Les composants de Vue.js

1 Créer des composants Vue.js

Avec **Vue.js** nous pouvons créer des composants et les utiliser. Un composant est composé d'un template, de données et de méthodes.

Fichier /src/main/resources/static/message.js

```
export default {

    // partie template
    // ------------------------------------------
    template: `<p :class='clazz'>{{msg}}</p>`,

    // partie propriétés
    // ------------------------------------------
    props: {
        clazz: String, // une chaine
        text: {
            type: String, // une chaine
            required: true // obligatoire
        }
    },

    // les données de ce composant
    // ------------------------------------------
    data() {
        return {
            msg: "Pas de message",
        }
    },

    // initialisation du composant
    // ------------------------------------------
    mounted() {
        this.msg = this.text;
    },

    // les méthodes de ce composant
    // ------------------------------------------
    methods: {
        change: function(e) {
            console.log("change message ");
            this.msg = e;
        },
        clear: function() {
            console.log("clear message ");
            this.msg = "";
        }
    }
}
```

Pour utiliser ce composant dans votre application Vue.js vous devez

- Changer l'élément `script` afin d'utiliser les modules :

dans le fichier app.jsp

```
<script src="${app}" type="module"></script>
```

- Importer ce composant (au début de la partie JS) :

dans le fichier /src/main/resources/static/app.js

```
import Message from './message.js';
```

- Ajouter ce composant à votre application (après `createApp`) :

dans le fichier /src/main/resources/static/app.js

```
const app = Vue.createApp(myApp);
app.component('Message', Message);
app.mount('#myApp');
```

- Utiliser ce composant dans la partie template de votre application. Vous remarquerez que les composants peuvent avoir des propriétés (deux dans notre exemple `text` et `clazz`).

dans le fichier app.jsp

```
...
<div id="myApp">
    <div class="container">
        ...

        <message text="Une info" clazz="alert alert-primary"></message>
        <message text="Une alerte" clazz="alert alert-warning"></message>
    </div>
</div>
...
```

- Si vous voulez agir sur un composant, vous devez l'identifier avec un proriété `ref` comme dans l'exemple ci-dessous. Vous pourrez ensuite accéder aux méthodes à partir de l'objet père avec la formulation `this.$refs.info.method()`.

dans le fichier app.jsp

```
<message ref="info" text="Compteur" clazz="alert alert-primary"></message>
```

⏩ **Travail à faire :** Utilisez ce mécanisme pour que le changement du compteur soit répercuté dans le message `info` avec la méthode `change`.

2 Composants liés

Un composant peut utiliser d'autres composants. Dans l'exemple ci-dessous nous définissons un compteur à partir d'un message :

Fichier /src/main/resources/static/counter.js

```
import Message from './message.js';

export default {

    // les composants utilisés
    // -----------------------------------------
    components: {
        "message": Message,
    },

    // le template
    // -----------------------------------------
    template: `<message
        clazz='alert alert-secondary'
        :text='counter.toString()'></message>`,

    // les données
    // -----------------------------------------
    data() {
        return {
            counter: 1000,
        }
    },

    // les méthodes
    // -----------------------------------------
    methods: {
        increment: function() {
            this.counter++;
        }
    }
}
```

⏩ **Travail à faire :** Utiliser ce composant dans votre application et faites en sorte d'appeler la méthode `increment` sur un événement associé à un bouton.

3 Une application Vue.js

Pour terminer, nous allons restructurer nos essais sous la forme d'une application complète **Vue.js** :

- Téléchargez l'application exemple [1] et décompressez l'archive.
- Ouvrez ce projet avec **IntelliJ IDEA**. L'IDE va vous proposer de réaliser une installation des packages nécessaires avec `npm install`. Ce projet est en effet basé sur `node` et `npm` (le node package manager).
- Tentez une exécution (vous devrez sûrement adapter les chemins pour trouver les exécutables `node` et `npm`).
- Explorez cette application (répertoire `src`) pour comprendre son fonctionnement :

1. app-vue.zip

```
index.html                          Page principale
src/
src/App.vue                         Composant principal
src/main.js                         Code JS d'initialisation
src/assets
src/assets/base.css                 Style CSS
src/components
src/components/Page1.vue            Le composant Page1
src/components/Page2.vue            Le composant Page2
src/components/NotFound.vue         Le composant page inconnue
src/components/Message.vue          |
src/components/Counter.vue          | Les composants précédents
src/components/Upper.vue            |
```

Note : Nous utilisons les fichiers `.vue` (SFC : Single file component) qui sont une version plus simple à utiliser pour les composants. On retrouve à l'intérieur le code JS, le code HTML et les données. Ces fichiers sont transformer en code JS équivalent par un compilateur exécuté sur le serveur par `node`. Vous retrouvez une nouvelle version des composants `Message` et `Counter`.

Travail à faire : Étudiez la nouvelle version de `Message` et `Counter` afin de bien comprendre le fonctionnement des SFC. Aidez vous de cette introduction [a].

a. https://fr.vuejs.org/guide/introduction.html

Note : Vous remarquerez également que l'application possède un mécanisme très simple dans le composant `App` pour effectuer un routage côté client.

Travail à faire : Ajoutez une quatrième page (qui utilise un autre `Message`).

Travail à faire : Modifiez le composant `Page1` pour utiliser des propriétés calculées [a].

a. https://fr.vuejs.org/guide/essentials/computed.html

Travail à faire : Dans `Page1`, l'expression `$refs.myCounter.increment()` est complexe. Comment la simplifier avec des `useTemplateRef` (doc [a]) ?

a. https://fr.vuejs.org/guide/essentials/template-refs.html

Travail à faire : Tentez d'utiliser le mécanisme provide/inject [a] pour une donnée fournie dans `App` et injectée dans `Message`.

a. https://vuejs.org/guide/components/provide-inject.html

Travail à faire : Intégrez votre application de gestion des films (listage, détails, suppression, modification, création) sous la forme de composants SFC. Réalisez ce travail étape par étape. Il est sans doute souhaitable de regrouper les opérations sur les données dans un composant (au format **Option**).

4 Packager l'application VueJS

Nous allons maintenant embarquer l'application VueJS dans l'application Sprint Boot afin de faciliter le déploiement.

⏩ Améliorer l'application frontend VueJS :

- Nous allons changer le menu pour utiliser des adresses relatives et non plus des adresses absolues (afin de changer le chemin de l'application) :

Ancienne version de `App.vue`

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="/#page1">Page 1</a>
    <a class="navbar-brand" href="/#page2">Page 2</a>
    <a class="navbar-brand" href="/#bad">Bad link</a>
</nav>
```

Nouvelle version de `App.vue`

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#page1">Page 1</a>
    <a class="navbar-brand" href="#page2">Page 2</a>
    <a class="navbar-brand" href="#bad">Bad link</a>
</nav>
```

- Nous allons ajouter dans `vite.config.js` un chemin afin de déplacer l'application frontend :

```
....
export default defineConfig({
    base: '/frontend',            // ligne à ajouter
    ...
})
```

- Lancer l'application VueJS et vérifiez son bon fonctionnement à l'adresse `http://localhost:5173/frontend`.
- Placez-vous dans le répertoire du projet VueJS et préparez une version distribuable de l'application (les codes JS, HTML et CSS sont compilés et préparés) :

```
$ cd "répertoire_de_votre_application_VueJS"
$ VUEAPP="$(pwd)"
$ npm run build
$ ls dist/
```

⏩ Intégrer l'application frontend à Spring-Boot :

- Placez-vous dans le répertoire du projet Spring-Boot et copiez la version distribuable de l'application VueJS dans les fichiers statiques :

```
$ cd répertoire_de_votre_application_Spring_Boot
$ rsync -av --del "$VUEAPP/dist/" src/main/resources/static/frontend/
```

- Lancez votre application Spring-Boot et vérifiez que l'application VueJS est accessible via l'URL : `http://localhost:8081/frontend/index.html`
- Préparez une version distribuable de l'application Spring-Boot :

```
$ cd répertoire_de_votre_application_Spring_Boot
$ mvn package
$ ls -l target/
```

- Lancez le frontend et le backend à partir du fichier `.jar` :

```
$ java -jar target/*.jar
```