Secteur Tertiaire Informatique
Filière étude - développement

Développement Web et Web Mobile

ReactJS

Développer des scripts clients dans une page web grâce à un Framework

Mise en pratique 2

Développeur Web et Web Mobile
Vincenzo Tinebra - AFPA - 2019

1. Mise en œuvre de React en mode console

Nous allons utiliser ***Node.js*** pour faire fonctionner React à partir de maintenant. Pour cela, la première chose à faire est d'installer ***Node*** sur notre machine.
Aller sur le lien suivant https://nodejs.org/fr/ et suivre les indications pour installer ***Node.js*** sur une machine Windows.
Une fois ***Node.js*** installé, ouvrons une fenêtre de commande (nous en avons une à disposition dans Visual Studio Code en choisissant dans le menu : Terminal -> New terminal).
Si l'installation s'est bien passée, nous pouvons taper :

```
node --version
```

et obtenir le numéro de version installée. Et taper :

```
npm --version
```

pour obtenir le numéro de version de npm. ***npm*** est le gestionnaire de paquets officiel de Node.js (pour plus d'information : https://docs.npmjs.com/about-npm/).

Installation de *create-react-app*

Create-React-App est un outil permettant de faciliter le développement d'applications web fondées sur React. Pour plus d'information : https://create-react-app.dev/docs/getting-started.
Toujours en mode console, se positionner dans un répertoire de travail (par exemple : *C:\Users\Documents\DWWM\react\applications*) puis taper la ligne de commande :

```
npx create-react-app ToDoList
```

L'instruction va permettre de créer notre première application complète et structurée ReactJS.
Après quelques minutes d'installation, on obtient la structure suivante :

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
```

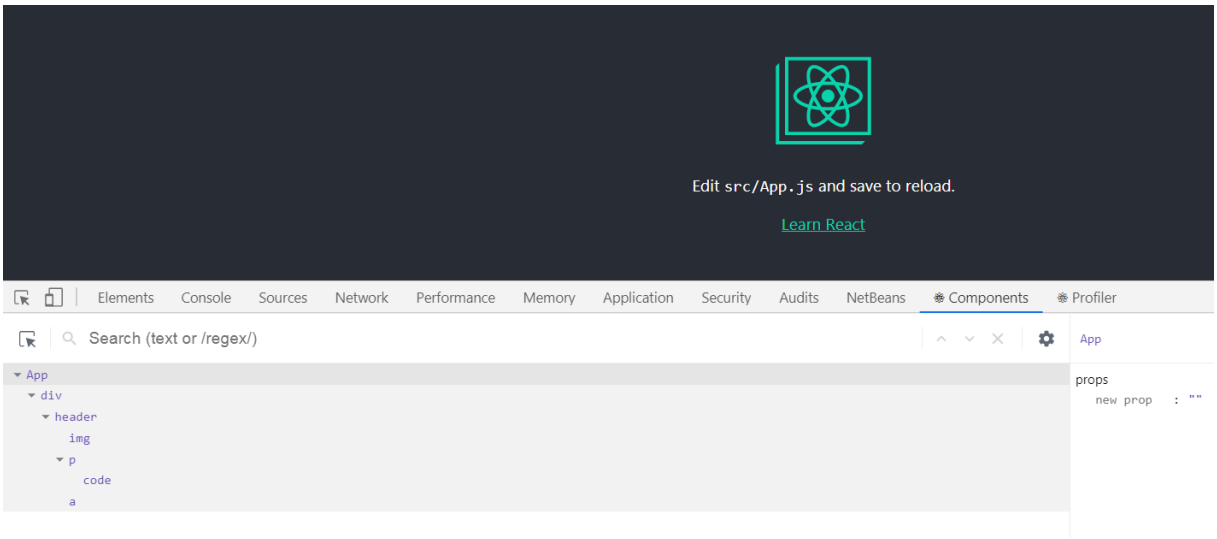
Nous pouvons tout de suite tester l'application en tapant :

```
cd ToDoList
```

puis

```
npm start
```

Un serveur node est lancé et l'application va fonctionner sur le port 3000.



On voit tout de suite le Composant ***App*** créé dans le navigateur. Ce composant est défini dans le fichier ***App.js*** dans le dossier ***source :***

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

On trouve :

- Les **imports** nécessaires au fonctionnement du composant,
- La définition du composant ***App*** en mode « ***fonction*** »,
- La fonction *return*() qui permet l'affichage du contenu,
- L'export **obligatoire** pour utiliser le composant dans un autre fichier (ici index.js)

Le fichier index.html sert de point d'entrée à l'application (https://create-react-app.dev/docs/folder-structure).

Le fichier index.js permet de monter le composant ***App*** sur le seul nœud contenu dans index.html ***<root>***

2. Définir et appeler une fonction dans un composant « fonction »

3. Les Hooks dans les composants « fonction »

Nous voyons ici que les composants sont définis par des fonctions. Dans de tels composant, nous ne pouvons pas utiliser setState pour gérer les states, nous allons utiliser le Hook d'état.

> **Remarque** : pourquoi avoir vu les composants sous formes de classe si nous devons utiliser les composants fonctions ? Simplement parce que c'est une manière de faire que nous pouvons retrouver dans des applications existantes et nous nous devons donc de connaitre la syntaxe.

Les ***Hooks*** sont une nouveauté de React 16.8. Ils permettent de bénéficier d'un état local et d'autres fonctionnalités de React sans avoir à écrire de classes. Nous nous contenterons d'étudier ici le hook d'état mais il en existe d'autres, nous pouvons également en créer.

Les Hooks sont des fonctions qui permettent de « se brancher » sur la gestion d'état local et de cycle de vie de React depuis des fonctions composants. Les Hooks ne fonctionnent pas dans des classes : ils vous permettent d'utiliser React sans classes.

```
import React, { useState } from 'react';

function Example() {
  // Déclare une nouvelle variable d'état, qu'on va appeler « count »
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Vous avez cliqué {count} fois</p>
      <button onClick={() => setCount(count + 1)}>
        Cliquez ici
      </button>
    </div>
  );
}
```

Dans le code ci-dessus, useState est un Hook (nous verrons ce que ça veut dire dans un instant). Nous l'appelons au sein d'une fonction composant pour y ajouter un état local. React va préserver cet état d'un affichage à l'autre. useState retourne une paire : la valeur de l'état actuel et une fonction qui vous permet de la mettre à jour. Vous pouvez appeler cette fonction depuis un gestionnaire d'événements, par exemple. Elle est similaire à this.setState dans une classe, à ceci près qu'elle ne fusionne pas l'ancien état et le nouveau.

Le seul argument de useState est l'état initial. Dans l'exemple précédent, c'est 0 puisque notre compteur démarre à zéro. Remarquez que contrairement à this.state, ici l'état n'est pas nécessairement un objet, même si ça reste possible. L'argument d'état initial n'est utilisé que pour le premier affichage.

Déclarer plusieurs variables d'état

Vous pouvez utiliser le Hook d'état plus d'une fois dans un seul composant :

```
function ExampleWithManyStates() {
  // Déclaration de multiples variables d'état !
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banane');
  const [todos, setTodos] = useState([{ text: 'Apprendre les Hooks' }]);
  // ...
}
```

A faire …

1. **Suivre les instructions du paragraphe 1 du document 02-React-TP.doc afin de créer notre première application ToDoList.**
2. **Remplir le paragraphe 1 du document de synthèse 02-react-synthese.doc.**

Les références avec les *hooks*

```
import React, { useRef } from 'react';
```

```
<input type="text" className="form-control" name="taskName" id="taskName" required ref={newTask} />
```

```
newTask.current.value
```

4. Installation de nouveaux modules nécessaires à l'application

Chaque module nécessaire à l'application va être installé en mode console. Par exemple, pour l'installation du module de gestion des ***routes*** (gestion des ***url*** des différents composants):

```
npm install react-router-dom --save
```

Le module installé est react-router-dom. L'instruction **- *-save*** permet d'ajouter ce module au fichier package.json qui gère toutes les dépendances de l'application.
Ce fichier va être utilisé pour réinstallé tous les modules node si nécéssaire.

Les routes

La bibliothèque JavaScript React est pensée pour vous aider à concevoir des applications monopage, comme l'est par exemple Facebook (les créateurs de React). Vous naviguez sur une seule page qui évolue selon vos actions. Il n'y a donc en principe pas de redirection vers une autre URL. Il est quand même possible de naviguer vers une autre URL grâce à un

composant de la librairie. Il existe ensuite différentes méthodes pour implémenter la redirection.

Pour pouvoir changer l'URL dans une application développée avec React, vous avez besoin d'un composant supplémentaire nommé "react-router-dom". Il vous permettra d'interagir avec le routeur de React pour modifier l'URL de la page courante, et donc effectuer une redirection.
Dans le composant le plus élevé de votre application, vous devez inclure le composant "BrowserRouter". Ce dernier vous permettra d'utiliser l'API "History" de HTML5 afin de manipuler l'historique du navigateur et ainsi changer les URL de navigation sans changer de page. Chaque routeur crée un objet « historique » pour garder une trace de l'emplacement actuel de la page.

La première solution pour effectuer une redirection est l'utilisation d'une balise <route>. Cette balise contiendra le bouton de navigation. Celui-ci permettra lors du clic de changer l'URL en cours de la page. Il est nécessaire d'utiliser une fonction de rendu pour afficher le bouton. Cela vous permet de modifier l'affichage de la façon dont vous le souhaitez. Vous pouvez utiliser cette méthode n'importe où dans votre application.

A faire …

1. **Suivre les instructions du document 02-React-TP.doc afin de terminer notre première application ToDoList.**
2. **Terminer de remplir le document de synthèse 02-react-synthese.doc.**

5. Cycle de vie des composants ReactJS

Le cycle de vie des composants est bien différent selon que l'on définit les composants en « class » ou en « fonction ».
Dans un premier temps, nous allons voir le cycle de vie d'un composant « class ».
Chaque composant de React a un cycle de vie que vous pouvez surveiller et manipuler pendant ses trois phases principales.
Ces trois phases sont les suivantes : « Monting », « Updating » et « Unmounting ». Voici les principales méthodes appelées mais il y en a d'autres.

componentDidMount()

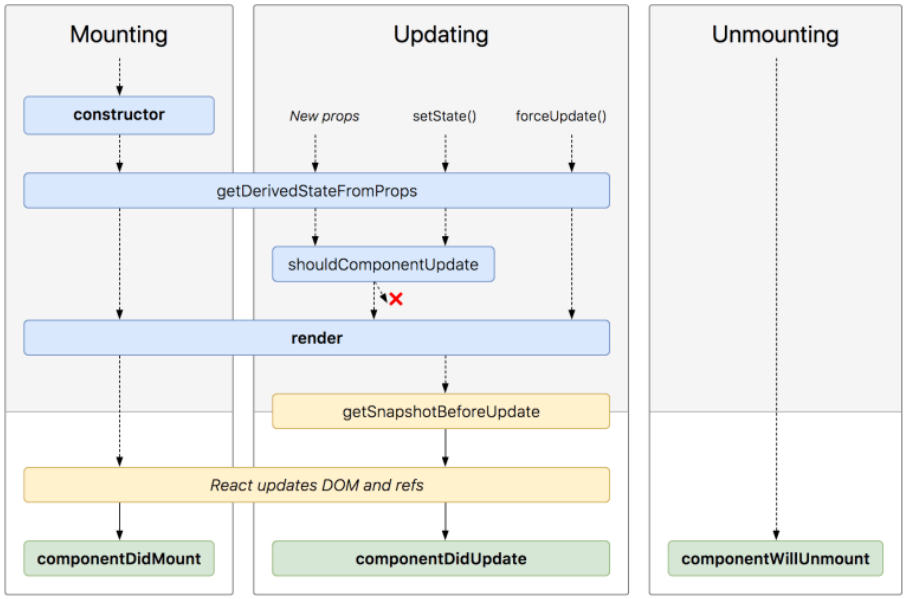
La méthode ***componentDidMount***() est appelée après le rendu du composant.
C'est là que vous lancez les instructions une fois le composant déjà placé dans le DOM. Par exemple, pour mettre du contenu via un Web Service dans un élément.

componentDidUpdate()

La méthode ***componentDidUpdate***() est appelée après la mise à jour du composant dans le DOM.

componentWillUnmount()

La phase suivante du cycle de vie est le retrait d'un composant du DOM, ou le « unmounting » comme aime l'appeler React.
React n'a qu'une seule méthode intégrée qui est appelée lorsqu'un composant est démonté : ***componentWillUnmount***()



Pour les composants « fonction », on utilisera les hooks « useEffect » :

Que fait useEffect ?
On utilise ce Hook pour indiquer à React que notre composant doit exécuter quelque chose après chaque affichage. React enregistre la fonction passée en argument (que nous appellerons « effet »), et l'appellera plus tard, après avoir mis à jour le DOM.

6. Utiliser une API avec ReactJS

Axios est l'un des clients HTTP les plus populaires pour les navigateurs et Node.js.
Il est simple, léger et facile à personnaliser. De plus, il fonctionne très bien avec React et de nombreux autres frameworks.

Axios est une librairie qui permet l'auto-conversion vers JSON. Il vous protège également par défaut contre cross-site request forgery (XSRF).
Axios s'installe dans le projet de la manière habituelle, avec la commande npm :

```
npm install axios
```

Nous avons vu auparavant que la meilleure façon de charger des données externes dans un composant React est de le faire à l'intérieur de la fonction componentDidMount() qui est exécutée juste après le montage de notre composant.
Après avoir récupéré les données, nous les ajoutons typiquement à l'état, afin qu'elles soient prêtes à être utilisées par notre application.

```
axios.get('http://localhost:3333/items')
```

Nous utilisons la méthode get() et nous passons l'URL où se trouve la ressource.
Axios fait sa magie et nous renvoie une réponse.
Si la requête est réussie, elle est passée à la méthode then() où nous l'utilisons pour définir un « state ».
Si la requête n'est pas réussie, nous récupérons une erreur qui est passée à la méthode catch() et nous pouvons l'afficher sur la console ou d'une autre manière.
Voici l'appel complet à Axios dans componentDidMount() :

```
componentDidMount() {
  axios.get('http://localhost:3333/items')
    .then(response => this.setState({items: response.data}))
    .catch(err => console.log(err))
}
```