React

Réaliser une application web

Matthieu Rocher - 2024

Qu-est ce que React

- Bibliothèque JavaScript open source réalisée par Meta
- Utilise un DOM (Document Object Model) virtuel
- Une des librairie les plus utilisées dans le monde de la tech
- Fonctionnement par composant réutilisable

Pourquoi React

- Facile à apprendre
- Composants réutilisables
- Opportunités d’emploi
- Performances améliorées grace au DOM virtuel

Démarrer avec React

Les librairies utiles

- ViteJS : *Afin de vous simplifier la vie a démarrer une projet react*

https://vitejs.dev/

- Emotion / TailwindCSS : *Designer simplement vos composants*

https://tailwindcss.com/ ou **https://emotion.sh/docs/introduction**

- Recoil : *Avoir un état partagé dans votre application*

https://recoiljs.org/fr/

- React-router : *Créer un système de routage pour votre application*

https://reactrouter.com/en/main

- React-Query : *Simplifier les requêtes web et la mise en cache*

https://tanstack.com/query/v3/

Composants

- Un composant react est une fonction Js qui peut prendre en entrées des arguments (appelés props) et qui retourne du JSX

```
function MyComponent(){
  return <p>Je suis un composant simple react</p>
}

function MyComponentWithArguments(props){
  return <p>Je suis un composant avec des propriétés
{props}</p>
}
```

Hook

- Les Hook en React sont des fonctions JS préfixés sur mot « use » et qui permettent d'effectuer n'importe quel type de tache.

```
function useSampleHook(){
  const sayHello = (name:string) => {
    return `Hello ${name}`
  }

  return sayHello
}
```

Props

- Ce sont les propriétés que vous passez à votre composant react
- Elles peuvent être de tout type et même un autre composant react !
- Par défaut, les props d’un composant react ont l’element « children » qui vous permet d’imbriquer vos composants

State

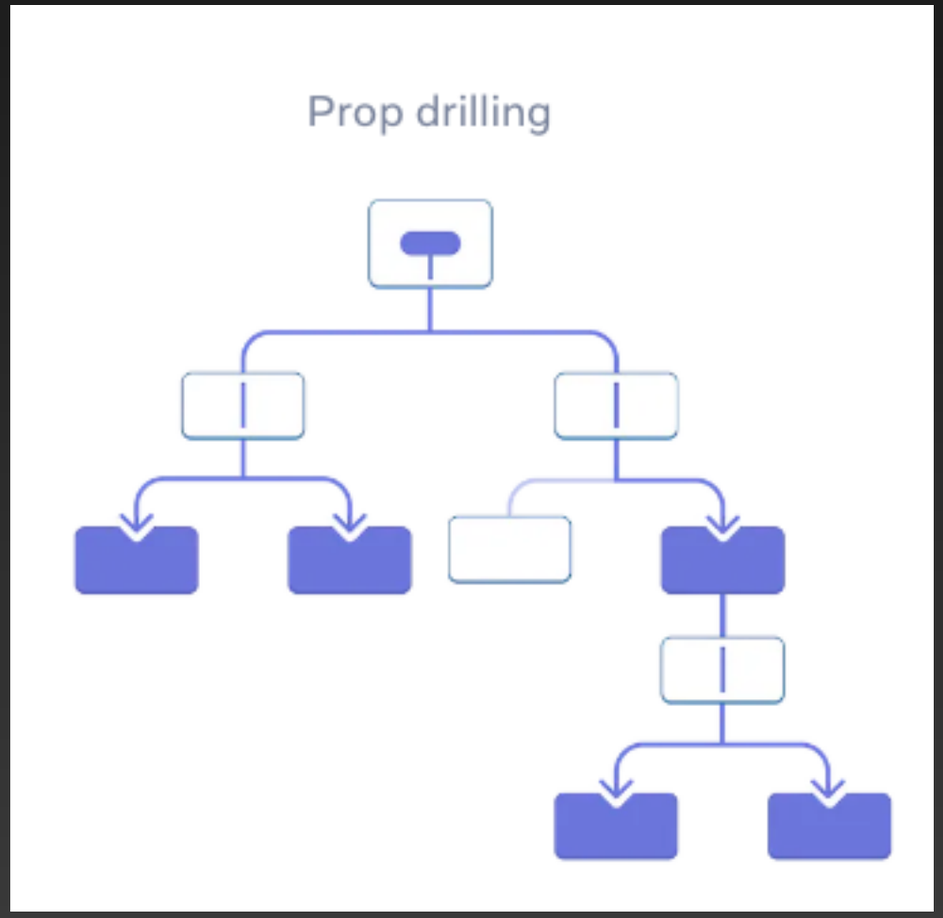
- Il s'agit de l'état interne de votre composant
- Chaque modification de state crée un nouveau rendu de votre composant (il est rechargé dans le DOM)

- Utilise le « hook » useState

```
function SampleComponent(props){
  const [isOnline,setIsOnline] = useState(false)
  return <>
    <p>Je suis un composant avec des propriétés {props}</p>
    {isOnline ? <p>Je suis connecté</p> : <p>Je ne suis pas connecté</p>}
    <button onClick={()=>setIsOnline(true)}>Me connecter</button>
  </>
}
```

Context

- Lorsque vous passez des props d’un composant parent a son enfant, son petit enfant etc, se pose la question d’utiliser un context. On appelle cela le « Prop drilling »



Context

- Utiliser un context, c’est passer une propriété de votre application dans toute l’arborescence de vos composants React. Cette propriété n’est donc plus passé en props de vos composants sous jacent. Le code est plus lisible
- Il faut utiliser « createContext » et le hook « useContext »

Reducer

Un remplaçant de redux

- Un reducer permet de consolider les mises à jours de l'état d'un composant
- Il s'utilise via le hook useReducer
- Ce hook prend 2 paramètres : une fonction « reducer » et un état initial
- Le hook renvoi un tuple avec l'état et la fonction de changement d'état

Reducer

```tsx
enum TaskAvailableActions {
  ADD = "ADD",
  REMOVE = "REMOVE",
  CHECK = "CHECK",
  UNCHECK = "UNCHECK",
}

interface TaskAction {
  type: TaskAvailableActions;
  payload: Task | { id: string };
}

const [tasks, dispatch] = useReducer(taskReducer, []);

function taskReducer(state: Task[], action: TaskAction) {
  const { type, payload } = action;
  switch (type) {
    case TaskAvailableActions.ADD: {
      return [ ...state, {  ...payload }];
    }
    case TaskAvailableActions.REMOVE: {
      return state.filter((task) => task.id !== payload.id);
    }
    default:
      return state;
  }
}
```

Combiner Context + Reducer

- L’avantage d’utiliser un reducer c’est aussi de pouvoir le passer aux sous composant de l’arborescence afin de pouvoir modifier les données d’un context

Récupérer un élément html

- Il est possible de récupérer un élément HTML en react
- Il n’est pas recommandé d’utiliser le JS standard (document.getElementBy…) mais plutôt d’utiliser un hook appelé useRef
- useRef est pertinent à être utiliser dans un formulaire afin d’éviter les re-render à chaque fois qu’un utilisateur écris un charactère dans un input

Utiliser et réagir à un état externe

- Il est possible de définir un état de votre composant qui sera stocké ailleurs dans votre navigateur par exemple en « stockage session » ou « stockage local »
- Il faut utiliser le hook useSyncExternalStore
- Ce dernier demande l'instanciation d'un store avec 2 méthodes : subscribre et getSnapshot

useSyncExternalStore

Que fait ce hook ?

```tsx
import { useSyncExternalStore } from "react";

type IsOnlineState = "false" | "true";

export const useIsOnlineStore = () => {
  if (!store.getSnapshot()) {
    setIsOnlineState("false");
  }
  const isOnline = useSyncExternalStore(store.subscribe, store.getSnapshot);
  return { isOnline, setIsOnlineState };
};

const store = {
  getSnapshot: () => localStorage.getItem("isOnline") as IsOnlineState,
  subscribe: (listener: () => void) => {
    window.addEventListener("storage", listener);
    return () => window.removeEventListener("storage", listener);
  },
};

function setIsOnlineState(newValue: IsOnlineState) {
  window.localStorage.setItem("isOnline", newValue);
  window.dispatchEvent(
    new StorageEvent("storage", { key: "isOnline", newValue })
  );
}
```

Les autres hooks

- useMemo => pour garder en cache le résultat d'un calcul entre les « re-render » d'un composant
- useCallback => pour garder en cache une fonction entre les « re-render » d'un composant
- useId => générer un uuid