Web Client - **#4**

[Vue.js - SPA / Router]

Cyril Rouyer

https://vuejs.org

Sommaire

- Application VueJS SPA
- Intégration d’axios
- Composants .vue
- Router

Application VueJS SPA

(Single Page Application)

SPA 1

La création de l’application

- Via un terminal
- Assurez-vous d’avoir une version à jour de Node.js
- npm create vue@latest

SPA 2

La création de l'application

```
✔ Project name: … <your-project-name>
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes
✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit testing? … No / Yes
✔ Add an End-to-End Testing Solution? … No / Cypress / Playwright
✔ Add ESLint for code quality? … No / Yes
✔ Add Prettier for code formatting? … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

SPA 3

Le lancement

- À l’issue de l’installation :
  - Rentrez dans le dossier de votre application
  - npm i pour installer les paquets
  - npm run dev pour lancer l’application

Intégration d’axios

Intégration d'axios 1

npm i

- Pour installer un nouveau paquet (ex : axios, sass, vuetify …), il faut utiliser npm install [nom_du_paquet]

- npm i est un alias

- -s pour sauvegarder le paquet dans votre fichier package.json (utile pour partager la structure de votre app via git).

- --dev pour indiquer qu'il ne s'agit que d'une dépendance utile pendant les développements (ex : sass)

Intégration d’axios 2

Axios

- Pour installer axios : npm i -s axios
- Pour utiliser axios dans votre code (en haut de vos scripts) :

  import axios from ‘axios’

Intégration d’axios 3

Rappels

- Exemples d’appels :
- axios.get(url, {params: {ID: 34, order: ‘name’}})
- axios.post(url, {name: ‘John’, age: 56}, {headers: { “Content-type”: “application/json”}})
- axios.put(…)
- axios.delete(…)
- Tous les verbes HTTP sont gérés

Intégration d'axios 4

Config

- Pour éviter de répéter toujours la même url, les mêmes headers, etc… dans vos appels Ajax

- Vous pouvez créer une instance d'axios pré-configurée

```
export default axios.create({
    baseURL: 'http://masuperapi.com',
    headers: {
        'Content-Type': 'application/json',
        'Authorization': 'ma_super_api_key'
    }
})
```

Intégration d’axios 5

Config

- Pour utiliser l’instance d’axios ainsi exportée

```
import api from ‘…’

api.post(‘/maressource’, {name: ‘titi’}).then(({data}) => {
  this.gameCode = data.code
})
```

- Dans cet exemple, plus besoin du domaine ou des headers à chaque appel

Composants .vue

Composants .vue 1

Principe

- En mode SPA, VueJS propose un format de fichier dédié pour gérer ses composants : .vue
- Ils permettent de regrouper toutes la logique du composant dans un seul fichier
- Ces fichiers intègrent à leur racine:
  - Le template du composant : <template>…</template>
  - Le JS : <script>…</script>
  - Les styles du composant : <style>…</style>
- Truc cool : le template n’est pas à coder dans des back-quotes :`<div>…</div>`

Composants .vue 2

Exemple

```
<script>
  import api from ‘@/api’

  export default {
    data() {
      return {
        message: “Hello World”
      }
    }
  }
</script>

<template>
  <div><span>{{message}}</span></div>
</template>

<style scoped>
    span {
      color: #123456;
    }
</style>
```

Composants .vue 3

Styles : scoped

- Par défaut les styles que vous allez ajouter dans la balise <style> du composant ont une portée sur l'ensemble de votre application

- Pour limiter la portée de vos styles uniquement à votre composant, vous pouvez utiliser l'attribut scoped

<style scoped> … </style>

Composants .vue 4

Styles : SASS / SCSS

- Si vous êtes habitués à travailler avec SASS ou SCSS vous pouvez le faire dans votre application
- Il faut installer le préprocesseur :
  - npm i -s --dev sass
  - npm i -s —dev sass-loader
- Puis indiquer le langage à utiliser dans votre balise style avec lang="…"

```
<style scoped lang="sass"> … </style>
```

Router

Router 1

Définition

- Lib extérieure à Vue mais développée par la même équipe : https://router.vuejs.org

- Permet de créer une logique de navigation via des routes (portion dynamique d'url)

- Permet les nested-routes (notion de cascade dans le routing)

- Mais toujours en mode SPA

- Grâce à HTML 5 History qui permet de changer et parser l'url

Router 2

Installation

- Au moment de l'installation VueJS vous propose de l'installer
- Mais si vous avez dit "No", vous pouvez l'installer à posteriori

```
npm install -s vue-router
```

- import {createApp} from 'vue'
- import router from 'myRouter'
- const app = createApp({…})
- app.use(router) //c'est un plugin

Router 3

Utilisation / template

```
<div id="app">
  <h1>Bonjour l'application !</h1>
  <p>
    <!-- utilisez le composant router-link pour la navigation. -->
    <!-- spécifiez le lien en le passant à la prop `to` -->
    <!-- `<router-link>` sera rendu en tag `<a>` par défaut -->
    <router-link to="/foo">Aller à Foo</router-link>
    <router-link to="/bar">Aller à Bar</router-link>
  </p>
  <!-- balise pour le composant router-view -->
  <!-- le composant correspondant à la route sera rendu ici -->
  <router-view />
</div>
```

Router 4

Utilisation / template

- <router-link> permet de créer des liens vers une route
- <router-view> permet d'indiquer l'emplacement où sera affiché le contenu de la route demandée par l'utilisateur
- Le router-link dont la cible correspond à la route courante aura automatiquement la classe **.router-link-active** (pratique pour mettre en avant l'item courant dans un menu par exemple)

Router 5

Utilisation / JS

```
import {createRouter, createWebHistory} from ‘vue-router’
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }

const routes = [
  { path: '/foo', name: ‘foo’, component: Foo },
  { path: '/bar', name: ‘bar’, component: Bar }
]

const router = createRouter({
   //indique quel mode d’historique on veut soit hashtag soit
   classique (createWebHashHistory pour le hashtag)
  history: createWebHistory(import.meta.env.BASE_URL),
  routes // raccourci pour `routes: routes`
})

const app = Vue.createApp(App)
app.use(router)
app.mount(‘#app’)
```

Router 6

Utilisation / JS

- Chaque route est définie à minima par un path et un component
- Le composant indiqué sera celui rendu si l'url correspond au path de la route
- Le router est composé de routes (tableau de routes)
- On intègre le router à l'application directement dans les options de notre instance de Vue

Router 7

Utilisation / JS

- Le fait d'intégrer le router comme une option de l'instance de notre Vue nous permet des raccourcis dans les composants :

- this.$router (permet d'accéder à l'instance du router)

- this.$route (permet d'accéder à la route courante)

Router 8

Utilisation / $route.params

- Le path d'une route peut contenir un placeholder dynamique

- { path: '/users/**:id**', component: UserShow }

- Où ':id' sera remplacé par une vraie valeur (dans l'exemple l'id de l'utilisateur dont on veut consulter sa fiche)

- Dans les composants, on peut récupérer la valeur réelle grâce **this.$route.params**

- /users/**:username**/posts/**:id** -> /users/**bob**/posts/**34** -> this.$route.params => {username: 'bob', id: 34}

Router 9

beforeRouteUpdate

- Vue-router cherche toujours à optimiser la réutilisation du code des templates

- Par conséquent, si l'utilisateur passe de /users/34 à /users/35, il ne va pas considérer que la route a été modifiée et ne sera pas réactif

- Il faut alors ajouter un hook dans le composant et gérer soi-même le côté réactif :

```
…
beforeRouteUpdate (to, from){
  …
  axios.get(`/users/${to.params.id}`).then(({data}) => this.currentUser = data)
  …
}
```

Router 10

Utilisation / navigation

- On a déjà vu <router-link to="…"> qui permet de diriger l'utilisateur sur une route en cliquant sur le router-link
- On peut aussi diriger l'utilisateur côté JS
- **this.$router.push("…")** permet ainsi d'envoyer l'utilisateur sur la route indiquée

Router 11

Exemples de push

- this.$router.push(‘/home’)
- this.$router.push({path: ‘/home’})
- this.$router.push({name: ‘user’, params: {id: 1234}})
- this.$router.push({path: ‘/register’, query: {plan: ‘private’}}) //—> …/register?plan=private
- This.$router.push({path: ‘/about’, hash: ‘#team’})

// —> …/about#team

Router 12

Même principe avec <router-link>

- Les exemples décrits précédemment s’appliquent aussi au router-link (dans les templates)

<router-link class=“has-text-dark” **:to**="{name: 'project-show-all', params: {id: project.id}}”>{{project.all}}</router-link>

Router 13

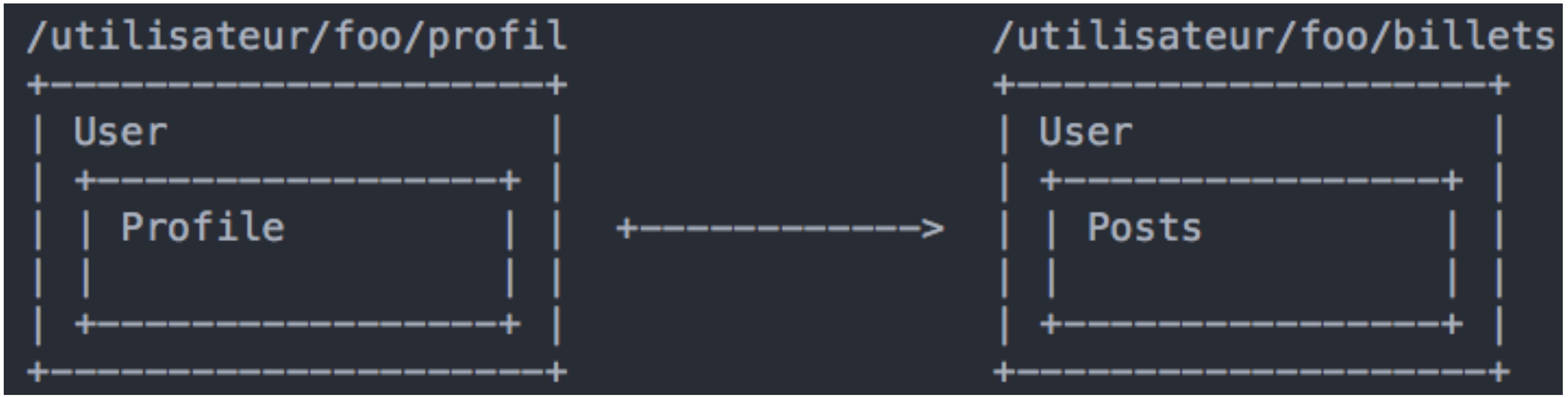
Push / Replace / Go

- Le router expose deux autres méthodes :
  - router.replace() : fonctionne exactement de la même façon que push à l'exception du fait qu'elle ne réalise pas une nouvelle entrée dans l'historique, elle remplace la dernière
  - router.go() : prend en paramètre un entier, et indique de combien d'entrées avant ou après la page courante il faut naviguer vers. Ex : router.go(4) , router.go(-1)

Router 14

Nested routes

- Une application réelle peut parfois nécessiter plusieurs niveaux de décomposition des routes :



- Vue-router permet de gérer autant de niveaux que l’on souhaite !

Router 15

Nested routes

/utilisateur/34/profile

```
<div id="app">
  <router-view></router-view>
</div>
```

```
const User = {
  template: `
    <div class="user">
      <h2>Utilisateur {{ $route.params.id }}</h2>
      <router-view></router-view>
    </div>
  `
}
```

```
const router = createRouter({
  history: …,
  routes: [
    { path: '/utilisateur/:id', component: User,
      children: [
        {
          path: 'profile',
          component: UserProfile
        },
        {
          path: 'posts',
          component: UserPosts
        }
      ]
    }
  ]
})
```

Router 16

Route nommée

- On a vu un peu plus tôt qu'un router.push() peut prendre une option "name" dans le premier argument

  this.$router.push({name: 'user', params: {id: 1234}})

- Voici comment attribuer un nom à une route :

```
const router = createRouter({
  history: …,
  routes: [
    {
      path: '/utilisateur/:userId',
      name: 'user',
      component: User
    }
  ]
})
```

Router 17

Vues nommées

- Parfois on a besoin d'avoir plusieurs vues côte à côte, pas forcément imbriquées

- Les vues nommées permettent, pour une route donnée, de rendre plusieurs composants différents en même temps

Router 18

Vues nommées

```
<router-view class="view one"></router-view>
<router-view class="view two" name="a"></router-view>
<router-view class="view three" name="b"></router-view>
```

```
const router = new VueRouter({
  routes: [
    {
      path: '/',
      components: {
        default: Foo,
        a: Bar,
        b: Baz
      }
    }
  ]
})
```

Router 19

Redirect

- Il est parfois utile de rediriger l'utilisateur lorsqu'il arrive sur une route en particulier

- Pour cela, on peut utiliser dans la signature de la route, l'option **redirect** qui permet de définir la route vers laquelle l'utilisateur devra être redirigé

Router 20

Redirect / exemple

```
{
    path: '/profile',
    name: 'profile',
    redirect: {name: 'user-infos'},
    components: {content: Profile},
    children: [
      {
        path: 'infos',
        name: 'user-infos',
        component: ProfileInfos
      },
      {
        path: 'filters',
        name: 'user-filter',
        component: ProfileFilters
      }
    ]
},
```

Router 21

Alias

- Redirect change l'url, la fonction réalise un push dans l'historique
- Dans l'exemple, avec 'alias' en visitant /b, l'url restera /b mais l'utilisateur verra l'équivalent de /a.

```
const router = new VueRouter({
  routes: [
    { path: '/a', component: A, alias: '/b' }
  ]
})
```

Router 22

Découplage du component

- Chaque route prend un composant à afficher, ce qui revient à coupler le composant à la route

- On perd ainsi l'indépendance du composant (qui par nature devrait pouvoir être utilisé quelque soit le contexte)

- vue-router permet de découpler la route du composant en s'appuyant sur les props du composant

Router 23

Découplage du component

```
const User = {
  props: ['id'],
  template: '<div>Utilisateur {{ id }}</div>'
}
const router = new VueRouter({
  routes: [
    { path: '/utilisateur/:id', component: User, props: true }
```

Router 24

History

- Le routeur attend un mode de gestion de l'historique de navigation à travers sa propriété **history**
- Trois modes sont proposés par vue-router :
  - Hash Mode :
    - un hashtag est injecté dans l'url, pas très joli (ex : https://mondomaine.com/**#**/users/12
    - via history: createWebHashHistory()
  - HTML5 Mode :
    - Des vraies urls (ex : https://mondomaine.com/users/12)
    - Plus compliqué à déployer en production (virtualhost)
    - via history: createWebHistory()
  - Memory Mode : utilisé dans des situations où il n'y a pas de browser (Node env. et SSR) —> history: createMemoryHistory()

Router 25

Mode (exemple)

```
import {createRouter, createWebHistory} from ‘vue-router’

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

Router 26

Hooks

- On a vu plus tôt que **beforeRouteUpdate** permet d'intercepter un changement de params ou de query string au sein d'une même route (/foo/1 -> /foo/2)

- Il existe plusieurs autres hooks, qui fonctionnent soit de façon **globale** (à l'échelle du router), soit de façon **locale** (à l'échelle de la route ou du composant)

Router 27

Hooks

- Inscription globale (pratique pour tester si l'utilisateur est connecté ou a bien le droit d'accéder à la route désirée - ACL)

```
const router = createRouter({ ... })

router.beforeEach(async (to, from) => {
  // ...
})
```

Router 28

Hooks - beforeEach

- beforeEach(async (to, from) => {…}) : s’applique à chaque changement de route ou params
- La callback est asynchrone
- to : l’objet route cible vers laquelle on navigue
- from : l’objet route d’où l’on vient
- Peut retourner :
  - false : arrête la navigation
  - Un objet représentant une nouvelle route. Ex : {name: ‘Login’}
  - rien, true, undefined : continue la navigation (vers to)

Router 29

Hooks - afterEach

- afterEach((to, from) => { …}) : est appelée après chaque fin de navigation, déclaration globale également (sur le router directement).

```
router.afterEach((to, from) => {
  // ...
})
```

Router 30

Hooks - beforeEnter

- beforeEnter((to, from, next) => { … }) est exécutée avant d'entrer dans une route en particulier.
- Si params, query, ou hash changent, elle n'est pas exécutée
- C'est une inscription **locale**, directement dans l'objet Route, du router.

```
const router = createRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      beforeEnter: (to, from) => {
        // ...
      }
    }
  ]
})
```

Router 31

Hooks - beforeEnter

- **return false** coupe la navigation
- Il est possible de lui passer un tableau de fonctions.
- Toutes les fonctions passées en paramètre seront alors exécutées

```
const router = createRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      beforeEnter: [checkAutorizations, cleanParams]
    }
  ]
})
```

Router 32

Hooks - beforeRouteEnter

- beforeRouteEnter((to, from, next) => { … }) est un hook à inscription **locale,** dans le composant directement

- Pratique si on veut charger les éléments depuis le serveur avant de les afficher, on peut bloquer le rendu de la page tant que next() n’est pas appelée

- Vu que c’est appelé avant de rendre le component, la fonction n’a pas accès à this

- Next prend alors une fonction de callback donnant l’instance en argument

Router 33

Hooks - beforeRouteEnter

```
const Foo = {
  template: `...`,
  data: {
    products: [],
  },
  beforeRouteEnter (to, from, next) {
    api.get('/products').then( resp => {
      next( vm => {
      vm.products = resp.data
    })
  })
```

Router 34

Hooks - beforeRouteLeave

- beforeRouteLeave((to, from) => { … }) est un hook à inscription **locale,** dans le composant directement

- Il est appelé avant de quitter le composant courant

- Pratique par exemple pour demander confirmation à l’utilisateur

- Il a accès à this, pas besoin de next()

- **return false** arrête la navigation, l’utilisateur reste sur le composant

Router 35

Hooks - beforeRouteLeave

```
beforeRouteLeave (to, from) {
  const answer = window.confirm('Voulez-vous vraiment quitter
cette page ? Vos changements seront perdus.')
  if (! answer) {
    return false
  }
}
```

Router 36

Metas

- Vous pouvez ajouter des meta-données sur chaque route
- Pratique par exemple pour gérer les droits d’accès

Router 37

Metas

```
routes: [
    {
      path: '/foo',
      component: Foo,
      children: [
        {
          path: 'bar',
          component: Bar,
          // un champ `meta`
          meta: { requiresAuth: true }
        }
      ]
    }
]
```

```
router.beforeEach((to, from) => {
  if (to.meta.requiresAuth && !auth.isLoggedIn()) {
    return {path: ‘/login’}
})
```