

Routing

Mettez en place une application complexe avec les routes

Animé par Mazen Gharbi

Pourquoi gérer les routes côté front ?

- ▷ Nous développons une Single Page Application
- ▷ Sans « **deep-linking** », les utilisateurs ne peuvent pas partager des URLs pour accéder à une vue ou plus exactement une "route".
- ▷ VueJS offre 2 possibilité pour l'affichage de vos urls :
 - › Hashbang mode
 - › HTML5 mode
- ▷ Chaque "route" doit avoir une URL unique et explicite permettant de la retrouver ;

Vue Router

- ▷ Le système de routing officiel de Vue.js s'appelle vue-router
 - › Il faut l'installer en lançant la commande « `npm install vue-router` »
 - › Pour forcer la compatibilité avec Vite : « `npm install vue-router@4` »
- ▷ Il apporte de nombreuses fonctionnalités :
 - › Gestion des routes imbriquées ;
 - › Mise en place des paramètres de route ;
 - › Scroll automatique ;
 - › *etc.*
- ▷ Très inspiré de « *ui.router* » d'AngularJS

Mise en place

- ▷ On commence par créer nos routes
 - › Chaque chemin doit être associé à un composant

src/routes/index.js

```
const routes = [  
  { path: '/', component: Home },  
  { path: '/contact', component: Contact }  
];  
  
const router = createRouter({  
  history: createWebHistory(),  
  routes  
});
```

- ▷ Il faut ensuite créer un système de routing à intégrer à notre app

index.js

```
import App from './App.vue'  
import router from './routes';  
  
createApp(App).use(router).mount('#app');
```

Affichage de la vue en cours

[Testez ce code !](#)

- Comment indiquer où le contenu de la route chargée doit s'afficher ?

src/App.vue

```
<template>  
  <router-view></router-view>  
</template>  
  
<script setup>  
</script>
```



Sera remplacé automatiquement
en fonction de la route actuelle

Routes nommées

▷ Pour améliorer la flexibilité de votre code, Vue vous propose d'attribuer des noms à vos routes

- › Pratique si dans 2 ans vous décidez de changer « contact » par « infos »
- › Désormais, on préférera cette écriture

src/routes/app.routes.js

```
...  
{  
  path: "/",  
  component: Home,  
  name: "home"  
},  
{  
  path: "/contact",  
  component: Contact,  
  name: "contact"  
}  
...
```

Redirections

- ▷ Mettons en place un petit menu :

MENU : [Home](#) [Contact](#)

- ▷ Pour ce faire, il va falloir rediriger au moment du click
 - › « href » n'est plus adapté

```
<div>
```

```
  <span>MENU : </span>
```

```
  <router-link to="/">Home</router-link>
```

```
  <router-link :to="{name: 'contact'}">Contact</router-link>
```

```
</div>
```

src/App.js

Redirections avec le modèle

[Testez ce code !](#)

- ▷ Certains cas nécessiteront une redirection à partir du modèle
- ▷ Pour ce faire, il sera nécessaire de récupérer l'instance VueRouter

src/App.js

```
...  
methods: {  
  onClickRedirect() {  
    setTimeout(() => {  
      this.$router.push({ name: 'contact' });  
    }, 5000);  
  }  
}  
...
```


Paramètres de route

- ▷ Nous pourrions avoir besoin d'afficher un contenu différent en fonction des paramètres transmis à la "route"
- ▷ Les paramètres seront déclarés dans la propriété « path »

```
...  
    {  
      path: '/contact/:name',  
      component: Contact,  
      name: 'contact'  
    }  
...
```

src/routes/app.routes.js


- ▷ Comment passer les paramètres lors d'une redirection ?

Paramètres de route

[Testez ce code !](#)

src/App.js

```
<div>
  <span>MENU : </span>
  <router-link to="/">Home</router-link>
  <router-link :to="{name: 'contact', params: {name: 'Coucou'}}">Contact</router-link>
  <router-link to="/contact/Toto">Contact 2</router-link>
</div>
```



Avec route nommée

▷ Plus qu'à récupérer ces paramètres :

```
<template>
  <div>
    <p>Page Contact {{ this.$route.params.name }}</p>
  </div>
</template>
```

▷ Et si on redirigeait vers la même route avec un paramètres différent?

Ecouter le changement de route

[Testez ce code !](#)

▷ Soit avec un Hook :

```
beforeRouteUpdate(to, from, next) {  
  this.valueDisplay = to.params.name;  
  next();  
},
```

▷ ou avec un Watcher :

```
watch: {  
  '$route'(to, from) {  
    this.valueDisplay = to.params.name;  
  }  
}
```

Routes imbriquées

- ▷ Configurer toutes les routes à la racine de l'application peut être **très lourd**
 - › VueJS nous permet de définir une **arborescence de routes**
- ▷ Le composant racine définit toutes les routes
 - › Fait appel aux routes des composants enfants
- ▷ Les routes imbriquées sont des routes qui vivent au sein d'autres routes
- ▷ La propriété « **children** » nous permettra de gérer les composants enfant

Routes imbriquées - Définitions

- ▷ Comme tout à l'heure, la première étape va être de définir les routes enfants

```
export default [  
  {  
  
    },  
  {  
  
    }  
];
```

```
    path: 'article',  
    component: Article,  
    name: 'home.article'
```

```
    path: 'jeux',  
    component: Game,  
    name: 'home.jeux'
```

src/routes/home.routes.js

Pour ne pas surcharger un
fichier route, on crée un nouveau
fichier par route enfant

Pour éviter les clash de nom, on préfixe
le nom de la route par le nom de la route père

Routes imbriquées - Children

► Puis on déclare les enfants au niveau du père :

```
import homeRoutes from './home.routes';
```

src/routes/app.routes.js

```
...
```

```
routes: [  
  {
```

```
    path: '/home',  
    component: Home,  
    name: 'home',  
    redirect: {
```

```
      name: 'home.article'
```

```
    },  
    children: homeRoutes
```

```
  },
```

```
...
```

Permet d'indiquer que l'on souhaite atterrir sur la route « home.article » par défaut

Nous permet de déclarer les routes enfants

Routes imbriquées - Affichage

[Testez ce code !](#)

▷ Enfin, il va être nécessaire d'indiquer où les routes enfants doivent s'afficher

```
src/pages/home.js

<div>
  <span>SOUS MENU : </span>
  <router-link :to="{name: 'home.article'}">Articles</router-link>
  <router-link :to="{name: 'home.jeux'}">Jeux</router-link>
</div>

<div style="border: 1px solid black; padding: 20px;">
  <router-view></router-view>
</div>
```

Middlewares

- ▷ Le système de routing nous propose un cycle de vie applicable à chaque changement de route
 - › Nous pourrions surcharger certaines étapes
- ▷ Ces fonctions sont des Guards
- ▷ Nous avons 3 catégories de Guards :
 - › **Globaux**, relatifs au système de routing général ;
 - › Par **route** et ses enfants ;
 - › Par **composant** ;

Guards globaux

[Testez ce code !](#)

```
/* Mise en place de nos Guards globaux */
const router = createRouter({
  history: createWebHistory(),
  routes
});

router.beforeEach((to, from, next) => {
  console.log(`Changement de route enclenché de ${from.name} vers ${to.name}`);

  if (Math.random() > 0.5) {
    console.log("Changement de route acceptée.");
    next();
  } else {
    console.log("Refusé");
  }
});
```

index.js

- ▷ La fonction `next()` permet d'enclencher la suite de l'appel
- ▷ On peut lui passer en paramètre une nouvelle route :
 - › `next({path: '/'});`

Guards spécifiques aux routes

src/routes/app.routes.js

```
{
  path: '/contact',
  component: Contact,
  name: 'contact',
  beforeEnter: (to, from, next) => {
    console.log(`Tentative d'accès à ${to.name}`);
    if (Math.random() > 0.5) {
      next();
    } else {
      console.log('Refusé.');
```

```
    }
  }
},

{
  path: '/home',
  component: Home,
  name: 'home',
  beforeEnter: (to, from, next) => {
    console.log('Accès à un enfant de home !')
    next();
  },
  redirect: {
    name: 'home.article'
  },
  children: homeRoutes,
}
```

« In-Component Guards »

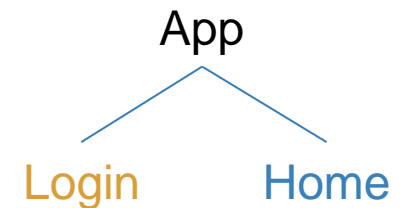
src/pages/articles.js

```
export default Vue.component('articles', {
  template: `
    <div>
      <h1>Page des Articles</h1>
    </div>
  `,
  beforeRouteEnter (to, from, next) {
    // Appelé juste avant que la route ne soit confirmé (next)
    // N'a évidemment pas accès au "this" du composant appelé car pas encore créé à ce moment
  },
  beforeRouteUpdate (to, from, next) {
    // Appelé quand la route qui a appelé le composant actuel change
  },
  beforeRouteLeave (to, from, next) {
    // Appelé JUSTE avant de quitter la route actuelle
    // Nous avons encore accès à this
  }
});
```

TP - Sécuriser notre application



- ▷ Nous allons encore une fois récupérer notre TP sur les films afin de le **sécuriser**
- ▷ Pour ce faire, ajoutez une page login empêchant l'accès
 - › Le composant App racine devra proposer 2 routes enfants : **Login** & **Home**
- ▷ **Home** contiendra la liste des films et **Login** contiendra un simple bouton « Me connecter » qui permettra d'accéder à la page Home !
- ▷ *(Bonus) - Si déjà connecté, rediriger vers Home*
- ▷ *- Empêcher accès à Home si non connecté*
- ▷ *- Implémenter la persistance de connexion avec localStorage*



Questions