Routing

Client-Side vs. Server-Side Routing

Routing on the server side means the server is sending a response based on the URL path that the user is visiting. When we click on a link in a traditional server-rendered web app, the browser receives an HTML response from the server and reloads the entire page with the new HTML.

In a Single-Page Application (SPA), however, the client-side JavaScript can intercept the navigation, dynamically fetch new data, and update the current page without full page reloads. This typically results in a more snappy user experience, especially for use cases that are more like actual "applications", where the user is expected to perform many interactions over a long period of time.

In such SPAs, the "routing" is done on the client side, in the browser. A client-side router is responsible for managing the application's rendered view using browser APIs such as History API or the `hashchange` event.

Official Router

Watch a Free Video Course on Vue School

Vue is well-suited for building SPAs. For most SPAs, it's recommended to use the officially-supported Vue Router library. For more details, see Vue Router's documentation.

Simple Routing from Scratch

listening to browser `hashchange` events or using the History API.

Here's a bare-bone example:

```vue
<script setup>
import { ref, computed } from 'vue'
import Home from './Home.vue'
import About from './About.vue'
import NotFound from './NotFound.vue'

const routes = {
  '/': Home,
  '/about': About
}

const currentPath = ref(window.location.hash)

window.addEventListener('hashchange', () => {
  currentPath.value = window.location.hash
})

const currentView = computed(() => {
  return routes[currentPath.value.slice(1) || '/'] || NotFound
})
</script>

<template>
  <a href="#/">Home</a> |
  <a href="#/about">About</a> |
  <a href="#/non-existent-path">Broken Link</a>
  <component :is="currentView" />
</template>
```

Try it in the Playground

Edit this page on GitHub

Previous
Tooling

Next
State Management