

Laravel 11, Vue 3 এবং Inertia.js গাইড

PDF Version:

Vue.js 3 সম্পূর্ণ গাইডলাইন (Laravel 11 সহ)

Table of Contents

Getting Started

1. Introduction
2. Quick Start

Essentials

3. Creating an Application
4. Template Syntax
5. Reactivity Fundamentals
6. Computed Properties
7. Class and Style Bindings
8. Conditional Rendering
9. List Rendering
10. Event Handling
11. Form Input Bindings
12. Watchers
13. Template Refs
14. Components Basics
15. Lifecycle Hooks

Components In-Depth

16. Registration

- 17. Props
- 18. Events
- 19. Component v-model
- 20. Fallthrough Attributes
- 21. Slots
- 22. Provide / Inject
- 23. Async Components

Reusability

- 24. Composables
- 25. Custom Directives
- 26. Plugins

Built-in Components

- 27. Transition
- 28. TransitionGroup
- 29. KeepAlive
- 30. Teleport
- 31. Suspense

Scaling Up

- 32. Single-File Components
- 33. Tooling
- 34. Routing
- 35. State Management
- 36. Testing
- 37. Server-Side Rendering (SSR)

Best Practices

- 38. Production Deployment

39. Performance

40. Accessibility

41. Security

Getting Started (শুরু করা)

Introduction (পরিচিতি)

Vue.js হলো একটি **progressive JavaScript framework**, যা **SPA (Single Page Application)** তৈরি করতে ব্যবহৃত হয়। এটি সহজেই **Laravel 11** এর সাথে ইন্টিগ্রেট করা যায়।

Quick Start (দ্রুত শুরু)

Vue.js ব্যবহার করার ৩টি উপায় আছে:

- 1 **CDN** ব্যবহার করা (সরাসরি **HTML** ফাইলে ব্যবহার করা যায়)
- 2 **Vite + Vue.js** ব্যবহার করা (**Laravel 11** এর সাথে **Best Practice**)
- 3 **Vue CLI** ব্যবহার করা (বড় অ্যাপ তৈরির জন্য)

Laravel 11 + Vue 3 ইনস্টল করার সঠিক পদ্ধতি

1. Setup Laravel 11 Project

Task: Open your terminal and run:

```
composer create-project laravel/laravel project-name  
cd project-name
```

2. Setup Environment Variables

Task: Rename `.env.example` to `.env` and configure your database

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=db_name
DB_USERNAME=root
DB_PASSWORD=
```

3. Install Laravel Breeze for Inertia.js

Task: Open your terminal and run:

```
composer require laravel/breeze --dev
php artisan breeze:install vue --inertia

npm install @inertiajs/vue3 @vitejs/plugin-vue

npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p

npm install
npm run dev
```

4. Run Migration

Task: Open your terminal and run:

```
php artisan migrate
```

5. Configure Tailwind CSS

Modify `tailwind.config.js`:

```
export default {
  content: [
    './resources/**/*.blade.php',
```

```
"/resources/**/*.vue",
"/resources/**/*.js",
],
theme: {
  extend: {},
},
plugins: [],
};
```

Modify `resources/css/app.css`:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Update `vite.config.js`:

```
import { defineConfig } from 'vite';
import laravel from 'laravel-vite-plugin';
import vue from '@vitejs/plugin-vue';

export default defineConfig({
  plugins: [
    laravel({
      input: ['resources/css/app.css', 'resources/js/app.js'],
      refresh: true,
    }),
    vue(),
  ],
});
```

6. Setup Inertia Middleware

Modify `bootstrap/app.php`:

```
use Inertia\Middleware\HandleInertiaRequests;

$app->middleware([
    HandleInertiaRequests::class,
]);
```

Vue.js 3 এর প্রয়োজনীয় কনসেপ্ট (Essentials)

1. Creating an Application (Vue App তৈরি করা)

কাজ:

Vue অ্যাপ তৈরি করতে `createApp` মেথড ব্যবহার করা হয়। এই মেথড Vue-এর একটি নতুন ইনস্ট্যান্স তৈরি করে।

```
<template>
  <div>
    <h1>{{ message }}</h1>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const message = ref('Welcome to Vue 3!');
</script>
```

2. Template Syntax (Vue টেমপ্লেট সিনট্যাক্স)

কাজ:

Vue-তে ডাটা বাইন্ডিং করার জন্য `{{ }}` এবং `v-bind` ব্যবহার করা হয়।

```
<template>
  <div>
    <p>{{ message }}</p>
    <p :title="tooltip">Hover over me</p>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const message = ref('Hello Vue 3!');
const tooltip = ref('This is a tooltip message');
</script>
```

3. Reactivity Fundamentals (React System কিভাবে কাজ করে)

কাজ:

Vue.js এর সবচেয়ে শক্তিশালী ফিচার হল **reactivity system**, যা ডাটা পরিবর্তনের সাথে সাথে UI আপডেট করে।

```

<template>
  <div>
    <p>Count: {{ state.count }}</p>
    <button @click="state.count++">Increase</button>
  </div>
</template>

<script setup>
import { reactive } from 'vue';

const state = reactive({ count: 0 });
</script>

```

4. Computed Properties (গণনাযোগ্য প্রোপার্টি)

কাজ:

Vue.js-এ কিছু মান স্বয়ংক্রিয়ভাবে আপডেট করার জন্য **computed** প্রোপার্টি ব্যবহার করা হয়।

```

<template>
  <div>
    <p>Number: {{ num }}</p>
    <p>Squared: {{ squared }}</p>
    <button @click="num++">Increase</button>
  </div>
</template>

<script setup>
import { ref, computed } from 'vue';

const num = ref(5);
const squared = computed(() => num.value * num.value);
</script>

```

5. Class and Style Bindings (CSS ক্লাস ও স্টাইল ডায়নামিকভাবে পরিবর্তন করা)

```

<template>
  <div>
    <p :class="{ active: isActive }">Styled Text</p>
    <button @click="isActive = !isActive">Toggle</button>
  </div>
</template>

<script setup>

```

```
import { ref } from 'vue';

const isActive = ref(false);
</script>

<style>
.active {
  color: red;
  font-weight: bold;
}
</style>
```

6. Conditional Rendering (শর্তসাপেক্ষে HTML দেখানো বা লুকানো)

```
<template>
  <div>
    <p v-if="isLoggedIn">Welcome back!</p>
    <p v-else>Please log in</p>
    <button @click="isLoggedIn = !isLoggedIn">Toggle</button>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const isLoggedIn = ref(false);
</script>
```

7. List Rendering (একটি তালিকা ডায়নামিকভাবে দেখানো)

```
<template>
  <ul>
    <li v-for="user in users" :key="user.id">{{ user.name }}</li>
  </ul>
</template>

<script setup>
import { ref } from 'vue';

const users = ref([ { id: 1, name: 'John' }, { id: 2, name: 'Jane' } ]);
</script>
```


8. Event Handling (ইভেন্ট হ্যান্ডলিং করা)

```
<template>
  <div>
    <button @click="count++">Clicked {{ count }} times</button>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const count = ref(0);
</script>
```

9. Form Input Bindings (ইনপুট ফিল্ডের সাথে ডাটা লিঙ্ক করা)

```
<template>
  <div>
    <input v-model="name" placeholder="Enter your name" />
    <p>Hello, {{ name }}</p>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const name = ref("");
</script>
```

10. Watchers (একটি ভেরিয়েবলের পরিবর্তন পর্যবেক্ষণ করা)

```
<template>
  <div>
    <p>Count: {{ count }}</p>
    <button @click="count++">Increase</button>
  </div>
</template>

<script setup>
import { ref, watch } from 'vue';

const count = ref(0);

watch(count, (newValue, oldValue) => {
  console.log(`Count changed from ${oldValue} to ${newValue}`);
});
</script>
```

11. Template Refs (ডাইরেক্টলি DOM এলিমেন্ট অ্যাক্সেস করা)

```
<template>
  <div>
    <input ref="myInput" />
    <button @click="focusInput">Focus</button>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const myInput = ref(null);

const focusInput = () => {
  myInput.value.focus();
};
</script>
```

12. Registration (কম্পোনেন্ট নিবন্ধন করা)

কাজ: Vue.js-এ কম্পোনেন্ট নিবন্ধন করার দুটি পদ্ধতি রয়েছে:

- **Global Registration** (Vue অ্যাপে একবার নিবন্ধন করলে যেকোনো স্থানে ব্যবহার করা যাবে)
- **Local Registration** (শুধুমাত্র নির্দিষ্ট কম্পোনেন্টের মধ্যে ব্যবহার করা যাবে)
- **Global Registration (Laravel 11 + Vue 3)**

```
import { createApp } from 'vue';
import App from './App.vue';
import HelloWorld from './components/HelloWorld.vue';

const app = createApp(App);
app.component('HelloWorld', HelloWorld);
app.mount('#app');
```

- **Local Registration**

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

```
<script setup>
import HelloWorld from './components/HelloWorld.vue';
</script>
```

13. Props (প্রপস ব্যবহার করা)

কাজ: Props ব্যবহার করে আমরা Parent কম্পোনেট থেকে Child কম্পোনেটে ডাটা পাঠাতে পারি।

- Parent Component (App.vue)

```
<template>
  <ChildComponent message="Hello from Parent!" />
</template>

<script setup>
import ChildComponent from './components/ChildComponent.vue';
</script>
```

- Child Component (ChildComponent.vue)

```
<template>
  <p>{{ message }}</p>
</template>

<script setup>
defineProps(['message']);
</script>
```

14. Events (ইভেন্ট হ্যান্ডলিং করা)

কাজ: Child কম্পোনেট থেকে Parent কম্পোনেটে ডাটা পাঠানোর জন্য ইভেন্ট ব্যবহার করা হয়।

- Parent Component (App.vue)

```
<template>
  <div>

    <ChildComponent @send-message="receiveMessage" />
    <p>{{ receivedMessage }}</p>
  </div>
</template>

<script setup>
```

```
import { ref } from 'vue';
import ChildComponent from './components/ChildComponent.vue';

const receivedMessage = ref("");
const receiveMessage = (msg) => {
  receivedMessage.value = msg;
};
</script>
```

- **Child Component (ChildComponent.vue)**

```
<template>
  <button @click="$emit('send-message', 'Hello from Child!')">Send Message</button>
</template>
```

15. Component v-model (কম্পোনেন্টে v-model ব্যবহার করা)

কাজ: Vue 3-এ v-model ব্যবহার করে Parent এবং Child কম্পোনেন্টের মধ্যে Two-way Data Binding করা যায়।

- **Parent Component (App.vue)**

```
<template>
  <ChildComponent v-model="message" />
  <p>Parent Message: {{ message }}</p>
</template>

<script setup>
import { ref } from 'vue';
import ChildComponent from './components/ChildComponent.vue';

const message = ref("");
</script>
```

- **Child Component (ChildComponent.vue)**

```
<template>
  <input v-model="inputValue" @input="$emit('update:modelValue', inputValue)" />
</template>

<script setup>
import { ref } from 'vue';

defineProps(['modelValue']);
```

```
const inputValue = ref("");  
</script>
```

16. Fallthrough Attributes (অতিরিক্ত অ্যাট্রিবিউট হস্তান্তর করা)

কাজ: Child কম্পোনেন্টের Root Element-এ Parent Component থেকে প্রাপ্ত সব Attribute স্বয়ংক্রিয়ভাবে যোগ হয়।

- Parent Component (App.vue)

```
<template>  
  <ChildComponent class="custom-class" title="Hello" />  
</template>  
  
<script setup>  
import ChildComponent from './components/ChildComponent.vue';  
</script>
```

- Child Component (ChildComponent.vue)

```
<template>  
  <button>Click Me</button>  
</template>
```

ফলাফল:

```
<button class="custom-class" title="Hello">Click Me</button>
```

17. Slots (কম্পোনেন্টের ভিতরে কাস্টম কন্টেন্ট পাঠানো)

কাজ: Slots ব্যবহার করে Parent কম্পোনেন্ট থেকে Child কম্পোনেন্টের ভিতরে ডাটা পাঠানো যায়।

- Parent Component (App.vue)

```
<template>  
  <ChildComponent>
```

```
<p>This is a custom slot content.</p>
</ChildComponent>
</template>

<script setup>
import ChildComponent from './components/ChildComponent.vue';
</script>
```

- **Child Component (ChildComponent.vue)**

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

18. Provide / Inject (ডাটা শেয়ার করা)

কাজ: Provide / Inject ব্যবহার করে Parent থেকে অনেকগুলো Nested Child কম্পোনেন্টে ডাটা শেয়ার করা যায়।

- **Parent Component (App.vue)**

```
<template>
  <ChildComponent />
</template>

<script setup>
import { provide } from 'vue';
import ChildComponent from './components/ChildComponent.vue';

provide('appMessage', 'Hello from Provide!');
</script>
```

- **Child Component (ChildComponent.vue)**

```
<template>
  <p>{{ injectedMessage }}</p>
</template>

<script setup>
import { inject } from 'vue';

const injectedMessage = inject('appMessage');
```

```
</script>
```

19. Async Components (অ্যাসিনক্রোনাস কম্পোনেন্ট লোড করা)

কাজ: Lazy Load করা যায়, যখন কম্পোনেন্টের লোডিং সময় বেশি লাগে।

```
<template>
  <Suspense>
    <AsyncComponent />
    <template #fallback>
      <p>Loading...</p>
    </template>
  </Suspense>
</template>

<script setup>
import { defineAsyncComponent } from 'vue';

const AsyncComponent = defineAsyncComponent(() =>
import('./components/AsyncComponent.vue'));
</script>
```

20. Composables (Reusable Functions ব্যবহার করা)

কাজ: Vue 3-এ **Composables** ব্যবহার করে আমরা **Reusable Functions** তৈরি করতে পারি, যা কম্পোনেন্টগুলোর মধ্যে কোড পুনঃব্যবহারযোগ্য করে।

• Counter Composable (useCounter.js)

```
import { ref } from 'vue';

export default function useCounter() {
  const count = ref(0);
  const increment = () => count.value++;
  return { count, increment };
}
```

• Component এ ব্যবহার করা (Counter.vue)

```
<template>
  <div>
    <p>Count: {{ count }}</p>
```

```

    <button @click="increment">Increase</button>
  </div>
</template>

<script setup>
import useCounter from '../composables/useCounter';

const { count, increment } = useCounter();
</script>

```

21. Custom Directives (নিজস্ব ডিরেক্টিভ তৈরি করা)

কাজ: Vue.js-এ `v-*` ডিরেক্টিভ ব্যবহার করা যায়। আমরা নিজস্ব ডিরেক্টিভও তৈরি করতে পারি।

• Custom Directive তৈরি করা (highlight.js)

```

export default {
  beforeMount(el, binding) {
    el.style.backgroundColor = binding.value;
  }
};

```

• Component এ Custom Directive ব্যবহার করা

```

<template>
  <p v-highlight="yellow">Highlighted Text</p>
</template>

<script setup>
import { createApp } from 'vue';
import highlight from '../directives/highlight';

const app = createApp({});
app.directive('highlight', highlight);
</script>

```

22. Plugins (Vue প্লাগিন তৈরি ও ব্যবহার করা)

কাজ: Vue.js প্লাগিন ব্যবহার করে আমরা গ্লোবাল ফাংশন, কম্পোনেন্ট এবং ডাইরেক্টিভ যুক্ত করতে পারি।

• Custom Plugin তৈরি করা (myPlugin.js)

```

export default {
  install(app) {

```



```
app.config.globalProperties.$greet = (name) => `Hello, ${name}!`;
}
};
```

● Plugin ব্যবহার করা (main.js)

```
import { createApp } from 'vue';
import App from './App.vue';
import myPlugin from './plugins/myPlugin';

const app = createApp(App);
app.use(myPlugin);
app.mount('#app');
```

1. Component এ Plugin ব্যবহার করা

```
<template>
  <p>{{ $greet('Vue User') }}</p>
</template>
```

23. Transition (অ্যানিমেশন প্রয়োগ করা)

কাজ: **Vue.js-এ Transition** ব্যবহার করে CSS অ্যানিমেশন সহজে যোগ করা যায়।

```
<template>
  <div>
    <button @click="show = !show">Toggle</button>
    <transition name="fade">
      <p v-if="show">Hello Vue!</p>
    </transition>
  </div>
</template>

<script setup>
import { ref } from 'vue';
const show = ref(true);
</script>

<style>
.fade-enter-active, .fade-leave-active {
  transition: opacity 0.5s;
}
.fade-enter, .fade-leave-to {
  opacity: 0;
}
```

```
</style>
```

24. TransitionGroup (একাধিক এলিমেন্ট অ্যানিমেট করা)

```
<template>
  <button @click="addItem">Add Item</button>
  <transition-group name="list" tag="ul">
    <li v-for="item in items" :key="item">{{ item }}</li>
  </transition-group>
</template>

<script setup>
import { ref } from 'vue';
const items = ref([1, 2, 3]);
const addItem = () => items.value.push(items.value.length + 1);
</script>

<style>
.list-enter-active, .list-leave-active {
  transition: all 0.5s;
}
.list-enter, .list-leave-to {
  opacity: 0;
  transform: translateY(30px);
}
</style>
```

25. KeepAlive (কম্পোনেন্ট স্টেট সংরক্ষণ করা)

```
<template>
  <keep-alive>
    <component :is="currentView" />
  </keep-alive>
</template>

<script setup>
import { ref } from 'vue';
import ViewA from './ViewA.vue';
import ViewB from './ViewB.vue';

const currentView = ref(ViewA);
</script>
```

26. Teleport (একটি এলিমেন্ট অন্য স্থানে রেন্ডার করা)

```
<template>
  <teleport to="body">
    <div class="modal">
      <p>This is a teleported modal!</p>
    </div>
  </teleport>
</template>
```

27. Suspense (অ্যাসিনক্রোনাস কম্পোনেন্ট হ্যান্ডল করা)

```
<template>
  <Suspense>
    <template #default>
      <AsyncComponent />
    </template>
    <template #fallback>
      <p>Loading...</p>
    </template>
  </Suspense>
</template>

<script setup>
import { defineAsyncComponent } from 'vue';
const AsyncComponent = defineAsyncComponent(() => import('./AsyncComponent.vue'));
</script>
```

28. Single-File Components (SFCs)

কাজ: Vue.js-এ **Single-File Components (SFCs)** ব্যবহার করলে **HTML, JavaScript ও CSS** একই ফাইলে রাখা যায়, যা কোড ব্যবস্থাপনা সহজ করে।

29. একটি সাধারণ SFC (ExampleComponent.vue)

```
<template>
  <div>
    <h1>{{ message }}</h1>
  </div>
</template>

<script setup>
import { ref } from 'vue';
const message = ref('Hello from SFC!');
</script>

<style scoped>
```

```
h1 {  
  color: blue;  
}  
</style>
```

Laravel Blade ফাইলের মধ্যে **Vue** কম্পোনেন্ট লোড করা:

```
<div id="app">  
  <example-component></example-component>  
</div>
```

30. Tooling (Vue Devtools & Vite)

কাজ: Vue.js-এর জন্য **Vue Devtools** এবং **Vite** ব্যবহার করে দ্রুত ডেভেলপমেন্ট ও ডিবাগিং করা যায়।

❖ **Vue Devtools** ইনস্টল করা (Chrome/Firefox Extension)

- [Vue Devtools Chrome Extension](#)
- [Vue Devtools Firefox Extension](#)

❖ **Vite** ব্যবহার করে **Vue.js** প্রোজেক্ট তৈরি করা

```
npm create vite@latest my-vue-app --template vue  
cd my-vue-app  
npm install  
npm run dev
```

31. Routing (Vue Router ব্যবহার করা)

কাজ: Vue Router ব্যবহার করে **SPA (Single Page Application)** তৈরি করা যায়।

• **Vue Router** সেটআপ (**router.js**)

```
import { createRouter, createWebHistory } from 'vue-router';  
import Home from '../views/Home.vue';  
import About from '../views/About.vue';  
  
const routes = [  
  { path: '/', component: Home },  
  { path: '/about', component: About }  
];  
  
const router = createRouter({  
  history: createWebHistory(),  
  routes  
});
```

```
export default router;
```

- **Vue অ্যাপে Router ব্যবহার করা (main.js)**

```
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

const app = createApp(App);
app.use(router);
app.mount('#app');
```

32. State Management (Pinia & Vuex)

কাজ: Vue.js-এ **Pinia** বা **Vuex** ব্যবহার করে গ্লোবাল স্টেট ম্যানেজমেন্ট করা যায়।

- **Pinia** ইনস্টল করা

`npm install pinia`

- **Pinia Store তৈরি করা (store.js)**

```
import { defineStore } from 'pinia';
import { ref } from 'vue';

export const useCounterStore = defineStore('counter', () => {
  const count = ref(0);
  const increment = () => count.value++;
  return { count, increment };
});
```

- **Component এ Store ব্যবহার করা**

```
<template>
  <div>
    <p>Count: {{ counter.count }}</p>
    <button @click="counter.increment">Increase</button>
  </div>
```

```
</template>
```

```
<script setup>
```

```
import { useCounterStore } from '../store';
```

```
const counter = useCounterStore();
```

```
</script>
```

33. Computed Properties & Watch ব্যবহার করা

Avoid methods, use computed for better performance.

```
<template>
```

```
<p>Full Name: {{ fullName }}</p>
```

```
</template>
```

```
<script setup>
```

```
import { ref, computed } from 'vue';
```

```
const firstName = ref('John');
```

```
const lastName = ref('Doe');
```

```
const fullName = computed(() => `${firstName.value} ${lastName.value}`);
```

```
</script>
```

Inertia.js v2 Documentation

1. Introduction (পরিচিতি)

Task: Inertia.js হলো একটি modern frontend framework যা Laravel-এর সাথে Vue, React, বা Svelte ব্যবহার করে Single Page Application (SPA) তৈরি করতে সাহায্য করে। এটি API তৈরি না করেও, সার্ভার-সাইড রেন্ডারিং-এর সুবিধা দিয়ে থাকে।

Example: Laravel এবং Vue 3 ব্যবহার করে একটি Inertia.js ভিত্তিক অ্যাপ বানানোর জন্য নিচের উদাহরণটি দেখুন।

```
import { useForm } from '@inertiajs/vue3';

const form = useForm({
  name: "",
  email: "",
  password: ""
});

const submit = () => {
  form.post('/register');
};
```

2. Core Concepts (মূল ধারণা)

Task: Inertia.js এর ৩টি গুরুত্বপূর্ণ কনসেপ্ট:

- **Pages** – Vue/React/Svelte-তে তৈরি ফ্রন্টএন্ড পেজ।
- **Requests & Responses** – Laravel থেকে Vue-তে ডাটা পাঠানোর উপায়।
- **Shared Data** – Inertia middleware ব্যবহার করে global data শেয়ার করা।

Example:

```
return Inertia::render('Dashboard', [
  'user' => auth()->user()
]);
```

3. Who is it for? (কার জন্য)

Task:

- যদি SPA তৈরি করতে চান **Laravel + Vue/React** ব্যবহার করে।
- API ছাড়াই **JavaScript Framework** ইন্টিগ্রেট করতে চান।

4. How it Works? (কীভাবে কাজ করে)

Task: Inertia.js PHP framework (Laravel) এবং JavaScript framework (Vue) কে সংযুক্ত করে।

Example:

```
Route::get('/dashboard', function () {  
    return Inertia::render('Dashboard', [  
        'user' => auth()->user()  
    ]);  
});
```

5. The Protocol (প্রটোকল)

Task: Inertia.js AJAX-based request-response format ব্যবহার করে।

Example:

Request:

```
form.post('/profile/update');
```

Response:

```
return Inertia::render('Profile', [  
    'user' => $request->user()  
]);
```

6. Redirects (রিডাইরেক্ট করা)

Task: Inertia.js এ **redirect** করার জন্য **redirect()** ব্যবহার করা হয়, যাতে **Laravel session flash data** **Vue**-তে পাঠাতে পারে।

Example:

```
return redirect('/dashboard')->with('success', 'Profile updated successfully!');
```


Vue Component (resources/js/Pages/Dashboard.vue)

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

const page = usePage();
</script>

<template>
  <p v-if="page.props.success">{{ page.props.success }}</p>
</template>
```

7. Title & Meta (পেজের টাইটেল এবং মেটা সেট করা)

Task: Inertia.js title এবং meta tags সেট করার জন্য **usePage()** ও **head** ব্যবহার করা হয়।

Example:

Vue Component (resources/js/Pages/Home.vue)

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

<template>
  <Head title="Home Page" />
  <h1>Welcome to Home Page</h1>
</template>
```

8. Links (Inertia Link ব্যবহার করা)

Task: Inertia.js এ **Link** কম্পোনেন্ট ব্যবহার করে Vue-এর মধ্যে SPA-like navigation করা হয়।

Example:

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

<template>
  <Link href="/dashboard">Go to Dashboard</Link>
</template>
```

9. Manual Visits (ম্যানুয়ালি ভিজিট পাঠানো)

Task: `Inertia.visit()` ব্যবহার করে ম্যানুয়ালি নেভিগেট করা যায়।

Example:

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

const goToProfile = () => {
  Inertia.visit('/profile');
};
</script>

<template>
<button @click="goToProfile">Go to Profile</button>
</template>
```

10. File Uploads (ফাইল আপলোড করা)

Task: `Inertia.js` এ `useForm()` দিয়ে ফাইল আপলোড করা যায়।

Example:

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

const form = useForm({
  avatar: null
});

const submit = () => {
  form.post('/upload', {
    forceFormData: true
  });
};
</script>

<template>
<form @submit.prevent="submit">
  <input type="file" @change="form.avatar = $event.target.files[0]">
  <button type="submit">Upload</button>
</form>
</template>
```

11. Validation (ভ্যালিডেশন সেট করা)

Task: Laravel থেকে validation errors Vue-তে পাঠানো হয়।

Example:

```
use Illuminate\Http\Request;
use Inertia\Inertia;

Route::post('/register', function (Request $request) {
    $request->validate([
        'name' => 'required',
        'email' => 'required|email',
        'password' => 'required|min:6',
    ]);

    return redirect('/dashboard');
});
```

Vue Component ([resources/js/Pages/Register.vue](#))

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

const form = useForm({
    name: "",
    email: "",
    password: ""
});

const submit = () => {
    form.post('/register');
};
</script>

<template>
<form @submit.prevent="submit">
    <input v-model="form.name" placeholder="Name">
    <p v-if="form.errors.name">{{ form.errors.name }}</p>

    <input v-model="form.email" placeholder="Email">
    <p v-if="form.errors.email">{{ form.errors.email }}</p>

    <input v-model="form.password" type="password" placeholder="Password">
    <p v-if="form.errors.password">{{ form.errors.password }}</p>

    <button type="submit">Register</button>
</form>
</template>
```

```
</form>
</template>
```

12. Shared Data (সকল পেজে ডাটা শেয়ার করা)

Task: কিছু ডাটা, যেমন **authenticated user**, **settings**, **flash messages**, সব পেজে **globally** শেয়ার করা যেতে পারে।

Example: `app/Http/Middleware/HandleInertiaRequests.php`

```
namespace App\Http\Middleware;

use Inertia\Middleware;
use Illuminate\Http\Request;

class HandleInertiaRequests extends Middleware
{
    public function share(Request $request)
    {
        return array_merge(parent::share($request), [
            'auth' => [
                'user' => $request->user(),
            ],
            'flash' => [
                'message' => session('message'),
            ]
        ]);
    }
}
```

Vue Component (`resources/js/Layouts/AppLayout.vue`)

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

const page = usePage();
</script>

<template>
<p v-if="page.props.flash.message">{{ page.props.flash.message }}</p>
<slot />
</template>
```

13. Partial Reloads (কিছু অংশ রিফ্রেশ করা)

Task: কিছু নির্দিষ্ট **props reload** করার জন্য **only()** ব্যবহার করা যায়।

Example:

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

const refreshNotifications = () => {
  Inertia.reload({ only: ['notifications'] });
};
</script>

<template>
<button @click="refreshNotifications">Reload Notifications</button>
<ul>
<li v-for="notification in usePage().props.notifications" :key="notification.id">
  {{ notification.message }}
</li>
</ul>
</template>
```

14. Deferred Props (ডাটা লোড ডিলে করা)

Task: কিছু ডাটা লোড করা সময় সাপেক্ষ হতে পারে, তাই **lazy props** ব্যবহার করা হয়।

Example:

```
return Inertia::render('Dashboard', [
  'user' => auth()->user(),
  'analytics' => Inertia::lazy(fn () => getAnalyticsData()),
]);
```

Vue Component:

```
<script setup>
defineProps({ user: Object, analytics: Object });
</script>

<template>
  <h1>Welcome, {{ user.name }}</h1>
  <p v-if="analytics">Page views: {{ analytics.views }}</p>
</template>
```

15. Polling (নির্দিষ্ট সময় পর পর ডাটা আপডেট করা)

Task: Inertia.js ব্যবহার করে কিছু নির্দিষ্ট সময় পর পর ডাটা রিফ্রেশ করা যায়।

Example:

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

const fetchData = () => {
  Inertia.reload({ only: ['notifications'] });
};

onMounted(() => {
  setInterval(fetchData, 10000); // Every 10 seconds
});
</script>

<template>
  <h1>Live Notifications</h1>
</template>
```

16. Prefetching (আগে থেকে ডাটা লোড করা)

Task: Inertia.js এর prefetching ব্যবহার করে লিঙ্কের ডাটা আগে থেকে লোড করা যায়।

Example:

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

```
<template>
  <Link href="/dashboard" prefetch>Go to Dashboard</Link>
</template>
```

17. Load When Visible (প্রয়োজন হলে ডাটা লোড করা)

Task: কোনো Vue কম্পোনেন্ট স্ক্রিনে এলে তখন ডাটা লোড করানো যেতে পারে।

Example:

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

const loaded = ref(false);
const loadData = () => {
  if (!loaded.value) {
    Inertia.reload({ only: ['reports'] });
    loaded.value = true;
  }
};
</script>

<template>
  <div v-intersect="loadData">
    <h1>Reports</h1>
  </div>
</template>
```

18. Merging Props (প্রপস মার্জ করা)

Task: একই কম্পোনেন্টে নতুন props merge করতে হলে **merge** ফাংশন ব্যবহার করা হয়।

Example:

```
return Inertia::render('Dashboard', [
  'user' => auth()->user(),
  'settings' => [
    'theme' => 'dark',
    'notifications' => true,
  ]
]);
```

Vue Component:

```
<script setup>
defineProps({
  user: Object,
  settings: { type: Object, default: () => ({ theme: 'light', notifications: false }) }
});
</script>

<template>
<h1>Welcome, {{ user.name }}</h1>
<p>Theme: {{ settings.theme }}</p>
</template>
```

19. Remembering State (পেজ স্টেট মনে রাখা)

Task: পেজ **reload** বা **navigate** করলেও **form** ডাটা বা পেজ স্টেট মনে রাখা যায়।

Example:

```
<script setup>
import { useForm, useRemember } from '@inertiajs/vue3';

const form = useRemember(useForm({
  search: '',
  filter: 'all'
})));
</script>

<template>
<input v-model="form.search" placeholder="Search">
<select v-model="form.filter">
  <option value="all">All</option>
  <option value="active">Active</option>
</select>
</template>
```

20. Security & Advanced Features

- **Security** (সিকিউরিটি নিশ্চিত করা)

Task: Inertia.js Laravel এর নিরাপত্তা ফিচার ব্যবহার করে **authentication**, **authorization**, **CSRF protection**, এবং **encryption** নিশ্চিত করে।

- **Authentication** (অথেনটিকেশন সেটআপ করা)

Task: Laravel-এর **auth middleware** ব্যবহার করে Inertia.js-এ **authentication** নিশ্চিত করা।

Example:

Laravel Route (routes/web.php)

```
Route::middleware(['auth'])->get('/dashboard', function () {  
    return Inertia::render('Dashboard', [  
        'user' => auth()->user()  
    ]);  
});
```

Vue Component (resources/js/Pages/Dashboard.vue)

```
<script setup>  
defineProps({ user: Object });  
</script>  
  
<template>  
  <h1>Welcome, {{ user.name }}</h1>  
</template>
```

21. Authorization (ব্যবহারকারী অনুমতি নিয়ন্ত্রণ করা)

Task: Policies এবং Gates ব্যবহার করে user permission নিয়ন্ত্রণ করা হয়।

Example: Policy Middleware ব্যবহার করা

```
use App\Models\User;  
use Illuminate\Support\Facades\Gate;  
  
Gate::define('view-dashboard', function (User $user) {  
    return $user->isAdmin();  
});  
  
Route::get('/dashboard', function () {  
    if (!Gate::allows('view-dashboard')) {  
        abort(403);  
    }  
    return Inertia::render('Dashboard');  
});
```

22. CSRF Protection (Cross-Site Request Forgery প্রতিরোধ করা)

Task: Inertia.js Laravel CSRF protection ব্যবহার করে।

Example: Form Submission

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

const form = useForm({
  name: "",
  _token: document.querySelector('meta[name="csrf-token"]').content
});

const submit = () => {
  form.post('/profile/update');
};
</script>
```

23. History Encryption (ইতিহাস এনক্রিপশন করা)

Task: Inertia.js ব্যবহার করে session-based state encryption করা হয়, যাতে sensitive data client-side এ না থাকে।

Example: Laravel session ব্যবহার করে state সংরক্ষণ করা।

```
session(['history' => encrypt(json_encode($request->all()))]);
```

24. Asset Versioning (CSS/JS নতুন ভার্সন লোড করানো)

Task: `version()` ব্যবহার করে নতুন assets ফোর্স করা হয়।

Example:

```
use Inertia\Inertia;

Inertia::version(function () {
  return md5_file(public_path('mix-manifest.json'));
});
```

25. Events (ইভেন্ট ব্যবস্থাপনা করা)

Task: Inertia.js ইভেন্ট ব্যবস্থাপনা ব্যবহার করে পেজ রিফ্রেশ বা লোডের উপর অ্যাকশন নেওয়া যায়।

Example: Form Submission Event

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

Inertia.on('success', (event) => {
  console.log('Form submitted successfully!', event);
});
</script>
```

26. Progress Indicators (লোডিং প্রগ্রেস দেখানো)

Task: Inertia.js progress bar যোগ করতে nprogress ব্যবহার করা হয়।

Installation:

npm install nprogress

Example:

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

Inertia.on('start', () => NProgress.start());
Inertia.on('finish', () => NProgress.done());
</script>
```

27. Scroll Management (স্ক্রল অবস্থান মনে রাখা)

Task: Inertia.js scroll position রিসেট করে বা পূর্বের অবস্থায় ফিরিয়ে নেয়।

Example:

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

const scrollPosition = useRemember({ top: 0 });
```

```

window.addEventListener('scroll', () => {
  scrollTop.top = window.scrollY;
});
</script>

<template>
  <div>
    <h1>Page Title</h1>
  </div>
</template>

```

28. Server-side Rendering (SSR)

Task: Inertia.js SSR (Server-Side Rendering) সাপোর্ট দেয়।

Example: Inertia SSR ইমপ্লিমেন্ট করা

```

import createServer from '@inertiajs/server';
import { createInertiaApp } from '@inertiajs/vue3';

createServer((page) =>
  createInertiaApp({
    page,
    resolve: (name) => import(`./Pages/${name}.vue`),
    setup({ App, props }) {
      return h(App, props);
    },
  })
);

```

29. Testing (ইনটিগ্রেশন টেস্ট করা)

Task: Laravel ও Inertia.js-এ Pest/PHPUnit দিয়ে টেস্ট করা যায়।

Example: Inertia response টেস্ট করা

```

use Inertia\Testing\AssertableInertia;

test('dashboard page loads', function () {
  $this->actingAs(User::factory()->create())
    ->get('/dashboard')
    ->assertInertia(fn (AssertableInertia $page) =>







```

```
$page->component('Dashboard')  
    ->has('user')  
    );  
});
```

Laravel 11 + Vue 3 + Inertia.js

E-Commerce CRUD with Image Upload

This is a **step-by-step** guide to building an **E-Commerce CRUD** system with **Laravel 11**, **Vue 3**, **Inertia.js**, and **Tailwind CSS**. The project includes:

-  **Product Create**
-  **Product Read (List & Single View)**
-  **Product Update**
-  **Product Delete**
-  **Image Upload**
-  **Fully Responsive UI with Tailwind CSS**

Step 1: Install Laravel 11

First, install Laravel 11 and navigate to the project folder.

```
composer create-project laravel/laravel laravel_vue_inertia
cd laravel_vue_inertia
```

Step 2: Install Inertia.js, Vue 3, and Tailwind CSS

```
composer require laravel/breeze --dev
php artisan breeze:install vue --inertia

npm install @inertiajs/vue3 @vitejs/plugin-vue

npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p

npm install
npm run dev
```

Step 3: Configure Tailwind CSS

Modify **tailwind.config.js**:

```
export default {
  content: [
    "./resources/**/*.blade.php",
    "./resources/**/*.vue",
    "./resources/**/*.js",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

Modify **resources/css/app.css**:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Update **vite.config.js**:

```
import { defineConfig } from 'vite';
import laravel from 'laravel-vite-plugin';
import vue from '@vitejs/plugin-vue';

export default defineConfig({
  plugins: [
    laravel({
      input: ['resources/css/app.css', 'resources/js/app.js'],
      refresh: true,
    }),
    vue(),
  ],
});
```

Step 4: Setup Inertia Middleware

Modify **bootstrap/app.php**:

```
use Inertia\Middleware\HandleInertiaRequests;

$app->middleware([
    HandleInertiaRequests::class,
]);
```

Step 5: Setup Database

Update **.env** file:

```
DB_DATABASE=ecommerce
DB_USERNAME=root
DB_PASSWORD=
```

Run migration:

```
php artisan migrate
```

Step 6: Create Product Model & Migration

php artisan make:model Product -m

Modify **database/migrations/xxxx_xx_xx_create_products_table.php**:

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->text('description');
            $table->decimal('price', 10, 2);
            $table->integer('stock');
            $table->string('image')->nullable();
            $table->timestamps();
        });
    }

    public function down(): void {
        Schema::dropIfExists('products');
    }
};
```

Model: Product.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model {
    use HasFactory;
}
```



```
protected $fillable = [  
    'name',  
    'description',  
    'price',  
    'stock',  
    'image',  
];  
}
```

Run migration:

```
php artisan migrate
```

Step 7: Create Product Controller

```
php artisan make:controller ProductController
```

Modify **app/Http/Controllers/ProductController.php**:

```
<?php  
  
namespace App\Http\Controllers;  
  
use App\Http\Resources\ProductResource;  
use App\Models\Product;  
use Illuminate\Foundation\Application;
```

```

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Redirect;
use Illuminate\Support\Facades\Route;
use Inertia\Inertia;

class ProductController extends Controller {

    protected function getSharedProps() {
        return [
            'canLogin'      => Route::has( 'login' ),
            'canRegister'   => Route::has( 'register' ),
            'laravelVersion' => Application::VERSION,
            'phpVersion'    => PHP_VERSION,
        ];
    }

    public function index() {
        $products = Product::latest()->get();

        return Inertia::render( 'Products/Index', array_merge(
            ['products' => ProductResource::collection( $products
)],
            $this->getSharedProps()
        ) );
    }

    public function create() {
        return Inertia::render( 'Products/Create',
            $this->getSharedProps()
        );
    }

    public function store( Request $request ) {
        $request->validate( [
            'name'          => 'required|string|max:255',
            'description'   => 'required|string',
            'price'         => 'required|numeric',
            'stock'         => 'required|integer',
            'image'         =>

```

```

'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
    ] );

    if ( $request->hasFile( 'image' ) ) {
        $image = $request->file( 'image' );
        $imageName = time() . '.' .
$image->getClientOriginalExtension();
        $image->move( public_path( 'products' ), $imageName );
        $imagePath = 'products/' . $imageName;
    } else {
        $imagePath = null;
    }

    Product::create( array_merge( $request->all(), ['image' =>
$imagePath] ) );

    return Redirect::route( 'product.index' )->with( [
        'success' => 'Product created successfully',
    ] );

    // session()->flash( 'success', 'Product created
successfully' ); // ✅ Ensure flash message is set

    // return Redirect::route( 'product.index' );

    // return redirect()->route( 'product.index' )->with(
'message', 'Product created successfully' );
}

public function edit( Product $product ) {
    return Inertia::render( 'Products/Edit', array_merge(
        ['product' => $product],
        $this->getSharedProps()
    ) );
}

public function update( Request $request, Product $product ) {

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

    $request->validate( [

```

```

        'name'          => 'required|string|max:255',
        'description' => 'required|string',
        'price'         => 'required|numeric',
        'stock'         => 'required|integer',
        'image'         => 'nullable|image|max:2048',
    ] );

    if ( $request->hasFile( 'image' ) ) {

        if ( $product->image ) {
            $oldImagePath = public_path( $product->image );
            if ( file_exists( $oldImagePath ) ) {
                unlink( $oldImagePath );
            }
        }

        $image = $request->file( 'image' );
        $imageName = time() . '.' . $image->getClientOriginalExtension();
        $image->move( public_path( 'products' ), $imageName );
        $imagePath = 'products/' . $imageName;
    } else {
        $imagePath = $product->image;
    }

    $product->update( array_merge( $request->all(), ['image' => $imagePath] ) );

    return Redirect::route( 'product.index' )->with( [
        'success' => 'Product updated successfully',
    ] );

}

public function show( Product $product ) {
    return Inertia::render( 'Products/Show', array_merge(
        ['product' => $product],
        $this->getSharedProps()
    ) );
}

```

```

    }

    public function destroy( Product $product ) {
        if ( $product->image ) {

            $imagePath = public_path( $product->image );
            if ( file_exists( $imagePath ) ) {
                unlink( $imagePath );
            }
        }

        $product->delete();

        return Redirect::route( 'product.index' )->with( [
            'success' => 'Product deleted successfully',
        ] );
    }
}

```

Step 8: Define Routes

Modify **routes/web.php**:

```

<?php

use App\Http\Controllers\ProductController;
use App\Http\Controllers\ProfileController;
use Illuminate\Foundation\Application;
use Illuminate\Support\Facades\Route;
use Inertia\Inertia;

Route::get( '/', function () {
    return Inertia::render( 'Home', [
        'canLogin'    => Route::has( 'login' ),
        'canRegister' => Route::has( 'register' ),
        'laravelVersion' => Application::VERSION,
        'phpVersion'   => PHP_VERSION,
    ] );
} )->name( 'home' );

```

```
Route::get( '/dashboard', function () {
    return Inertia::render( 'Dashboard' );
} )->middleware( ['auth', 'verified'] )->name( 'dashboard' );

Route::middleware( 'auth' )->group( function () {
    Route::get( '/profile', [ProfileController::class, 'edit'] )->name( 'profile.edit' );
    Route::patch( '/profile', [ProfileController::class, 'update'] )->name( 'profile.update' );
    Route::delete( '/profile', [ProfileController::class, 'destroy'] )->name( 'profile.destroy' );
} );

Route::resource( 'product', ProductController::class );
Route::post( 'product/{product}', [ProductController::class, 'update'] )->name( 'products.update' );

require __DIR__ . '/auth.php';
```

Step 9: Frontend Setup

Now, I will create the Vue frontend files for **product listing, create, update, delete, and single product view**. Each module will be provided separately.

Let's add each module **step by step** for better clarity.

Step 1: Product List Page (Index.vue)

This page displays all products with options to **View, Edit, and Delete**.

Create **resources/js/Pages/Products/Index.vue**:

```
<script setup>
import { Link, usePage, Head } from '@inertiajs/vue3';
import { ref, onMounted } from 'vue';
import AppLayout from '@Layouts/AppLayout.vue';

defineProps({
  products: Object,
  canLogin: Boolean,
  canRegister: Boolean,
  laravelVersion: String,
  phpVersion: String
});

const page = usePage();
const showMessage = ref(!page.props.flash?.success); // ✅ Initialize if flash message exists

// ✅ Auto-hide flash message after 3 seconds
onMounted(() => {
  if (showMessage.value) {
    setTimeout(() => {
      showMessage.value = false;
    }, 3000);
  }
});
</script>
<template>
  <app-layout :canLogin="canLogin" :canRegister="canRegister">

    <Head title="Products" />

    <div class="max-w-4xl mx-auto mt-8 p-4">
      <!-- ✅ Flash message with auto-hide -->
      <transition name="fade">
        <div v-if="showMessage"
          class="mb-4 p-4 bg-green-100 text-green-700 rounded dark:bg-green-900
dark:text-green-300">
          {{ page.props.flash.success }}
        </div>
      </transition>

      <h1 class="text-2xl font-bold mb-4">Product List</h1>

      <Link :href="route('product.create')" class="bg-blue-500 text-white px-4 py-2
```

```

rounded">
  Create Product
</Link>

<ul class="mt-4">
  <li v-for="product in products.data" :key="product.id"
    class="flex justify-between items-center bg-gray-100 p-3 rounded mt-2">

    <div class="flex items-center space-x-4">
      
      <span>{{ product.name }} - ${{ product.price }}</span>
    </div>

    <div>
      <Link :href="route('product.show', product.id)" class="text-green-600
mr-3">View</Link>
      <Link :href="route('product.edit', product.id)" class="text-blue-600
mr-3">Edit</Link>
      <button @click="$inertia.delete(route('product.destroy', product.id))"
        class="text-red-600">Delete</button>
    </div>
  </li>
</ul>
</div>
</app-layout>
</template>

<style scoped>
/* ✅ Fade transition effect */
.fade-enter-active,
.fade-leave-active {
  transition: opacity 0.5s;
}

.fade-enter-from,
.fade-leave-to {
  opacity: 0;
}
</style>

```

Step 2: Create Product Page (Create.vue)

This page allows users to add a new product.

Create **resources/js/Pages/Products/Create.vue**:


```

<script setup>
import AppLayout from '@/Layouts/AppLayout.vue';
import { useForm, Head } from '@inertiajs/vue3';

defineProps({
  canLogin: Boolean,
  canRegister: Boolean,
  laravelVersion: String,
  phpVersion: String
});

const form = useForm({
  name: "",
  description: "",
  price: "",
  stock: "",
  image: null
});

const submit = () => {
  form.post(route('product.store'));
};
</script>

<template>
  <app-layout :canLogin="canLogin" :canRegister="canRegister">

    <Head title="Product|Create" />
    <div class="max-w-lg mx-auto mt-8 p-4">
      <h1 class="text-2xl font-bold">Create Product</h1>

      <form @submit.prevent="submit" class="mt-4 space-y-3">
        <input v-model="form.name" placeholder="Name" class="border p-2 w-full">
        <p v-if="form.errors.name" class="text-red-500">{{ form.errors.name }}</p>

        <textarea v-model="form.description" placeholder="Description" class="border p-2
w-full"></textarea>
        <p v-if="form.errors.description" class="text-red-500">{{ form.errors.description
}}</p>

        <input v-model="form.price" type="number" placeholder="Price" class="border p-2
w-full">
        <p v-if="form.errors.price" class="text-red-500">{{ form.errors.price }}</p>

        <input v-model="form.stock" type="number" placeholder="Stock" class="border p-2
w-full">
        <p v-if="form.errors.stock" class="text-red-500">{{ form.errors.stock }}</p>

```

```

        <input type="file" @change="event => form.image = event.target.files[0]"
class="border p-2 w-full">
        <p v-if="form.errors.image" class="text-red-500">{{ form.errors.image }}</p>

        <button type="submit" :disabled="form.processing"
        class="bg-blue-500 text-white px-4 py-2 rounded">Save</button>
    </form>
</div>
</app-layout>
</template>

```

Step 3: Edit Product Page (Edit.vue)

This page allows users to update product details.

Create **resources/js/Pages/Products/Edit.vue**:

```

<script setup>
import AppLayout from '@/Layouts/AppLayout.vue';
import { useForm, Head } from '@inertiajs/vue3';

const props = defineProps({
  product: Object, canLogin: Boolean,
  canRegister: Boolean,
  laravelVersion: String,
  phpVersion: String
});

const form = useForm({
  name: props.product?.name || '',
  description: props.product?.description || '',
  price: props.product?.price || 0,
  stock: props.product?.stock || 0,
  image: null
});

const submit = () => {

```

```

    form.post(route('products.update', props.product.id));
  };
</script>

<template>
  <app-layout :canLogin="canLogin" :canRegister="canRegister">

    <Head title="Product|Edit" />
    <div class="max-w-lg mx-auto mt-8 p-4">
      <h1 class="text-2xl font-bold">Edit Product</h1>

      <form @submit.prevent="submit" class="mt-4 space-y-3">
        <div>
          <input v-model="form.name" placeholder="Name" class="border p-2 w-full">
          <p v-if="form.errors.name" class="text-red-500">{{ form.errors.name }}</p>
        </div>

        <div>
          <textarea v-model="form.description" placeholder="Description" class="border
p-2 w-full"></textarea>
          <p v-if="form.errors.description" class="text-red-500">{{ form.errors.description
}}</p>
        </div>

        <div>
          <input v-model="form.price" type="number" placeholder="Price" class="border
p-2 w-full">
          <p v-if="form.errors.price" class="text-red-500">{{ form.errors.price }}</p>
        </div>

        <div>
          <input v-model="form.stock" type="number" placeholder="Stock" class="border
p-2 w-full">
          <p v-if="form.errors.stock" class="text-red-500">{{ form.errors.stock }}</p>
        </div>

        <div>
          <input type="file" @change="event => form.image = event.target.files[0]"
class="border p-2 w-full">
          <p v-if="form.errors.image" class="text-red-500">{{ form.errors.image }}</p>
        </div>

        <button type="submit" :disabled="form.processing"
class="bg-blue-500 text-white px-4 py-2 rounded">Update</button>
      </form>
    </div>
  </app-layout>
</template>

```

Step 4: Single Product View Page (Show.vue)

This page displays the details of a single product.

Create **resources/js/Pages/Products/Show.vue**:

```
<script setup>
import { defineProps } from 'vue';
import { Link, Head } from '@inertiajs/vue3';
import AppLayout from '@Layouts/AppLayout.vue';

defineProps({
  product: Object,
  canLogin: Boolean,
  canRegister: Boolean,
  laravelVersion: String,
  phpVersion: String
});

</script>
<template>
  <app-layout :canLogin="canLogin" :canRegister="canRegister">

    <Head title="Product|Show" />
    <div class="max-w-lg mx-auto mt-8 p-4">
      <h1 class="text-2xl font-bold">{{ product.name }}</h1>
      
      <p class="mt-4 text-gray-700">{{ product.description }}</p>
      <p class="text-lg font-semibold">Price: ${{ product.price }}</p>
      <p class="text-lg font-semibold">Stock: {{ product.stock }}</p>
      <Link :href="route('product.index')" class="text-blue-500 mt-4 inline-block">Back to
Products</Link>
    </div>
  </app-layout>
</template>
```

✓ Summary of the Modules

- **Step 1: Product List Page (`Index.vue`)** → Displays all products.
- **Step 2: Create Product Page (`Create.vue`)** → Form to add a new product.
- **Step 3: Edit Product Page (`Edit.vue`)** → Update an existing product.
- **Step 4: Single Product View (`Show.vue`)** → View details of a single product.

Now, your **full frontend CRUD system** is ready! 🎉🚀 Let me know if you need further modifications.