React

Il a un peu picolé le Capitaine Hook, il est dans tous ses états

Lors de la quête précédente, nous avons vu comment faire passer des données entre composants parents et enfants, et utiliser ces données dans l'affichage avec nos composants. Nous avons vu aussi que lorsque les **props** d'un composants changent, le composant peut être re-rendu pour mettre à jour l'affichage avec les nouvelles données.
Pour l'instant les données sont statiques. Donc nos composants le sont aussi. C'est à dire qu'une fois que le rendu de nos pages est fait grâce aux composants, il ne changera pas.

Pour rendre notre application plus **interactive**, il faudrait pouvoir modifier le rendu de nos composants en fonction des actions utilisateurs, comme un clic sur un bouton.

Imaginons un composant *<Bouton />* qui créé en JSX un bouton HTML avec un texte « Cliquez moi ! ». Si nous voulions rendre ce composant interactif, il faudrait pouvoir modifier l'affichage du composant en fonction du clic.
Par défaut le composant afficherait le bouton. Après le clic, le composant afficherait un message de type « merci pour le clic ! ».

Essayons de mettre cela en place. La première chose que nous devons faire est de créer le composant en question et de l'appeler dans notre *App.js*. Dans le composant nous allons ensuite créer une variable *isButtonClicked,* contenant un booléen qui par défaut aura pour valeur *false*. Cette variable contient l'info de savoir si le bouton a été cliqué ou non.
En fonction de la valeur de cette variable, notre composant va afficher le bouton cliquable ou alors le message de remerciement pour le clic :

```
const Button = () ⇒ {
  const isButtonClicked = false;

  return (
    <div>
      {isButtonClicked ? (
        <p>Merci, quel beau click !</p>
      ) : (
        <button>Cliquez moi !</button>
      )}
    </div>
  );
};

export default Button;
```

Pour que notre application réagisse au click sur le bouton nous allons devoir créer un **Event Listener**. Avec React, on peut associer directement des event listeners aux éléments dans le JSX avec un attribut comme on le ferait en HTML. Il faut ensuite associer à l'event listener une fonction.

Par exemple, pour créer un event listener au click sur le bouton et appeler une fonction registerClick :

HTML + JS

```
<button onclick="registerClick()">
  Cliquez moi !
</button>
```

JSX

```
<button onClick={registerClick}>
  Cliquez moi !
</button>
```

La fonction *registerClick()* recevra en premier paramètre un **objet *Event***. En javascript vanilla (c'est à dire sans librairie donc sans React), cet objet est envoyé par le navigateur et contient de nombreuses informations et fonctionnalités sur l'évènement en question (le clic).

A noter qu'avec React l'objet reçu dans la fonction est de type **SyntheticEvent**. Cet objet est une sur-couche de l'objet **Event** du navigateur. C'est à dire qu'il contient tout ce que contient l'objet **Event** du navigateur, fonctionne à peu de chose près de la même manière, mais ajoute quelques éléments comme notamment une meilleure gestion de la compatibilité entre les navigateurs.

La fonction *registerClick()* doit donc modifier la variable *isButtonClicked* pour lui donner la valeur *true*. Idéalement, le composant devra, après le clic, afficher le message de remerciement et non plus le bouton.

Button.js

```jsx
const Button = () => {
  let isButtonClicked = false;

  const registerClick = (event) => {
    console.log("hello");
    isButtonClicked = true;
  };

  return (
    <div>
      {isButtonClicked ? (
        <p>Merci, quel beau click !</p>
      ) : (
        <button onClick={registerClick}>Cliquez moi !</button>
      )}
    </div>
  );
};

export default Button;
```

Il ne nous reste plus qu'à tester notre bouton. Au clic, le *console.log()* s'affiche bien dans la console du navigateur !
Mais… Le message de remerciement n'est pas affiché à la place du bouton !

Si le message s'affiche dans la console c'est que notre event listener a fonctionné. La valeur de la variable a donc bien été modifiée, mais l'affichage lui n'a pas changé…

Résumons ce qu'il vient de se passer :

Le composant a été rendu dans le navigateur une première fois avec la valeur de la variable à *false*.
Nous avons modifié cette valeur avec un event listener. Mais le composant n'a pas été re-rendu dans le navigateur avec cette nouvelle valeur : le composant du navigateur est donc bloqué avec la valeur de la variable à false, donc avec l'affichage du bouton et non l'affichage du message.
En d'autres termes, l'état du composant n'a pas changé.

Et même si nous parvenions à forcer le composant a être re-rendu, que se passerait t'il ?

Le composant afficherait toujours le bouton, car la valeur de la variable serait toujours à false. Pourquoi ?

Car demander à ce que le composant soit re-rendu, c'est re-executer la fonction *Button*. Et rappelez vous la notion de **scope**. Ici la variable *isButtonClicked* est déclarée localement dans la fonction. Elle n'existe que durant l'exécution de la fonction. A chaque appel de la fonction, la variable est donc re-créée et son ancienne valeur détruite.

Comment faire donc pour pouvoir créer une variable dans un composant, modifier sa valeur à la suite d'une action utilisateur (comme un clic), demander à React de re-rendre le composant tout en conservant la valeur de l'ancienne variable ? En d'autres termes, **comment demander à React de modifier l'état d'un composant et de le re-rendre** ?

C'est justement le but du « **hook** » ***useState()***.

Qu’est ce qu’un **hook** ?

C’est tout simplement une **fonction javascript,** qui permet d’utiliser des fonctionnalités de React. Ces fonctions de hook commencent généralement par « use ».

Ici le hook *useState()* permet de créer un **état local pour un composant et de le conserver.** C’est à dire de pouvoir créer une variable qui enregistre une valeur qui est conservée même quand le composant est re-rendu. Cette variable est nommée **variable « d’état »**. Le petit plus ?
La modification de la valeur de la variable d’état va automatiquement entrainer un re-render du composant en conservant la modification de valeur de la variable.

En d’autres termes, c’est exactement ce que l’on cherchait à faire !

Modifions donc notre composant pour utiliser le hook *useState()* pour créer une variable d’état *isButtonClicked*.

```jsx
import { useState } from "react";

const Button = () => {
  const [isButtonClicked, setIsButtonClicked] = useState();

  const registerClick = (event) => {
    setIsButtonClicked(true);
  };

  return (
    <div>
      {isButtonClicked ? (
        <p>Merci, quel beau click !</p>
      ) : (
        <button onClick={registerClick}>Cliquez moi !</button>
      )}
    </div>
  );
};

export default Button;
```

Décomposons l'utilisation l'utilisation du hook *useState()* dans le composant :

Nous importons d'abord la fonction *useState()* depuis la librairie React.
useState() est ensuite utilisé pour créer la variable *isButtonClicked* qui est donc une variable d'état.
Mais *useState()* retourne un tableau avec deux éléments : la valeur actuelle de la variable d'état et une fonction pour mettre à jour cette valeur. Par convention, la fonction qui permet de modifier la valeur commence par « set ». Nous pouvons donc lire la valeur de la variable d'état et la modifier.

En paramètre de *useState()*, on peut passer la valeur par défaut de notre variable *isButtonClicked*. Ici c'est false.

On utilise généralement la syntaxe de *de-structuration* pour récupérer directement les deux éléments du tableau retourné par *useState()* :

Reprenons donc le processus d'affichage de notre composant, cette fois avec l'utilisation d'un état local (avec le hook *useState()*) :

- Le composant est appelé une première fois. On utilise le hook *useState()* pour créer la variable d'état *isButtonClicked* (en lui donnant une valeur par défaut) ainsi qu'une fonction *setIsButtonClicked()* pour pouvoir modifier la valeur de la variable.
- Le composant retourne le JSX qui utilise une condition pour l'affichage : ici *isButtonClicked* est false, donc le JSX créé le bouton
- Au click sur le bouton, l'event listener est déclenché : la fonction registerClick est donc appelée. On utilise la fonction *setIsButtonClicked()* pour modifier la valeur de la variable d'état.
- La modification de la variable d'état entraine un re-render du composant. La fonction Button est donc re-executée.
- On utilise le hook *useState()*. La variable d'état *isButtonClicked* est re-créée et permet cette fois de récupérer la valeur mise à jour lors du render précédent. *isButtonClicked* est donc égale à *true*.
- Le composant retourne le JSX, qui cette fois créé le message de remerciement

useState() nous permet donc de maintenir la valeur de la variable, c'est à dire un état, entre les render.