YAVMDB 

Projet VueJS Master 2

Ernesto Artigas*

13 février 2022

*ernart99@gmail.com

Table des matières

Résumé

Ce rapport porte sur le projet final du cours de VueJS de Master 2 WedSCi de l'Université du Littoral de la Côte d'Opale [1]. L'application est fournie dans une image Docker à monter ou à compiler, comme décrit dans le README.md. Il faut créer un .env et le placer à la racine du projet.

.env

```
VUE_APP_AUTH_KEY_V3 = 33da563807ef7bb066ebb41b281c9a3e
```

1 Liste des fonctionnalités et spécificités de l'application.

— L'approche par composants :

1. Structure interne : data, method, computed `src/components/MovieCard.vue`

```
data() { return { url: "", }; },
...
computed: {
        localizedDate: function () {
                if (this.movie.releaseDate == "Date Unknown")
                        return this.movie.releaseDate;
                let date = new Date(this.movie.releaseDate);
                return date.toLocaleDateString(
                        window.navigator.userLanguage ||
                            window.navigator.language, {
                        year: "numeric", month: "short", day:
                            "numeric",
                });
        },
        genres: function () {
                let numberOfGenre = "",
                genres = "";
                try {
                        numberOfGenre = this.movie.genres.
                            length > 1 ? "Genres" : "Genre";
                        // Ternary if to only get the commas
                            between genres.
                        for (let i = 0; i < this.movie.genres.
                            length; i++) {
                                genres +=
                                i != this.movie.genres.length
                                    - 1 ? `${this.movie.genres
                                    [i].name}, ` : this.movie.
                                    genres[i].name;
```

1. Site web de l'université

```
22                          }
23                          genres = this.movie.genres.length != 0
                                ? genres : "Aucun";
24                  } catch (error) { null; }
25                  return `${numberOfGenre} : ${genres}`;
26          },
27  },
```

On déclare nos différentes parties du composants, data pour les données, methods pour les fonctions disponibles et computed pour des valeurs qui vont se mettre à jour avec un traitement particulier. Ici, localizedData nous sert à convertir une date au format UTC à un format plus lisible, ici jour, mois et année. Les fonctions computed sont extrêmement prattiques pour rapidement créer des valeurs dynamique, comme les différents genres à trier par ordre alphabétiques et avec la bonne orthographe pour prendre en compte un ou plusieurs genres.

2. Cycle de vie : mounted. `src/views/SearchResults.vue`

```
1  mounted() {
2          this.searchQuery = this.$route.params.query;
3          tmdbControllerInstance
4                  .getMoviesByName(this.searchQuery)
5                  .then((data) => {
6                          this.moviesArray = data;
7                  })
8                  .catch(() => {
9                          null;
10                 });
11         this.isSearchCompleted = true;
12 },
```

On utilise l'option mounted pour lancer une fonction dès que le composant est monté. Il faut cependant rajouter une option dans le router, car le composant de recherche de film est réutilisé quand on fait une seconde recherche sans revenir à l'accueil. Sans l'option `:key="$route.fullPath"` sur le `router-view` de `App.vue`, le composant ne se remonte pas et ne se met pas à jour.

3. Rendu déclaratif. `src/views/SearchMovie.vue`

```
1  <b-card-text style="font-size: 1.2rem" class="mb-5">
2          {{ movie.overview }}
3  </b-card-text>
```

On utilise l'opérateur pour injecter dans le code HTML des données, méthodes ou computed. Cet opérateur permet de garantie la mise à jour en cas de modification.

4. Directives pour la génération de code « Front » : v-if, v-for, v-bind. `src/views/SearchResults.vue`

```
1   <b-alert v-if="hasSearchNoResults" show variant="danger">
2          Nous n'avons trouve de film pour votre recherche.
3  </b-alert>
4  <div v-else>
5          <div class="grid">
6                  <MovieCard
```

```
7                           v-for="movie in moviesArray"
8                           :key="movie.id"
9                           :movie="movie"
10                          style="margin-bottom: 15px">
11                  </MovieCard>
12          </div>
13  </div>
```

Ici, j'utilise le v-if pour afficher si la recherche n'a rien retourné comme résultats, le v-else devant être dans la balise suivant le v-if pour fonctionner et ici afficher les résultats. On utilise le v-for pour créer autant d'instances du composant MovieCard qu'il y a de résultats, ici d'instance de l'objet Movie dans un tableau. Le v-bind, pouvant être aussi être écrit en :, est utilisé pour donner à notre instance de MovieCard une props.

src/components/Navbar.vue

```
1  <b-button v-if="user != null" @click="disconnect">
2          Deconnexion
3  </b-button>
4  <b-button v-else v-b-modal.connection>Connexion</b-button>
5  ...
6  <p v-if="!isEmailCorrect" class="error-text">
7          Email : toto@yavmdb.com
8  </p>
9  ...
10 <p v-if="!isPasswordCorrect" class="error-text">
11         Mot de passe : 123456789
12 </p>
13 ...
14 <b-button
15     variant="primary"
16     @click="submitUser"
17     :disabled="!isFormValid">
18         Connexion
19 </b-button>
```

Ici, Navbar possède plusieurs v-if dans différentes situations. D'abord on peut afficher un bouton pour afficher un formulaire de connexion si l'utilisateur n'est pas connecté, soit un bouton de déconnexion dans le cas contraire. Dans le formulaire, si le mail et le mot de passe sont incorrects (c'est ici un code de test et l'identification est implémentée dans le but de démontrer les fonctionnalités de VueJS) on le précise à l'utilisateur. Enfin, tant que les champs sont invalides on bloque le bouton de connexion avec un v-bind.

src/App.vue

```
1  <b-alert
2          show
3          v-show="isAlertVisible"
4          v-if="user != null"
5          variant="success">
6                  {{ user }} est connecte.
7  </b-alert>
```

```
<b-alert show v-show="isAlertVisible" v-else variant="danger">
        Pas connecte.
</b-alert>
```

Ici, j'affiche soit une alerte pour prévenir l'utilisateur qu'il est déconnecté, soit qu'il est connecté, avec dans les méthodes un booléen utilisé pour faire disparaître l'alert après 3 secondes.

5. Gestion événementielle : v-on ou @. `src/components/Navbar.vue`

```
<b-input
        v-model="searchQuery"
        @keyup.enter="goToSearchResults"
        placeholder="Rechercher un film">
</b-input>
<b-input-group-append>
        <b-button @click="goToSearchResults" variant="
            secondary">
                <b-icon-search> </b-icon-search>
        </b-button>
</b-input-group-append>
```

L'utilisation de v-on ou @ est extrêmement pratique. Ici, on donne au bouton la fonction à l'ancer quand l'évènement `click` est déclenché. Pour plus d'ergonomie, appuyer sur la touche entrer quand on est dans le champ de recherche permet également de lancer la fonction `goToSearchResults`, grâce à l'évènement `keyup.enter`.

6. Formulaires et liaison bidirectionnelle : v-model. `src/components/Navbar.vue`

```
<b-input
    v-model="searchQuery"
    @keyup.enter="goToSearchResults"
    placeholder="Rechercher un film"></b-form-input>
        ...
        <b-form-input
                id="mail-input"
                v-model="mail"
                :class="{ 'error-input': !isEmailCorrect }"
                required>
        </b-form-input>
        ...
        <b-form-input
                id="password-input"
                type="password"
                v-model="password"
                :class="{ 'error-input': !isPasswordCorrect }"
                required>
        </b-form-input>
```

Le v-model nous permet de lier une valeur à une autre, ici le champ pour entrer le mail et le mot de passe sont liés à leurs valeurs respectives grâce au v-model. On peut effectuer ensuite des traitements sur ces valeurs, comme détaillés plus haut avec des v-if pour changer dynamiquement la classe d'une balise ou afficher / faire disparaître un

message d'erreur. Ici il est également utilisé pour récupérer les éléments à rechercher dans une variable, ici `searchQuery`.

7. Watch `src/views/SearchMovie.vue`

```
watch: {
        ratingValue: function (newValue) {
                alert(
                        `Vous avez vote ${newValue} ${newValue != 1 ? "etoiles" : "etoile"}.`
                );
        },
},
```

Pour prévenir l'utilisateur sur la note qu'il a donné à un film, on utilise un watch qui va surveiller la valeur. A chaque changement elle provoquera une alert pour prévenir l'utilisateur avec un message personnalisé.

— Composants paramétrés `src/components/MovieCard.vue`

```
<template>
...
<b-card :img-src="movie.poster" overlay :footer="localizedDate">
<template #header>
        <b-card-title class="text-overflow">{{ movie.title }}</b-card-title>
</template>
</b-card>
...
</template>

<script>
...
props: {
        movie: {
                type: Movie,
                default: new Movie(),
        },
},
...
</script>
```

Il y a deux moyens de créer descomposants paramétrés, utiliser des props ou des slots. Le composant MovieCard utilise des props, ici des instances de la classe Movie avec tous les éléments nécessaires pour générer une page (utilisation du poster dans la balise `b-card`). Bootstrap-Vue utilise beaucoup de slots pour facilement personnaliser des composants, ici on peut modifier le `header` de la `b-card` grâce à un slot nommé `header` et lui donner un attribut de notre props.

— Gestion évenementielle inter-composants `src/components/Navbar.vue`

```
methods: {
        submitUser() {
                if (this.isFormValid) {
```

```
4                           this.$emit("emit:user", "Toto");
5                           this.hideModal();
6                           this.user = "Toto";
7                   }
8           },
9           disconnect() {
10                  this.$emit("disconnect:user");
11                  (this.mail = ""), (this.password = ""), (this.
                        user = null);
12          },
13  }
```

src/App.vue

```
1  <template>
2  ...
3  <Navbar @emit:user="receiveUser" @disconnect:user="
       disconnectUser"></Navbar>
4  ...
5  </template>
6
7  <script>
8  ...
9  methods: {
10          receiveUser(userFromNavbar) {
11                  this.user = userFromNavbar;
12                  this.isAlertVisible = true;
13                  setTimeout(() => (this.isAlertVisible = false),
                        3000);
14          },
15          disconnectUser() {
16                  this.user = null;
17                  this.isAlertVisible = true;
18                  setTimeout(() => (this.isAlertVisible = false),
                        3000);
19          },
20 },
21 ...
22 </script>
```

Un composant fils peut envoyer des données à un composant parent grâce à l'utilisation d'un `emit`, permettant de renvoyer un message avec une donnée. Ici, le composant `Navbar` envoie soit `emit:user` avec le nom de l'utilisateur, soit `disconnect:user` sans données. On récupère ces valeurs avec une méthodes et on les traite, ici pour afficher l'alert prévenant de la connexion ou déconnexion de l'utilisateur, décrite plus haut dans l'explication des v-if.

— Utilisation d'API externes avec Axios `src/utilities/TMDBController.js`

```
1  async getMoviesByName(movieTitle) {
2  return new Promise((resolve, reject) => {
3          axios
```

```
4           .get(`https://api.themoviedb.org/3/search/movie?api_key=
                ${this.apiKey}&query=${movieTitle}&language=fr-FR`)
5           .then((response) => {
6           let movieArray = [];
7           for (let i = 0; i < response.data.results.length; i++) {
8                   movieArray.push(
9                   new Movie(
10                  response.data.results[i].id,
11                  response.data.results[i].title,
12                  response.data.results[i].poster_path,
13                  response.data.results[i].release_date,
14                  response.data.results[i].vote_average,
15                  response.data.results[i].original_language
16                  )
17                  );
18          }
19          resolve(movieArray);})
20          .catch(function (error) {
21                  reject(error);
22          });
23          }
24  );
25  }
```

src/views/SearchResults.js

```
1   mounted() {
2           this.searchQuery = this.$route.params.query;
3           tmdbControllerInstance
4                   .getMoviesByName(this.searchQuery)
5                   .then((data) => {
6                           this.moviesArray = data;
7                   })
8                   .catch(() => {
9                           null;
10                  }
11          );
12          this.isSearchCompleted = true;
13  },
```

Pour plus de proprétés de codes, j'utilise une instance de l'objet `TMDBController` dans les différents composants pour récupérer le contenu des requêtes. Ici en lançant le composant `SearchResults` on récupère le contenu de la requête en lui donnant comme paramètre le titre du film pour itérer sur chaque élément du tableau avec un v-for, comme montré ci-dessus.

— Routage au sein d'une application Vue `src/router/index.js`

```
1   const routes = [{
2           path: "/",
3           component: Home,
4   }, {
```

```
5           path: "/search/:query",
6           component: SearchResults,
7   }, {
8           path: "/movie/:id",
9           component: SearchMovie,
10  },];
```

src/App.vue

```
1   <router-view :key="$route.fullPath" />
```

src/components/Navbar.vue

```
1   methods: {
2           goToSearchResults() {
3                   // To trigger the no results alert on the
                        SearchResults vue.
4                   this.searchQuery = this.searchQuery == "" ? " "
                        : this.searchQuery;
5                   this.$router.push(`/search/${this.searchQuery}`)
                        ;
6           },
7   }
```

src/components/MovieCard.vue

```
1   <router-link :to="url" style="text-decoration: none; color:
        inherit">
2   ...
3   </router-link>
```

On définit nos routes dans l'index.js et on peut spécifier si les routes ont besoin d'un paramètre avec `:valeur`. Le `App.vue` affiche les différents composants avec le `router-view`. Il y a deux façons d'accéder à une route, soit par la fonction `$router.push` en lui donnant la route et la valeur à mettre dans l'url, soit `router-link` avec l'url, ici une computed pour plus de lisibilité.

2 Conclusion

Ce projet a été une très bonne expérience pour me familiariser avec la création d'un projet de A à Z dans le développement web, me faisant concevoir l'interface de l'application et ses fonctionnalités. Avec les différents outils appris au cours de ce travail, je serais capable de travailler sur des interfaces plus complètes et de lier l'application avec une API, pour par exemple gérer les utilisateurs et permettre de sauvegarder les notes données par l'utilisateur pour chaque film.