TABLE DES MATIERES

1. Introduction
    a. Présentation
    b. Fonctionnement
    c. Composant
    d. JSX
    e. Props
    f. State
2. Utilisation

REACT

Version 0.1
Rédigé et mis en page par
Romain Fournier.

1.a. Introduction → Présentation

C’est quoi React ?

Une bibliothèque JavaScript open source.

Inventée par Jordan Walke, un ingénieur de Facebook en 2013.

Permet de créer des interfaces utilisateurs.

Il est possible de mettre en place une « SPA » (Single Page Application) :
Un site web qui se comporte comme une application, un chargement initial d’un document HTML, et plus besoin de requête un nouveau fichier sur le serveur (hors communication avec une BDD)

Fonctionnement →

1.a. Introduction → Fonctionnement

Orientée composant

- Un composant est un conteneur qui va retourner des éléments HTML avec ou sans expressions grâce à la syntaxe JSX.

Deux approches

- Classe : syntaxe lourde, peut gérer un état local et le cycle de vie d'un composant
- Fonction : syntaxe simple, **ne** peut **pas** gérer d'état local ni de cycle de vie

Un DOM virtuel

- Une « copie » du DOM réel est créé lors d'une mise à jour d'un state, React s'en sert pour le comparer au DOM réel et appliquer uniquement les modifications nécessaires.

La réconciliation

- Le nom de la mécanique utilisé pour comparer l'arbre de composant virtuel et avec le réel.

Les Hooks

- Fonctionnalité majeure « moderne », utiliser le hook d'état avec un composant fonctionnel lui confère une gestion d'état local
- On reconnait un hook par son préfix « use »

Composant →

1.b. Introduction → Composant

Note : Montage/démontage

Props slide 6-7-8
State slide 9

Classe

```
class Counter extends React.Component{
    constructor(props){
        super(props);
        this.state = { count: 0 }
    }

    render(){
        return (
            <div>
              <p>Count : {this.state.count}</p>
              <button onClick={() => this.setState({count: this.state.count +
1})}>        +</button>
            </div>
        );
    }
}
```

- Une classe qui hérite du module Component de React
- On est dans une classe, donc un objet, utilisation du mot-clé « this »
- Une variable state qui est un objet
- Une méthode render qui va retourner le JSX
- Nativement il y a la fonction de MàJ d'une state en classe qui se nomme « setState »

Fonction

```
function Counter(){
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>Count : {count}</p>
            <button onClick={() => setCount(count +
1)}>+</button>
        </div>
    );
}
```

- Une fonction nommée en PascalCase
- Un Hook « useState » qui retourne un Array
  - [count, setCount] est une déstructuration de Array, le 1er élément est le state, le 2nd la fonction de MàJ
  - useState(0) signifie affecte la valeur 0 au state
  - Par convention, la fonction de MàJ se préfix de « set »
  - Retourne le JSX directement

Du coup, c'est quoi du JSX ? →

1.c. Introduction → JSX

JSX pour JavaScript XML.

C’est une extension de JavaScript.

- Toute balise auto-fermante doit avoir un « / » avant le « > »
- Les attributs for et class deviennent respectivement htmlFor et className
- TOUS les attributs sont au format camelCase
- Les accolades {} permettent d’interpréter une expression

```
<img src="car.png" alt="chat roux dans son
panier" />

<label htmlFor="firstname">Firstname :</label>

<button className="btn">Add</button>

function Card(props) {
    return <h3>{props.children}</h3>
}
```

A besoin d’un compilateur (*Babel*), afin de traduire du code JS moderne en code JS plus ancien (ES5).
Ici il permet de transformer le JSX en JavaScript.

Tiens, c’est quoi ce « props » ? →

1.d. Introduction → Props

Props (pour properties) est un objet, réservé à React qui se « débloque » lors de l'utilisation d'un attribut dans un composant.

C'est une donnée rendue disponible dans un Composant enfant par son Composant parent.

Tel un attribut HTML, on va pouvoir « créer » notre propre attribut en lui donnant un nom et une valeur.

Cette paire (attribut/valeur) sera disponible dans le composant enfant en tant que nouvelle propriété dans l'objet props.

Exemple →

1.d. Introduction → Props → Exemple

```
function Wrapper(){
    return (
        <Card label="Hello from France !" />
    );
}

function Card(props) {
    return (
        <article>
            <h3>{props.label}</h3>
        </article>
    )
}
```

Wrapper retourne un composant Card.
Transmission d'une donnée à ce dernier.

A partir du moment où l'on créé cet attribut, on active l'objet props dans lequel cette paire attribut/valeur deviendra une propriété.

Dans le composant Card, on aura :

```
props = { label: "Hello from France !"}
```

D'où le chaînage props.label qui permettra d'avoir en sortie un niveau de titre 3 avec ''Hello from France ! ''

```
function Wrapper(){
    return (
        <div>
            <Card label="Hello from France !" />
            <Card label="Hello from Japan !" />
        </div>
    );
}

function Card(props) {
    return (
        <article>
            <h3>{props.label}</h3>
        </article>
    )
}
```

Je peux maintenant invoquer ce composant Card autant de fois que je veux, juste en changeant la valeur de « label », j'obtiens mon composant réutilisable !

Ces exemples démontrent des composants auto-fermants mais quid de composant avec une ouverture et une fermeture classique …

Suite →

1.c. Introduction → Props → Exemple

```jsx
function Wrapper(){
    return (
        <div>
            <Card>
                <h3>Lorem, ipsum dolor. </h3>
                <p>Lorem ipsum dolor, sit amet consectetur adipisicing elit.</p>
                <p>Voyage</p>
                <p>By : Lorem Ipsum</p>
            </Card>

            <Card>
                <h3>Lorem, ipsum dolor.</h3>
                <p>Lorem ipsum dolor, sit amet...</p>
                <p>Multimédia</p>
                <p>By : Lorem Ipsumus</p>
            </Card>
        </div>
    );
}

function Card(props) {
    return (
        <article>
            {props.children}
        </article>
    )
}
```

Il y a également une propriété native qui se débloque toute seule sous une condition.

Elle est ici dans le composant Card, « children », chainé sur props.

Elle permet de récupérer tout contenu entre la balise ouvrante et fermante d'un composant.

Ici, les « h3 » et « p ».

State →

1.e. Introduction → State

Un state est un état (variable) locale au composant dans lequel il est déclaré, on va parler « d'état local »,

Il possède un nom, une valeur et une fonction de mise à jour.

Quand un state est mis à jour, il déclenche à nouveau le montage du composant.

Voici sa syntaxe.
Pour un composant en classe

- Déclaration
- `this.state = { count: 0 }`

- Mise à jour
- `this.setState({count: this.state.count + 1})`

Pour un composant fonctionnel :

- Déclaration
- `const [count, setCount] = useState(0);`

- Mise à jour
- `setCount(count + 1)`

2. Utilisation

On peut utiliser React sur un site web classique, en complément.
On va charger les 2 scripts nécessaires depuis un CDN.

```
<!-- Le cœur de la lib' -->
<script
src="https://cdnjs.cloudflare.com/ajax/libs/react/18.2.0/umd/react.production.min.js"></script>
<!- Pour manipuler le DOM -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/react-dom/18.2.0/umd/react-
dom.production.min.js"></script>
```

Avec ça nous sommes en mesure de créer nos premières lignes de react.

```
const Text = React.createElement("p", {style: {color: "red"}}, "Lorem ipsum dolor sit amet.");
ReactDOM.render( Text, document.getElementById("root"));
```

Donne

Lorem ipsum dolor sit amet.

C'est quoi cette syntaxe ? ..
Du Javascript simple… sans JSX…
Sur la 5ème slide, on a vu qu'on avait besoin d'un « compilateur » pour transformer le JSX.

La même avec Babel →

2. Utilisation

Charger Babel

```
<!-- Compilateur Babel, pour "traduire" le JSX en JS -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
standalone/7.22.10/babel.min.js"></script>

// Composant Text en JSX
const Text = () => <p style={{color:"red"}}>Lorem ipsum dolor sit amet.</p>
ReactDOM.render( <Text/>, document.getElementById("root"));

Rappel sans JSX :
const Text = React.createElement("p", {style: {color: "red"}}, "Lorem ipsum dolor sit amet.");
ReactDOM.render( Text, document.getElementById("root"));
```

Nous allons travailler avec cette méthode la 1ère journée, ensuite on verra un outil optimisé pour la création d'application en React

HAVE FUN