Frontend

Introduction

Laravel is a backend framework that provides all of the features you need to build modern web applications, such as routing, validation, caching, queues, file storage, and more. However, we believe it's important to offer developers a beautiful full-stack experience, including powerful approaches for building your application's frontend.

There are two primary ways to tackle frontend development when building an application with Laravel, and which approach you choose is determined by whether you would like to build your frontend by leveraging PHP or by using JavaScript frameworks such as Vue and React. We'll discuss both of these options below so that you can make an informed decision regarding the best approach to frontend development for your application.

Using PHP

PHP and Blade

In the past, most PHP applications rendered HTML to the browser using simple HTML templates interspersed with PHP echo statements which render data that was retrieved from a database during the request:

```
<div>
    <?php foreach ($users as $user): ?>
        Hello, <?php echo $user->name; ?> <br />
    <?php endforeach; ?>
</div>
```

In Laravel, this approach to rendering HTML can still be achieved using views and Blade. Blade is an extremely light-weight templating language that provides convenient, short syntax for displaying data, iterating over data, and more:

```
<div>
    @foreach ($users as $user)
        Hello, {{ $user->name }} <br />
    @endforeach
</div>
```

When building applications in this fashion, form submissions and other page interactions typically receive an entirely new HTML document from the server and the entire page is re-rendered by the browser. Even today, many applications may be perfectly suited to having their frontends constructed in this way using simple Blade templates.

Growing Expectations

However, as user expectations regarding web applications have matured, many developers have found the need to build more dynamic frontends with interactions that feel more polished. In light of this, some developers choose to begin building their application's frontend using JavaScript frameworks such as Vue and React.

Others, preferring to stick with the backend language they are comfortable with, have developed solutions that allow the construction of modern web application UIs while still primarily utilizing their backend language of choice. For example, in the Rails ecosystem, this has spurred the creation of libraries such as Turbo Hotwire, and Stimulus.

Within the Laravel ecosystem, the need to create modern, dynamic frontends by primarily using PHP has led to the creation of Laravel Livewire and Alpine.js.

Livewire

Laravel Livewire is a framework for building Laravel powered frontends that feel dynamic, modern, and alive just like frontends built with modern JavaScript frameworks like Vue and React.

When using Livewire, you will create Livewire "components" that render a discrete portion of your UI and expose methods and data that can be invoked and interacted with from your application's frontend. For example, a simple "Counter" component might look like the following:

```php
namespace App\Http\Livewire;

use Livewire\Component;

class Counter extends Component
{
    public $count = 0;

    public function increment()
    {
        $this->count++;
    }

    public function render()
    {
        return view('livewire.counter');
    }
}
```

And, the corresponding template for the counter would be written like so:

```html
<div>
    <button wire:click="increment">+</button>
    <h1>{{ $count }}</h1>
</div>
```

As you can see, Livewire enables you to write new HTML attributes such as `wire:click` that connect your Laravel application's frontend and backend. In addition, you can render your component's current state using simple Blade expressions.

For many, Livewire has revolutionized frontend development with Laravel, allowing them to stay within the comfort of Laravel while constructing modern, dynamic web applications. Typically, developers using Livewire will also utilize Alpine.js to "sprinkle" JavaScript onto their frontend only where it is needed, such as in order to render a dialog window.

If you're new to Laravel, we recommend getting familiar with the basic usage of views and Blade. Then, consult the official Laravel Livewire documentation to learn how to take your application to the next level with interactive Livewire components.

Starter Kits

If you would like to build your frontend using PHP and Livewire, you can leverage our Livewire starter kit to jump-start your application's development.

Using React or Vue

Although it's possible to build modern frontends using Laravel and Livewire, many developers still prefer to leverage the power of a JavaScript framework like React or Vue. This allows developers to take advantage of the rich ecosystem of JavaScript packages and tools available via NPM.

However, without additional tooling, pairing Laravel with React or Vue would leave us needing to solve a variety of complicated problems such as client-side routing, data hydration, and authentication. Client-side routing is often simplified by using opinionated React / Vue frameworks such as Next and Nuxt; however, data hydration and authentication remain complicated and cumbersome problems to solve when pairing a backend framework like Laravel with these frontend frameworks.

In addition, developers are left maintaining two separate code repositories, often needing to coordinate maintenance, releases, and deployments across both repositories. While these problems are not insurmountable, we don't believe it's a productive or enjoyable way to develop applications.

Inertia

Thankfully, Laravel offers the best of both worlds. Inertia bridges the gap between your Laravel application and your modern React or Vue frontend, allowing you to build full-fledged, modern frontends using React or Vue while leveraging Laravel routes and controllers for routing, data hydration, and authentication — all within a single code repository. With this approach, you can enjoy the full power of both Laravel and React / Vue without crippling the capabilities of either tool.

After installing Inertia into your Laravel application, you will write routes and controllers like normal. However, instead of returning a Blade template from your controller, you will return an Inertia page:

```php
<?php

namespace App\Http\Controllers;

use App\Models\User;
use Inertia\Inertia;
use Inertia\Response;

class UserController extends Controller
{
    /**
     * Show the profile for a given user.
     */
    public function show(string $id): Response
    {
        return Inertia::render('users/show', [
            'user' => User::findOrFail($id)
        ]);
    }
}
```

An Inertia page corresponds to a React or Vue component, typically stored within the `resources/js/pages` directory of your application. The data given to the page via the `Inertia::render` method will be used to hydrate the "props" of the page component:

```jsx
import Layout from '@/layouts/authenticated';
import { Head } from '@inertiajs/react';

export default function Show({ user }) {
    return (
        <Layout>
            <Head title="Welcome" />
            <h1>Welcome</h1>
            <p>Hello {user.name}, welcome to Inertia.</p>
        </Layout>
    )
}
```

As you can see, Inertia allows you to leverage the full power of React or Vue when building your frontend, while providing a light-weight bridge between your Laravel powered backend and your JavaScript powered frontend.

Server-Side Rendering

If you're concerned about diving into Inertia because your application requires server-side rendering, don't worry. Inertia offers server-side rendering support. And, when deploying your application via Laravel Cloud or Laravel Forge, it's a breeze to ensure that Inertia's server-side rendering process is always running.

Starter Kits

If you would like to build your frontend using Inertia and Vue / React, you can leverage our React or Vue application starter kits to jump-start your application's development. Both of these starter kits scaffold your application's backend and frontend authentication flow using Inertia, Vue / React, Tailwind, and Vite so that you can start building your next big idea.

Bundling Assets

Regardless of whether you choose to develop your frontend using Blade and Livewire or Vue / React and Inertia, you will likely need to bundle your application's CSS into production ready assets. Of course, if you choose to build your application's frontend with Vue or React, you will also need to bundle your components into browser ready JavaScript assets.

By default, Laravel utilizes Vite to bundle your assets. Vite provides lightning-fast build times and near instantaneous Hot Module Replacement (HMR) during local development. In all new Laravel applications, including those using our starter kits, you will find a `vite.config.js` file that loads our light-weight Laravel Vite plugin that makes Vite a joy to use with Laravel applications.

The fastest way to get started with Laravel and Vite is by beginning your application's development using our application starter kits, which jump-starts your application by providing frontend and backend authentication scaffolding.

> [!NOTE] For more detailed documentation on utilizing Vite with Laravel, please see our dedicated documentation on bundling and compiling your assets.