TP3 : Vue.js - Dr Mad et son laboratoire

Pour réaliser les exercices de ce TP en Vue.js, nous allons décomposer chaque tâche étape par étape. Voici comment aborder les différentes sections du TP, en appliquant les principes des props et des événements dans Vue.js.

1. La barre de navigation

1.1 Objectifs et contraintes L'objectif est de créer un composant `NavBar` réutilisable qui émet un événement à chaque clic sur un bouton. Le composant parent va capturer cet événement pour naviguer vers la route appropriée.

1.2 Mise en place Composant `NavBar.vue`

Voici le code de base pour le composant `NavBar.vue` :

```
<template>
  <div>
    <button v-for="(button, index) in titles" :key="index" :style="{ color: button.color }" @click="handleClic
      {{ button.text }}
    </button>
  </div>
</template>

<script>
export default {
  name: 'NavBar',
  props: {
    titles: Array
  },
  methods: {
    handleClick(index) {
      this.$emit('menu-clicked', index);
    }
  }
}
</script>

<style scoped>
</style>
```

- Le composant reçoit une prop `titles` qui est un tableau d'objets avec les propriétés `text` et `color` pour chaque bouton.
- La méthode `handleClick` émet un événement `menu-clicked` avec l'index du bouton sur lequel l'utilisateur a cliqué.

Composant parent `App.vue`

Dans `App.vue`, vous intégrerez le composant `NavBar` et capturerez l'événement `menu-clicked` pour rediriger l'utilisateur.

```
<template>
  <div>
    <NavBar :titles="menuItems" @menu-clicked="navigate"/>
    <router-view/>
  </div>
</template>

<script>
import NavBar from './components/NavBar.vue';

export default {
  name: 'App',
  components: {
    NavBar
  },
  data() {
    return {
```

```
      menuItems: [
        { text: 'Viruses', color: 'blue' },
        { text: 'Compte bancaire', color: 'green' },
        { text: 'Login', color: 'red' }
      ]
    };
  },
  methods: {
    navigate(index) {
      const routes = ['/', '/bank/account', '/login'];
      this.$router.push(routes[index]);
    }
  }
}
</script>
```

- `menuItems` est un tableau d'objets qui contient les titres et les couleurs des boutons.
- `navigate` est une méthode qui redirige vers la route appropriée en fonction de l'index du bouton.

2. Une liste avec cases à cocher

2.1 Objectifs et contraintes Nous devons créer un composant `CheckedList` qui affiche une liste d'objets JSON avec des cases à cocher et des boutons. Ce composant émet des événements chaque fois qu'une case est cochée ou qu'un bouton est cliqué.

2.2 Mise en place Composant `CheckedList.vue`

Voici le code pour le composant `CheckedList.vue` :

```
<template>
  <div>
    <div v-for="(item, index) in data" :key="index">
      <div v-if="itemCheck">
        <input type="checkbox" v-model="checked[index]" @change="handleCheckedChange(index)"/>
      </div>
      <span v-for="field in fields" :key="field">{{ item[field] }}</span>
      <div v-if="itemButton.show">
        <button @click="handleItemButtonClick(index)">{{ itemButton.text }}</button>
      </div>
    </div>
    <div v-if="listButton.show">
      <button @click="handleListButtonClick">{{ listButton.text }}</button>
    </div>
  </div>
</template>

<script>
export default {
  name: 'CheckedList',
  props: {
    data: Array,
    fields: Array,
    itemCheck: Boolean,
    checked: Array,
    itemButton: Object,
    listButton: Object
  },
  methods: {
    handleCheckedChange(index) {
      this.$emit('checked-changed', index);
    },
    handleItemButtonClick(index) {
```

```
      this.$emit('item-button-clicked', index);
    },
    handleListButtonClick() {
      this.$emit('list-button-clicked');
    }
  }
}
</script>

<style scoped>
</style>
```

- Le composant prend en props un tableau de données (`data`), les champs à afficher (`fields`), un tableau de cases à cocher (`checked`), ainsi que des options pour les boutons d'items et le bouton global à la fin de la liste.
- Il émet des événements `checked-changed`, `item-button-clicked` et `list-button-clicked` lorsqu'une case ou un bouton est cliqué.

2.3 Tests Dans `VirusesView`, utilisez le composant `CheckedList` pour afficher la liste des virus.

Modifications dans `VirusesView.vue` :

```
<template>
  <div>
    <CheckedList
      :data="filteredViruses"
      :fields="['nom', 'price']"
      :itemCheck="true"
      :checked="selected"
      :itemButton="{ show: true, text: 'Details' }"
      :listButton="{ show: true, text: 'Show Selected' }"
      @checked-changed="handleCheckedChange"
      @item-button-clicked="handleItemButtonClick"
      @list-button-clicked="handleListButtonClick"
    />
  </div>
</template>

<script>
import CheckedList from './components/CheckedList.vue';

export default {
  name: 'VirusesView',
  components: {
    CheckedList
  },
  data() {
    return {
      viruses: [
        { nom: 'Virus A', price: 10, stock: 5, inSale: true },
        { nom: 'Virus B', price: 20, stock: 0, inSale: false },
      ],
      selected: []
    };
  },
  computed: {
    filteredViruses() {
      // Implémentez ici un filtrage basé sur les critères nécessaires
      return this.viruses;
    }
  },
  methods: {
```

```
    handleCheckedChange(index) {
      const virus = this.filteredViruses[index];
      const virusIndex = this.viruses.findIndex(v => v.nom === virus.nom);
      if (this.selected.includes(virusIndex)) {
        this.selected = this.selected.filter(i => i !== virusIndex);
      } else {
        this.selected.push(virusIndex);
      }
    },
    handleItemButtonClick(index) {
      const virus = this.filteredViruses[index];
      alert(`Nom: ${virus.nom}, Stock: ${virus.stock}, En vente: ${virus.inSale}`);
    },
    handleListButtonClick() {
      const selectedViruses = this.selected.map(index => this.viruses[index]);
      alert('Viruses sélectionnés: ' + selectedViruses.map(v => v.nom).join(', '));
    }
  }
}
</script>
```

- Le tableau `selected` garde les indices des virus sélectionnés.
- Les méthodes `handleCheckedChange`, `handleItemButtonClick` et `handleListButtonClick` gèrent les événements émis par le composant `CheckedList`.