Mise en place d'une API Rest avec Spring

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/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/app.js

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

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

dans le fichier /src/main/resources/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 pouvez l'identifier (avec un proriété `ref` comme dans l'exemple ci-dessous)

dans le fichier app.jsp

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

Vous pouvez ensuite accéder aux méthodes à partir de l'objet père avec la formulation `this.$refs.info.method()`. **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/counter.js

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

export default {

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

    // le template
    // -----------------------------------------
    template: <messageclazz='alert alert-secondary' :text='counter.toString()'></me

    // 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.

3 Routage côté client

Créer les trois composants ci-dessous :

Fichier /src/main/resources/page1.js

```
export default {
    template: <h1>Page 1</h1>,
}
```

Fichier /src/main/resources/page2.js

```
export default {
    template: <h1>Page 2</h1>,
}
```

Fichier /src/-main/resources/-notFound.js

```
export default {
    template: <h1>Page not found</h1>,
}
```

et utilisez les pour mettre en oeuvre un routage côté client :

fichier ex-routage.jsp

```
<%@ include file="/WEB-INF/jsp/header.jsp"%>

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

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

        <component :is="currentView" />

    </div>
</div>
<script type="module">

import Page1 from './page1.js';
import Page2 from './page2.js';
import NotFound from './notFound.js';

const routes = {
  '/': Page1,
  '/page2': Page2
}

const app = Vue.createApp( {
  data() {
    return {
      currentPath: window.location.hash
    }
  },
  computed: {
    currentView() {
      return routes[this.currentPath.slice(1) || '/'] || NotFound
    }
  },
  mounted() {
    window.addEventListener('hashchange', () => {
         this.currentPath = window.location.hash
       })
  }
});

app.mount('#myApp');
</script>

<%@ include file="/WEB-INF/jsp/footer.jsp"%>
```