Introduction à Zustand

1. Introduction

Lorsque l'on développe une application React, l'un des défis courants est de partager un état entre plusieurs composants sans lien de parenté direct. Plutôt que de recourir au passage de props, une option est d'utiliser les contextes, bien que cela puisse devenir complexe. Une alternative plus simple est proposée par Zustand pour gérer l'état partagé de manière plus directe.

1. Installation

npm i zustand

1. Création du store

Dans un fichier js on va créer le store qu’on partagera :

import {create} from "zustand"

export const useStore = create((set)=>{

info : 'etat'

});

La variable s’appelle useStore, elle commence par use pour signifier qu’on utilisera un hook.

La méthode create de Zustand prend en parametre une fonction, cette fonction prend un setter qui permettra de modifier l’état et le corps correspond à l’état initial.

On export la variable.

1. Utilisation

Dans un component pour utiliser le store on déclare une variable qui prendra le useStore.

Cela reverra par défaut l’état initial, qu’on peut destructurer pour utiliser.

Ce qu’on a destructurer peut etre utiliser au niveau du composant.

import { useStore } from "../store/store"

export function CmpFirst(){

const {info} = useStore()

return <div>

{info}

</div>

}

On constate l’affiche de la donnée du store :

* Je le rajoute sur un deuxieme composant au niveau d’un input de la manière que précédemment sur value.

import { useStore } from "../store/store"

export function CmpSecond(){

const {info} = useStore()

return <div>

<input value={info} />

</div>

}

3) Implémenter la modification

Dans le store on va pouvoir implémenter des fonctions pour modifier l’état en utilisant le setter.

Ici updateInfo qui prend en paramètre une nouvelle info pour mettre à jour l’état.

import { create } from 'zustand'

export const useStore = create((set) => ({

info: 'info',

updateInfo(info){

set({info:info})

}

}))

On va pouvoir recuperer à partir de ce hook le updateInfo qu’on va pouvoir utiliser.

import { useStore } from "../store/store"

export function CmpSecond(){

const {info, updateInfo} = useStore()

return <div>

<input value={info} onChange={e=> updateInfo(e.target.value)} />

</div>

}

On peut constater que les deux composants sont parfaitement synchronisés.

(UNIQUEMENT QUAND C’EST DES PROPRIETES DE PREMIER NIVEAU)

Sans aucune autre modification supplémentaire.

Une image contenant Police, texte, capture d’écran, blanc

Description générée automatiquement

1. Modification en dehors des composants

Il est possible de faire des modifications des stores en dehors du store voici un exemple :

La méthode du store setState fournit par zustand permet de modifier un état (ici toute les seconde)

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

On peut constater ici qu’on n’a pas eu besoin de préciser les autres états du store.

Zustand fera la fusion avec les états précédents sans qu’on lui dise de les garder.

Avec cette méthode l’ensemble des composants avec le store sont mis à jour à chaque changement dans le store même pour les composants qui n’utilise que des données non changeantes.

On peut y remédier :

On peut préciser sur le composant spécifique juste le state qui nous interesse.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Le test de rafraichissement peut etre fait avec les outils de debugs sous chrome (React developper tools) en cochant Highlight updates afin de voir les MAJ en direct.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

1. Possibilité de mettre les fonctions en dehors du state

Il est possible de faire des fonctions en dehors pour modifier l’état plutôt que de le laisser dans le state.

Il n’y a pas de méthode meilleure qu’une autre.

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement