Vue.js

M2i

Alan Piron-Lafleur

Création d’une SPA avec Vue CLI

Création d’une SPA avec Vue CLI

Une SPA (Single Page Application) n’est pas un site web one page.

Il s’agit d’une application qui ne charge qu'un seul document web, puis met à jour le contenu du corps de ce document.

Ainsi, une seule page est chargée = pas de chargement de page pendant la navigation.

Création d’une SPA avec Vue CLI

Nous avons jusqu’ici créé nos applications Vue.js en utilisant le CDN et en utilisant une arborescence maison, ce qui est possible dans le cas de petits projets.

Cela nous a également servi à aborder les fondamentaux de Vue.js

Maintenant que les v-bind, les v-model, les events, les customs events

les composants, les props, les hooks, les methods, les computed et les calls API n’ont plus de secret pour vous, intéressons nous à la création de projet Vue.js sous architecture professionnelle… on fera la même chose, mais nos fichiers seront rangés différemment.

Création d’une SPA avec Vue CLI

Allons voir la doc de Vue Cli : https://cli.vuejs.org/guide/installation.html

Attention, vous devez d’abord vous assurer que node soit à jour (version > 10 pour vue Cli mais version > 16 pour vue js 3).
Faites un node -v pour le savoir.

Création d'une SPA avec Vue CLI

Si problème pour installer node avec linux :
https://github.com/nodejs/help/wiki/Installation

Si problème avec le proxy pour le npm install vue cli :
Il faut connaître l'adresse du proxy et faire un : npm config set proxy <adresse>

Création d'une SPA avec Vue CLI

```
npm install -g @vue/cli
```

puis

vue create cours_spa -> choisir vue 3

ensuite, un cd cours_spa puis un npm run serve

Votre app est dispo sur localhost:8080

Installer Vue Router

Installer Vue Router

Comme nous l'avons vu précédemment, le terme « Applications monopages » (SPA) indique qu'une seule page est chargée. Cette dernière est ensuite mise à jour dynamiquement en fonction des besoins.

Bien que cela soit puissant, il y a tout de même un défaut principal : les utilisateurs ont besoin d'URL différentes pour distinguer les différentes pages les unes des autres. Sinon, à chaque fois qu'un utilisateur essaie de revenir en arrière, il obtiendra toujours la même page !

Installer Vue Router

Installons Vue Router : vue add router

A la demande d'installation d'history mode, répondons "non". L'history mode est une configuration qui permet diverses opérations sur les urls dont nous n'aurons pas besoin, la version de base étant suffisante.

Plusieurs fichiers ont été modifiés :

main.js -> router importé et configuré

App.vue -> la logique de notre app a été automatiquement déplacée dans des views.

Arborescence de notre application

Arborescence de notre application

public/index.html : le fichier qui contient notre app qui sera montée. Les fichiers JS seront automatiquement injectés dedans.

src/assets : dans ce dossier, nous mettrons toutes les images et fichiers js et css de notre application. Ils sont gérés par WebPack.

components : dossier qui contiendra les composants (donc les pages) de l'application.

router/index.js : la configuration de notre router. Il importe les views et se charge de la configuration de la navigation.

views : toutes les vues de l'application sont ici.

app.vue : c'est LE fichier qui configure notre application et qui appelle les différents composants à utiliser.

Arborescence de notre application

main.js : le point d'entrée de notre application qui est injecté dans index.html. C'est ce fichier qui indique comment monter l'application

A quoi vous fait penser cette arborescence ?

A l'architecture M V C !
Nous venons de mettre en place quelque chose qui ressemble au MVC au niveau de notre frontend, pas mal ! 😍

Dans Vue.js, c'est Le MVVM

Arborescence de notre application

Si vous naviguez de la page Home à la page About, vous verrez qu'un diese s'intercale dans l'url. Cela vient du mode history du router.

Rdv dans router/index.js pour changer : createWebHashHistory en createWebHistory

(dans l'import ligne 1 et dans la config history ligne 21)

Principe de base de l’archi MVVM

Le MVVM sous Vue.js fonctionne ainsi—---> une requête arrive !

-> public/index.html est chargé

-> mains.js est lu : il importe les dépendances, utilise le router et monte l’app

-> App.vue est lu : il déclare obligatoirement <router-view/> car c’est dans ce composant hérité de Vue Router que viendra se placer le contenu de la page demandée par le visiteur.

-> le router est utilisé via son fichier router/index.js : création des routes et association des composants

-> la page demandée est affichée par <router-view> : le router sait quelle page charger

-> (optionnellement) les composants sont chargés et appelés par la vue

Exercice

Afficher le tableau suivant.
Précision : les data se mettent dans la vue, pas dans le composant.
Précision2 : le v-for s'écrira ainsi (explications à la correction) :

```
<tr v-for="personne in list_inscrits" :key="personne">
```

Correction

Voir le repo github associé au cours

Le call API dans une SPA

Le call API dans une SPA

Rien d’exceptionnellement nouveau, le call API dans notre SPA se fera via axios. La vue est responsable des données, elle passera les users au composant qui fera l’affichage.

Je vous montre.

Le call API dans une SPA

Pour installer axios : npm install axios

Dans InscritsList.vue :

```
data() {
    return {
      personnes: []
    }
  },
  methods: {
    // Récupère 3 utilisateurs via l'API Random Users
    async fetchUsers() {
      const userResponse = await
axios.get('https://randomuser.me/api/?results=3');
      this.personnes = userResponse.data.results;
    }
```

Le call API dans une SPA

Dans le composant InscritsList :

```
<tbody>
      <tr v-for="personne in list_inscrits" :key="personne">
        <td>{{ personne.name.first }}</td>
        <td>{{ personne.name.last }}</td>
        <td>{{ personne.registered.age }}</td>
      </tr>
    </tbody>
```

L’anti-pattern Vue.js

L’anti-pattern Vue.js

Pour comprendre l’anti-pattern de Vue.js, vous allez réaliser un exercice plutôt insolite : l’objectif sera de faire apparaître une erreur.

Nous voulons réaliser l’habituel bouton compteur : au clic sur le bouton, la valeur est multipliée par deux.

1 : créez une nouvelle route “Compteur” et ajoutez la au menu

2 : créez votre vue et le composant qui affichera un bouton

3 : la vue passe la valeur du compteur via une props au composant

4 : le composant utilise la props et l’affiche dans le bouton

5 : créez un événement incremente dans votre composant : au clic sur le bouton, la valeur du compteur est multipliée par deux

Message d’erreur à obtenir : Unexpected mutation of "val" prop

L’anti-pattern Vue.js

Votre vue :

```
<template>
    <ButtonCompteur :val="val" />
</template>

<script>
import ButtonCompteur from '@/components/ButtonCompteur.vue'

export default {
    name: 'CompteurDisplay',
    components: {
        ButtonCompteur
    },
    data(){
```

L’anti-pattern Vue.js

Votre composant ::

```
<template>
  <button type="button" @click="incremente">{{ val }}</button>
</template>

<script>
export default {
  name: 'ButtonCompteur',
  props: {
    val: Number
  },
  methods:{
    incremente(){
      this.val = this.val * 2 ;
```

L’anti-pattern Vue.js

Vue.js considère le fait de modifier la valeur d’une prop passée à un composant comme une opération interdite.
La valeur de la prop doit restée “intacte”.

Pour qu’un composant Vue.js puisse récupérer la valeur d’une prop et la modifier, il faut qu’il la clone dans ses datas.
Ainsi, le composant agira non sur la prop mais sur ses propres datas

L’anti-pattern Vue.js

Ajoutez ce code à composant :

```
data(){
    return{
      compteur:this.val
    }
  },
```

Les mixins

Les mixins

Pour illustrer le principe de mixin, imaginons que nous voulons envoyer un mail à deux endroits de notre super site :

- si le compteur dépasse 30 : on envoie un mail
- si on clic sur le bouton envoyer l'email d'inscription (que l'on va rajouter) sur la liste des utilisateurs : on envoie un mail

Créons une fonction sendMail() qui fera un alert("Email envoyé"); dans le composant InscritsList et ButtonCompteur

Les mixins

- Dans ButtonCompteur, ça donne :

```
methods:{
    incremente(){
      this.compteur = this.compteur * 2;
      if(this.compteur > 30){
        this.sendMail();
      }
    },
    sendMail(){
      alert('Email envoyé !');
    }
  }
```

Les mixins

Le problème d’un code non DRY (Don’t Repeat Yourself) est évident : la méthode d’envoie d’email se répète une fois de trop.

Les mixins offrent une manière flexible de créer des fonctionnalités réutilisables pour les composants de Vue.

Je vous montre.

Les mixins

Le problème d’un code non DRY (Don’t Repeat Yourself) est évident : la méthode d’envoie d’email se répète une fois de trop.

Créons un dossier mixins et un fichier sendMail.js

Les mixins

Dans mixins/sendMail.js

```
export default {
    methods: {
      sendMail() {
        alert('Email envoyé');
      }
    }
  };
```

Les mixins

Dans vos composants :

```
import sendMail from "@/mixins/sendMail";

export default {
  name: 'InscritsList',
  props: {
    list_inscrits: Array
  },
  mixins: [sendMail]
}
```

Les modes et variables globales

Les modes

Pour finir, intéressons nous aux deux modes d'environnements : developpement et production.

Le mode développement : npm run serve … nous le connaissons, c'est pour le dev.

En production, on ne veut pas que d'éventuelles erreurs s'affichent.. on préfèrera une belle page.

Créeons un composant NotFound avec une simple phrase.

Les modes

```
<template>
    <h1>Oups, vous avez trouvé les coulisses !</h1>
</template>

<script>

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

Les modes

Dans index.js (router) :

```
import NotFound from '../components/NotFound.vue'
```

Ajouter au router :

```
  {
    path: '/404', name: 'NotFound', component: NotFound
  },
  {
    path: '/:catchAll(.*)', redirect: '404'
  }
```

Les modes

Le mode production de Vue se génère grâce à : npm run build

Cela créera un dossier dist/

Pour mettre notre site en ligne, il suffira uniquement de mettre le contenu de dist/ sur un serveur web (dans un dossier www ou htdocs généralement) pour le rendre live !

Variables globales

En ce qui concerne les variables globales, l'utilisation est la suivante :

Modifiez votre main.js : `createApp(App).use(router).mount('#app')`

Devient :

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

Ensuite, il suffit de déclarer les globales :

```
app.config.globalProperties.$prenom = "Bob"
```

Et d'y accéder : `this.$prenom`

Formation

THE END