Introduction to Inertia.js

[Introduction]

Hello my name is Lee Crosdale,

A bit on my background:

At the start of my career I was a Microsoft Dynamics AX Developer, building SSRS reports and making various modifications to the AX codebase. I also built an Android app (which as far as I'm aware is still in use).

I then moved into Web Development for a company named ParcelBroker (Magpie Solutions), where I was the IT Manager / Developer on the project, I spent a lot of time refactoring a legacy PHP site into Laravel, as well as building some internal tools, both with Laravel and some other languages, Node, Python etc.

I'm currently the Lead Laravel and PHP Developer for Enovate (Point to boss if he is in the room), where I work on various Laravel and CraftCMS projects.

You can find me on twitter **@leecrosdale**.
I also stream on twitch: **twitch.tv/crosdale**

[Who are you]

- Go around the room and do introductions

[Talk overview]

What is inertia
Who is inertia for
Setup
Code examples
Questions
End

[What is Inertia]

Inertia allows you to create fully client-side rendered, single-page apps, without much of the complexity that comes with modern SPAs. It does this by leveraging existing server-side frameworks.

Inertia has no client-side routing, nor does it require an API. Simply build controllers and page views like you've always done!

Inertia isn't a framework, nor is it a replacement to your existing server-side or client-side frameworks. Rather, it's designed to work with them. Think of Inertia as glue that connects the two. Inertia does this via adapters. We currently have three official client-side adapters (React, Vue.js, and Svelte) and two server-side adapters (Laravel and Rails).

[Who is Inertia for]

Inertia was designed for development teams who typically build server-side rendered applications using frameworks like Laravel,

But what happens when these developers want to replace their server-side rendered views with a modern JavaScript-based single-page app front-end? The answer is always "you need to build an API". Because that's how modern SPAs are built.

This means building a REST or GraphQL API. It means figuring out auth for that API. It means client-side state management. It means setting up a new Git repo. It means setting up another hosting account for the API. And this list goes on. It's a complete paradigm shift.

Inertia allows you to build a fully JavaScript-based single-page app without all this added complexity.

[Code examples]

[Installation]

So the first part we will install is the server side, I've already set up a Laravel site by typing laravel new sitename.

https://inertiajs.com/installation

[Server Side]

We can then run the composer require to pull in the laravel adapter.

```
composer require inertiajs/inertia-laravel
```

After that's installed, we can create the 'root template' this is what gets loaded first on a page visit.

We can delete out the welcome.blade.php as we don't need that

It's not required but recommended that we set up asset versioning straight away.
We can goto the AppServiceProvider in Laravel to set this up.

I usually create a 'bootInertia' function.

```
public function boot()
{
    $this->bootInertia();
}

private function bootInertia()
{
    // If you're using Laravel Mix, you can
    // use the mix-manifest.json for this.
    Inertia::version(function () {
        return md5_file(public_path('mix-manifest.json'));
    });
}
```

We also need to import Inertia

```
use Inertia\Inertia;
```

We can also go to the webpack.mix.js and add in

```
mix.version();
```

[Server Side post setup]

Before we switch to setting up the client side adapter I'm going to quickly make a controller named 'ImageController' and a model named 'Image'

I will just define some fields in the migration.

```
Schema::create('images', function (Blueprint $table) {
    $table->bigIncrements('id');
    $table->string('image');
    $table->string('title');
    $table->timestamps();
});
```

And enable it in the database seeder class
`$this->call(ImagesTableSeeder::class);`

I'm also going to create a web route so we can eventually use this controller.

```
Route::resource('image', 'ImageController');
```

(Explain about resource with pa route:list)

I'm also going to create a seeder and a factory so we can quickly put some fake data in.

Pa make:seeder ImagesTableSeeder
Pa make:factory ImageFactory

Seeder

```
factory(\App\Image::class, 10)->create();
```

Factory

```
use App\Image;
use Faker\Generator as Faker;

$factory->define(Image::class, function (Faker $faker) {
    return [
```

```
        'image' => $faker->imageUrl(),
        'title' => $faker->title,
    ];
});
```

If we try to load the page we will get an error

http://127.0.0.1:8000/image/1

Because we need to set up the client side adapter!

[Client Side]

First we need to install vue, laravel has a package for this

composer require laravel/ui --dev

Php artisan ui vue

Now we can run npm install && npm run dev

Cool now we can install inertia

```
npm install @inertiajs/inertia @inertiajs/inertia-vue
```

We can then go into the vue app.js and place this code in here

```
require('./bootstrap');

window.Vue = require('vue');

import { InertiaApp } from '@inertiajs/inertia-vue';

Vue.use(InertiaApp);



const app = document.getElementById('app');


new Vue({
  render: h => h(InertiaApp, {
      props: {
          initialPage: JSON.parse(app.dataset.page),
          resolveComponent: name => require(`./Pages/${name}`).default,
      },
  }),
}).$mount(app);
```

If we run npm run dev now, we will get an error, because we don't have anything in the Pages folder!

Lets create that and add a vue component in there.

```
<template>
  <div class="container">
      <div class="row justify-content-center">
          <div class="col-md-8">
              <div class="card">
                  <div class="card-header">Example Component</div>

                  <div class="card-body">
                      {{ image }}
                  </div>
              </div>
          </div>
      </div>
  </div>
</template>
```

```
<script>
  export default {
      mounted() {
          console.log('Component mounted.')
      },
      props: [
          'image'
      ]
  }
</script>
```

Ok, if we now serve up the page, we can see there is a vue component loading, and because of our ImageController it's passing in the data to the vue component as a prop.

You can see that we are missing our image url so we can go to the controller and add that in.

```
return Inertia::render('Image/Show', [
  'image' => $image->only(
      'id',
      'title',
      'image'
  ),
]);
```

And now that is being passed through.

Let's get the vue component to show the image, hopefully the lorempixel website is up or this image won't render

```
<template>
  <div class="container">
      <div class="row justify-content-center">
          <div class="col-md-8">
              <div class="card">
                  <div class="card-header">{{ image.title }}</div>

                  <div class="card-body">
                      Rendering: {{ image.image }}
                      <img :src="image.image" />
                  </div>
              </div>
```

```
        </div>
    </div>
  </div>
</template>

<script>
  export default {
    mounted() {
        console.log('Component mounted.')
    },
    props: [
        'image'
    ]
  }
</script>
```

We can delete the ‘Components/ExampleComponent.vue’ folder and file as we don’t need it.

[Routing]

With Inertia all routing is defined server-side. Meaning you don't need Vue Router or React Router. Simply create routes using your server-side framework of choice.

Some server-side frameworks allow you to generate URLs from named routes. However, you will not have access to those helpers client-side.

Except now, you will!

The easiest way is to just pass them through server side via the controller.

```
class UsersController extends Controller
{
    public function index()
    {
        return Inertia::render('Users/Index', [
            'users' => User::all()->map(function ($user) {
                return [
                    'id' => $user->id,
```

```
                'name' => $user->name,
                'email' => $user->email,
                'edit_url' => URL::route('users.edit', $user),
            ];
        }),
        'create_url' => URL::route('users.create'),
    ]);
  }
}
```

But since we are using laravel, there is a package named Ziggy that will help us out here.

If you're using Laravel, the Ziggy library does this for you automatically via a global route() function. If you're using Ziggy with Vue.js, it's helpful to make this function available as a custom $route property so you can use it directly in your templates.

Lets install ziggy

composer require tightenco/ziggy

We also pass them through to a $route variable in Vue

```
Vue.prototype.$route = (...args) => route(...args).url()
```

We also need to add

```
@routes
```

To the app.blade.php in the <head> tags

[Add `{{ $route('image.create') }} <br/>`
Somewhere in the component so we can see that the url is working.]

[Responses]

Just a quick word as we've already seen responses (it's the page we are looking at).

To ensure that pages load quickly, only return the minimum data required for the page. Also, be aware that all data returned from the controllers will be visible client-side, so be sure to omit sensitive information.

[Creating links]

Now usually when you're creating a site you and you wanted to create a link, you would use the a href="" tag. Inertia is slightly different.

We use

```
<inertia-link :href="$route('image.create')">Create New</inertia-link>
```

Inertia links.

Lets see what happens when I add this in.

You will see that the page has errored. But, what's this? A modal has popped up!
All server side errors will automatically pop up as modals in inertia.

This one is because the create method in the controller doesn't exist.

I just want to quickly show you how Laravel resource controllers work.

I'm going to delete this controller, and create a new one with the --resource tag.

You'll see that the page is now fleshed out with all the same functions that we saw earlier when I did the php artisan route:list command.

I'll add my original code back in, but also add

```
return Inertia::render('Image/Create');
```

To the create function.

As well I'll create a 'Create.vue' in the Pages/Image folder.

For now I'll just copy the show.vue and delete the content out.

```
<div class="card-header">Create a new Image record</div>
```

```
<div class="card-body">
```

```
</div>
```

You'll now see that when I click the link, something important is happening, even though our routing is server side, inertia is doing that request for us, so the page change happens client side, we don't get that horrible white flash, it's a true SPA!

We can also hard refresh the page and you'll see that the page loads as normal too.

[Forms]

Cool so lets build a form in this create page.

I'm not going to bore you with building a form so I'll copy and paste this in:

```
<form @submit.prevent="submit">
  <label for="title">Title:</label>
  <input id="title" v-model="form.title" />
  <label for="image">Image URL:label>
  <input id="image" v-model="form.image" />
  <button type="submit">Submit</button>
</form>
```

The javascript will look like this:

```
export default {
  data() {
      return {
          form: {
              title: null,
              image: null
          }
      }
  },
  methods: {
      submit() {
          this.$inertia.post(this.$route('image.store'), this.form);
      }
  }

}
```

We use the this.$inertia.post to post the laravel controller.

We can test this out, and see once again, we get the modal, if we add

```
dd($request->all());
```

Into the store function, we will see the debug data.

Let's put some actual code, this isn't what I'd usually do but it's nice and simple for this example

```
$image = Image::create(
    $request->validate([
        'title' => 'required',
        'image' => 'required'
    ])
);

return redirect()->route('image.show', $image);
```

Let's try that out.. Oops we need to add fillable to the image model.

We can do that with

```
protected $fillable = ['title', 'image''];
```

Or the less safe

```
protected $guarded = [];
```

Errors are a bit weird in Inertia, but we need to add this to the AppServiceProvider via Inertia::share so we can get them passed back to the component.

```
Inertia::share([
    'errors' => function () {
        return Session::get('errors')
            ? Session::get('errors')->getBag('default')->getMessages()
            : (object) [];
    },
]);
```

Then we can add

```
<div v-if="$page.errors.title">{{ $page.errors.title[0] }}</div>
```

To see the errors on the page.

[Pages]

I won't spend too long here but one thing I feel is important is the ability to have layouts.

I'll just past in this code for a layout:

```
<template>
  <main>
    <header>
      <inertia-link href="/">Home</inertia-link>
      <inertia-link href="/about">About</inertia-link>
      <inertia-link href="/contact">Contact</inertia-link>
    </header>
    <article>
      <slot />
    </article>
  </main>
</template>

<script>
  export default {
    props: {
      title: String,
    },
    watch: {
      title: {
        immediate: true,
        handler(title) {
          document.title = title
        },
      },
    },
  }
</script>
```

Lets create an index to see all the images we made:

```
public function index()
{
    return Inertia::render('Image/Index', ['images' => Image::all()]);
}
```

```
<template>
    <div class="container">
        <div class="row justify-content-center">
            <div class="col-md-8">
                <div class="card">
                    <div class="card-header">All Images</div>

                    <div class="card-body">
                        <img v-for="image in images" :src="image.image" />
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>

<script>
    export default {
        mounted() {
            console.log('Component mounted.')
        },
        props: [
            'images'
        ]
    }
</script>
```

We import the new layout

```
import Layout from '../../Shared/Layout';
```

Add the component:

```
components: {
  Layout
},
```

Add the layout the to Index vue component

```
<layout title="All images">
[[content]]
</layout>
```

There are some more things out of the scope of this such as persistent layouts, for example if you had audio playing, e.g a podcast, you might want that to keep playing throughout the whole site.

[Shared Data]

We are close to the end, one last thing we might want to do is share data across all the components, for example Logged in user data, or in this example, the app.name

We add this to the AppServiceProvider just like we did with the error (it is the exact same thing but I just wanted to cover it on it's own).

```
// Synchronously
Inertia::share('app.name', Config::get('app.name'));
```

[Questions]

That's me done, any questions?