**Ans: to the question no:- 01**

**What is Inertia js**:

It is called the modern monolith because it comes with a new approach to building classic server-driven web apps. For creating modern SPAs. It allows you to create fully client-side rendered, single-page apps, without any complexity. Inertia isn't a framework.

**Working procedure:** You can bulid morden application using Inertia with your server-side web frame work like Laravel, rails without using their template. Rather you can use entire frontend using react, Vue, svelte . Inertia works like a middle man between the server-side & client side with out using any api.

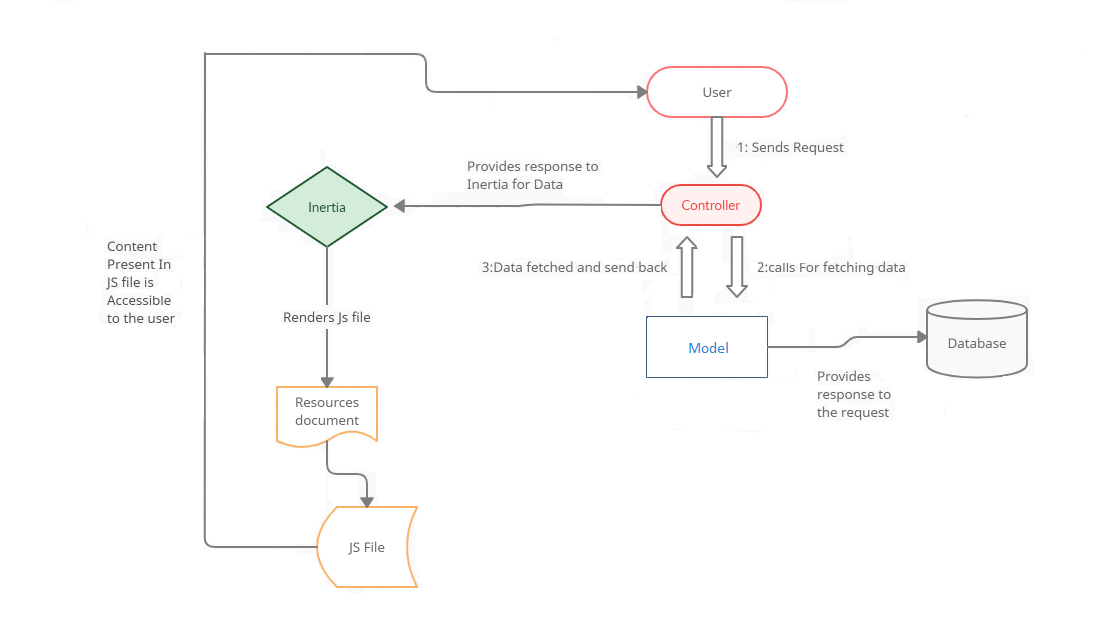


Fig: working procedure of inertia

**Ans: to the question no:- 02**

**Comparison of SSR and CSR:**

1. **SSR :**

SSR stands for server-side rendering. it is the ability of a web application to render the web page on the server instead of rendering it in the browser. when the page arrived on the client-side, it is fully rendered. it is because the server-side has fully rendered the page before it was sent by the server to the client. when the request is received on the server-side, it will compile everything, if the content of the page needed data from the database, the server will do that, then render the data into the fully rendered page and then send it to the client as the response. now, what if the client navigates to a different route? every time the client navigates to a different route, the server will do the work all over again.

1. **CSR:**

CSR stands for client-side rendering. overall, CSR is the opposite of SSR. If the SSR renders the page on the server-side, CSR renders the page on the client-side. when the request is received on the server, it will not render the page, instead, the server will send a single page that will be the skeleton of the page to the client. the server sends the page along with the JavaScript file. later, the js will turn the page into a fully rendered page. so where is the content? what if the page needs to take data from the database? Then, the Api comes in. the client will make a request to the Api to take the data and then render it to the page. Lastly, what if the client navigates to a different route? do the server will send the page again ? the server will not send the page again, instead, the client will re-render the page according to the route that client requested. so the page that is used, is always the same page as the first request.

**Difference between CSR and SSR:**

The main difference between CSR and SSR is where the page is rendered. SSR renders the page on the server-side and CSR renders the page on the client-side. Client-side manages the routing dynamically without refreshing the page every time the client requests a different route.

**Advantage of SSR:**

1. If SEO is your priority, typically when you are building a blog site and you want everyone who searching on google go to your website, then SSR is your choice.
2. If your website needs a faster initial loading.
3. If the content of your website doesn't need much user interaction.

**Advantage of CSR:**

* when SEO is not your priority.
* if your site has rich interactions.
* if you are building a web application.

**Disadvantages of Server-side Rendering :**

1. Increased server load.
2. More complexity
3. Delayed Interactivity
4. Increased bandwidth usage

**Disadvantages of client-side Rendering :**

1. Decreased server load.
2. Less complexity
3. Less Delayed Interactivity
4. Decreased bandwidth usage

**Ans: to the question no:- 03**

**Features:**

Inertia.js is a powerful JavaScript framework designed to simplify building data-driven, client-side interactive web applications. It seamlessly integrates with backend frameworks like Laravel and Ruby on Rails, enabling developers to create modern, dynamic user interfaces. Here are some key features of Inertia.js:

1. **Data-Driven UI**: Inertia.js allows you to fetch and display data from the server and update the UI without a full page reload. It achieves this by making AJAX requests to your server, which responds with the necessary data. This feature is crucial for creating dynamic web applications.

Practical Example:

// Fetch data from the server and update the UI

const data = Inertia.get('/products/1');

**Client-Side Routing**: Inertia.js provides client-side routing capabilities, which means you can navigate between different parts of your application without traditional page loads. It maintains the back and forward buttons' functionality, mimicking server-side routing for a smoother user experience.

**Practical Example:**

// Navigate to a different page with client-side routing

Inertia.visit('/dashboard');

**Shared Controllers**: Inertia.js allows you to share controller logic between the server and client. You write your server-side logic once and use it on the client side as well, reducing duplication and maintaining consistency.

Practical Example:

// Server-side controller

public function show(Product $product)

{

return Inertia::render('Product/Show', ['product' => $product]);

}

// Client-side controller

Inertia.share({

props: {

product: null,

},

});

**Form Handling**: Inertia.js simplifies form handling by automatically managing form submissions and validation errors, all while keeping your user on the same page for a smoother user experience.

Practical Example:

<!-- A form handled by Inertia.js -->

<form

method="post"

action="/products/1"

as="put"

data-inertia

>

<input type="text" name="name">

<button type="submit">Update</button>

</form>

Inertia.js's features combine to create a more efficient and user-friendly web application development experience. Its ability to streamline data-driven UI updates, provide client-side routing, enable shared controllers, and simplify form handling makes it a valuable tool for developers aiming to build responsive and interactive web applications.

**Ans: to the question no:- 04**

**Integration with Laravel:**

Creating a basic project using Laravel and Inertia.js is a straightforward process. I'll guide you through the steps to set up a simple web application that includes at least one page managed by Inertia.js. In this example, we'll create a basic task management application.

**Step 1: Set up Laravel Project**

* Create a new Laravel project:

composer create-project laravel/laravel example-app

**Step 2: Set up Inertia.js and Vue.js**

composer require inertiajs/inertia-laravel

**Step 3: Setup Laravel for vue.js**

* update middleware

php artisan inertia:middleware

* modify /app/http/kernel.php

'web' => [

// ...

\App\Http\Middleware\HandleInertiaRequests::class,

],

**Step 3: Root template**

* update /resources/js/app.js

import { defineConfig } from 'vite';

import laravel from 'laravel-vite-plugin';

export default defineConfig({

plugins: [

laravel({

input: ['resources/css/app.css', 'resources/js/app.js'],

refresh: true,

}),

],

});

**Step 4: Setup vue with vite**

npm install @inertiajs/vue3

npm i @vitejs/plugin-vue

* update /resources/js/app.js

import { defineConfig } from 'vite';

import laravel from 'laravel-vite-plugin';

export default defineConfig({

plugins: [

laravel({

input: ['resources/css/app.css', 'resources/js/app.js'],

refresh: true,

}),

],

});

* update /vite.config.js

import vue from '@vitejs/plugin-vue'

export default {

plugins: [

vue(),

laravel({

input: ['resources/css/app.css', 'resources/js/app.js'],

refresh: true,

}),

],

}

**Step 5: Create Vue Applicatio**

* create a Directory resources/js/Pages/
* create test component resources/js/Pages/Welcome.vue

<script setup>

import {ref, reactive} from 'vue'

</script>

<template>

<div>Welcome</div>

</template>

<style scoped>

</style>

**Step 6: Update Home route with new sample vue component**

* update routes/web.php:

use Inertia\Inertia;

Route::get('/', function(){

return Inertia::render(component:'Welcome');

});

**Step 7: Run the Application**

* Compile the assets:

npm run dev

* Start the development server:

php artisan serve.

* Visit <http://localhost:8000/> inyour browser to see the blog posts.

You've now set up a basic Laravel project with Inertia.js and created a simple web application with at least one page managed by Inertia.js. This example illustrates how Inertia.js seamlessly integrates with Laravel, allowing you to build dynamic, client-side interactive web applications while utilizing the power of Laravel on the backend.

**Add new controller**

1. Define a route in routes/web.php:

Route::get('/posts', 'PostController@index');

1. Create a controller:

php artisan make:controller PostController

1. Create a Model:

php artisan make:model Post -m

1. Update Model:

Schema::create('posts', function (Blueprint $table) {

$table->id();

$table->bigInteger('user\_id')

->foreign('user\_id')

->references('id')

->on('users');

$table->string('slug', 255)

->unique(); // Index

$table->string('title', 255);

$table->text('content');

$table->timestamps();

});

1. Run migration:

php artisan migrate

1. In the PostController, define the index method:

use App\Models\Post;

public function index() {

$posts = Post::all();

return inertia('Posts/Index', ['posts' => $posts]);

}

1. Create a new Vue component at resources/js/Pages/Posts/Index.vue:

<template>

<div>

<h1>Blog Posts</h1>

<ul>

<li v-for="post in posts" :key="post.id">

<h2>{{ post.title }}</h2>

<p>{{ post.content }}</p>

</li>

</ul>

</div>

</template>

<script>

export default {

props: ['posts'],

};

</script>

**Ans: to the question no:- 05**

**Client-Side Components:**

Using a frontend framework like Vue.js alongside Inertia.js is a powerful combination for building client-side components in a web application. This approach leverages the strengths of both technologies to create dynamic, interactive user interfaces. Here's how it works and how data is exchanged between the server and client:

1. **Setting Up Vue.js with Inertia.js**: To use Vue.js with Inertia.js, you typically include the Vue.js library in your project and create Vue components as you would in a traditional Vue.js application. Inertia.js handles the server communication and page navigation, while Vue.js takes care of creating and managing client-side components. You can choose which parts of your application are managed by Vue.js and which are handled by Inertia.js.
2. **Data Exchange**: Inertia.js facilitates data exchange between the server and client by making AJAX requests to your server and fetching data without full page reloads. When you load a page, Inertia.js sends an initial set of data from the server to the client. This data is often used to populate the initial state of Vue.js components.

For example, in a Laravel/Inertia.js project, you might pass data from the server to the client like this:

return Inertia::render('Products/Index', [

'products' => Product::all(),

]);

In your Vue component, you can access this data like so:

export default {

props: {

products: Array,

},

};

**Interactivity**: Vue.js takes over when it comes to adding interactivity and dynamic behavior to your web application. You can create Vue components and utilize the data passed from the server to create interactive user interfaces. Vue.js can make additional AJAX requests to fetch or submit data as needed, but Inertia.js handles the main navigation and communication with the server.

For example, in your Vue component, you can define methods to fetch additional data or update existing data:

methods: {

fetchData() {

// Make an AJAX request to fetch more data

Inertia.get('/products/more')

.then(response => {

// Update component state with the new data

this.products = response.products;

});

},

},

By combining Inertia.js and Vue.js, you can create a seamless and efficient workflow for developing client-side components within a server-rendered application. Inertia.js simplifies server-client communication, page navigation, and data exchange, while Vue.js enhances your application's interactivity and component-based development, resulting in a well-rounded and responsive user experience.