Comprehensive Guide to Vue Router: Creating Multi-Page Applications with Routing, Navigation, and Error Handling

Introduction

In modern web development, creating dynamic, multi-page applications (MPAs) that feel seamless and fast is essential. Vue.js, a progressive JavaScript framework, offers powerful tools to build such applications efficiently. While initial projects often involve a single page with multiple nested components, real-world applications require multiple pages or routes—such as About, Contact, Blogs, and Jobs. This guide explores how Vue handles multi-page routing, the setup of Vue Router, dynamic route parameters, navigation techniques, redirects, and error handling, all structured to give you a deep understanding of building robust Vue applications.

Center: Deep Dive into Vue Routing

1. Single Page Applications (SPAs) vs. Multi-Page Applications (MPAs)

- **SPAs**: Load a single HTML page; Vue takes control of routing and component switching without reloading the page.
- **MPAs**: Multiple distinct pages, each with its own URL, often requiring server requests for each page.

Vue's Approach:
Vue is inherently a single-page application framework, but with Vue

Router, it can simulate multi-page navigation by dynamically switching components based on the URL, without reloading the page.

---

2. Understanding Vue Routing

Concept	Explanation
Routing	The process of mapping URLs to components.
Vue Router	Official routing library for Vue.js, enabling navigation between components via URLs.
History Mode	Uses HTML5 History API for clean URLs (e.g., `/about`) instead of hash-based URLs (e.g., `#/about`).

How Vue Router Works:
- When a user visits `mysite.com/about`, Vue Router intercepts the request. - It dynamically loads the About component into the `<router-view>` placeholder. - Navigation is handled client-side, avoiding full page reloads.

---

3. Setting Up Vue Router

Step-by-step process:

- **Generate a new Vue project with Vue CLI**:

```
vue create ninja-jobs
```

    - Select features, including Router.
    - Choose History Mode for cleaner URLs.

- **Folder Structure**:

```
src/
├── components/
```

```
├── views/
│   ├── Home.vue
│   ├── About.vue
│   ├── NotFound.vue
│   └── Jobs/
│       ├── Jobs.vue
│       └── JobDetails.vue
├── router/
│   └── index.js
└── App.vue
```

- **Router Configuration (`src/router/index.js`)**:

```
import { createRouter, createWebHistory } from 'vue-router'
import Home from '../views/Home.vue'
import About from '../views/About.vue'
import NotFound from '../views/NotFound.vue'
import Jobs from '../views/Jobs/Jobs.vue'
import JobDetails from '../views/Jobs/JobDetails.vue'

const routes = [
  { path: '/', name: 'Home', component: Home },
  { path: '/about', name: 'About', component: About },
  { path: '/jobs', name: 'Jobs', component: Jobs },
  { path: '/jobs/:id', name: 'JobDetails', component: JobDetails, props
  // Catch-all route for 404
  { path: '/:catchAll(.*)', name: 'NotFound', component: NotFound }
]

const router = createRouter({
  history: createWebHistory(),
  routes
})
```

```
export default router
```

Key Points: - Use `createWebHistory()` for clean URLs. - Define routes with `path`, `name`, and `component`. - Use `props: true` to pass route params as component props. - Add a catch-all route for 404 pages.

---

4. Using `<router-view>` and `<router-link>`

- `<router-view>`: Placeholder where matched components are injected.
- `<router-link>`: Navigational link that intercepts clicks and updates the URL without reloading.

Example (`App.vue`):

```
<template>
  <div>
    <nav>
      <router-link to="/">Home</router-link>
      <router-link to="/about">About</router-link>
      <router-link :to="{ name: 'Jobs' }">Jobs</router-link>
    </nav>
    <router-view></router-view>
  </div>
</template>
```

Benefits: - `<router-link>` prevents full page reloads. - Active links automatically get classes like `router-link-active`, which can be styled.

---

5. Dynamic Routes and Route Parameters

- **Purpose**: Show details for specific items, e.g., a job with ID `123`.

- **Setup**:

```
{ path: '/jobs/:id', name: 'JobDetails', component: JobDetails, props:
```

- **Accessing Route Params**:

```
<script>
export default {
  props: ['id'],
  mounted() {
    console.log('Job ID:', this.id)
  }
}
</script>
```

- **In the component**:

```
<template>
  <div>
    <h1>Job Details for ID: {{ id }}</h1>
  </div>
</template>
```

- **Passing Params via `<router-link>`**:

```
<router-link :to="{ name: 'JobDetails', params: { id: job.id } }">
  {{ job.title }}
</router-link>
```

---

6. Programmatic Navigation and History Control

- **Navigate programmatically**:

```
this.$router.push({ name: 'Home' }) // Redirects to Home
```

- **Go back/forward in history**:

```
this.$router.go(-1) // Back one page
this.$router.go(1)  // Forward one page
```

- **Replace current route**:

```
this.$router.replace({ name: 'About' }) // Replaces current history ent
```

---

7. Redirects and Route Guards

- **Redirects**:

```
{ path: '/old-route', redirect: '/new-route' }
```

- **Example**: Redirect `/all-jobs` to `/jobs`.

- **404 Catch-All Route**:

```
{ path: '/:catchAll(.*)', component: NotFound }
```

- **Custom 404 Page**:

```
<template>
  <div>
    <h2>404</h2>
    <h3>Page Not Found</h3>
  </div>
</template>
```

---

8. Active Link Styling

- Vue automatically adds classes like `router-link-active`.
- Customize styles:

```
.router-link-active {
  color: white;
  background-color: crimson;
  padding: 10px;
  border-radius: 4px;
}
```

- Use `exact-active` for exact matches.

---

9. Advanced: Lazy Loading Routes

- Improve performance by loading components only when needed:

```
{
  path: '/about',
  component: () => import('../views/About.vue')
}
```

Outro

Vue Router transforms Vue.js from a simple component-based framework into a powerful tool for building **multi-page, dynamic web applications**. By understanding how to set up routes, handle parameters, navigate programmatically, implement redirects, and manage errors with 404 pages, you can craft seamless user experiences that feel like traditional multi-page sites but with the speed and fluidity of a single-page application.

Remember: - Use `<router-link>` for navigation to leverage Vue's client-side routing. - Define route parameters for dynamic content. - Implement catch-all routes for error handling. - Control navigation programmatically with `$router`. - Organize your folder structure for scalability.

With these tools, you're well-equipped to develop sophisticated Vue applications that are fast, user-friendly, and maintainable.

Happy coding! Your Vue routing mastery is just beginning.