1. What is Inertia JS?

Inertia.js allows developers to write single-page applications (SPAs) using classic server-side routing and controllers. Inertia JS was created to be the glue between your backend and frontend frameworks. Inertia tightly couples the backend to the frontend so that developers need not write APIs. For example, if you'd be using Laravel & Vue, you'd no longer have to install Axios and make a request to your Laravel application.

Inertia.js takes a singular position in the SPA solution space between fully client-side SPAs and MPAs (multi-page application). Inertia refers to its approach in the following terms:

> *"*
>
> **Inertia is a new approach to building classic server-driven web apps. We call it *the modern monolith*.**
>
> **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!**

2. Comparison of SSR and CSR

Server-side rendering (SSR) and client-side rendering (CSR) are two different approaches to rendering web applications. The main difference between the two is where the HTML is generated. In SSR, the HTML is generated on the server and sent to the client in a fully rendered state. In CSR, the HTML is generated on the client using JavaScript.

Advantages and Disadvantages of SSR

Advantages:

- Faster initial load times: SSR can deliver a fully rendered page to the client more quickly than CSR, since the browser does not need to download and execute any JavaScript before the page can be displayed.
- Better SEO: SSR pages are more easily readable by search engines, which can lead to better SEO performance.
- Improved accessibility: SSR pages can be more accessible to users with disabilities, since they do not require JavaScript to be enabled.

Disadvantages:

- Increased server load: SSR can generate more load on the server, since the server needs to generate the HTML for each page request.
- Reduced interactivity: SSR pages can be less interactive than CSR pages, since the server needs to generate a new page for every user interaction.

The best approach for your web application will depend on your specific needs and requirements. If you need a website with fast initial load times and good SEO performance, then SSR is a good choice. If you need a web application with a high degree of interactivity and responsiveness, then CSR is a good choice.

Many modern web applications use a hybrid approach that combines SSR and CSR. For example, the initial page load may be rendered on the server, but then CSR can be used to update the page dynamically as the user interacts with it.

Here is a table that summarizes the key differences between SSR and CSR:

Characteristic	SSR	CSR
Where is the HTML generated?	Server	Client
Initial load time	Faster	Slower
SEO performance	Better	Worse
Accessibility	Better	Worse
Interactivity	Reduced	Improved
Server load	Increased	Reduced
Development and maintenance	More complex	Easier

3. Inertia.js Features

Key Features

- Fast and responsive UIs: Inertia.js uses a server-rendering approach to deliver data to the client, which eliminates the need for any loading spinners or progress bars. This results in fast and responsive UIs that feel like native apps.
- Smooth page transitions: Inertia.js uses a clever technique to update the browser's history state when navigating between pages. This allows for smooth page transitions without any jarring reloads.
- Shared controllers: Inertia.js allows you to share controllers between your server-side and client-side code. This simplifies development and makes it easier to keep your codebase in sync.
- Automatic data hydration: Inertia.js automatically hydrates your client-side state with data from the server. This means that you can start interacting with your UI immediately, without having to wait for any data to load.
- Support for popular JavaScript frameworks: Inertia.js supports a variety of popular JavaScript frameworks, including Vue, React, and Svelte. This makes it easy to use Inertia.js in any existing project.

Role in Data-driven UI

Inertia.js is a modern JavaScript framework that allows you to build data-driven UIs without the need for a heavy API. It does this by using the browser's history state to keep track of the current page and its data. When a user navigates to a new page, Inertia.js will fetch the necessary data from the server and then render the page component. This eliminates the need for full-page reloads, resulting in a faster and more responsive user experience.

Code Example:

```
<template>
  <div>
    <h1>{{ page.props.name }}</h1>
  </div>
</template>

<script setup>
import { usePage } from '@inertiajs/vue3'

const page = usePage();
</script>
```

Role in Client-Side Routing

Inertia.js also provides a simple way to implement client-side routing. With Inertia.js, you can define routes in your server-side code and then navigate between those routes using JavaScript on the client-side.

When you navigate between routes using Inertia.js, the client will send a request to the server for the data needed to render the next page. The server will respond with a page object that contains the data and HTML for the page. The client will then render the page and update the browser's history state.

This approach to client-side routing is fast and efficient, and it allows you to maintain a single source of truth for your routes.

Code Example

```
<template>
  <div>
    <Link href="/users/create">Create User</Link>
  </div>
</template>

<script setup>
import { Link } from '@inertiajs/vue3'
</script>
```

Role in Shared Controllers

Another key feature of Inertia.js is the ability to share controllers between your server-side and client-side code. This simplifies development and makes it easier to keep your codebase in sync.

For example, let's say you have a controller that handles user registration. With Inertia.js, you can share this controller between your server-side and client-side code. This means that you can use the same controller to handle user registration on both the front-end and back-end of your application.

```
<?php

// Server-side controller
class UserController extends Controller
{
    public function create()
    {
        return Inertia::render('Users/Create');
    }
}
```

5. Client-Side Components

To use Vue.js and Inertia.js together, firstly need to install the Inertia Vue adapter. Once installed, you can initialize your Inertia application in your main JavaScript file.

```
import { createInertiaApp } from '@inertiajs/vue3'

const app = createInertiaApp({
  // ...
})
```

Each component will represent a page in your application. You can use the Inertia.visit() method to navigate between pages.

```
<script setup>
import { Inertia } from '@inertiajs/vue3'

defineProps({
  // ...
})

async function visit() {
  await Inertia.visit('User/Show', {
    userId: 1,
  })
}
</script>

<template>
  <button @click="visit()">Visit User</button>
</template>
```

If the user navigate to a new page, Inertia will make a request to the server and retrieve the corresponding Vue component. The server will then render the component and send it to the client as HTML.

Data exchange between server and client

Inertia.js provides a simple way to exchange data between the server and client. When you render a page, you can pass data to it using the Inertia.render() method.

```php
use Inertia\Inertia;

class UserController extends Controller
{
  public function show(User $user)
  {
    return Inertia::render('User/Show', [
      'user' => $user,
    ])
  }
}
```

The data that you pass to the Inertia.render() method will be available to the Vue component that renders the page. You can access the data using the props object.

```vue
<script setup>
import { Inertia } from '@inertiajs/vue3'

defineProps({
  user: Object,
})

// ...
</script>
```

You can also send data from the client to the server using the Inertia.post() method. This can be useful for submitting forms or sending AJAX requests.

```vue
<script setup>
import { Inertia } from '@inertiajs/vue3'

defineProps({
  // ...
})

async function submit() {
  await Inertia.post('User/Update', {
    name: 'John Doe',
    email: 'john.doe@example.com',
  })
}
</script>

<template>
  <form @submit="submit()">
    <input type="text" v-model="name" />
    <input type="email" v-model="email" />
    <button type="submit">Submit</button>
  </form>
</template>
```

Conclusion

Vue.js and Inertia.js are a powerful combination for building full-stack applications. Vue.js provides a simple and flexible way to build user interfaces, while Inertia.js provides a lightweight and efficient way to exchange data between the server and client.

Here are some of the benefits of using Vue.js and Inertia.js together:

- Improved performance: Inertia.js can improve the performance of your application by pre-rendering your Vue components on the server.
- Better SEO: Inertia.js can also improve the SEO of your application by sending your Vue components to the client as HTML. This makes it easier for search engines to index your pages.
- Simple and efficient data exchange: Inertia.js provides a simple and efficient way to exchange data between the server and client.
- Flexibility: Vue.js and Inertia.js can be used to build a wide variety of applications, from simple websites to complex web applications.

If you are looking for a powerful and flexible way to build full-stack applications, then Vue.js and Inertia.js is a great combination to consider.