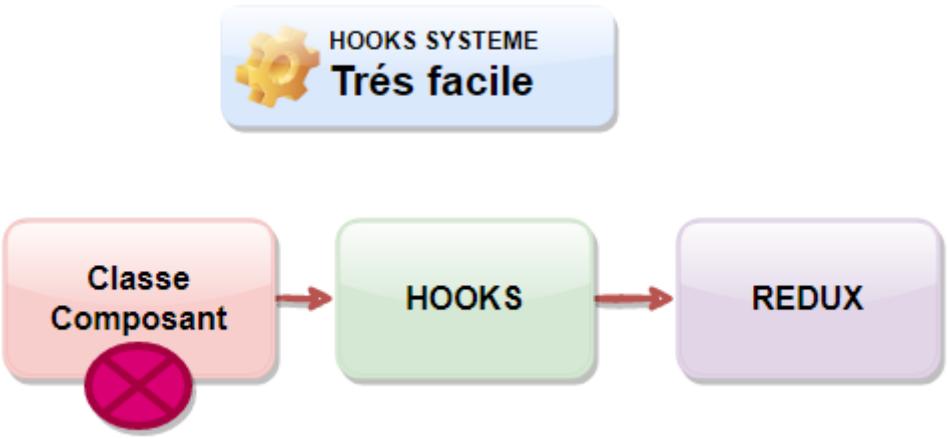
12. Incorporer des données dans une application React avec les hooks



12.1. Utilisation des hooks pour gérer l'etat de composant créé par fonction

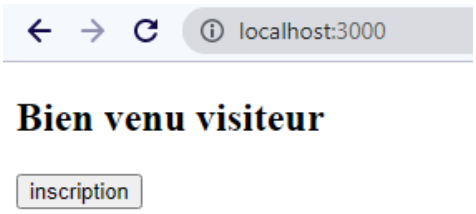
Les Hooks ont été ajoutés à React dans la version 16.8.

Les Hooks permettent aux composants fonctionnels d'accéder à l'état et à d'autres fonctionnalités de React. Pour cette raison, les composants de classe ne sont généralement plus nécessaires.

Bien que les Hooks remplacent généralement les composants de classe, il n'est pas prévu de supprimer des classes de React.

Reprenant l'exemple précèdent en utilisant cette fois ci un composant AutreMessage créé par fonction

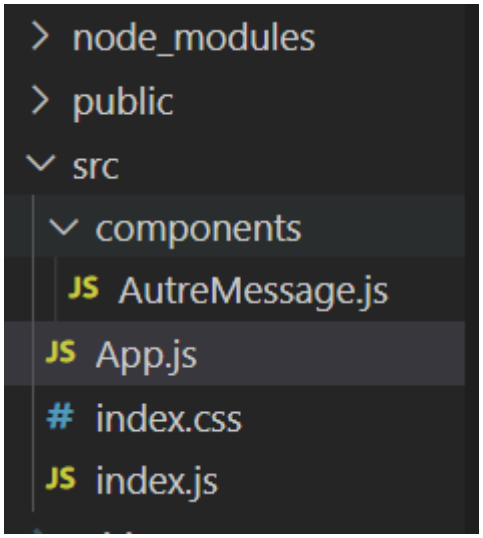
Pour expliquer l'objet state, considérons que nous souhaitions écrire un composant Message simple qui permet d'afficher le rendu suivant



Si l'utilisateur clique sur le bouton inscription

Le rendu devient

Vous allez remarquer que après le clic sur le bouton le message Bien Venu visiteur change pour devenir votre inscription est effectuée, le texte de bouton inscription change pour devenir merci



Créer le fichier AutreMessage.js dans le dossier src/components

AutreMessage.js

```
import React, { useState } from "react";
export default function AutreMessage() {

  const [message, setMessage] = useState("Bien venu visiteur");
  const [btnMessage, setBtnMessage] = useState("inscription");

  function inscription() {
    setMessage("votre inscription est effectuée");
    setBtnMessage("merci");
  }

  return (
    <div>
      <h2>{message}</h2>
      <button onClick={() => inscription()}>{btnMessage}</button>
    </div>
  );
}
```

App.js

Le fichier App est créer directement dans le dossier src

```
import React from 'react'
import AutreMessage from './components/AutreMessage';
export default function App() {
    return (
      <div>
        <AutreMessage/>

      </div>
    );
  }
```

index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
const root=ReactDOM.createRoot(document.getElementById("root"))
root.render(<App/>);
```

Vous allez remarquer que cette fois ci on a créé le composant App dans fichier App.js

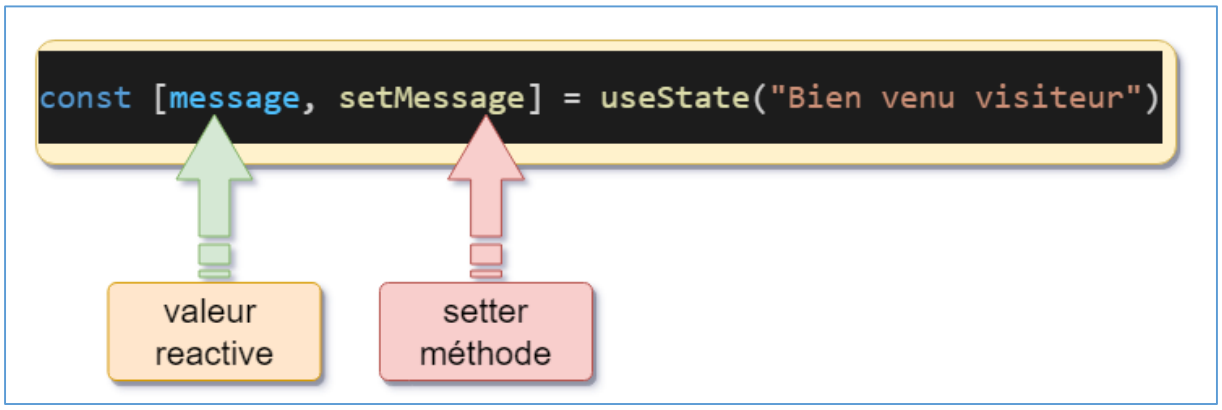
explication du code de composant fonctionnel AutreMessage

Pour utiliser le Hook useState il faut l'importer

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

Pour cet exemple on a besoin de gérer l'état de message et le texte du bouton.

```
const [message, setMessage] = useState("Bien venu visiteur");
```



Le hook useState est une fonction fournie par REACT qui vous permet d'ajouter les fonctionnalités de React dans votre composant fonctionnel.

Le hook useState est une fonction qui retourne un Array contenant deux éléments

Le premier élément c'est la propriété message que l'on peut utiliser dans le composant fonctionnel, la propriété message est initialisée par l'argument de la fonction useState, dans ce cas message est initialisé par "Bien venu visiteur".

Le deuxième élément retourné c'est la fonction setMessage.
la méthode setMessage permet de changer la valeur de la propriété message puis lance un rafraîchissement de user interface.

Pour cette écriture

```
<button onClick={() => inscription()}>{btnMessage}</button>
```

Vous allez remarquer qu'on a utilisé un arrow fonction qui fait appel à la fonction inscription après chaque événement click du bouton.

On peut utiliser aussi un callBack fonction

```
<button onClick={inscription}>{btnMessage}</button>
```

```
function inscription() {
    setMessage("votre inscription est effectuée");
    setBtnMessage("merci");
  }
```

```
    setMessage("votre inscription est effectuée");
```

setMessage modifie la valeur de la propriété message puis déclenche un rafraîchissement de l'interface, de même pour setBtnMessage

Attention si on fait appel de la fonction inscription directement

```
<button onClick={inscription()}>{btnMessage}</button>
```

Vous allez avoir ce message d'erreur sur le console

```
Uncaught Error: Too many re-renders. React limits the number of renders to prevent an infinite loop.
    at renderWithHooks (react-dom.development.js:16317:1)
    at updateFunctionComponent (react-dom.development.js:19588:1)
    at beginWork (react-dom.development.js:21601:1)
    at HTMLUnknownElement.callCallback (react-dom.development.js:4164:1)
    at Object.invokeGuardedCallbackDev (react-dom.development.js:4213:1)
```

Ceci est dû a une boucle infinie, en effet la fonction inscription est appelée au chargement de l'interface, cette même fonction inscription contient

Les instructions

```
setMessage("votre inscription est effectuée");
setBtnMessage("merci");
```

qui a leurs tours déclenchent un ré-render de l'interface,et voilà on est dans une boucle infinie

Remarque : Attention Il ne faut pas changer directement les propriétés d'états !!!!

```
let [message, setMessage] = useState("Bien venu visiteur");
let [btnMessage, setBtnMessage] = useState("inscription");

function inscription() {
  message="votre inscription est effectuée";
  btnMessage="merci";
  console.log(message);
  console.log(btnMessage);
}
```

On a changé const par let

Puis dans la méthode inscription on a changé directement les propriétés d'Etats

On remarque que après le click l'interface n a pas changé

Le rendu du console :

```
votre inscription est effectuée
merci
```

Donc il faut faire Attention !!!!!

Exercice d'application

Réalisation d'un compteur

Le click sur le bouton incrementer incrémente le compteur

Le click sur le bouton decrementer décrémente le compteur

La valeur du compteur doit être affiché

D'où on a besoin d'une propriété d'état on va la nommée valeur

Composant Compteur.js

```
import React, { useState } from "react";
export default function Compteur(){
const [valeur,setValeur]=useState(0)
    return(
        <div>
            <h1>compteur: {valeur}</h1>
            <input type="button" value="incrementer"/>
            <input type="button" value="decrementer"/>

        </div>
    )
}
```

Index.js

```
import React from "react"
import ReactDOM from "react-dom/client";
import Compteur from "./Compteur";
const root=ReactDOM.createRoot(document.getElementById("root"))
root.render(<Compteur/>);
```

On remarque que l'interface affiche la valeur 0 car

```
const [valeur,setValeur]=useState(0)
```

La propriété valeur est initialisé par 0

Maintenant on gère les événements clicks des deux boutons

le code devient :

```
import React, { useState } from "react";
export default function Compteur(){
const [valeur,setValeur]=useState(0)

function incrementer(){
    setValeur(valeur+1)
}
function decrementer(){
    setValeur(valeur-1)
}
    return(
        <div>
            <h1>compteur: {valeur}</h1>
            <input type="button" value="incrementer" onClick={incrementer}/>
            <input type="button" value="decrementer" onClick={decrementer}/>

        </div>
    )
}
```

On peut l'écrire autrement en utilisant un arrow fonction :

```
import React, { useState } from "react";
export default function Compteur(){
const [valeur,setValeur]=useState(0)
const incrementer=()=>{setValeur(valeur+1)}
const decrementer=()=>{setValeur(valeur-1)}
    return(
        <div>
            <h1>compteur: {valeur}</h1>
            <input type="button" value="incrementer" onClick={incrementer}/>
            <input type="button" value="decrementer" onClick={decrementer}/>

        </div>
    )
}
```

Ou encore

```
import React, { useState } from "react";
export default function Compteur(){
const [valeur,setValeur]=useState(0)

    return(
        <div>
            <h1>compteur: {valeur}</h1>
<input type="button" value="incrementer" onClick={()=>{setValeur(valeur+1)}}/>
<input type="button" value="decrementer" onClick={()=>{setValeur(valeur-1)}}/>

        </div>
    )
}
```

C’est à vous de choisir ;)

Si on assimile la logique des états (state) en React ça va nous permettre d’ouvrir les portes de React

12.2. useEffect :

Le Hook useEffect, ajoute la possibilité d'effectuer des effets secondaires à partir d'un fonctionnel composant . Il a le même objectif que componentDidMount, componentDidUpdate et componentWillUnmount dans les classes React, mais unifié en une seule API.

useEffect s'exécute après le rendu initial et après chaque mise à jour, donc par défaut, il est cohérent avec ce qu'il a rendu et vous pouvez désactiver ce comportement si vous le souhaitez pour des raisons de performances et/ou si vous avez une logique spéciale.

useEffect est déclaré à l'intérieur du composant, pour ce la useEffect nous donne un accès aux variables d'état.

Si nous voulons utiliser un useEffet de notre composant, importer **useEffect** à partir de **react**

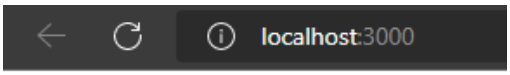
```
import React ,{useEffect, useState} from "react";
```

et en suite nous demandons à React quoi faire après chaque changement

```
useEffect(()=>{document.title=nom;},[nom])
```

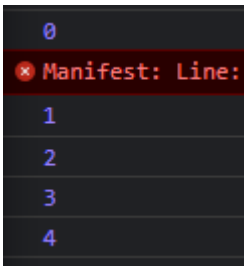
Exemple 1 :

Prenant l'exemple de compteur, on souhaite afficher sur la console la valeur de compteur. pour ce faire, le code doit être exécuté après chaque rendu de l'interface, le bon emplacement du code sera dans useEffect



compteur: 4

incrementer decrementer



Le code :

```
import React, { useEffect, useState } from "react";
export default function Compteur(){
const [valeur,setValeur]=useState(0)
function incrementer(){setValeur(valeur+1)}
function decrementer(){setValeur(valeur-1)}
```

```
useEffect(()=>{console.log(valeur)})
    return(
        <div>
            <h1>compteur: {valeur}</h1>
            <input type="button" value="incrementer" onClick={incrementer}/>
            <input type="button" value="decrementer" onClick={decrementer}/>
        </div>)
}
```

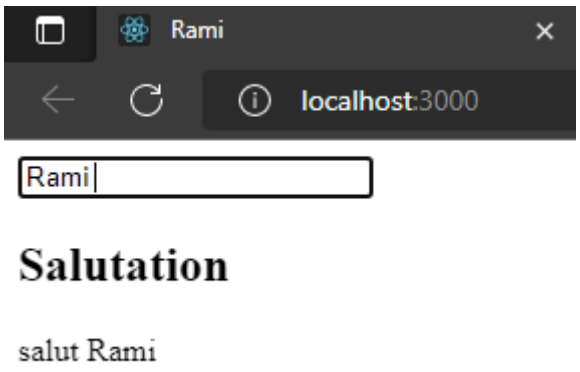
On peut aussi utiliser useEffect avec le deuxième argument :

```
useEffect(()=>{console.log(valeur)},[valeur])
```

comme ça on est sure que l'exécution se fait seulement si la valeur est modifiée.

Exemple 2 :

Le titre de document va prendre la valeur du nom après chaque modification du nom



Le composant fonctionnel Salutation permet d'afficher automatiquement le Nom dans le paragraphe en bas puis aussi dans le titre du document.

Le nom est modifié par l'élément input

Indes.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
const root=ReactDOM.createRoot(document.getElementById("root"))
root.render(<App/>);
```

App.js

```
import React from 'react'

import Salutation from './components/Salutation';
export default function App() {
    return (
      <div>
```

```
        <Salutation/>
      </div>
    );
  }
```

Salutation.js

```
import React ,{useEffect, useState} from "react";

export default function Salutation(props){
const [nom,setNom]=useState("Rami")
function changeNom(e){
setNom(e.target.value)
}
useEffect(()=>{document.title=nom;},[nom])
    return(
        <div>
            <input type="text"  value={nom} onChange={changeNom}></input>
            <h2>Salutation</h2>
            <p>salut {nom} </p>
        </div>
    )
}
```

```
useEffect(()=>{document.title=nom;},[nom])
```

useEffect est exécutée en passant en argument un arrow fonction qui contient le code qui va être executé, Le deuxième argument est un Array optionnel ,si il est omis useEffect s'exécute après le rendu initial et après chaque mise à jour de tous les propriétés states .

Si le deuxième argument est renseigné, useEffect s'execute après le rendu initial et après la modification seulement des propriétés qui se trouvent dans l'Array

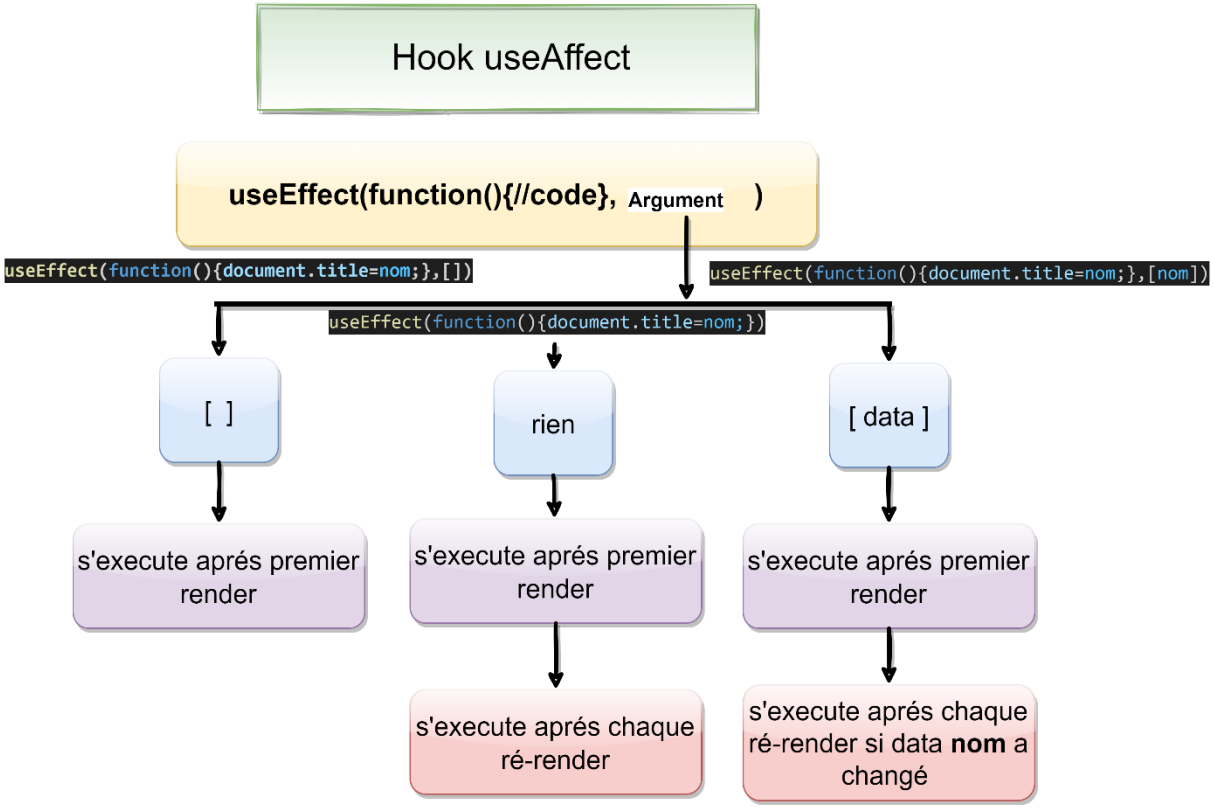
Dans notre exemple

```
useEffect(()=>{document.title=nom;},[nom])
```

useEffect s'exécute après le rendu initial et après la modification seulement de la propriété nom

d'où le code **document.title=nom**; va être exécuté que si la propriété nom est modifiée

Les différents comportements de useEffect selon le deuxième argument



Argument peut être :

- ✓ []
- ✓ [prop1,prop2…..]
- ✓ rien

Exercice d'application

On reprend l'exercice précèdent, en ajoutant le prenom et l'âge cette fois ci on souhaite que le titre de la page prend les valeurs nom et le prenom
Au chargement

Le rendu est :

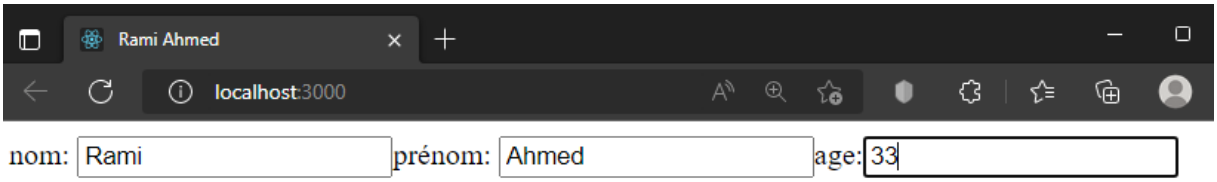
Les propriétés d'états prennent les valeurs d'initialisation

```
const [nom,setNom]=useState("---")
```

```
const [prenom,setPrenom]=useState("---")
const [age,setAge]=useState(0)
```

On remarque que le titre du document prend aussi les valeurs initiales

Après la saisie des informations on a le rendu suivant :



Salutation

salut Rami Ahmed vote age est:33

Le titre du document est mis à jour aussi le message est mis à jour

Le code du composant Salutation

```
import React ,{useEffect, useState} from "react";

export default function Salutation(props){
const [nom,setNom]=useState("---")
const [prenom,setPrenom]=useState("---")
const [age,setAge]=useState(0)
function changeNom(e){
    setNom(e.target.value)
}
function changePrenom(e){
    setPrenom(e.target.value)
}
function changeAge(e){
     setAge(e.target.value)
}
useEffect(function(){document.title=nom+" "+prenom;},[nom,prenom])
    return(
        <div>
 <label>nom:</label>
<input type="text"  value={nom} onChange={changeNom}></input>

<label>prénom:</label>
<input type="text"  value={prenom} onChange={changePrenom}></input>
<label>age:</label>
<input type="text"  value={age} onChange={changeAge}></input>
 <h2>Salutation</h2>
 <p>salut {nom} {prenom} vote age est:{age}</p>
 </div>
    )
}
```

Analysant ce code

```
useEffect(function(){document.title=nom+" "+prenom;},[nom,prenom])
```

Deuxième argument contient l'Array [nom,prenom]

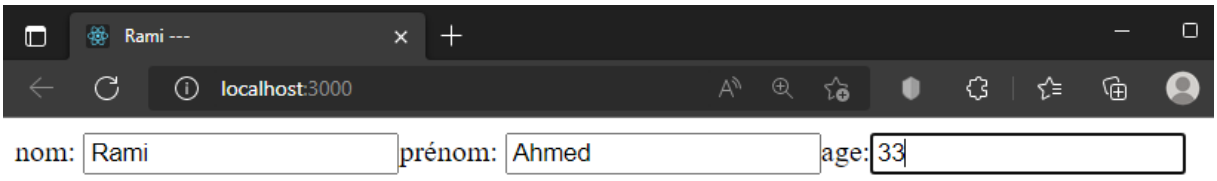
Dans cette situation le deuxième argument de useEffect est un Array contenant les propriétés d'états nom et prenom,par conséquence useEffect s'exécute au premier rendu et a chaque modification des propriétés d'états nom et prenom

Maintenant si on utilise cette écriture

```
useEffect(function(){document.title=nom+" "+prenom;},[nom])
```

Deuxième argument contient [nom]

Le rendu après avoir renseigné les informations



Salutation

salut Rami Ahmed vote age est:33

UseEffect est exécutée au premier rendu et seulement au changement du nom,car le deuxième argument de useEffect est [nom].

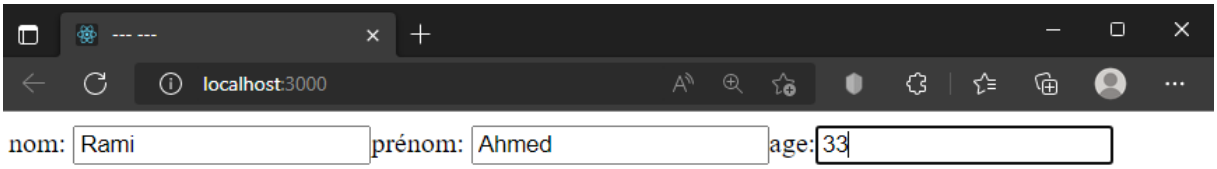
Par contre le message est toujours mis à jour

Regardons maintenant si on utilise cette écriture

```
useEffect(function(){document.title=nom+" "+prenom;},[])
```

Deuxième argument contient un Array vide

Le rendu relatif à cette écriture après avoir renseigné les informations :



Salutation

salut Rami Ahmed vote age est:33

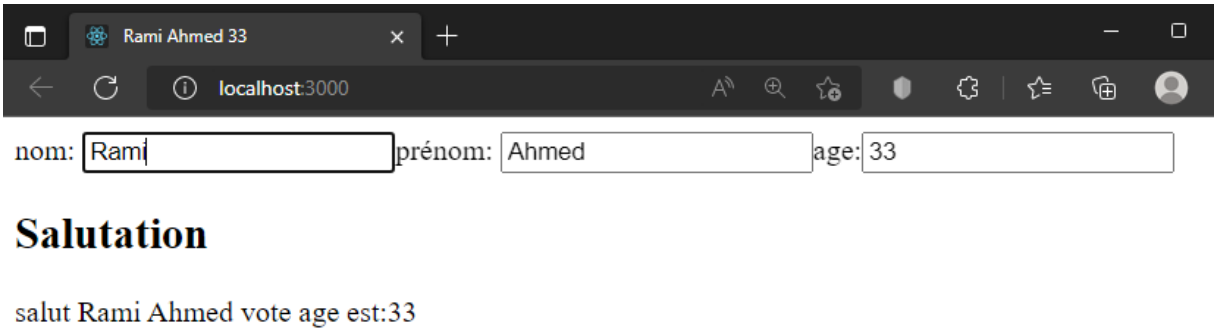
Vous allez remarquer que useEffect est exécuté seulement au premier rendu,

En fin si on utilise cette écriture :

```
useEffect(function(){document.title=nom+" "+prenom+" "+age;})
```

useEffect sans deuxième argument

Le rendu relatif à cette écriture après avoir renseigné les informations :



Vous allez remarquer que useEffect maintenant est executé au premier rendu et a tous les mis à jour des propriétés d'état nom, prenom et age

Quiz :

Question 1 :

Jetez un œil au code suivant. Après l'avoir exécuté, combien de console.log vous attendriez-vous à voir, et quand les verriez-vous ?

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

import ReactDOM from 'react-dom';

const App = () => {

  useEffect(() => {

    console.log('TEST!');

  }, []);

  return <div>test composant</div>;

};

ReactDOM.render(<App />, document.querySelector('#root'));
```

a- Je vais voir un log de 'TEST', il est affiché après le composant est rendu
b- Je ne vais voir aucun log
c- Je vais voir un log de 'TEST', il est affiché avant le composant est rendu
d- Je vais voir deux log de 'TEST',ils sont affichés après le composant est rendu

Question 2 :

Jetez un œil au code suivant. Imaginez qu'il soit exécuté, puis qu'un utilisateur a cliqué trois fois sur l'élément bouton. Combien d'instructions de log 'TEST' vous attendriez-vous à voir imprimées ?

```
import React, { useEffect, useState } from 'react';
import ReactDOM from 'react-dom';
const App = () => {
const [count, setCount] = useState(0);

 useEffect(() => {
console.log('TEST!');
}, []);

 const onClick = () => {
setCount(count + 1); };

  return (
<div>
 <h1>Count: {count}</h1>
<button onClick={onClick}>Click me!</button>
</div>
);};

ReactDOM.render(<App />, document.querySelector('#root'));
```

a- Je vais voir trois console.log 'TEST'
b- Je vais voir quatre console.log 'TEST'
c- Je vais voir un console.log 'TEST'

Question 3 :

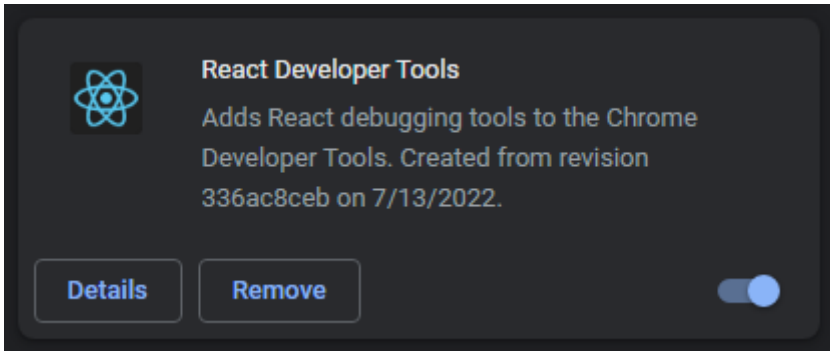
Jetez un œil à l'extrait de code suivant. C'est identique à la dernière question, sauf que maintenant useEffect a un tableau avec un seul argument à l'intérieur.

Combien de fois vous attendez-vous à voir l'instruction de console.log 'TEST' après qu'un utilisateur a cliqué trois fois sur le bouton ?

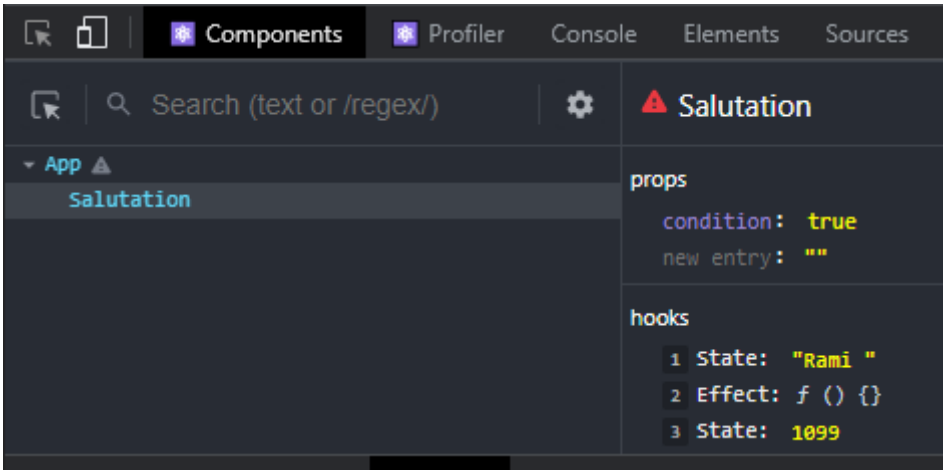
```
import React, { useEffect, useState } from 'react';
import ReactDOM from 'react-dom';

const App = () => {
const [count, setCount] = useState(0);

 useEffect(() => {console.log('TEST!');

  }, [count]);

  const onClick = () => {
   setCount(count + 1);};

return (
<div>
<h1>Count: {count}</h1>
<button onClick={onClick}>Click me!</button>
</div>)}
ReactDOM.render(<App />, document.querySelector('#root'));
```

a- Je vais voir un console.log 'TEST'
b- Je vais voir deux console.log 'TEST'
c- Je vais voir trois console.log 'TEST'
d- Je vais voir quatre console.log 'TEST'

L'extension React Dévelope Tools, permet de faire le debugging



Voir le volet Component



12.3. Utilisation de useEffect pour charger les données d'un API

Le chargement des données provenant d'un API se fait seulement au chargement de user interface c-à-d au premier rendu du composant

Soit le endPoint suivant :

https://jsonplaceholder.typicode.com/posts

qui retourne un array des posts sous la forme

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati exceptur",
    "body": "test"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "test2"
  },……..]
```

Soit l'interface

App.js

```
import React, { useEffect, useState } from "react";
//import usePosts from './hooks/usePosts'
export default function App(){
const [posts,setPosts]=useState([])
useEffect(()=>{
    fetch('https://jsonplaceholder.typicode.com/posts')
    .then(res=>res.json())
    .then(response=>setPosts(response))
},[])
return (<div>
    <h1>liste des posts à partir d'un API</h1>
    <ul>
    {posts.map(post=>{
        return (<li key={post.id}>{post.title}</li>)
    })}
    </ul>
</div>)
}
```

Index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
const root=ReactDOM.createRoot(document.getElementById("root"))
root.render(<App/>);
```

Comme vous remarquez on fait l'appel de fetch dans la fonction callback de useEffect,le deuxième argument de useEffect contient [],de ce fait fetch est exécuté seulement au premier chargement de user interface, par conséquent si on effectue des mise à jour de propriétés state, fetch ne sera pas réexécuté.

12.4. Construire vos propres Hooks

Les customs Hooks ou bien les Hooks personnalisés sont des fonctions réutilisables.

Lorsque vous avez une logique de composant qui doit être utilisée par plusieurs composants, nous pouvons extraire cette logique dans un Hook personnalisé.

Les Hooks personnalisés commencent par convention par "use". Exemple : usePosts.

Dans l'exemple précèdent la logique de récupération des Posts par fetch peut être utilisé par plusieurs composants, pour ce faire on externalise cette logique dans Hooks usePosts.

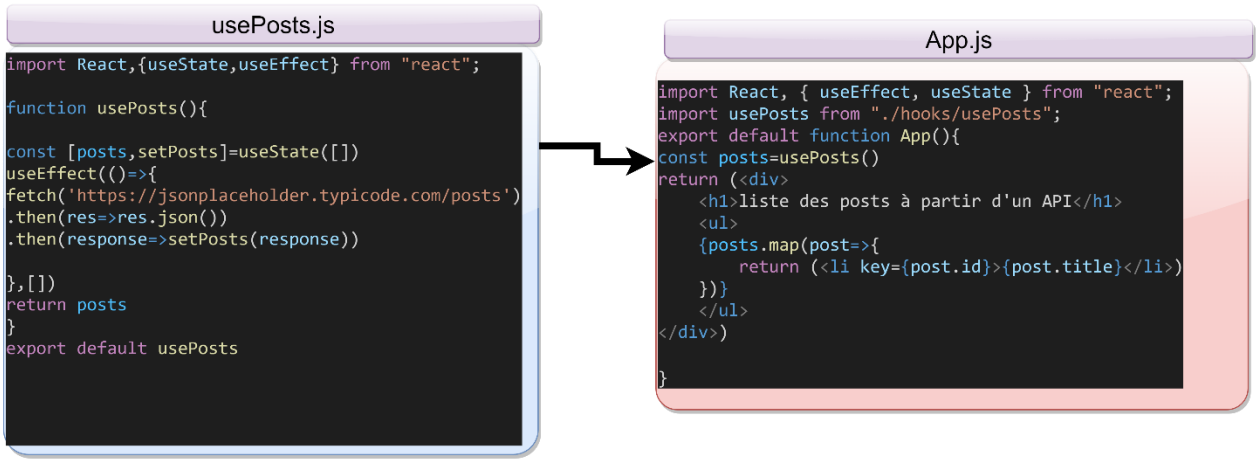
Par convention on met le fichier usePosts.js dans le dossier hooks

usePosts.js

```
import React,{useState,useEffect} from "react";

function usePosts(){
const [posts,setPosts]=useState([])
useEffect(()=>{
fetch('https://jsonplaceholder.typicode.com/posts')
.then(res=>res.json())
.then(response=>setPosts(response))

},[])
return posts
}
export default usePosts
```

App.js

```
import React, { useEffect, useState } from "react";
import usePosts from "./hooks/usePosts";
export default function App(){
const posts=usePosts()
return (<div>
    <h1>liste des posts à partir d'un API</h1>
    <ul>
    {posts.map(post=>{
        return (<li key={post.id}>{post.title}</li>)
    })}
    </ul>
</div>)
}
```




```
import usePosts from "./hooks/usePosts";
```

```
const posts=usePosts()
```

Posts fait référence à posts retourné par la fonction **usePosts** du hook usePosts.js

Par conséquent chaque composant qui nécessite cette logique peut utiliser ce hook usePosts