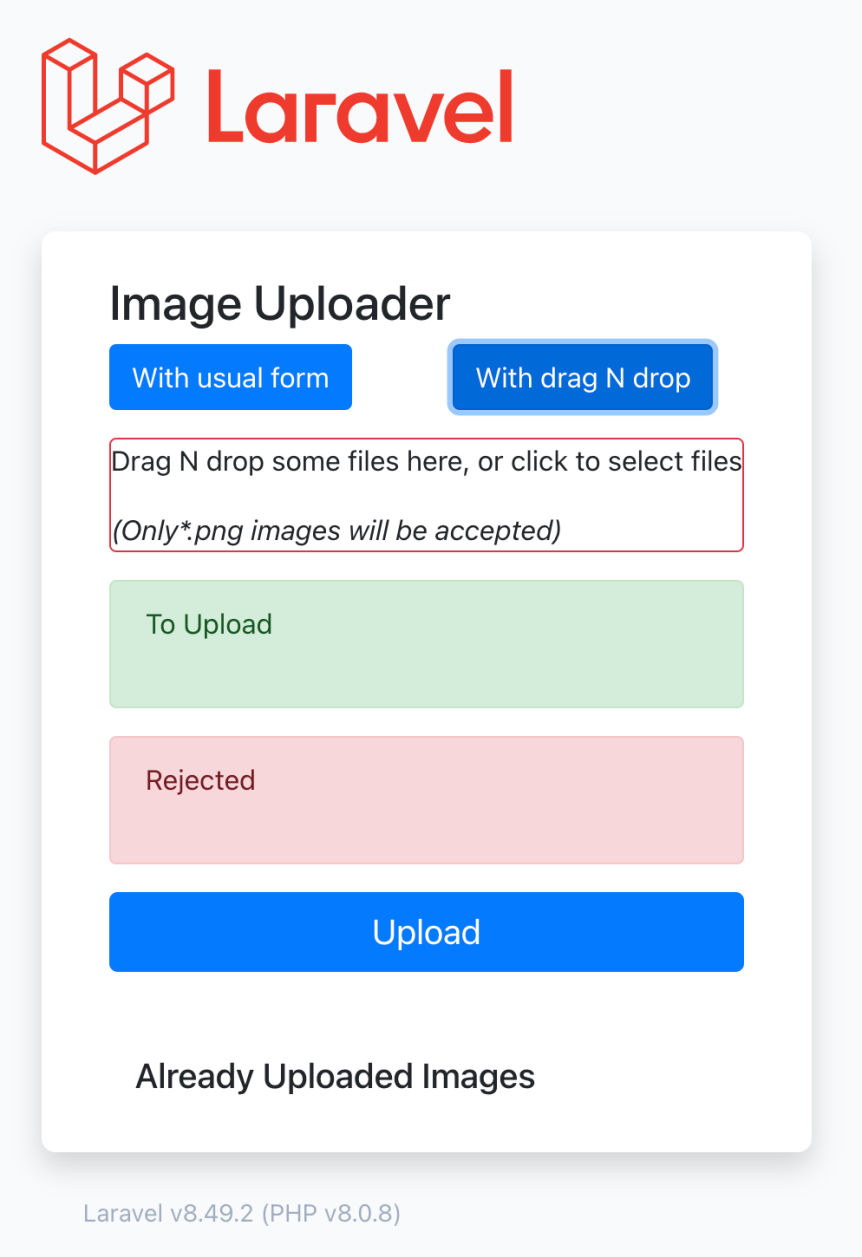
Select png image

Choose Files No file chosen

Upload

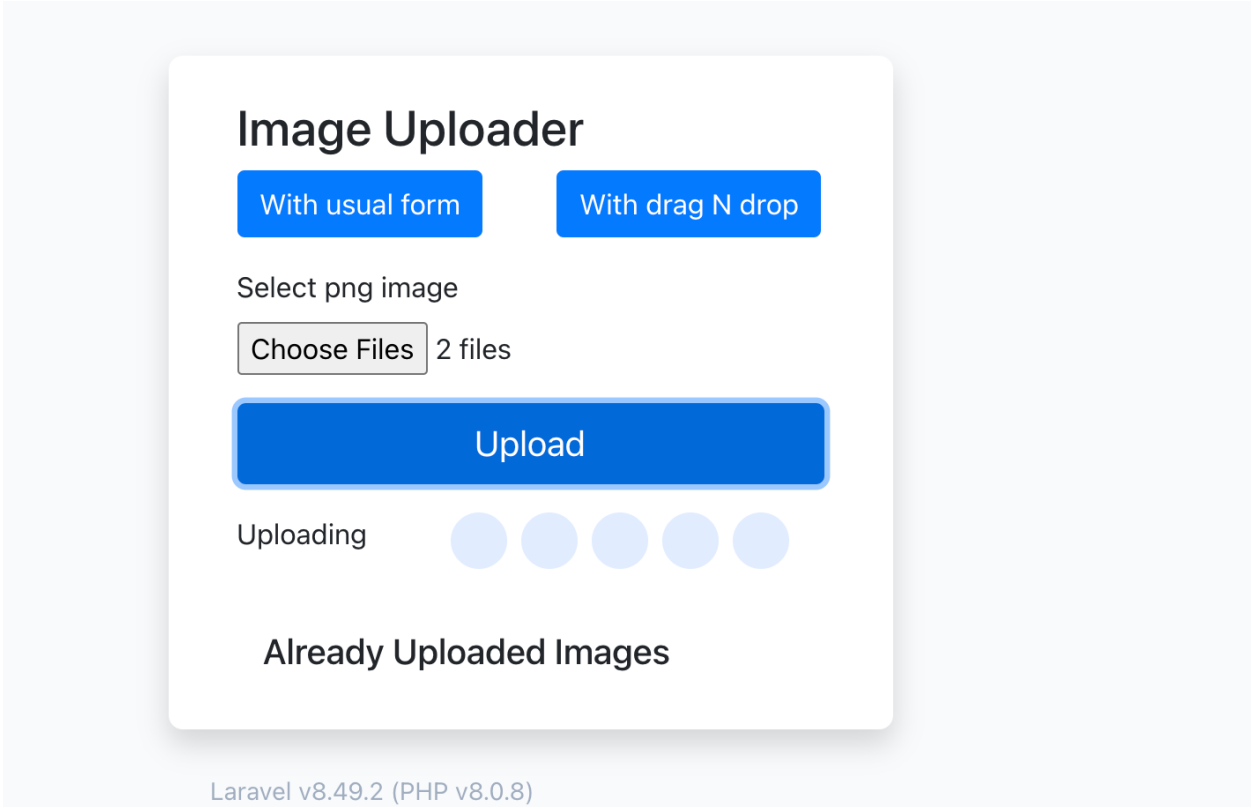
Already Uploaded Images

When you pushed the other button, you will the following form

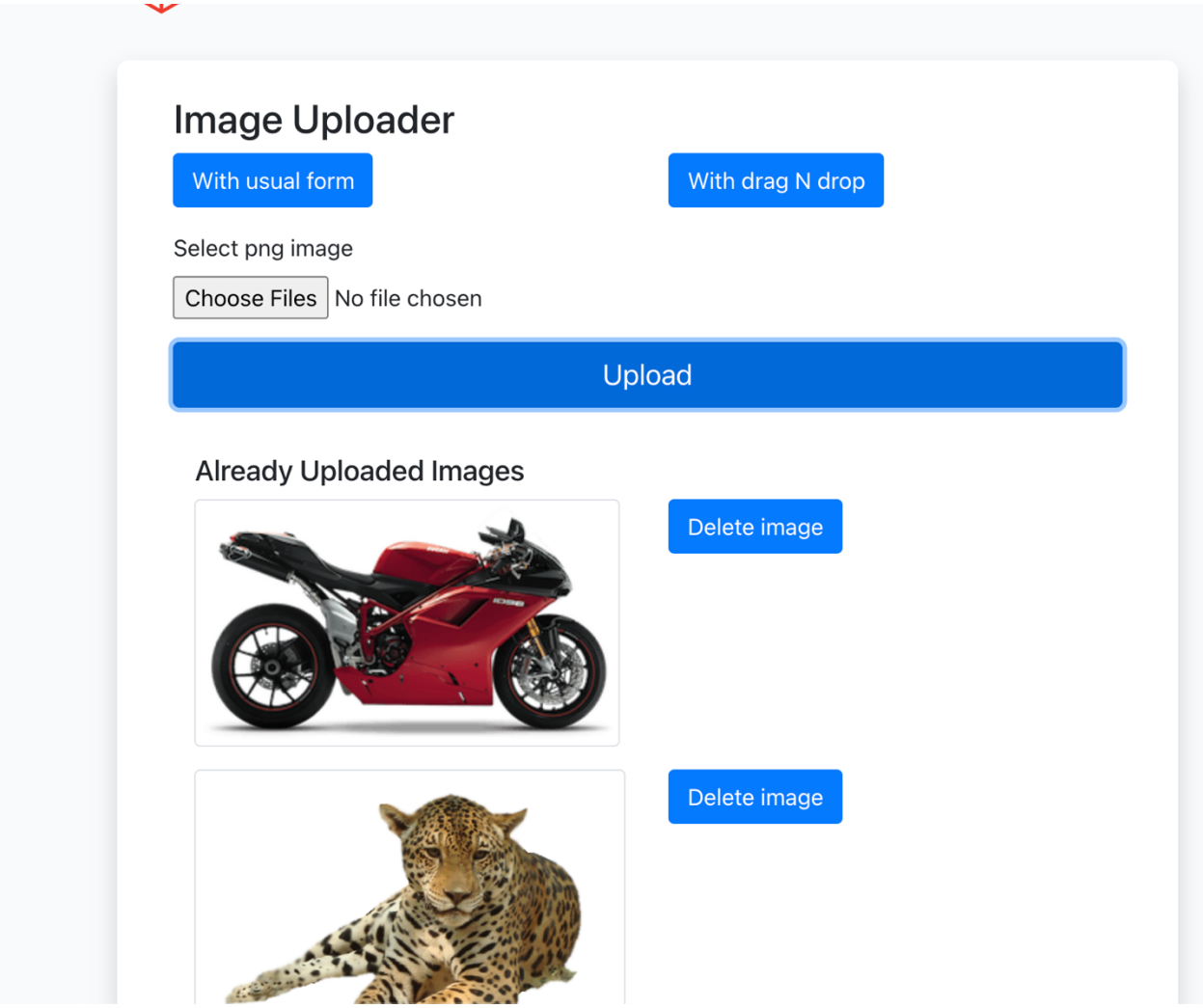


In either case once you have selected the images and hit upload you will see a spinner that indicates that the images are uploading. If all the images where successfully uploaded the app will let you know by showing a green text and display the images at the bottom. If there's something wrong the app will let you know, and no images will be displayed.

Like this:



Laravel v8.49.2 (PHP v8.0.8)



The delete button will allow you to delete the image you want. If you try to upload images that were already uploaded the process will exclude those to avoid uploading them.

Some technical aspects of the app

1) The App uses local storage to save the uploaded images and from there are served to display them on the app. I know it was requested to use session but due the short time I was not able to implement it. It was easier at the time to use local storage, which is not always consider a good practice.

2) In order to request API calls I use Axios, and I don't use it directly on the components I normally try to create classes to handle specific processes that eventually uses Axios in the background. These are the files that eventually use the Axios ../resources/classes/ImageManagerHandler.js and

../resources/classes/UserAuthHandler.js

3) The app uses Class and functional components, although I normally work with class components, I'm not that familiar with functional component and hooks, which is kind of the new thing in react. On the path of polishing my React and JS skills.

4) Certain Laravel routes renders blade views, on those views I add the div with a certain id to render in it the react components needed to make the app as SPA as requested.

5) The backend is using PHP 8.0 but I'm familiar working with other versions like 5.4 and 7.4. I tried to follow the rule of having slim controllers as much as possible. Although sometimes this creates several function/object jumps and several file creations.

6) I use the middleware feature to validate the parameters from the requests. Those middlewares are in ../app/Http/Middleware/LoginParamsMiddlewareValidator.php and ../app/Http/Middleware/ImageMangerParamsMiddlewareValidator.php

7) I normally try to create independent route files that are in: ../routes/ImageManagerRoutes.php and routes/LoginRoutes.php. This requires a small config to Laravel.

8) I like to work with what I know as the repository pattern, I created one for the User model located in ../app/Repositories/UserRePository.php, although it is not used. I like to work with this pattern because it concentrates in a single place all DB related queries and avoids having those queries spread all over the controllers. I had the intention to create a model to save into the DB the url of the uploaded images, but due time I was not able to do it.

9) The controller that deals with the image uploading uses a class that is called ../.app/Services/RxFloDevImageUploader.php. This class uses the Guzzle Http Client to perform the upload to the test.rxflodev.com server. It is implementing an Interface. The idea of this is to have the ability to have multiple servers to upload like Azure, AWS, etc. All these Uploaders needs to have those methods from the Interface to warrantee the correct functionality of the process.

10) For the different responses I normally try to use what I know as the decorator pattern. The idea of the decorator is to receive certain information and return an index array with all the needed values the response must have. This avoids returning different response formats. Eventually this array is converted to Json and return as response from the controller.