**Comment fonctionne l'utilisation des contextes avec React ?**

Le contexte permet de faire passer des données à travers les composants de notre arborescence sans avoir à passer par les props.

Elle permet de faire passer des données qu’on peut considérer globales et dont on aurait besoin dans toute notre application ou presque.

utiliser le Contexte avec des pincettes car il rend la réutilisation des composants plus difficile.

UseContext : Depuis les hooks, on peut utiliser le hook useContext pour récupérer les infos du contexte plus facilement, sans avoir un composant supplémentaire sous la forme de context.consumer

Provider:

Chaque objet contexte a un Provider, qui permet aux composants qui l’utilisent de s’abonner aux mises à jour du contexte et qui possède une prop value qu’il leur transmet.

Pour mettre à jour le contexte, on peut lui passer une fonction qui permettra aux composants qui l’utilisent de le mettre à jour.

**Comment fonctionne l'utilisation des hooks avec React ?**

Fonction qui permet de se brancher sur des fonctionnalités React. Permet d’ajouter l'état local React et les cycles de vie à des composants fonctionnels, ce qui permet de ne plus utiliser les classes.

Les hooks permettent d’extraire la logique d’un composant pour la réutiliser et la tester de façon indépendante sans modifier la hiérarchie de nos composants.

Ils nous permettent aussi de séparer les différentes petites fonctions qui sont liées à plusieurs composants.

Il existe plusieurs types de hooks prédéfinis et inclus avec React comme useState pour le state local, qu’on utilise avec une variable qui correspond au state actuel et une fonction pour modifier ce state (ex : [compteur, setCompteur] = useState(valeur initiale))

On a le choix entre utiliser plusieurs hooks d’états ou d’en utiliser un seul, en créant un objet à la façon de this.state, mais il est conseillé de gérer les différents states séparément.

Par convention, le nom des hooks doit toujours commencer par « use ».

**Comment fonctionne l'utilisation des effects avec React ?**

On peut imaginer useEffect comme l’ensemble des fonctions componentDidMount, componentDidUpdate et componentWillUnMount. Cela dit, à la différence de celles-ci, useEffect n'empêche pas le navigateur d’actualiser l’écran, ce qui permet un rendu plus fluide.

Par défaut il est appelé à chaque rendu, mais il est possible de sauter l'exécution, pour cela il suffit de passer une liste comme second argument optionnel. Si la liste est vide, useEffect ne sera appelé qu’au moment du componentDidMount.

UseEffect est appelé à chaque rendu, ce qui permet d'exécuter des actions à tout moment et donc de ne pas avoir à attendre d’être à la bonne étape du cycle de vie du composant react.

Il existe 2 types d’effects: ceux qui requièrent un “nettoyage” après utilisation et ceux qui ne le requièrent pas. Les effect peuvent retourner une fonction qui nettoie après elle.

Le nettoyage se fait donc si on retourne la fonction de nettoyage dans l’effect, et lors de la phase componentDidUnmount.