TP2 : Vue.js - Dr Mad et son laboratoire

1°/ Affichage de listes avec `v-for`

Objectif : Dans le composant `BankAccountView.vue`, afficher la liste des transactions sous forme de puces avec les informations sur le montant et la date de chaque transaction. De plus, transformer la date au format lisible.

Solution :

1. Modifiez le fichier `BankAccountView.vue` pour utiliser `v-for` et itérer sur `accountTransactions` afin d'afficher chaque transaction sous forme de `<li>`.
2. Ajoutez une méthode dans `methods` pour transformer la date au format voulu.

```
<template>
  <div>
    <p>Transactions :</p>
    <ul>
      <li v-for="(transaction, index) in accountTransactions" :key="index">
        {{ transaction.amount }} - {{ formatDate(transaction.date.$date) }}
      </li>
    </ul>
  </div>
</template>

<script>
export default {
  methods: {
    formatDate(date) {
      const dateObj = new Date(date);
      const day = String(dateObj.getDate()).padStart(2, '0');
      const month = String(dateObj.getMonth() + 1).padStart(2, '0');
      const year = dateObj.getFullYear();
      const hours = String(dateObj.getHours()).padStart(2, '0');
      const minutes = String(dateObj.getMinutes()).padStart(2, '0');
      const seconds = String(dateObj.getSeconds()).padStart(2, '0');
      return `${day}/${month}/${year} at ${hours}:${minutes}:${seconds}`;
    }
  }
}
</script>
```

2°/ Champ de saisie et `v-model`

Objectif :

- Ajouter un champ de saisie avec `v-model` pour filtrer les virus en fonction du prix.
- Ajouter un filtrage supplémentaire pour le nom du virus et la disponibilité en stock.

Solution pour le filtre de prix :

1. Dans `VirusesView`, initialisez une variable `priceFilter` et une méthode `filterVirusesByPrice` dans `computed`.

```
<template>
  <div>
    <label for="filterprice">Prix inférieur à : </label>
    <input v-model.number="priceFilter" id="filterprice" type="number" />

    <ul>
      <li v-for="(virus, index) in filterVirusesByPrice" :key="index">{{ virus.nom }} : {{ virus.price }}</li>
    </ul>
  </div>
</template>
```

```
<script>
export default {
  data() {
    return {
      priceFilter: 0
    };
  },
  computed: {
    ...mapState(['viruses']),
    filterVirusesByPrice() {
      return this.priceFilter > 0
        ? this.viruses.filter(virus => virus.price < this.priceFilter)
        : this.viruses;
    }
  }
}
</script>
```

Solution pour le filtre de nom :

1. Ajoutez une variable `nameFilter` et une méthode `filterVirusesByName` dans `computed`.

```
<template>
  <div>
    <label for="filtername">Nom du virus : </label>
    <input v-model="nameFilter" id="filtername" type="text" />

    <ul>
      <li v-for="(virus, index) in filterVirusesByName" :key="index">{{ virus.nom }} : {{ virus.price }}</li>
    </ul>
  </div>
</template>

<script>
export default {
  data() {
    return {
      nameFilter: ''
    };
  },
  computed: {
    ...mapState(['viruses']),
    filterVirusesByName() {
      return this.viruses.filter(virus => virus.nom.toLowerCase().includes(this.nameFilter.toLowerCase()));
    }
  }
}
</script>
```

Solution pour le filtre de stock :

1. Ajoutez une variable `stockFilter` et une méthode `filterVirusesByStock` dans `computed`.

```
<template>
  <div>
    <label for="filterstock">En stock : </label>
    <input v-model="stockFilter" type="checkbox" id="filterstock" />

    <table v-if="filteredViruses.length > 0">
      <thead>
        <tr>
          <th>Nom</th>
```

```
          <th>Prix</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="(virus, index) in filteredViruses" :key="index">
          <td>{{ virus.nom }}</td>
          <td>{{ virus.price }}</td>
        </tr>
      </tbody>
    </table>
  </div>
</template>

<script>
export default {
  data() {
    return {
      stockFilter: false
    };
  },
  computed: {
    ...mapState(['viruses']),
    filteredViruses() {
      return this.viruses.filter(virus => (this.stockFilter ? virus.inStock : true));
    }
  }
}
</script>
```

3°/ Rendu conditionnel avec `v-if`

Objectif : Rendre les filtres activés et désactivés en fonction d'une case à cocher.

Solution :

1. Ajoutez une variable `filterPriceActive` pour contrôler l'affichage du filtre de prix.

```
<template>
  <div>
    <span>Filtres :</span>
    <label for="filterpriceactive">par prix</label>
    <input type="checkbox" v-model="filterPriceActive" id="filterpriceactive">
    <hr />
    <div v-if="filterPriceActive">
      <label for="filterprice">prix inférieur à : </label>
      <input v-model="priceFilter" id="filterprice">
    </div>
  </div>
</template>

<script>
export default {
  data() {
    return {
      filterPriceActive: false,
      priceFilter: 0
    };
  }
}
</script>
```

4°/ Pour aller plus loin

4.1°/ Filtrage multi-critères Vous pouvez combiner les trois filtres pour appliquer un filtrage successif sur les critères sélectionnés. Voici une méthode `computed` qui les combine.

```
<template>
  <div>
    <ul>
      <li v-for="(virus, index) in filteredViruses" :key="index">
        {{ virus.nom }} : {{ virus.price }}
      </li>
    </ul>
  </div>
</template>

<script>
export default {
  data() {
    return {
      priceFilter: 0,
      nameFilter: '',
      stockFilter: false
    };
  },
  computed: {
    ...mapState(['viruses']),
    filteredViruses() {
      return this.viruses.filter(virus => {
        const matchesPrice = this.priceFilter > 0 ? virus.price < this.priceFilter : true;
        const matchesName = virus.nom.toLowerCase().includes(this.nameFilter.toLowerCase());
        const matchesStock = this.stockFilter ? virus.inStock : true;
        return matchesPrice && matchesName && matchesStock;
      });
    }
  }
}
</script>
```

4.2°/ Rendre cohérent l'affichage dans `BankAccountView`

1. Désactivez les boutons si le champ est vide ou si le format du numéro de compte est incorrect.
2. Mettez à jour le store pour gérer l'état d'erreur du numéro de compte.

```
<template>
  <div>
    <input v-model="number" type="text" />
    <button :disabled="!isValidAccount || accountNumberError !== 1">Consulter le solde</button>
    <button :disabled="!isValidAccount || accountNumberError !== 1">Voir les transactions</button>
    <p v-if="accountNumberError === -1">Numéro de compte invalide</p>
  </div>
</template>

<script>
export default {
  computed: {
    ...mapState(['accountNumberError']),
    isValidAccount() {
      const accountRegex = /^[A-Za-z0-9]{22}-\d{7}$/;
      return accountRegex.test(this.number);
    }
  },
  data() {
```

```
    return {
      number: ''
    };
  }
}
</script>
```

Dans le store :

```
mutations: {
  setAccountNumberError(state, errorCode) {
    state.accountNumberError = errorCode;
  }
}
```

Conclusion

Ces exercices montrent l'utilisation de `v-for`, `v-model`, `v-if`, et des computed properties pour gérer l'affichage dynamique et le filtrage dans une application Vue.js. Les solutions proposées respectent les bonnes pratiques et permettent de rendre l'application réactive et fonctionnelle.