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 1

Développeur Web et Web Mobile
Vincenzo Tinebra - AFPA - 2019

1. Présentation

ReactJS est une bibliothèque JavaScript créée par Facebook. Elle est souvent considérée comme la "vue" d'une interface utilisateur MVC (Model-View-Controller). Cela a du sens si l'on considère que la seule fonction qui doit être implémentée dans React est la fonction « render » qui fournit ce que doit afficher le navigateur.

2. Prérequis

- Connaissance de HTML et JavaScript.
- JSX doit avoir été présenté.
- Installation de l'outil « React Developer Tools » sur le navigateur

Ce document utilise les fichiers sources fournis nommés « **premiers essais** ».

3. Le DOM et le DOM virtuel de React

La modification d'un nœud du DOM demande de recharger l'ensemble de la page web (code HTML, CSS, JavaScript, ...), cela prend du temps. React propose une solution différente appelée DOM virtuel. Le DOM virtuel est une représentation rapide et en mémoire du DOM réel, et c'est une abstraction qui nous permet de traiter JavaScript et DOM comme s'ils étaient réactifs. Voyons comment ça marche :

1. chaque fois que l'état de votre modèle de données change, le DOM virtuel de React est mis à jour rapidement
2. React calcule ensuite la différence entre les deux représentations DOM virtuels : la représentation DOM virtuel **précédente** et la représentation DOM virtuel **actuelle** qui a été calculée après la modification des données. La différence entre les deux représentations est ce qu'il faut changer dans le DOM réel.
3. React met à jour uniquement ce qui doit être changé dans le DOM réel.

4. Débuter avec React

Il existe de nombreuses options pour télécharger et installer React et ses différentes dépendances, mais la façon la plus rapide de commencer à faire quelque chose avec React est de simplement utiliser les fichiers JavaScript directement depuis le CDN :

```
<head>
  <meta charset="UTF-8" />
  <title>Hello React!</title>
  <script src="https://unpkg.com/react/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
```

Comme indiqué dans le diaporama de présentation, nous avons besoin de React et de Babel. Nous allons utiliser cette structure simplifiée pour découvrir les différents concepts de React.
Nous verrons dans un deuxième temps, les outils (utilisés en mode de commande) pour installer et utiliser ReactJS. Une structure plus complexe et plus structurée nous permettra de créer tous les composants dont nous avons besoin.

Pourquoi Babel ?

Les navigateurs actuels ne savent pas interpréter le JSX et dernières syntaxes JavaScript. Babel est un outil qui est principalement utilisé pour convertir le code ECMAScript 2015+ en une version compatible de JavaScript pour les navigateurs ou environnements actuels et anciens.

5. Fonctions composants et composants à partir d'une de classe

Le moyen le plus simple de définir un composant consiste à écrire une fonction JavaScript :

```
function Welcome(props) {
  return <h1>Bonjour, {props.name}</h1>;
}
```

Cette fonction est un composant React valide, elle accepte un argument « props » (qui signifie « propriétés ») contenant des données, et renvoie un élément React. Nous appelons de tels composants des « fonctions composants », car ce sont littéralement des fonctions JavaScript.
Nous pouvons également utiliser une classe ES6 pour définir un composant :

```
class Welcome extends React.Component {
  render() {
    return <h1>Bonjour, {this.props.name}</h1>;
  }
}
```

Les deux composants ci-dessus sont équivalents du point de vue de React. La représentation fonctionnelle est cependant plus récente.
Les classes possèdent quelques fonctionnalités supplémentaires dont nous discuterons dans les prochaines sections, nous utiliserons la notation classe dans un premier temps dans cette formation.

Hello World

Entrons dans le vif du sujet et commençons par créer notre premier fichier « Hello World » :

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8" />
```

```
  <title>Hello React!</title>
  <script src="https://unpkg.com/react/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="mountNode"></div>
  <script type="text/babel">
      const mountNode = document.getElementById('mountNode');
      class MyComponent extends React.Component {
        render() {
          return (
            <h1>Hello, world!</h1>
          );
        }
      }
      ReactDOM.render(<MyComponent />, mountNode);
    </script>
</body>
</html>
```

Quelques explications :

Nous avons, pour le moment, une page html classique contenant un script de type « text/babel ». Ce script contient le code source React.

Comme nous l'avons dit, React est orienté « composant ». Nous voyons ici une syntaxe permettant de créer le composant « MyComponent » en créant une classe héritant de la classe React « React.Component ». Une fonction de rendu qui contient du JSX. Ici, seuls les instructions html sont apparentes.

Ensuite, le composant va être monté sur un nœud HTML, ici, « mountNode ».

Ouvrons notre fichier index.html dans un navigateur, nous obtenons notre « Hello World »

Un peu de JSX

Prenons le deuxième exemple :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>Hello React!</title>
  <script src="https://unpkg.com/react/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="mountNode"></div>
  <script type="text/babel">
      const mountNode = document.getElementById('mountNode');
      const message = "Hello Word ! ";
      class MyComponent extends React.Component {
        render() {
          return (
            <h1>{message}</h1>
          );
```

```
      }
    }
    ReactDOM.render(<MyComponent />, mountNode);
  </script>
</body>
</html>
```

La variable JavaScript *message* est créée dans le script et affichée dans la fonction render() à l'aide du JSX (pour plus de précision sur JSX : https://fr.reactjs.org/docs/introducing-jsx.html)

De la même manière, on peut afficher les variables objet JavaScript :

```
<script type="text/babel">
    let mountNode = document.getElementById('mountNode1');
    const message= {
      text: 'Bienvenue',
      erreur: 'Au revoir'
    };

    class MyComponent extends React.Component {
      render() {
        return (
        <div>
          <h1>{message.text}</h1>
          <h1>{message.erreur}</h1>
        </div>
        );
      }
    }

    ReactDOM.render(<MyComponent name="you" color ="blue"/>, mountNode);
  </script>
```

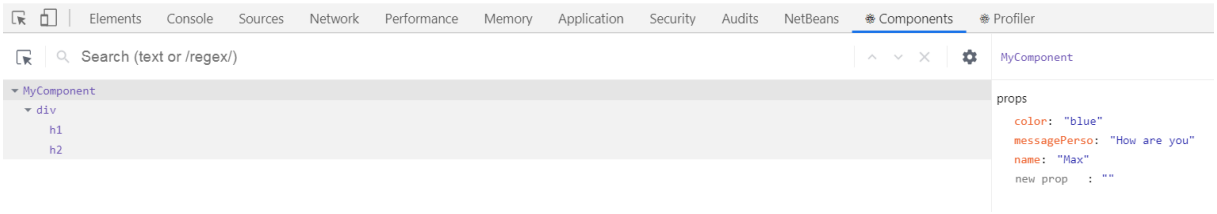
6. Les props

Les **props** (propriétés) sont des objets JavaScript **passés d'un élément parent à un élément enfant** (et non l'inverse) avec certaines propriétés qui sont considérées comme immuables, c'est-à-dire celles qui ne doivent pas être modifiées.
Lors de la création d'éléments DOM avec React, on peut passer l'objet props avec des propriétés qui représentent les attributs HTML tels que la classe, le style ou autre ...
Exemple :

```
<body>
  <div id="mountNode"></div>

  <script type="text/babel">
    const mountNode = document.getElementById('mountNode');
    class MyComponent extends React.Component {
      render() {
        return (
          <div>
            <h1 style={{color: this.props.color}}>Hello, {this.props.name}!</h1>
```

```
            <h2>{this.props.messagePerso}</h2>
          </div>
        );
      }
    }
ReactDOM.render(<MyComponent name="Max" color ="blue" messagePerso="How are you" />,
      mountNode);
  </script>
</body>
```

Si nous ouvrons notre fichier dans un navigateur sur lequel est installé « Dev Developer Tools », nous avons les informations sur notre composant *MyComponent* :



Nous retrouvons nos 3 propriétés envoyées lors de l'affichage du composant *MyComponent* :

```
ReactDOM.render(<MyComponent name="Max" color ="blue" messagePerso="How are you" />,
mountNode);
```

A faire …

1. **Ouvrir le fichier 3b et analyser le code afin de le comprendre. Poser des questions au formateur si besoin.**

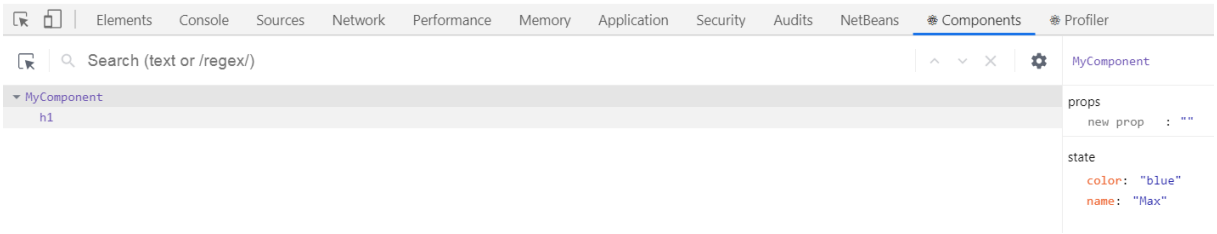
7. Les states

Un composant ReactJS possède des *propriétés* comme nous venons de le voir, elles ne sont pas amenées à varier. Il possède également un état défini par un "state", c'est à dire un état à l'instant t. Celui-ci change et entraine une mise à jour du DOM virtuel retourné en sortie.
N.B. La mise en pratique des *states* diffère selon que nous utilisons les composants créés avec des classes ou des fonctions. Dans un premier temps, nous allons voir la syntaxe et l'utilisation sur des classes.

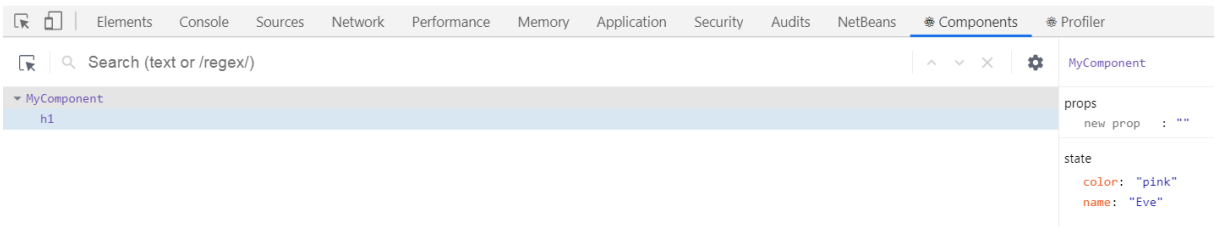
```
<body>
  <div id="mountNode"></div>

  <script type="text/babel">
      var mountNode = document.getElementById('mountNode');

      class MyComponent extends React.Component {
        constructor() {
          super();
          this.state = {name: "Max",
                        color: "blue"};
```

```
          this.changeName = this.changeName.bind(this)
        }

        changeName() {
          this.setState({name: "Eve"});
          this.setState({color: "pink"});
        }

        render() {
          return (
            <h1 style={{color: this.state.color}} onClick={this.changeName}>
              Hello, {this.state.name}!
            </h1>
          );
        }
      }

      ReactDOM.render(<MyComponent />, mountNode);
    </script>
</body>
```

Ouvrer le fichier « 4. le state » dans un navigateur. Avant de cliquer sur le texte, nous pouvons voir le composant avec ses states et ses props :



Nous voyons les 2 ***states*** créés, *name* et *color* avec leurs valeurs.
Après le clic, les 2 ***states*** changent de valeur et le DOM est mis à jour :



Avec la syntaxe « classe », les states sont initialisés dans un constructeur. On utilise la fonction setState() pour mettre à jour un state. La mise en place dans des composants fonctions est différente, nous le verrons ultérieurement.

A faire …

1. **Analyser le fichier pour comprendre chaque instruction. Poser des questions au formateur si nécessaire.**

2. **Réaliser les exercices 01 et 02 du document ReactJS Travaux Pratiques.**

8. Communication entre composants

Comme nous l'avons vu, une application React sera formée de plusieurs « composants » simples et autonomes. L'imbrication de ces composants formera une application plus complexe. Les composants doivent donc communiquer entre eux.

Imbrication de composants

Lorsque qu'un composant est créé, il peut être utilisé par un autre composant. Il est appelé par une balise au nom du composant.

```
<script type="text/babel">
      var mountNode = document.getElementById('mountNode');

      class Bonjour extends React.Component {
        render() {
          return (
            <div>
              <h1>Hello Tout Le Monde</h1>
            </div>
          );
        }
      }
      class MyComponent extends React.Component {
        constructor() {
          super();
          this.state = {textSaisi: ""}
          this.recopie = this.recopie.bind(this)
        }

        recopie() {
          this.setState({textSaisi: this.resultInput.value});
        }

        render() {
          return (
            <>
              <Bonjour/>
              <input ref={(message) => this.resultInput = message} onChange={this.recopie}
/>
              <p>{this.state.textSaisi}</p>
            </>
          );
        }
      }
```

Ici, le composant « Bonjour » est créé. Il est appelé par le composant « MyComponent » simplement par la balise `<Bonjour/>`
Deux choses à noter :

Les références

Selon docs React, Les refs fournissent un moyen d'accéder aux nœuds du DOM ou éléments React créés dans la méthode de rendu.
Dans notre exemple, la référence crée permet d'accéder à la balise *Input*, elle est stockée dans *resultInput*. On accède donc à sa valeur par `this.resultInput.value`.
Ici, on peut récupérer la valeur de l'input pour l'afficher ailleurs dans la page.

Les fragments

Dans un composant React, il est courant de renvoyer plusieurs éléments. Ces éléments doivent être regroupés, sinon une erreur est levée. Les fragments nous permettent de grouper une liste d'enfants sans ajouter de nœud (par exemple) supplémentaire au DOM.

```
return (
    <React.Fragment>
      <ChildA />
      <ChildB />
      <ChildC />
    </React.Fragment>
  );
```

Pour simplifier, comme dans notre exemple le fragment peut être noté comme ceci :

```
return (
    <>
      <ChildA />
      <ChildB />
      <ChildC />
    </>
  );
```

Communication Parent-Enfant

Maintenant que nous comprenons l'état et que nous savons les initialiser, les modifier et les afficher, nous avons besoin d'un moyen de communication entre composants. Une autre façon de le dire est que nous avons besoin qu'un enfant soit capable de parler et d'interagir avec ses parents. Le parent devra également être en mesure de transmettre les données à son enfant. Heureusement, React rend cela simple grâce aux propriétés.

Ce sont des données qui sont transmises d'un parent à ses enfants et qui sont mises à disposition par l'intermédiaire de this.props sur la composante enfant. Elles sont représentées sous forme d'attributs sur les composants du JSX que nous écrivons.

```
  <script type="text/babel">
      var mountNode = document.getElementById('mountNode');

      class InputComponent extends React.Component {
        constructor() {
          super();
          this.state = {query: ""};
```

```
        this.update = this.update.bind(this);
      }

      update() {
        this.setState({query: this.input.value});
      }

      render() {
        return (
          <>
            <input placeholder={this.props.valDefaut} ref={(input) => this.input = input}
onChange={this.update} />
            <p>{this.state.query}</p>
          </>
        );
      }
    }

    class MainComponent extends React.Component {
      constructor() {
        super();
        this.state = {name: "you"};
      }

      render() {
        return (
          <>
            <h1>
              Hello, {this.state.name}!
            </h1>
            <InputComponent valDefaut="Votre nom"/>
          </>
        );
      }
    }

    ReactDOM.render(<MainComponent color="blue" />, mountNode);
  </script>
```

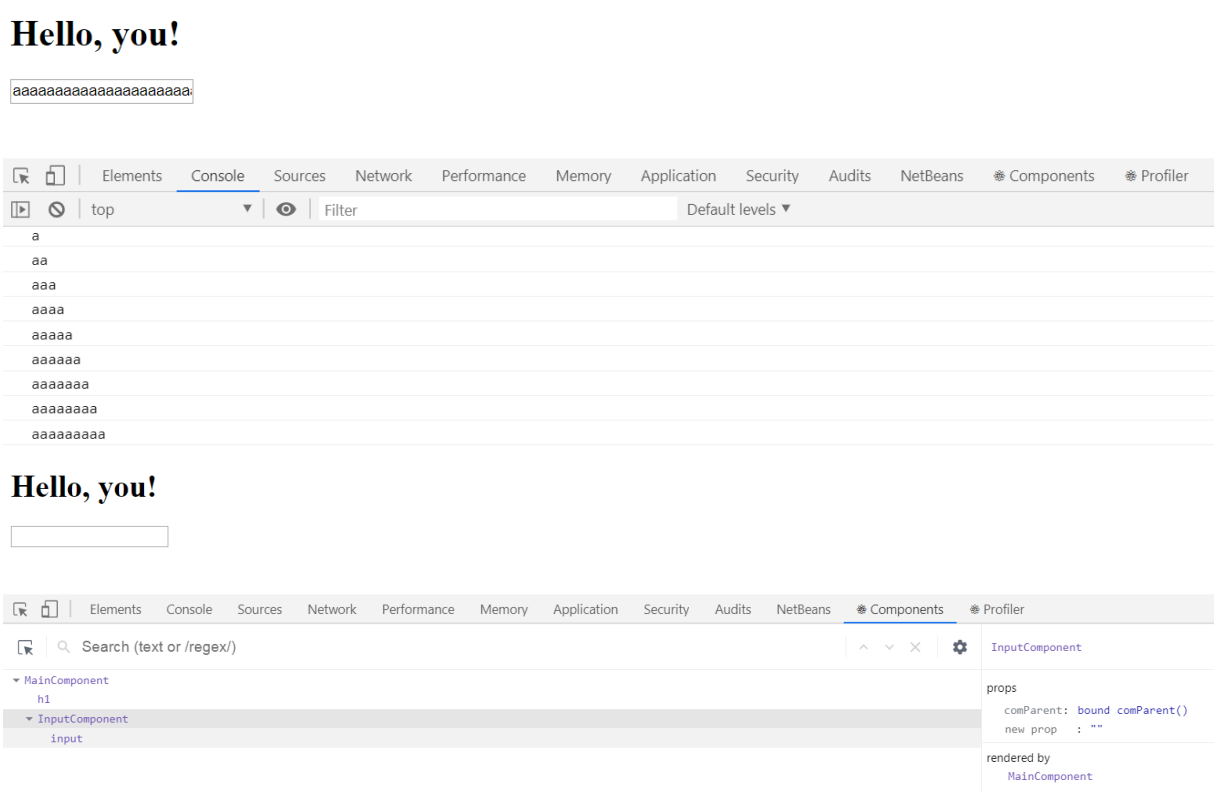
Communication Enfant- Parent

La communication entre enfants et parents est un peu plus compliquée. La façon standard de le faire est de demander au parent de passer une fonction à l'enfant par l'intermédiaire des propriétés. L'enfant appelle ensuite cette fonction à un moment donné et lui transmet une valeur à laquelle le parent est censé réagir. Nous écrivons ensuite la fonctionnalité pour la réaction du parent à l'intérieur du composant parent.

```
  <script type="text/babel">
    var mountNode = document.getElementById('mountNode');

    class InputComponent extends React.Component {
        constructor(props){
            super(props);
```

```
            this.update = this.update.bind(this);
          }

      update() {
        this.props.comParent(this.result.value);
      }

      render() {
        return (
          <>
            <input ref={(input) => this.result = input} onChange={this.update} />
          </>
        );
      }
    }

    class MainComponent extends React.Component {
      constructor() {
        super();
        this.state = {name: "you"};
        this.comChild = this.comChild.bind(this)
      }

      comChild(value) {
        console.log(value);
      }

      render() {
        return (
          <>
            <h1>
              Hello, {this.state.name}!
            </h1>
            <InputComponent comParent = {this.comChild} />
          </>
        );
      }
    }

    ReactDOM.render(<MainComponent color="blue" />, mountNode);
  </script>
```

Ici, le composant InputComponent doit communiquer avec le MainComponent pour lui envoyer la valeur saisie dans la balise *input*.
La valeur va être récupérée dans la fonction comParent(). Pour cela, elle est envoyée à l'aide des props au composant InputComponent.
La fonction update() fait appel à comParent() et envoie en paramètre, la valeur de l'input. Elle est ensuite affichée dans la console du navigateur dans le composant MainComponent.



A faire ...

1. **Réaliser l'exercice 03 du document ReactJS Travaux Pratiques.**

2. **Ecrire une fiche de synthèse sur les concepts importants vus à rendre au formateur.**

9. Les fonctions Composants

Comme nous l'avons dit en début de document, il existe deux façons de définir des composants ReactJS. Nous avons vu l'utilisation de Classe, nous allons voir l'utilisation de fonctions :

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8" />
    <title>Hello React!</title>
    <script src="https://unpkg.com/react/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>

<body>
    <div id="root"></div>
    <script type="text/babel">
        function Welcome(props) {
        return <h1>Bonjour, {props.name}</h1>;
        }
```

```
        function App() {
        return (
            <div>
                <Welcome name="Sara" />
                <Welcome name="Cahal" />
                <Welcome name="Edite" />
            </div>
                );
            }
        ReactDOM.render(
            <App />,
            document.getElementById('root')
                );
    </script>
</body>

</html>
```

Nous avons créés deux composants : App et Welcome.
App fait appel à Welcome trois fois.
Le résultat est le même, la syntaxe est cependant plus synthétique. Dans la littérature, vous trouverez les deux représentations.

A faire …

1. **Ré-écrire le code du paragraphe « 6. Les props » en créant des Composants fonctions.**
2. **Ré-écrire le code de l'exercice « 6. Tableaux » en créant des Composants fonctions.**

10. Les States et les composants « fonction »

Parce qu'un composant « fonction » n'est qu'une simple fonction JavaScript, nous ne pouvons pas utiliser setState() dans ce composant. C'est la raison pour laquelle on les appelle aussi composants fonctionnels sans état.
Heureusement, depuis React 16.8., les *Hooks* ont été créés. Ils permettent de bénéficier d'un état local et d'autres fonctionnalités de React sans avoir à écrire de classes.
Nous verrons leur utilisation dans la deuxième partie de cette formation.