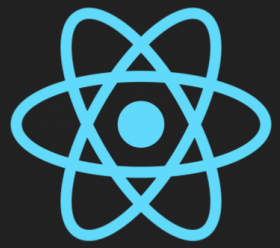


REACT live 5

Formateur :
TAUPIN Jérémy

State React

L'état (state) d'un composant React est une propriété qui contient des données qui peuvent être utilisées pour gérer la logique de l'application et afficher des données dans la vue.

Il est généralement défini dans le constructeur d'un composant en utilisant la méthode setState pour le mettre à jour.

L'état peut être utilisé pour stocker des informations telles que les données de formulaire, les données de l'application ou les variables de l'interface utilisateur.

Il est important de noter que l'état ne doit jamais être modifié directement, il faut utiliser setState pour mettre à jour l'état afin de garantir que la vue est correctement mise à jour.

L'état peut être partagé entre les composants en utilisant des contextes ou en passant des propriétés (props) entre les composants.

Incorporer un state dans un composant par Class

Dans un composant déclaré via une class, le state est déclaré avec les props dans le constructeur (constructor) grâce à la valeur this.state

Les modifications de valeurs quant à elle , sur le state se font ensuite avec la fonction setState.

Ces 2 valeurs sont récupérer en étendant Component de React

```
import React, { Component } from 'react'

class MyComponent extends Component {
    constructor(props) {
        super(props)
        this.state = {
            count: 0,
            text: 'Hello World'
        }
    }

    handleClick = () => {
        this.setState(prevState => ({
            count: prevState.count + 1
        }))
    }

    handleChange = (event) => {
        this.setState({ text: event.target.value })
    }

    render() {
        return (
            <div>
                <p>Count: {this.state.count}</p>
                <button onClick={this.handleClick}>
                Increment
                </button>
                <br />
                <input
                  type="text"
                  value={this.state.text}
                  onChange={this.handleChange}
                 />
                <p>Text: {this.state.text}</p>
            </div>
        )
    }
}
export default MyComponent
```

Explication

Le constructeur prend en arguments les propriétés (props) vue dans le live précèdent.
Il va ensuite déclarer les state avec this.state (ici this correspond à la Class MyComponent)

Ici, les valeurs ne sont pas définitive, il s'agit principalement d'initialisé les variables du state qui contiendront les données et leur attribuer une valeure initiale, par exemple count aura comme valeur initial 0

Nous pouvons voir qu'as l'intérieur de la function handleClick, nous ne modifions pas directement this.state.count mais passons par la function this.setState. Nous verrons cela dans le prochain slide.
Ici setState reçois comme valeur prevState (l'état précèdent de l'état) et lui ajoute 1

L'utilisation du state se fait ensuite simplement grâce à this(la classe).state(l'état).count(la clé). ce dernier renverra la valeure actuel du state

```
import React, { Component } from 'react'

class MyComponent extends Component {
    constructor(props) {
        super(props)
        this.state = {
            count: 0,
            text: 'Hello World'
        }
    }

    handleClick = () => {
        this.setState(prevState => ({
            count: prevState.count + 1
        }))
    }

    handleChange = (event) => {
        this.setState({ text: event.target.value })
    }

    render() {
        return (
            <div>
                <p>Count: {this.state.count}</p>
                <button onClick={this.handleClick}>
                Increment
                </button>
                <br />
                <input
                  type="text"
                  value={this.state.text}
                  onChange={this.handleChange}
                 />
                <p>Text: {this.state.text}</p>
            </div>
        )
    }
}
export default MyComponent
```

Fonctionnement de React : Les Cycles de vie

Les composants React ont un cycle de vie qui décrit les différentes étapes que les composants traversent depuis leur création jusqu'à leur suppression.

Le cycle de vie des composants est important pour comprendre comment les données et les fonctionnalités sont gérées au sein d'une application React.

En résumé, le cycle de vie des composants est un processus clé pour comprendre comment ces derniers sont créés, mis à jour et supprimés dans votre application. Il permet aux développeurs de gérer efficacement les données et les fonctionnalités des composants, et de réaliser des actions spécifiques à chaque étape du cycle de vie.

Découvrons les !

1

Fonctionnement de React : Les Cycles de vie

La phase de montage

L'une des étapes clés du cycle de vie des composants React est la phase de montage. Cette phase se produit lorsque le composant est créé pour la première fois et il est utilisé pour initialiser l'état et les propriétés du composant. C'est également lors de cette phase que les méthodes de cycle de vie telles que componentDidMount() sont appelées.

Dynamisons notre application !

2

Fonctionnement de React : Les Cycles de vie

La phase de mise à jour

Une autre étape importante est la phase de mise à jour. Cette phase se produit lorsque les propriétés ou l'état du composant changent. La méthode componentDidUpdate() est appelée lors de cette phase pour permettre aux développeurs de réagir aux changements d'état et de propriétés. Les composants peuvent également utiliser la méthode shouldComponentUpdate() pour déterminer s'ils doivent être mis à jour ou non.

Libérons les ressources !

3

Fonctionnement de React : Les Cycles de vie

La phase de démontage

La phase de démontage est la dernière étape du cycle de vie des composants React. Elle se produit lorsque le composant est supprimé de la vue. Cette phase permet aux développeurs de nettoyer les ressources utilisées par le composant, tels que les écouteurs d'événements ou les timers. La méthode componentWillUnmount() est appelée lors de cette phase pour permettre aux développeurs de réaliser des actions de nettoyage.

Fonctionnement de React :
Les Cycles de vie

Le state et son utilitée

Après avoir vue ce qu'était les cycles de vie, nous pouvons expliquer pourquoi il est primordial d'utiliser le state et non une variable standard (let, const)

Fonctionnement d'un composant

L'intéraction avec le state, permet à React de savoir si le composant à reçu des modifications ou non. Cela lui permet d'agir en conséquence et mettre à jour ce dernier au besoin.

Une variable ne sera pas surveillée pour mettre à jour le composant

Agir sur le cycle de vie dans un composant par Class

Montage du composant

Avant le chargement du composant le code de la fonction ***componentWillMount*** sera exécuté

Après le chargement, il s'agira du code la fonction ***componentDidMount***

Mise à jour du composant

Avant la mise à jour du composant le code de la fonction ***componentWillUpdate*** sera exécuté

Avant la réception de nouvelle props, le code de la fonction ***componentWillReceiveProps*** sera exécuté

Après le chargement, il s'agira du code la fonction ***componentDidUpdate***

Le contexte de mise à jour du composant peut être défini avec ***shouldComponentUpdate***

Démontage du composant

Avant le démontage du composant et son retrait du DOM le code de la fonction ***componentWillUnmount*** sera exécuté

Evidemment il est possible d'agir sur le cycle de vie dans les composants, qu'ils soit par class ou par function.

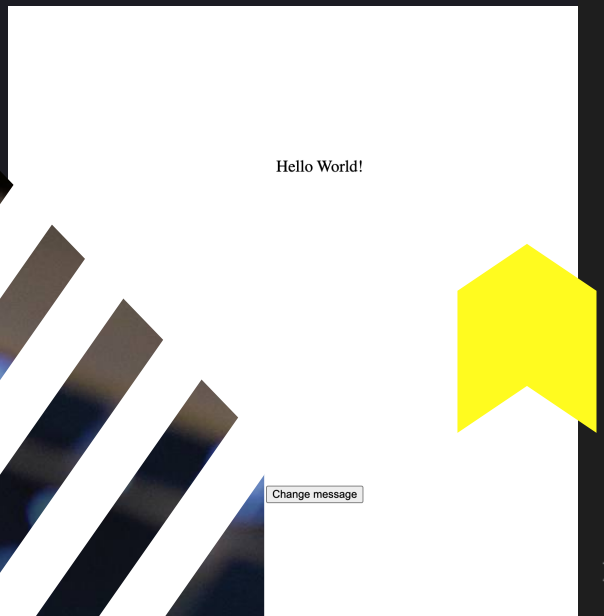
```
import React from 'react'
class MyComponent extends React.Component {
    componentWillMount() {
        // Ce bloc de code sera exécuté
        // juste avant que le composant soit monté
    }
    componentDidMount() {
        // Ce bloc de code sera exécuté dès
        // que le composant aura été monté
    }
    componentWillUnmount() {
        // Ce code sera exécuté juste avant
        // que le composant soit démonté et retiré du
DOM }
    componentWillUpdate() {
        // Ce code sera exécuté juste avant
        // que le composant soit mis à jour
    }
    componentWillReceiveProps() {
        // Ce code sera exécuté juste avant
        // que le composant soit mis à jour suite
        // au changement d'une props
    }
    componentDidUpdate() {
        // Ce code sera exécuté après
        // que le composant soit mis à jour
    }
    shouldComponentUpdate() {
        // Ce code permettra de définir dans quel
        // cas le composant doit être mis à jour
    }

}
export default MyComponent
```

class CheatSheet extends React.WWHS

Function intervenant sur le cycle de vie

```
componentWillMount() {
    // Ce bloc de code sera exécuté
    // juste avant que le composant soit monté
}
componentDidMount() {
    // Ce bloc de code sera exécuté dès
    // que le composant aura été monté
}
componentWillUnmount() {
    // Ce code sera exécuté juste avant
    // que le composant soit démonté et retiré du
DOM }
componentWillUpdate() {
    // Ce code sera exécuté juste avant
    // que le composant soit mis à jour
}
componentWillReceiveProps() {
    // Ce code sera exécuté juste avant
    // que le composant soit mis à jour suite
    // au changement d'une props
}
componentDidUpdate() {
    // Ce code sera exécuté après
    // que le composant soit mis à jour
}
shouldComponentUpdate() {
    // Ce code permettra de définir dans quel
    // cas le composant doit être mis à jour
}
```

11



Création d'une class avec state et props

```
import React, { Component } from 'react'
// Import de React avec Component (ici par destructuration)

class MyComponent extends Component {
// Déclaration du composant dans une class étendu de Component
// Le nom d'un composant débutera toujours par une majuscule
    constructor(props) {
     // Création du constructeur, recevant les props
     // Les props serviront à transmettre des informations
     //                 entre le composant parent et enfant
        super(props)
        // Appelle des props au parent
        this.state = { message: 'Hello World!' }
        // Initialisation du state avec une valeur initial
        // Le state permettra de gérer les données
    }

    handleClick = () => {
        // Création d'une fonction gérant un click
        // Le terme handle est à favoriser
        this.setState({ message: 'Hello React!' })
        // Changement d'une valeur du state avec la fonction
        //          setState à partir de this ( la classe )
    }

    render() {
      // Appel de la fonction de rendu ( render )
        return (
            // La fonction retourne alors le composant (ici une div)
            <div>
            {/*un seul élement peut être renvoyé*/}
                <p>{this.state.message}</p>
                {/*
                Insertion de la valeur du state message dans des
                balises p, ce dernier est appelé via this (la class)
                state (le state), message (la clé).

                Les éléments JS sont intégré via des accolades
                {} dans le code JSX afin de faire appel à la
                partie logique du code.
                */}
                <button onClick={this.handleClick}>
                {/*
                Appel de la fonction handleClick lors du click
                sur le button déclaré, ici la fonction est une
                référence (pas de parenthèse () ).
                Réaliser une référence évite que la fonction
                soit appelé une fois au chargement puis
                ignoré car déjà appelées.
                */}
                  Change message
                </button>
            </div>
        )
    }
}
```

Incorporer un state dans un composant par Fonction

Dans un composant déclaré via une fonction, le state est déclaré dans la fonction du composant grâce à la fonction useState de React.

Elle se déclare en 2 temps, la clé et la fonction agissant pour ses modifications.

useState prend ensuite la valeur par défaut en argument.

Les modifications de valeurs sont ensuite réalisé par la fonction défini précèdement.

useState peut être utilisé depuis l'import de React mais aussi directement par destructuration lors de l'import

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

// Ou import React from 'react'

const MyComponent = () => {
  const [message, setMessage] = useState('Hello World!')
  // Ou const ... = React.useState('Hello World!')
  const handleClick = () => {
    setMessage('Hello React!')
  }

  return (
    <div>
        <p>{message}</p>
        <button onClick={handleClick}>
          Change message
        </button>
    </div>
  )
}

export default MyComponent
```

Explication

Import de useState grâce à la destructuration depuis le package react

Initialisation du state (message) et de sa fonction (setMessage) grâce à la fonction useState, la valeure par défaut est passé en argument ('Hello World!')

Fonction appelant la fonction créée précèdement (setMessage), cette fonction modifiera le contenu du state message

Appel du state message directement par sa clé comme une variable standard

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

// Ou import React from 'react'

const MyComponent = () => {
  const [message, setMessage] = useState('Hello World!')
  // Ou const ... = React.useState('Hello World!')
  const handleClick = () => {
    setMessage('Hello React!')
  }

  return (
    <div>
      <p>{message}</p>
      <button onClick={handleClick}>
        Change message
      </button>
    </div>
  )
}

export default MyComponent
```

Agir sur le cycle de vie dans un composant Fonctionnel

Comme pour les composants de class, il est possible d'agir sur le cycle de vie dans les composants fonctionnel. Pour cela, nous utiliseront la fonction useEffect qui devrat être importé ou appelé depuis le package React.

useEffect

La fonction useEffect prend en entrée deux arguments :
- Une fonction qui décrit l'effet de bord à exécuter
- Un tableau de dépendances.

La fonction d'effet de bord est exécutée chaque fois que le composant est mis à jour, après le rendu.

Le tableau de dépendances spécifie les variables qui doivent être surveillées pour déclencher une nouvelle exécution de l'effet de bord.

Si aucun tableau de dépendances n'est fourni, l'effet de bord sera exécuté à chaque mise à jour du composant.

```
import React, {useEffect} from 'react'
const MyComponent = () => {
  useEffect(() => {
    // Ce bloc de code sera exécuté
    //  juste avant que le composant
    //  soit monté (remplace componentWillMount)
    return () => {
      // Ce code sera exécuté
      // juste avant que le composant
      // soit démonté et retiré du DOM
      // (remplace componentWillUnmount)
    }
  }, [])
  useEffect(() => {
    // Ce code sera exécuté après avoir
    // effectué une mise à jour des props
  }, [props])

  useEffect(() => {
    // Ce bloc de code sera exécuté dès
    // que le composant aura été monté
    // (remplace componentDidMount)
  }, [])

  return (
    // JSX ici
  )
}
```

Agir sur le cycle de vie dans un composant par Class

Montage du composant

Mise à jour du composant

Démontage du composant

useEffect peut être appelé par destructuration comme le code ci-contre, ou peut être appelé par React : **React.useState**

useEffect est donc très puissant, mais sans connaissance et lecture global du code, moins précis que les fonctions qu'il remplace dans un composant par class

Selon sa position et son argument, useEffect agira à diverse moment du cycle de vie

```
import React, {useEffect} from 'react'
const MyComponent = ( ) => {
  useEffect(( ) => {
    // Ce bloc de code sera exécuté
    //  juste avant que le composant
    //  soit monté (remplace componentWillMount)
    return ( ) => {
      // Ce code sera exécuté
      // juste avant que le composant
      // soit démonté et retiré du DOM
      // (remplace componentWillUnmount)
    }
  }, [])
  useEffect(( ) => {
    // Ce code sera exécuté après avoir
    // effectué une mise à jour des props
  }, [props])

  useEffect(( ) => {
    // Ce bloc de code sera exécuté dès
    // que le composant aura été monté
    // (remplace componentDidMount)
  }, [])

  return (
    // JSX ici
  )
}
```

const CheatSheet =() => {

Function intervenant sur le cycle de vie

```jsx
import React, {useEffect} from 'react'
const MyComponent = () => {
  useEffect(() => {
    // Ce bloc de code sera exécuté
    //  juste avant que le composant
    //  soit monté (remplace componentWillMount)
    return () => {
      // Ce code sera exécuté
      // juste avant que le composant
      // soit démonté et retiré du DOM
      // (remplace componentWillUnmount)
    }
  }, [])
  useEffect(() => {
    // Ce code sera exécuté après avoir
    // effectué une mise à jour des props
  }, [props])

  useEffect(() => {
    // Ce bloc de code sera exécuté dès
    // que le composant aura été monté
    // (remplace componentDidMount)
  }, [])

  return (
    // JSX ici
  )
}
```

Création d'une function avec state et props

```jsx
import React, {useState, useEffect} from 'react'
/*
    Import de react et 2 fonctions en destructuration :
            - useState & useEffect
    Pourrais être utilisé sans React :
            ~ import {useState, useEffect} from 'react' ~
*/

const MyComponent = (props) => {
  /*
  Création du composant dans une fonction (ici fléchée)
     - Toujours la première lettre du nom en majuscule
     - Passage des props en arguments si nécessaire dans le composant
  */
  const [message, setMessage] = useState('Hello')
  /*
  - Définition d'un state message et de sa fonction
     - Le state pourra être lu avec la 1ère valeur (message)
     - Le state pourra être modifié avec la 2nd valeur (setMessage)
  - La valeur par défaut est celle donnée en argument de useState
  => Ici appeler message renverra 'Hello'
  */
  const handleClick = () => {
    setMessage('Hello React!');
  }
  /*
  - Définition d'une fonction de click (handleClick)
     - La fonction modifiera le message grâce à setMessage et
     lui donnera la valeur = Hello React!
  => Après appel de la fonction message renverra 'Hello React!'
  */
  return (
    <div>
        <h1>{props.name}</h1>
        <p>{message}</p>
        <button onClick={handleClick}>Change message</button>
    </div>
  )
  /*
  - return renvoi l'élément du composant (ici une div)
  - La balise <h1> contiendra la valeur de la props name
  - La balise <p> contiendra le state message ('Hello' de base)
  - Le button, lui, fait référence à la fonction handleClick
       elle sera appelée uniquement en cas de click
  */
}
```

useState, useEffect & Axios

Voyons un cas concret d'utilisation de useState et useEffect, accompagné du package axios

- Partie axios
- Partie useEffect
- Partie useState

Dans cet exemple, le composant renverra une phrase de l'API advice en appelant cette dernière pendant le montage du composant (grâce à useEffect), le résultat de cette appel sera stocké dans advice du state (initié avec useState) et fera partie du return

```jsx
import { useState, useEffect } from 'react'
import axios from 'axios'

const MyComponent = () => {
    const [advice, setAdvice] = useState('')
    const apiURL =
'https://api.adviceslip.com/advice'
    useEffect(() => {
        const fetchData = () => {
            axios.get(apiURL)
            .then(response=>{
              setAdvice(response.data.slip.advice)
            })
            .catch(error=>{
                console.error(error)
            })
        }
        fetchData()
    }, [])

    return (
        <div>
            <p>{advice}</p>
        </div>
    )
}

export default MyComponent
```


axios est installable facilement grâce à ***npm i axios***

Fonctionnement du composant

Le composant est monté

```
import { useState, useEffect } from 'react'
import axios from 'axios'

const MyComponent = () => {
```

La state advice est initié

```
const [advice, setAdvice] = useState('')
```

La fonction useEffect est exécuté

```
useEffect(() => {
    const fetchData = () => {
        axios.get(apiURL)
        .then(response=>{
          setAdvice(response.data.slip.advice)
        })
        .catch(error=>{
            console.error(error)
        })
    }
    fetchData()
}, [])
```

Dans useEffect axios appelle l'URL

```
axios.get(apiURL)
.then(response=>{
  setAdvice(response.data.slip.advice)
})
.catch(error=>{
    console.error(error)
})
```

Dans la réponse axios , le state est modifié avec la valeur

```
setAdvice(response.data.slip.advice)
```

Le composant renvoi son code avec le state advice à jour

```
return (
    <div>
        <p>{advice}</p>
    </div>
)
```

La fonction useEffect va gérer le comportement général afin de charger les informations lors du chargement du composant

Le state va gérer le stockage de la réponse et sa mise en page

Axios va gérer la connexion avec l'API via une requête GET, comme le ferait fetch ou d'autres méthode.

Conclusions

En conclusion, useEffect et useState sont deux hooks essentiels dans React qui permettent de gérer les états et les effets dans une application.

useState est utilisé pour gérer les états locaux tandis que useEffect est utilisé pour effectuer des actions à chaque mise à jour de l'application.

En utilisant ces hooks de manière efficace, nous pouvons améliorer la qualité et la performance de nos applications React.

hooks ?

Les hooks sont des fonctions spéciales dans React qui permettent d'interagir avec les états et les effets d'une application sans avoir à utiliser de classes. Ils permettent aux développeurs de partager du code facilement réutilisable pour gérer les données, les mises à jour, les effets secondaires, etc.

Défi
Les hooks

Le compteur

Créer un composant qui sera utilisé comme compteur.

Le compteur sera stocké dans un state et affiché avec un bouton + et un bouton -.

```
import ListArticles from
'./Components/ListArticles'
const fakeDate = [
  {name : 'marguarita', price : 25},
  {name : '4 saisons', price: 32},
]
function App() {
  return (
    <div className="App">
          <ListArticles
                items={fakeDate}
                category="Pizza"/>
    </div>
    )
}

export default App
```

```
function ListArticles (props) {
        const itemsListing = props.items.map((item)=>
(         <h2> {item.name} : {item.price}</h2>
        ))
        return (
            <div>
            <h1>{category}</h1>
                {itemsListing}
            </div>
        )
}

export default ListArticles
```

L'API

Créez un composant qui appelera l'API advice et affichera le 7ème advice (id 7) dans un document.

(utilisez useState & useEffect)

Doc de l'API :
https://api.adviceslip.com/

```
import ListArticles from
'./Components/ListArticles'
const fakeDate = [
  {name : 'marguarita', price : 25},
  {name : '4 saisons', price: 32},
]
function App() {
  return (
    <div className="App">
          <ListArticles
                items={fakeDate}
                category="Pizza"/>
    </div>
    )
}

export default App
```

```
function ListArticles (props) {
        const itemsListing = props.items.map((item)=>
(         <h2> {item.name} : {item.price}</h2>
        ))
        return (
            <div>
            <h1>{category}</h1>
                {itemsListing}
            </div>
        )
}

export default ListArticles
```

Les Adresses

Grâce à l'API du gouvernement , la fonction map (vue dans le live précédent), le useState et useEffect, affichez les adresses de l'entreprise Studi.

l'adresse de l'api serait :
https://recherche-entreprises.api.gouv.fr/search?q=Studi

```
import ListArticles from
'./Components/ListArticles'
const fakeDate = [
  {name : 'marguarita', price : 25},
  {name : '4 saisons', price: 32},
]
fu        App() {
          (
23        className="App">
           <ListArticles
                 items={fakeDate}
```

```
function ListArticles (props) {
        const itemsListing = props.items.map((item)=>
(         <h2> {item.name} : {item.price}</h2>
        ))
        return (
            <div>
            <h1>{category}</h1>
                {itemsListing}
            </div>
        )
}

export default ListArticles
```