

XÂY DỰNG GIAO DIỆN TƯƠNG TÁC BACKEND

BÀI 8: VUE ROUTER VÀ AUTHENTICATION

- ◎ Kết thúc bài học này bạn có khả năng
 - Tìm hiểu khái niệm Router
 - Cài đặt Vue Router và những tính năng cơ bản
 - Kết hợp khái niệm Vue Router và Authentication



Phần I: Vue Router

- ❖ Router là gì?
- ❖ Giới thiệu Vue Router
- ❖ Cài đặt Vue Router
- ❖ Các tính năng cơ bản của Vue Router

Phần II: Authentication

- ❖ Router và Authentication

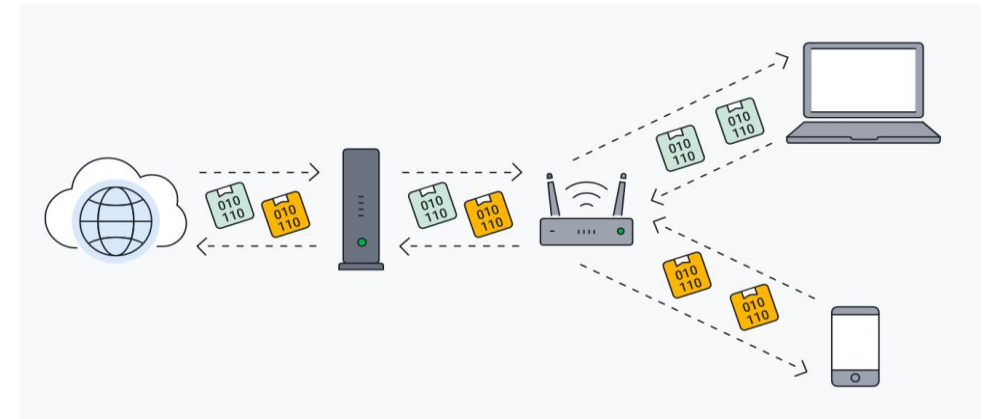


BÀI 8: **VUE ROUTER VÀ AUTHENTICATION**

PHẦN I: VUE ROUTER

Router là gì?

- ❑ Trong lĩnh vực phát triển web, khái niệm “router” dùng để chỉ một cơ chế hoặc công cụ được sử dụng nhằm xác định cách một ứng dụng (*application*) phản ứng (*respond*) lại một yêu cầu (*request*) từ client đến một điểm cuối xác định (*endpoint*) dựa trên URL.
- ❑ Các request được định tuyến (routes) đến các hàm (functions or actions) trong Controller thích hợp.
- ❑ Router đặc biệt cần thiết cho việc quản lý điều hướng, xử lý URLs trong ứng dụng web, đặc biệt là SPAs (single-page applications) và các Frameworks phía server.



Vue Router

- ☐ Vue Router là thư viện router chính thức dùng cho Vue.js
- ☐ Vue Router được tích hợp sâu với Vue.js core để xây dựng SPA (Single Page Application) một cách cực kỳ dễ dàng.

Các tính năng chính của Vue Router

- ☐ **Nested routes:** Cho phép tham chiếu components đến các đường dẫn lồng nhau trong ứng dụng.
- ☐ **Dynamic Routing:** Cho phép tạo routes có các phần giá trị động.
- ☐ **Programmatic Navigation:** Cho phép lập trình điều hướng trong ứng dụng sử dụng Javascript.
- ☐ **Route Guards:** là các hàm mà có thể được gắn vào trước khi đến một route hoặc sau khi rời một route (Hữu ích cho authentication, phân quyền).

Các tính năng chính của Vue Router (tiếp theo)

- ☐ **Lazy Loading:** Cho phép tách ứng dụng thành các khối nhỏ và chỉ tải các components khi cần thiết giúp cải thiện hiệu suất.
- ☐ **Named Routes:** Cho phép đặt tên cho routes.
- ☐ **Route Aliases:** Cung cấp phương án thay thế cho URLs, cho phép cùng component có thể được render với nhiều đường dẫn khác nhau.
- ☐ **Scroll Behavior:** Cho phép tùy biến hành vi cuộn trên ứng dụng khi điều hướng giữa các routes.
- ☐ **History Modes:** Hỗ trợ lịch các chế độ lịch sử để kiểm soát URLs

Cài đặt Vue Router

- ✓ **cd** vào thư mục dự án
- ✓ Chạy lệnh ở trên để cài đặt vue-router
- ✓ Nếu sử dụng Vue 2 thì cài đặt vue-router version 3.

```
bash
```

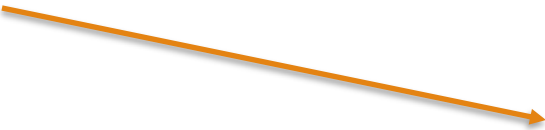
```
npm install vue-router@4
```

```
package.json ×
teaching > vue > my-vue > package.json > {} dependencies > vue-router
1  {
2    "name": "my-vue",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    "scripts": {
7      "dev": "vite",
8      "build": "vue-tsc -b && vite build",
9      "preview": "vite preview"
10   },
11   "dependencies": {
12     "vue": "^3.4.31",
13     "vue-router": "^4.4.3"
14   },
15   "devDependencies": {
16     "@vitejs/plugin-vue": "^5.0.5",
17     "typescript": "^5.2.2",
18     "vite": "^5.3.4",
19     "vue-tsc": "^2.0.24"
20   }
21 }
22
```

Cài đặt Vue Router

- ❑ Tiếp theo cần tạo file cấu hình router. Thông thường chúng ta sẽ tạo file index.js trong thư mục: src/router

- ❑ Ví dụ:



```
// src/router/index.js
import { createRouter, createWebHistory } from 'vue-router';
// import components for your routes
import Home from '../views/Home.vue';
import About from '../views/About.vue';

const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/about',
    name: 'About',
    component: About
  }
];

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
});

export default router;
```

Cài đặt Vue Router

- ❑ Thêm Router vào ứng dụng Vue bằng cách chỉnh sửa file src/main.js

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

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

Router Link & Router View

- ❑ Để điều hướng giữa các routes khác nhau, sử dụng component `<RouterLink>`
- ❑ Để hiển thị component tương ứng cho route hiện tại, sử dụng `<RouterView>`

```
<nav>
  <RouterLink to="/">Go to Home</RouterLink>
  <RouterLink to="/about">Go to About</RouterLink>
</nav>
<main>
  <RouterView />
</main>
```

Trong Vue.js bạn có thể sử dụng **PascalCase** hoặc **kebab-case**. Tức là việc viết `<RouterLink>` hay `<router-link>` đều hợp lệ. Tuy nhiên PascalCase được Vue SFC khuyến khích sử dụng.

Ví dụ:

- ☐ Trang web có menu gồm 2 menu items:
 - Go to Home
 - Go to About
- ☐ Tương ứng với mỗi menu item sẽ hiển thị nội dung bên dưới thanh menu.

Hello App!

Current route path: /

[Go to Home](#) [Go to About](#)

HomeView

[Go to About](#)

Các bước thực hiện ví dụ:

- ❑ Bước 1: Chỉnh sửa main.js để sử dụng router

```
1  import { createApp } from 'vue'
2  import router from './router'
3  import App from './app.vue'
4
5  createApp(App)
6    .use(router)
7    .mount('#app')
```

main.js

Các bước thực hiện ví dụ:

- ❑ Bước 2: Tạo 2 components là Home và About

```
1  <script>
2  export default {
3    methods: {
4      goToAbout() {
5        this.$router.push('/about')
6      },
7    },
8  }
9  </script>
10
11 <template>
12   <h2>HomeView</h2>
13   <button @click="goToAbout">Go to About</button>
14 </template>
```

HomeView.vue

```
1  <script setup>
2  import { computed } from 'vue'
3  import { useRoute, useRouter } from 'vue-router'
4
5  const router = useRouter()
6  const route = useRoute()
7
8  const search = computed({
9    get() {
10     return route.query.search ?? ''
11    },
12    set(search) {
13     router.replace({ query: { search } })
14    }
15  })
16 </script>
17
18 <template>
19   <h2>AboutView</h2>
20   <label>
21     Search: <input v-model.trim="search" maxlength="20">
22   </label>
23 </template>
```

AboutView.vue

Các bước thực hiện ví dụ:

- ❑ Bước 3: Tạo file router/index.js và cấu hình router

```
1 import { createMemoryHistory, createRouter } from 'vue-router'
2
3 import HomeView from './HomeView.vue'
4 import AboutView from './AboutView.vue'
5
6 const routes = [
7   { path: '/', component: HomeView },
8   { path: '/about', component: AboutView },
9 ]
10
11 const router = createRouter({
12   history: createMemoryHistory(),
13   routes,
14 })
15
16 export default router
```

router/index.js

- ❑ Bước 4: Sử dụng <RouterLink> để tạo liên kết đến component Home và About

```
1 <template>
2   <h1>Hello App!</h1>
3   <p>
4     <strong>Current route path:</strong> {{ $route.fullPath }}
5   </p>
6   <nav>
7     <RouterLink to="/">Go to Home</RouterLink>
8     <RouterLink to="/about">Go to About</RouterLink>
9   </nav>
10  <main>
11    <RouterView />
12  </main>
13 </template>
```

app.vue

demo

Tái hiện lại ví dụ đã thực hiện ở trên



Dynamic Routing

- ❑ Chúng ta thường cần sử dụng các routes có cùng hình mẫu (**pattern**) cho cùng component.
- ❑ Ví dụ: Hiển thị component: User dùng để hiển thị cho tất cả các user nhưng khác user-id.

```
import User from './User.vue'

// these are passed to `createRouter`
const routes = [
  // dynamic segments start with a colon
  { path: '/users/:id', component: User },
]
```

- Như vậy những URLs như **/users/johnny** hay **/users/tom** đều tham chiếu tới cùng một route.

Nested Routes

- ❑ Sử dụng route lồng nhau phù hợp với việc xây dựng hệ thống view phân cấp phức tạp.

```
const routes = [  
  {  
    path: '/user/:id',  
    component: User,  
    children: [  
      {  
        path: 'profile',  
        component: UserProfile  
      },  
      {  
        path: 'posts',  
        component: UserPosts  
      }  
    ]  
  }  
];
```

- **/user/tom/profile** => sẽ render UserProfile của Tom
- **/user/tom/posts** => sẽ render UserPosts - tất cả bài post của Tom

Programmatic Navigation

- ❑ Cho phép chuyển hướng trong ứng dụng sử dụng Javascript sử dụng đường dẫn (**route path**) hoặc truyền các đối tượng đối với việc điều hướng phức tạp hơn.
- ❑ Xem 2 ví dụ ở dưới đây:

```
javascript  
this.$router.push('/about');
```

```
javascript  
this.$router.push({ name: 'user', params: { id: 123 } });
```

Named Routes

- ❑ Cho phép đặt tên cho routes để dễ dàng hơn trong việc điều hướng trong lập trình.

```
const routes = [  
  {  
    path: '/user/:id',  
    name: 'user',  
    component: User  
  }  
];  
  
this.$router.push({ name: 'user', params: { id: 123 } });
```

Route Aliases

- ❑ Cho phép render cùng component đối với đường dẫn khác nhau.

```
const routes = [  
  {  
    path: '/profile',  
    alias: '/user/profile',  
    component: UserProfile  
  }  
];
```

Route Guards

- ❑ Là các functions có thể được gắn (hook) vào trước khi vào hoặc sau khi rời một route. Hữu dụng cho xác thực, phân quyền và lấy dữ liệu.

```
const router = createRouter({
  history: createWebHistory(),
  routes: [
    {
      path: '/dashboard',
      component: Dashboard,
      beforeEnter: (to, from, next) => {
        if (isAuthenticated()) {
          next();
        } else {
          next('/login');
        }
      }
    }
  ]
});
```

BÀI 8:
VUE ROUTER VÀ AUTHENTICATION

**PHẦN II: VUE ROUTER VÀ
AUTHENTICATION**

Authentication

- ❑ Xác thực (authentication) là một khía cạnh cực kì quan trọng đối với ứng dụng web.
- ❑ Để xử lý xác thực với Vue.js cần một số bước thiết lập bao gồm cài đặt các routes yêu cầu xác thực, kiểm soát việc login/logout và bảo vệ những tài nguyên của ứng dụng từ việc truy cập trái phép.



Cấu trúc cơ bản của xác thực thông qua Route

- ☐ beforeEach: kiểm tra xem user đã được xác thực hay chưa.
- ☐ Nếu chưa xác thực: chuyển hướng đến Login
- ☐ Nếu đã xác thực: cho phép di chuyển đến route đích.

```
// router/index.js
import { createRouter, createWebHistory } from 'vue-router';
import Home from '../views/Home.vue';
import Dashboard from '../views/Dashboard.vue';
import Login from '../views/Login.vue';

const routes = [
  { path: '/', name: 'Home', component: Home },
  { path: '/login', name: 'Login', component: Login },
  {
    path: '/dashboard',
    name: 'Dashboard',
    component: Dashboard,
    meta: { requiresAuth: true } // This route requires authentication
  }
];

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

// Navigation Guard
router.beforeEach((to, from, next) => {
  const isAuthenticated = false; // Replace with actual authentication check

  if (to.matched.some(record => record.meta.requiresAuth) && !isAuthenticated) {
    next({ name: 'Login' }); // Redirect to the login page if not authenticated
  } else {
    next(); // Proceed to the route
  }
});

export default router;
```

Cấu trúc cơ bản của xác thực thông qua Route

```
<!-- src/views/Login.vue -->
<template>
  <div>
    <h1>Login Page</h1>
    <button @click="login">Login</button>
  </div>
</template>

<script>
export default {
  name: 'Login',
  methods: {
    login() {
      // Implement login logic here
      // For example, set `isAuthenticated` to true and redirect to dashboard
      this.$router.push({ name: 'Dashboard' });
    }
  }
};
</script>
```

Login.vue

```
<!-- src/views/Home.vue -->
<template>
  <div>
    <h1>Home Page</h1>
    <router-link to="/dashboard">Go to Dashboard</router-link>
  </div>
</template>

<script>
export default {
  name: 'Home'
};
</script>
```

Home.vue

```
<!-- src/views/Dashboard.vue -->
<template>
  <div>
    <h1>Dashboard</h1>
    <p>Welcome to your dashboard. This area is protected and requires
  </p>
  </div>
</template>

<script>
export default {
  name: 'Dashboard'
};
</script>
```

Dashboard.vue

Khái niệm chính: Route Metadata

- ❑ Trong router/index.js, thuộc tính meta được sử dụng với giá trị requireAuth = true để xác định rằng route này yêu cầu phải được xác thực.

```
const routes = [  
  { path: '/', name: 'Home', component: Home },  
  { path: '/login', name: 'Login', component: Login },  
  {  
    path: '/dashboard',  
    name: 'Dashboard',  
    component: Dashboard,  
    meta: { requiresAuth: true } // This route requires authentication  
  },  
];
```

Khái niệm chính: Navigation Guard (beforeEach)

```
router.beforeEach((to, from, next) => {  
  const isAuthenticated = false; // Replace with actual authentication check  
  
  if (to.matched.some(record => record.meta.requiresAuth) && !isAuthenticated)  
    next({ name: 'Login' }); // Redirect to the login page if not authenticate  
  else {  
    next(); // Proceed to the route  
  }  
});
```

- ☐ router.beforeEach: chạy trước mỗi thay đổi về route.
- ☐ to.matched.some (record => record.meta.requiresAuth): kiểm tra nếu route đang được điều hướng yêu cầu xác thực.
- ☐ isAuthenticated: một giá trị kiểm tra xem user đã đăng nhập hay chưa.
- ☐ next({ name: 'Login' }): nếu user chưa được xác thực, chuyển hướng đến trang Login.
- ☐ next(): nếu user đã được xác thực hoặc route không yêu cầu xác thực, được phép tiếp tục.

Kiểm tra xác thực trong thực tế: isAuthenticated

- ❑ Ở ví dụ trên, `isAuthenticated = false`, nghĩa là user chưa được xác thực. Trong thực tế chúng ta cần thay thế bằng việc kiểm tra tình trạng xác thực như:
- Kiểm tra token: Tìm kiếm token hợp lệ có thể được lưu ở local storage hoặc cookies
 - Vuex Store: Sử dụng Vuex để lưu trữ toàn cục trạng thái xác thực

```
router.beforeEach((to, from, next) => {  
  const token = localStorage.getItem('authToken'); // Check if a token exists  
  
  if (to.matched.some(record => record.meta.requiresAuth) && !token) {  
    next({ name: 'Login' }); // Redirect to login if not authenticated  
  } else {  
    next(); // Proceed to the route  
  }  
});
```

- ☑ Vue Router là một công cụ mạnh mẽ tích hợp sâu với Vue.js, cung cấp nhiều tính năng giúp cho việc xây dựng điều hướng cho ứng dụng phức tạp một cách dễ dàng.
- ☑ Vue Router hỗ trợ route lồng nhau (nested routes), định tuyến động (dynamic routing), route guards, và nhiều tính năng khác cung cấp một công cụ đầy đủ để xây dựng ứng dụng hoàn chỉnh.
- ☑ Sử dụng Vue Router cho xác thực yêu cầu cài đặt các route cần xác thực, kiểm soát login, logout và sử dụng navigation guards để bảo vệ các thành phần nhạy cảm của ứng dụng.
- ☑ Sử dụng Vue Router cho xác thực có thể kết hợp với nhiều cơ chế xác thực phức tạp như OAuth, JWT và tích hợp với các dịch vụ phía backend.



Thank
You

