

PROJET 2 (PARTIE 1)

Pondération et échéance

- ☐ Ce travail peut être fait par groupe de 2 ou de façon individuelle.
- ☐ Ce travail compte pour 10 % de la note finale du bulletin, mais plus important encore, il vous prépare, avec les laboratoires effectués depuis le début de la session, à l'examen final.

- ☐ Date de remise de la solution : **Lundi, 9 décembre 2019 en fin de journée**

Une pénalité de 10 % par jour de retard, incluant les jours de congé, sera imposée, et ce, jusqu'à concurrence de 5 jours. (Après ce délai, la note attribuée est zéro.)

Vous serez évalués sur le produit final. Mieux vaut remettre une application incomplète, mais fonctionnelle qu'une qui semble complète, mais qui ne produit pas les bons résultats.

1. Pour débiter

L'objectif de ce projet est de vous permettre de vous familiariser avec React.

☉ Créez une application React w33_projet2 en utilisant l'approche vue en laboratoire.

```
$ npx create-react-app w33_projet2
```

☉ Installez le module express comme suit vous permettant d'utiliser express. Cette commande installe le module express nécessaire pour créer votre serveur web express.

```
$ npm install --save express
```

Votre projet est maintenant prêt à utiliser express et React. Ce que vous allez faire dans le cadre de ce projet.

Ouvrez le dossier w33_projet2 avec *Visual Studio Code*.

Dans votre répertoire racine, créez un fichier server.js avec le contenu suivant.

```
1  'use strict'
2
3  const express = require('express');
4
5  const portno = 3000; // Numéro de Port utilisé par l'application
6
7  const app = express();
8
9  // On utilise le module express static (http://expressjs.com/en/starter/static-files.html)
10 app.use(express.static(__dirname));
11
12 const server = app.listen(portno, () => {
13   const port = server.address().port
14   console.log(`L'application écoute sur le port ${port}.`);
15 });
```

Dans le fichier package.json, vous allez ajouter une ligne vous permettant de lancer votre serveur express. Modifier le fichier package.json comme suit

```
11  "scripts": {
12    "start": "react-scripts start",
13    "server": "nodemon server.js",
14    "build": "react-scripts build",
15    "test": "react-scripts test",
16    "eject": "react-scripts eject"
```

Faites un clic droit dans la racine de votre projet (évitez de cliquer sur un répertoire). Cliquez sur **Ouvrir dans un terminal**.

Lancez votre serveur express en lançant la commande suivante:

```
$ npm run server
```

Dans votre répertoire public, créez un fichier index.html avec pour contenu "Bienvenue dans le projet 2" dans une balise h2.

Allez dans votre navigateur et lancez la page <http://localhost:3000>. Votre application doit afficher le message "Bienvenue dans le projet 2". Sinon corrigez le problème.

Allez dans le terminal et lancez la partie cliente avec la commande suivante.

Vous devez voir votre application React lancée automatiquement dans votre navigateur.

```
$ npm start
```

Comme vous pouvez le remarquer, vous avez une application avec une partie cliente utilisant React et une partie serveur utilisant express cohabitant ensemble et exploitant les fonctionnalités de Node.js. Votre partie cliente va communiquer avec le serveur un peu plus tard dans le projet.

Dans le répertoire src de votre projet, créez un sous répertoire components qui va contenir vos composants associés à leur fichier css respectif.

Dans le répertoire components, créez une composante de classe provinces.js.

Créez un constructeur avec l'état suivant:

```
this.state = {  
  provinces: ['Alberta', 'Colombie-Britannique', 'Ile-  
du-Prince-Édouard', 'Manitoba', 'Nouveau-Brunswick', 'Nouvelle-  
Écosse', 'Ontario', 'Québec', 'Saskatchewan', 'Terre-Neuve-et-La-  
brador', 'Nunavut', 'Territoires du Nord\Ouest', 'Yukon'],  
  strFiltre: '',  
  binTrouve: false  
}
```

Modifiez votre fonction render pour qu'elle affiche dans une liste non ordonnée les différentes provinces du Canada. Vous pouvez utiliser la fonction map et une fonction fléchée comme suit. Vous devez comprendre les lignes suivantes au regard de ES6. On utilise la fonction map pour avoir les éléments de la liste non ordonnée en utilisant les données provenant du tableau provinces défini dans l'objet state du composant.

```
<ul>
  {this.state.provinces.map(currProvince => {
    return (
      <li key={currProvince}>
        {currProvince}
      </li>
    );
  })}
</ul>
```

Dans cette application, les données que vous allez utiliser pour les provinces (et plus tard les régions) vont provenir du serveur express. Modifiez votre application comme suit pour aller chercher vos données sur le serveur express.

Modifiez votre constructeur en initialisant province à un tableau vide comme suit.

```
constructor(props) {
  super(props)

  this.state = {
    provinces: [],
    strFiltre: ''
  }
}
```

Ajoutez une méthode `componentDidMount` à votre composant pour aller chercher vos données du serveur express. L'implémentation de base vous est fournie ci-dessous. On utilise les promesses pour traiter les données provenant du serveur. On modifie l'état du composant qui force un appel à la fonction `render` pour un nouvel affichage. (Voir Annexe)

```
componentDidMount() {
  fetch('http://localhost:3050/provinces')
    .then(function(res) {
      return res.json();
    }).then((jsonData)=> {
      console.log("Les provinces du Canada: " + jsonData.provinces)
      this.setState({
        provinces: jsonData.provinces
      })
    });
}
```

Ajout d'un filtre sur les provinces

Vous allez ajouter un filtre vous permettant d'afficher seulement les provinces contenant les lettres consécutives qui seront saisies par un utilisateur dans une zone de texte.

Ajoutez une zone de texte avec pour id `filtreID` à la composante `Provinces` dans la fonction `render` permettant à l'utilisateur de saisir des lettres pour filtrer l'affichage par province.

Pour y arriver, créer une fonction fléchée dans la classe `Provinces` vous permettant de filtrer les provinces.

```
handleChange = (event) => {
  this.setState({ strFiltre: event.target.value });
}
```

Modifiez votre zone de texte pour qu'il appelle la fonction `handleChange` à l'événement `onChange` et sa valeur doit correspondre au filtre défini dans l'état du composant.

```
<input
  id="filtreId"
  type="text"
  value={this.state.strFiltre}
  onChange={this.handleChange}
/>
```

Modifiez la fonction *render* pour qu'elle affiche seulement les provinces contenant les mots clés saisis par l'utilisateur.

Avant de faire *return* dans la fonction *render*, créez une variable *binTrouve* au début de la fonction vous permettant de vérifier si les lettres saisies par l'utilisateur sont contenues dans une province. Utilisez la fonction *some* comme suit pour ce faire en lui passant une fonction fléchée. Au départ, quand la chaîne est vide, tous les états sont affichés. Au fur et à mesure que l'utilisateur tape, le filtrage se fait de façon automatique.

```
let binProvincesFound = this.state.provinces.some(
  province=>province
    .toUpperCase()
    .includes(this.state.strFiltre.toUpperCase())
)
```

Modifiez la partie contenant votre liste non ordonnée *ul* comme suit:

```
<ul>
  {this.state.provinces.map(currProvince => {
    return (
      currProvince
        .toUpperCase()
        .includes(this.state.strFiltre.toUpperCase()) && (
          <li key={currProvince}>
            {currProvince}
          </li>
        )
    );
  })}
```

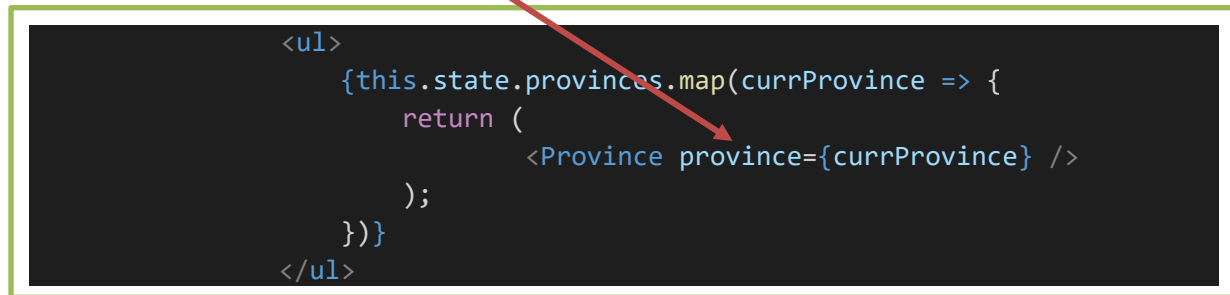
Si l'utilisateur saisi "a" dans la zone de texte, le contenu suivant lui sera affiché.



Un répertoire avec une bonne partie fonctionnelle du travail vous est fourni pour vous permettre de comprendre et de progresser assez rapidement et de bien situer les lignes de code données précédemment.

Travail à faire

1. Créez une nouvelle composante **Province** qui va vous permettre d'afficher les données d'une province provenant de la composante *Provinces*. La nouvelle composante enfant reçoit un attribut *province* provenant de la composante mère *Provinces*. Les lignes de la composante *Provinces* dans la fonction *render* seront modifiées comme suit au regard de l'ajout de la nouvelle composante.



```

<ul>
  {this.state.provinces.map(currProvince => {
    return (
      <Province province={currProvince} />
    );
  })}
</ul>

```

2. Le Canada ayant 5 régions réparties comme suit:
Atlantique: Terre-Neuve-Et-Labrador, Ile-du-Prince-Édouard, Nouvelle Écosse, Nouveau-Brunswick
Centre: Ontario, Québec
Prairies: Manitoba, Saskatchewan, Alberta
Côte Ouest: Colombie Britannique
Territoires du Nord: Nunavut, Territoires du Nord-Ouest, Yukon
Ajoutez une composante *Region* à votre application et faites les modifications nécessaires pour que vous puissiez manipuler les régions comme les provinces. Vous devez pouvoir filtrer les régions comme les provinces. Vous pouvez modifier le design à votre guise. Il suffit que le tout soit cohérent.
3. Ajoutez deux liens dans l'entête de votre page vous permettant de choisir *Provinces* ou *Regions*. La composante affichée doit tenir compte du lien sélectionné par l'utilisateur. Par défaut, les provinces seront affichées à l'utilisateur.
4. Sauvegarder votre application. Zippez-la avec le suffixe *v1* dépouillée de votre répertoire *node_modules*.
5. Modifiez l'application pour qu'elle vous affiche les différentes provinces dans un dropdown list.
6. Modifiez votre application pour que la première option dans le dropdown list région soit "Choisir votre région".
7. Modifiez votre application pour que la première option dans le dropdown list région soit "Choisir votre province".
8. Modifiez l'application pour qu'elle vous affiche les différentes régions dans un dropdown list et les provinces dans un autre dropdown list. Vous pouvez choisir par exemple de faire du dropdown list *Province* un enfant du dropdown list *Region* et lui passer la région sélectionnée par l'utilisateur dans le premier dropdown list.

9. Sauvegarder votre application. Zippez-la avec le suffixe v2 dépouillée de votre répertoire `node_modules`.
10. Modifiez votre application pour que quand l'utilisateur choisisse une région, les provinces qui s'affichent dans le dropdown list des provinces soit celle appartenant à la région sélectionnée.
Vous devez modifier le fichier `express` pour envoyer les bonnes informations à la partie cliente. Modifiez l'api en vous inspirant des fonctions qui vous sont fournies.
11. Sauvegarder votre application. Zippez-la avec le suffixe v3 dépouillée de votre répertoire `node_modules`.

Zippez l'ensemble de vos versions sous le nom `VotreNom_MatriculesDesMembres_Projet2`. Remettez sur Léa.

Références: React.js (Le framework JavaScript de Facebook) de Éric Sarrion

Annexe

Cycle de vie d'un composant

Lorsqu'un composant React est créé au moyen d'une classe, certaines méthodes internes à la classe sont appelées automatiquement par React. Ces méthodes peuvent être surchargées dans la classe du composant afin d'effectuer des traitements spécifiques lors du cycle de vie de ce composant.

Le cycle de vie d'un composant React comporte trois étapes fondamentales:

- la création du composant;
- la mise à jour du composant;
- la destruction du composant.

React propose des méthodes pour chacune des étapes du cycle de vie d'un composant.

Méthodes appelées à la création d'un composant

Lors de la création d'un composant, les méthodes suivantes sont appelées, dans l'ordre indiqué ci-après. Elles ne sont appelées qu'une seule fois (lors de la création du composant) sauf la méthode `render()` qui est appelée à chaque mise à jour.

- `Constructor(props)` : constructeur de la classe
- `componentWillMount()`: une fois le constructeur appelé
- `render()`: méthode qui est appelée lors de l'affichage du composant (ou en cas de mise à jour par `this.setState()`)
- `componentDidMount()`: une fois que le composant est affiché (méthode `render()`) suite à sa création, la méthode `componentDidMount()` est appelée. L'arborescence du DOM a été mise à jour.

Méthodes appelées à la mise à jour d'un composant

`componentWillReceiveProps(nextProps)`: lorsque le composant est de nouveau utilisé (avec de nouvelles propriétés ou non), cette méthode est appelée. Le paramètre `nextProps` indique les propriétés (attributs) qui lui sont transmises (qui peuvent être les mêmes que précédemment, indiquées dans `this.props`). Cette méthode a été introduite par React pour pallier le fait que dans le cas d'une nouvelle utilisation du même composant, le constructeur et les méthodes `componentWillMount()` et `componentDidMount()` n'étaient pas appelées une seconde fois., contrairement à `componentWillReceiveProps()`.

`shouldComponentUpdate(nextProps, nextState)`: si cette méthode retourne `false`, les méthodes décrites ci-après ne sont pas appelées (et le composant n'est pas mis à jour).

`componentWillUpdate(nextProps, nextState)`

`render()`: cette méthode affiche de nouveau le composant, en utilisant `this.state` et `this.props`. La méthode `render()` est appelée à condition que `shouldComponentUpdate()` n'ait pas retourné `false`.

`componentDidUpdate(prevProps, prevState)`: une fois le composant mis à jour, cette méthode est appelée. Cette méthode est appelée à condition que `shouldComponentUpdate()` n'ait pas retourné `false`.

Méthodes appelées à la mise à jour d'un composant

`componentWillUnmount()`: Cette méthode est appelée avant la destruction de l'élément React (et sa suppression de la page HTML).