Vue.js : routage

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Vue.js

Récapitulatif

- À la création d'un nouveau composant, on ajoute sa balise dans le template de `App.vue` ou un de ses composants enfants
- Souvent, dans les application Web, on n'affiche que le·s composant·s demandé·s
- Une route demandée $\Rightarrow$ un composant affiché

Vue.js

Gestion de route

- Mapping : route/composant
- Une nouvelle balise pour les liens : pour ne pas recharger la page (aspect **SPA** de **Vue.js**)
- Une nouvelle balise pour spécifier l'emplacement dédié à l'affichage du composant demandé

Vue.js

Gestion de route

- Mapping : route/composant
- Une nouvelle balise pour les liens : pour ne pas recharger la page (aspect **SPA** de **Vue.js**)
- Une nouvelle balise pour spécifier l'emplacement dédié à l'affichage du composant demandé

Remarque

À la création du projet, **Vue.js** nous a proposé l'ajout du module du routage.

Vue.js

Pour ajouter le module du routage avec Vue-CLI

```
vue add router
```

Vue.js

Pour ajouter le module du routage avec Vue-CLI

```
vue add router
```

Constats

- Création d'un fichier `index.js` dans un dossier `router` : fichier contenant le mapping route/composant
- Modification du fichier `main.js` : utilisation du router
- Création d'un dossier `views` avec deux composants : `AboutView.vue` et `HomeView.vue`
- Modification du fichier `App.vue` : remplacement du code précédent avec un menu avec deux éléments pointant vers `AboutView.vue` et `HomeView.vue`

Vue.js

Contenu de `main.js`

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

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

import "bootstrap/dist/css/bootstrap.min.css"
import "bootstrap/dist/js/bootstrap.bundle.min.js"
import "@fortawesome/fontawesome-free/css/all.css"
import "bootstrap-icons/font/bootstrap-icons.css"
import "./assets/css/style.css"
```

Vue.js

Contenu d'`index.js`

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'

const routes = [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/about',
    name: 'about',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import(/* webpackChunkName: "about" */ '../views/AboutView.vue')
  }
]

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

export default router
```

Vue.js

Explications

- `path` : chemin du composant.
- `name` : nom de la route à utiliser pour le débuggage et les redirections.
- `component` : composant à affiché pour le chemin spécifié.

Vue.js

Template de `App.vue`

```
<template>
  <nav>
    <router-link to="/">Home</router-link> |
    <router-link to="/about">About</router-link>
  </nav>
  <router-view/>
</template>
```

Vue.js

Template de `App.vue`

```
<template>
  <nav>
    <router-link to="/">Home</router-link> |
    <router-link to="/about">About</router-link>
  </nav>
  <router-view/>
</template>
```


Explications

- `<router-link>` : similaire à la balise `a` en **HTML** mais ne recharge pas la page.
- `<router-view/>` : indique l'emplacement d'affichage du composant associé à la route demandée.

Vue.js

Question

Quelle est la différence entre `views` et `components` ?

Vue.js

Question

Quelle est la différence entre `views` et `components` ?

Réponse

Vue.js recommande d'utiliser

- `views` pour les composants associés à au moins une route (utilisés par **vue-router**)
- `components` pour les composants enfants des composants définis dans `views`

Vue.js

Remarque

Pour récupérer le contenu précédent de `App.vue` :

- utilisez `Ctrl` + `z` et `Ctrl` + `y` pour déplacer le contenu précédent dans `AboutView.vue`.
- gardez le contenu actuel (le menu) dans `App.vue`.

Vue.js

Deux formes de paramètres de requête

- `/chemin/param1/param2`
- `/chemin?var1=value1&var2=value2`

Vue.js

Deux formes de paramètres de requête

- `/chemin/param1/param2`
- `/chemin?var1=value1&var2=value2`

Deux objets pour la récupération de ces paramètres

- `$route.params` pour `/chemin/param1/param2`
- `$route.query` pour `/chemin?var1=value1&var2=value2`

Vue.js

Pour la suite

Créons les trois composants suivants dans `views`

- `PersonneShowView`
- `PersonneDetailsView`
- `AdresseView`

Vue.js

Commençons par définir une route pour le composant `PersonneDetailsView` dans le tableau `routes` de `index.js`

```
{
    path: '/personne/:id',
    name: 'personne-details',
    component: PersonneDetailsView
}
```

Vue.js

Commençons par définir une route pour le composant `PersonneDetailsView` dans le tableau `routes` de `index.js`

```
{
    path: '/personne/:id',
    name: 'personne-details',
    component: PersonneDetailsView
}
```


Explication

On utilise le préfixe `:` pour `id` pour le définir comme paramètre (et non pas comme un path)

Commençons par déclarer le tableau `personnes` dans `PersonneDetailsView.vue`

```
<template>
</template>

<script>

export default {
    name: 'PersonneDetailsView',
    data() {
        return {
            personnes: [
                { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
                { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
                { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
            ]
        }
    }
}
</script>
```

Commençons par déclarer le tableau `personnes` dans `PersonneDetailsView.vue`

```
<template>
</template>

<script>

export default {
    name: 'PersonneDetailsView',
    data() {
        return {
            personnes: [
                { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
                { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
                { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
            ]
        }
    }
}
</script>
```

Objectif

Récupérer l'`id` de la barre d'adresse et afficher la personne correspondante dans `template`.

Vue.js

Pour récupérer les paramètres d'une route de la forme `personne/:id/` dans le `template`

```
<template>
    <h1>Détails de la personne {{ $route.params.id }}</h1>
</template>
<script>

export default {
    name: 'PersonneDetailsView',
    data() {
        return {
            personnes: [
                { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
                { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
                { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
            ]
        }
    }
}
</script>
```

Pour récupérer les paramètres d'une route de la forme `personne/:id/` dans le `script`

```
<template>
    <h1>Détails de la personne {{ id }}</h1>
</template>
<script>

export default {
    name: 'PersonneDetailsView',
    data() {
        return {
            personnes: [
                { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
                { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
                { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
            ]
        }
    },
    computed: {
        id() {
            return parseInt(this.$route.params.id)
        },
    }
}
</script>
```

Vue.js

Pour afficher les détails de la personne dont l'`id` est passé en paramètre

```
<template>
    <h1>Détails de la personne {{ id }}</h1>
    <p>Personne recherchée : {{ personne }}</p>
</template>
<script>

export default {
    name: 'PersonneDetailsView',
    data() {
        return {
            personnes: [
                { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
                { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
                { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
            ]
        }
    },
    computed: {
        id() {
            return parseInt(this.$route.params.id)
        },
        personne() {
            return this.personnes.find(elt => elt.id == this.id)
        }
    }
}
</script>
```

Vue.js

Nous pouvons utiliser une expression régulière pour indiquer que le paramètre `id` accepte uniquement les nombres

```
{
    path: '/personne/:id(\\d)',
    name: 'personne-details',
    component: HomeView
}
```

Vue.js

Commençons par définir une route pour le composant `AdresseView` **dans le tableau** `routes` **de** `index.js`

```
{
    path: '/adresse',
    name: 'adresse',
    component: AdresseView
},
```

Vue.js

Commençons par définir une route pour le composant `AdresseView` dans le tableau `routes` de `index.js`

```
{
    path: '/adresse',
    name: 'adresse',
    component: AdresseView
},
```


Remarque

Pas besoin d'inclure les paramètres dans la définition de la route

Vue.js

Contenu d'`AdresseView.vue`

```
<template>
    <h1>Adresse</h1>
    <ul>
        <li>Rue : {{ $route.query.rue }}</li>
        <li>Code postal : {{ $route.query.codePostal }}</li>
        <li>Ville : {{ $route.query.ville }}</li>
    </ul>
</template>

<script>
export default {
    name: 'AdresseView',
}
</script>
```

Vue.js

Pour récupérer les paramètres dans `script`

```
<template>
    <h1>Adresse</h1>
    <ul>
        <li>Rue : {{ adresse.rue }}</li>
        <li>Code postal : {{ adresse.codePostal }}</li>
        <li>Ville : {{ adresse.ville }}</li>
    </ul>
</template>

<script>
export default {
    name: 'AdresseView',
    computed: {
        adresse() {
            return this.$route.query
        }
    }
}
</script>
```

Vue.js

Exercice

- Considérons un composant `CalculView` (à créer)
- Ce composant est accessible via le chemin `calcul/:op`
- Les valeurs possibles de `op` sont `plus`, `moins`, `fois` et `div`
- Si l'adresse saisie dans la barre d'adresse contient `/calcul/plus?value1=2&value2=5`, la réponse attendue dans le template est `2 + 5 = 7`

Vue.js

Dans `script` de `PersonneDetails.vue`, supprimons `id` de `computed` et déclarons le comme `props`

```
<script>
export default {
    name: 'PersonneDetailsView',
    props: ['id'],
    data() {
        return {
            personnes: [
                { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
                { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
                { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
            ]
        }
    },
    computed: {
        personne() {
            return this.personnes.find(elt => elt.id == this.id)
        }
    }
}
</script>
```

Vue.js

Dans `index.js`, activons les `props` pour la route nommée `personne-details`

```
{
    path: '/personne/:id(\\d)',
    name: 'personne-details',
    props: true,
    component: PersonneDetailsView
},
```

Vue.js

Dans `index.js`, activons les `props` pour la route nommée `personne-details`

```
{
    path: '/personne/:id(\\d)',
    name: 'personne-details',
    props: true,
    component: PersonneDetailsView
},
```

Testez de nouveau la route `/personne/2` et vérifiez que la personne correspondante s'affiche toujours correctement.

Vue.js

Commençons par définir une route pour le composant `PersonneShowView` dans le tableau `routes` de `index.js`

```
{
    path: '/personne',
    name: 'personne-show',
    component: PersonneShowView
}
```

Vue.js

Commençons par définir une route pour le composant `PersonneShowView` dans le tableau `routes` de `index.js`

```
{
    path: '/personne',
    name: 'personne-show',
    component: PersonneShowView
}
```

Dans `template` de `App.vue`, ajoutons un lien vers `PersonneShowView`

```
<template>
  <nav>
    <router-link to="/">Home</router-link> |
    <router-link to="/about">About</router-link> |
    <router-link to="/personne">Personne</router-link>
  </nav>
  <router-view />
</template>
```

Vue.js

Dans `PersonneShowView.vue`**, définissons un lien avec paramètre vers** `PersonneDetailsView`

```
<template>
    <h1>Gestion de personnes</h1>
    <ul>
        <li v-for="(elt, index) in personnes" :key="index">
            <router-link :to="{
                name: 'personne-details',
                params: { id: elt.id } }
            ">
                {{ elt.nom }} {{ elt.prenom }}
            </router-link>
        </li>
    </ul>
</template>

<script>
export default {
    name: 'PersonneShowView',
    data() {
        return {
            personnes: [
                { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
                { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
                { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
            ]
        }
    }
}
</script>
```

Vue.js

Dans `template` **de** `AboutView`**, construisons un lien avec paramètres vers** `AdresseView`

```
<router-link
:to="{
  name: 'adresse',
  query: { rue: 'Paradis', ville: 'Marseille', codePostal: 13006 } }">
    Cliquez pour visiter Marseille...
</router-link>
```

Vue.js

Exercice

- Créez un nouveau composant `TableauView`
- Déclarez un tableau `numbers: [2, 3, 8, 1]` dans la fonction `data`
- Définissez une route `tableau/:id` / `id` étant l'indice de l'élément dans `numbers` à afficher dans le template
- Ajoutez deux liens `Suivant` et `Précédent` qui permettent de naviguer respectivement sur l'élément suivant et précédent de `numbers`
- Les deux liens `Suivant` et `Précédent` doivent permettre une navigation circulaire

Vue.js

Ajoutons un lien vers `AdresseView` **dans le** `template` **d'**`AdresseView`

```
<template>
    <h1>Adresse</h1>
    <ul>
        <li>Rue : {{ adresse.rue }}</li>
        <li>Code postal : {{ adresse.codePostal }}</li>
        <li>Ville : {{ adresse.ville }}</li>
    </ul>
    <router-link :to="{ name: 'adresse', query: { rue: 'Montparnasse',
      ville: 'Paris', codePostal: 75014 } }">
        Cliquez pour visiter Paris...
    </router-link>
</template>
```

Vue.js

Question

Le composant sera t-il recréé en cliquant sur un lien qui renvoie sur le même composant ?

Vue.js

Question

Le composant sera t-il recréé en cliquant sur un lien qui renvoie sur le même composant ?

Réponse

Non, **Vue.js** ne détruit pas l'instance pour recréer une nouvelle, il utilise l'ancienne.

Pour le vérifier, ajoutons le hook `created`

```
<script>
export default {
    name: 'AdresseView',
    computed: {
        adresse() {
            return this.$route.query
        }
    },
    created() {
        console.log('created adresse')
    },
}
</script>
```


Démarche

- Dans la barre d'adresse, saisissez `http://localhost:8080/adresse?rue=Paradis&ville=Marseille&codePostal=13006`.
- Lorsque le composant `AdresseView` s'affiche, cliquez sur le lien (`Cliquez pour visiter Paris...`).
- Vérifiez dans la console du navigateur que `created adresse` a été affiché une seule fois.

Vue.js

Pour réagir au changement de valeurs des paramètres

```
<script>
export default {
    name: 'AdresseView',
    computed: {
        adresse() {
            return this.$route.query
        }
    },
    created() {
        console.log('created adresse')
        this.$watch(
            () => this.$route.query,
            (next, previous) => {
                console.log(next, previous)
            }
        )
    }
}
</script>
```

Vue.js

Objectif

- Déplacer le menu dans un nouveau composant `MenuNav.vue` (à créer dans `components`)
- Déclarer `MenuNav` comme composant enfant de `App`

Vue.js

Créons le composant `MenuNav.vue` dans `components` avec le contenu suivant

```
<template>
    <nav>
        <router-link to="/">Home</router-link> |
        <router-link to="/about">About</router-link> |
        <router-link to="/personne">Personne</router-link>
    </nav>
</template>

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

Vue.js

Nouveau contenu d'`App.vue`

```
<template>
  <MenuNav></MenuNav>
  <router-view />
</template>

<script>
import MenuNav from './components/MenuNav.vue'

export default {
  name: 'App',
  components: {
    MenuNav
  }
}
</script>

<style>
  /* on garde les styles précédents */
</style>
```

Vue.js

Objectif

Afficher en gras l'élément actif du menu.

Vue.js

Dans `index.js`, spécifions le nom de classe à utiliser pour les liens actifs (ici `lien-actif`)

```
// imports + tableau de routes

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
  linkActiveClass: 'lien-actif'
})
```

Vue.js

Dans `style` de `MenuNav`, personnalisons le style de la classe `lien-actif`

```
<template>
    <nav>
        <router-link to="/">Home</router-link> |
        <router-link to="/about">About</router-link> |
        <router-link to="/personne">Personne</router-link>
    </nav>
</template>

<script>
export default {
    name: 'MenuNav'
}
</script>

<style scoped>
.lien-actif {
    font-style: italic;
}
</style>
```

Vue.js

Ajoutons le bouton suivant dans le `template` de `AboutView`

```
<button @click="gotoPersonne()">
  Consultez la fiche de Sophie
</button>
```

Vue.js

Ajoutons le bouton suivant dans le `template` de `AboutView`

```
<button @click="gotoPersonne()">
  Consultez la fiche de Sophie
</button>
```

Objectif

- En cliquant sur le bouton, la méthode `gotoPersonne()` (à définir dans `<script>` de `AboutView`) sera exécutée.
- La méthode `gotoPersonne()` doit nous rediriger vers le composant `PersonneDetailsView`.

Vue.js

Pour rediriger vers le composant `PersonneDetailsView`, on utilise la méthode `push`

```
gotoPersonne() {
    this.$router.push('/personne/3')
}
```

Vue.js

Pour rediriger vers le composant `PersonneDetailsView`, on utilise la méthode `push`

```
gotoPersonne() {
    this.$router.push('/personne/3')
}
```

On utilise aussi passer comme paramètre un objet spécifiant le `path`

```
gotoPersonne() {
    this.$router.push({ path: '/personne/3' })
}
```

Vue.js

Pour rediriger vers le composant `PersonneDetailsView`, on utilise la méthode `push`

```
gotoPersonne() {
    this.$router.push('/personne/3')
}
```

On utilise aussi passer comme paramètre un objet spécifiant le `path`

```
gotoPersonne() {
    this.$router.push({ path: '/personne/3' })
}
```

On peut aussi passer un objet spécifiant le `path` et les `params`

```
gotoPersonne() {
    this.$router.push({ path: '/personne', params: { id: 3 } })
}
```

Vue.js

On peut aussi utiliser le nom d'une route

```
gotoPersonne() {
    this.$router.push({ name: 'personne-details', params: { id: 3 } })
}
```

Vue.js

Exercice (à refaire avec la redirection)

- Reprenez le composant `TableauView`
- Ajoutez deux boutons `Suivant` et `Précédent` qui permettent de naviguer respectivement sur l'élément suivant et précédent de `numbers`
- Les deux boutons `Suivant` et `Précédent` doivent permettre une navigation circulaire

Vue.js

On peut rediriger vers un chemin existant

```
{
    path: '/person',
    redirect: '/personne'
},
```

Vue.js

On peut rediriger vers un chemin existant

```
{
    path: '/person',
    redirect: '/personne'
},
```

Explication

En demandant la route `/person` :

- `person` sera remplacé par `personne` dans la barre d'adresse,
- Le composant `PersonneShowView` sera affiché.

Vue.js

On peut aussi utiliser une route nommée

```
{
    path: '/person',
    redirect: { name: 'personne-show' }
},
```

Vue.js

On peut aussi utiliser une fonction pour une redirection dynamique

```
{
    path: '/adresse/:rue/:ville/:codePostal',
    redirect: to => {
      return { path: '/adresse', query: { ...to.params } }
    },
},
```

Vue.js

On peut aussi utiliser une fonction pour une redirection dynamique

```
{
    path: '/adresse/:rue/:ville/:codePostal',
    redirect: to => {
      return { path: '/adresse', query: { ...to.params } }
    },
},
```


Explication

- En demandant la route `/adresse/Paradis/Marseille/13006`, cette dernière sera remplacée par `/adresse?rue=Paradis&ville=Marseille&codePostal=13006`.
- Le composant `AdresseView` sera affiché.

Vue.js

`redirect` VS `alias`

- `redirect` : remplace le chemin demandé par un autre auquel un composant est associé.
- `alias` : permet d'associer plusieurs chemin à un même composant sans que l'un remplace l'autre dans la barre d'adresse.

Vue.js

Exemple avec deux alias

```
{
    path: '/',
    alias: ['/home', '/accueil'],
    name: 'home',
    component: HomeView
},
```

Vue.js

Exemple avec deux alias

```
{
    path: '/',
    alias: ['/home', '/accueil'],
    name: 'home',
    component: HomeView
},
```


Explication

- `HomeView` est accessible via trois chemins : `/`, `/home` et `accueil`.
- Le chemin demandé reste affiché après chargement du composant `HomeView`.

Vue.js

Créons un composant `NotFoundView` avec le contenu suivant

```
<template>
    <h1>404 : Page non trouvée</h1>
</template>

<script>
export default {
    name: 'NotFoundView',
}
</script>

<style scoped>
h1 {
    color: red;
}
</style>
```

Vue.js

Redirigeons tous les chemins demandés non-définis vers `NotFoundView` (Vue.js 3)

```
{
    path: '/:pathMatch(.*)',
    name: 'NotFound',
    component: NotFoundView
},
```

Vue.js

Redirigeons tous les chemins demandés non-définis vers `NotFoundView` (Vue.js 3)

```
{
    path: '/:pathMatch(.*)',
    name: 'NotFound',
    component: NotFoundView
},
```


Ou aussi (Vue.js 2)

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

Pour récupérer le chemin introuvable demandé (remplacez `catchAll` par `patMatch` si vous l'aviez utilisé pour définir la route)

```
<template>
    <h1>404 : Page non trouvée ({{ $route.params.catchAll }})</h1>
</template>

<script>
export default {
    name: 'NotFoundView',
}
</script>

<style scoped>
h1 {
    color: red;
}
</style>
```

Pour récupérer le chemin introuvable demandé (remplacez `catchAll` par `patMatch` si vous l'aviez utilisé pour définir la route)

```
<template>
    <h1>404 : Page non trouvée ({{ $route.params.catchAll }})</h1>
</template>

<script>
export default {
    name: 'NotFoundView',
}
</script>

<style scoped>
h1 {
    color: red;
}
</style>
```

Allez à `http://localhost:8080/x/y` et vérifiez le rendu suivant :
404 : Page non trouvée (x/y)

Vue.js

Le dernier `*` est nécessaire, pour `catchAll` et `pathMatch`, si nous voulions récupérer les sous chemins dans un tableau

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

Vue.js

Le dernier `*` est nécessaire, pour `catchAll` et `pathMatch`, si nous voulions récupérer les sous chemins dans un tableau

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


Allez à `http://localhost:8080/x/y` et vérifiez le rendu suivant :
404 : Page non trouvée (["x", "y"])

Vue.js

Dans le `template` de `PersonneDetails`, ajoutons un bouton qui permet de revenir à la page précédente

```
<button @click="pagePrecedente()">Retour</button>
```

Vue.js

Dans le `template` de `PersonneDetails`, ajoutons un bouton qui permet de revenir à la page précédente

```
<button @click="pagePrecedente()">Retour</button>
```

Dans la partie `methods` de `PersonneDetails`, définissons la méthode `pagePrecedente()` qui permet de revenir à l'arrière dans l'historique de la navigation

```
pagePrecedente() {
    this.$router.back()
}
```

Vue.js

On peut aussi utiliser la méthode `go` pour avoir le même résultat

```
pagePrecedente() {
    this.$router.go(-1)
}
```

Vue.js

On peut aussi utiliser la méthode `go` pour avoir le même résultat

```
pagePrecedente() {
    this.$router.go(-1)
}
```


Remarque

La méthode `go()` permet aussi d'aller loin dans l'historique de navigation.

Vue.js

Extrait du contenu d'`index.js`

```
const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
  linkActiveClass: 'lien-actif'
})
```

Trois modes d'historique en **Vue.js**

- **HTML5 Mode** (par défaut et recommandé)
- **Hash Mode** : ajoute le caractère # avant l'URL actuelle. À utiliser pour charger toute l'application dans le navigateur sans passer par le serveur (Mauvais pour le référencement).
- **Memory mode** : ne permet pas la manipulation de l'historique (`go`, `back`, `forward`...)

Vue.js

Pour utiliser `Hash Mode`**, par exemple, il suffit de remplacer** `createWebHistory` **par** `createWebHashHistory` **dans** `index.js`

```
const router = createRouter({
  history: createWebHashHistory(process.env.BASE_URL),
  routes,
  linkActiveClass: 'lien-actif'
})
```

Vue.js

Pour utiliser `Hash Mode`**, par exemple, il suffit de remplacer** `createWebHistory` **par** `createWebHashHistory` **dans** `index.js`

```
const router = createRouter({
  history: createWebHashHistory(process.env.BASE_URL),
  routes,
  linkActiveClass: 'lien-actif'
})
```


Remarque

Toutes les URL commencent maintenant par `http://localhost:8080/#/`.

Vue.js

Lazy loading ou **On-demand loading**

- Chargement du composant à la demande (pas au démarrage de l'application)
- Utilise l'import avec les promesses (**ES2020**)

Vue.js

Dans `index.js`, par défaut, Vue.js a importé le composant `AboutView` en mode lazy loading

```
{
    path: '/about',
    name: 'about',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import(/* webpackChunkName: "about" */ '../views/AboutView.vue')
}
```

Vue.js

Dans `index.js`, par défaut, Vue.js a importé le composant `AboutView` en mode lazy loading

```
{
    path: '/about',
    name: 'about',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import(/* webpackChunkName: "about" */ '../views/AboutView.vue')
}
```


Remarque

- Allez dans le **DevTools** du navigateur, cliquez sur l'onglet `Réseau` et notez la taille d'`app.js` (`239 Ko` pour mon cas).
- Remplacez tous les imports statiques par des imports utilisant les promesses et refaites l'opération précédente (item 1) : vérifiez que la taille de `app.js` est plus petite (`150 Ko` pour mon cas).

Vue.js

Lazy loading : avantages

- L'exécution du code inutile est évitée.
- Meilleure performance en temps et en espace (mémoire).

Lazy loading : inconvénients

- Écriture un peu plus complexe.
- Possibilité d'affecter le référencement et le classement du site Web sur les moteurs de recherche, à cause d'une mauvaise indexation du contenu déchargé.

Vue.js

Rappel

La route `personne/1` permet d'afficher les informations concernant la personne ayant l'identifiant 1.

Vue.js

Rappel

La route `personne/1` permet d'afficher les informations concernant la personne ayant l'identifiant 1.

Hypothèse

Et si chaque personne avait une (ou plusieurs) adresse et un (ou plusieurs) sports qu'on souhaiterait les afficher sur demande (pas dans le composant `PersonneDetails`).

Vue.js

Idée

- Route `personne/:id` ⇒ les détails d'une personne (sans sport ni adresse).
- Route `personne/:id/adresse` ⇒ les détails d'une personne + son adresse (sans sport).
- Route `personne/:id/sport` ⇒ les détails d'une personne + son sport (sans adresse).

Vue.js

Démarche

- Créer deux composants : `PersonneSportView` et `PersonneAdresseView`.
- Définir les routes `/adresse` et `sport` dans `index.js` comme routes enfants de la route `personne/:id`.
- Utiliser `router-view` et `router-link` dans `PersonneDetailsView`.

Vue.js

Contenu de `PersonneAdresseView.vue`

```
<template>
    <h2>Adresse </h2>
    <ul>
        <li>Ville : {{ adresse.ville }}</li>
        <li>Code postal : {{ adresse.codePostal }}</li>
    </ul>
</template>

<script>
export default {
    name: 'PersonneAdresseView',
    data() {
        return {
            personnes: [
                { id: 1, adresse: { ville: 'Marseille', codePostal: '13000' } },
                { id: 2, adresse: { ville: 'Paris', codePostal: '75000' } },
                { id: 3, adresse: { ville: 'Lille', codePostal: '59000' } }
            ]
        }
    },
    computed: {
        adresse() {
            return this.personnes.find(elt => elt.id == this.$route.params.id).adresse
        }
    }
}
</script>
```

Vue.js

Contenu de `PersonneSportView.vue`

```
<template>
    <h2>Sport </h2>
    <ul>
        <li>Type : {{ sport.type }}</li>
        <li>Nom : {{ sport.nom }}</li>
    </ul>
</template>

<script>
export default {
    name: 'PersonneSportView',
    data() {
        return {
            personnes: [
                { id: 1, sport: { type: 'collectif', nom: 'foot' } },
                { id: 2, sport: { type: 'individuel', nom: 'tennis' } },
                { id: 3, sport: { type: 'collectif', nom: 'hand' } }
            ]
        }
    },
    computed: {
        sport() {
            return this.personnes.find(elt => elt.id == this.$route.params.id).sport
        }
    }
}
</script>
```

Vue.js

Dans `index.js`, définissons les routes imbriquées

```
{
    path: '/personne/:id(\\d)',
    name: 'personne-details',
    props: true,
    component: PersonneDetailsView,
    children: [
      {
        path: 'sport',
        component: PersonneSportView,
      },
      {
        path: 'adresse',
        component: PersonneAdresseView,
      },
    ],
},
```

Vue.js

Dans `index.js`, définissons les routes imbriquées

```
{
    path: '/personne/:id(\\d)',
    name: 'personne-details',
    props: true,
    component: PersonneDetailsView,
    children: [
      {
        path: 'sport',
        component: PersonneSportView,
      },
      {
        path: 'adresse',
        component: PersonneAdresseView,
      },
    ],
},
```

Ne préfixez pas les routes imbriquées par `/`.

Vue.js

Ajoutons `router-view` **et** `router-link` **dans le** `template` **de** `PersonneDetailsView`

```
<template>
   <h1>Détails de la personne {{ id }}</h1>
   <p>Personne recherchée : {{ personne }}</p>
   <router-link :to="`/personne/${id}/adresse`">Adresse</router-link> |
   <router-link :to="`/personne/${id}/sport`">Sport</router-link>
   <router-view />
   <button @click="pagePrecedente()">Retour</button>
</template>
```

Vue.js

On peut aussi autoriser `props` pour les routes imbriquées afin de simplifier la récupération de l'`id` de la personne

```
{
    path: '/personne/:id(\\d)',
    name: 'personne-details',
    props: true,
    component: PersonneDetailsView,
    children: [
      {
        path: 'sport',
        component: PersonneSportView,
        props: true,
      },
      {
        path: 'adresse',
        component: PersonneAdresseView,
        props: true,
      },
    ],
},
```

Vue.js

Nouveau contenu de `PersonneAdresseView.vue`

```
<template>
    <h2>Adresse </h2>
    <ul>
        <li>Ville : {{ adresse.ville }}</li>
        <li>Code postal : {{ adresse.codePostal }}</li>
    </ul>
</template>

<script>
export default {
    name: 'PersonneAdresseView',
    props: ['id'],
    data() {
        return {
            personnes: [
                { id: 1, adresse: { ville: 'Marseille', codePostal: '13000' } },
                { id: 2, adresse: { ville: 'Paris', codePostal: '75000' } },
                { id: 3, adresse: { ville: 'Lille', codePostal: '59000' } }
            ]
        }
    },
    computed: {
        adresse() {
            return this.personnes.find(elt => elt.id == this.id).adresse
        }
    }
}
</script>
```

Vue.js

Nouveau contenu de `PersonneSportView.vue`

```
<template>
    <h2>Sport </h2>
    <ul>
        <li>Type : {{ sport.type }}</li>
        <li>Nom : {{ sport.nom }}</li>
    </ul>
</template>

<script>
export default {
    name: 'PersonneSportView',
    props: ['id'],
    data() {
        return {
            personnes: [
                { id: 1, sport: { type: 'collectif', nom: 'foot' } },
                { id: 2, sport: { type: 'individuel', nom: 'tennis' } },
                { id: 3, sport: { type: 'collectif', nom: 'hand' } }
            ]
        }
    },
    computed: {
        sport() {
            return this.personnes.find(elt => elt.id == this.id).sport
        }
    }
}
</script>
```

Vue.js

Remarque

On peut utiliser les clés `name`, `alias`, `redirect`, etc dans la définition des routes imbriquées.