**Redux**

1. **Redux Définition**
2. **Redux**

Redux est une bibliothèque de gestion d'état prévisible pour les applications JavaScript. Il est couramment utilisé avec des frameworks tels que React pour gérer l'état de l'application de manière centralisée.

Le principal concept de Redux est le "store" (magasin), qui est un objet JavaScript contenant l'état global de l'application. L'état de l'application ne peut être modifié que par des "actions", qui sont des objets décrivant un changement spécifique à l'état. Les actions sont ensuite envoyées au "reducer" (réducteur), une fonction pure qui spécifie comment l'état de l'application doit être modifié en réponse à chaque action.

Le réducteur prend l'état actuel de l'application et l'action en cours, puis retourne un nouvel état en appliquant les modifications nécessaires. Redux suit un principe d'immuabilité, ce qui signifie que l'état ne doit jamais être modifié directement, mais plutôt recréé à chaque modification.

Redux facilite également la gestion des effets secondaires et des requêtes asynchrones grâce à des bibliothèques complémentaires telles que Redux Thunk ou Redux Saga.

En utilisant Redux, les développeurs peuvent obtenir une traçabilité complète des changements d'état de l'application, ce qui facilite le débogage et le suivi des bugs. Il permet également de rendre les composants React plus modulaires et réutilisables, car ils peuvent accéder à l'état global de l'application et se mettre à jour en fonction des changements de cet état.

1. **Redux toolkit**

Redux Toolkit est un ensemble d'utilitaires et d'abstractions qui facilitent le travail avec Redux, une bibliothèque populaire de gestion d'état pour les applications JavaScript. Redux Toolkit est conçu pour simplifier de nombreux cas d'utilisation courants de Redux, notamment l'écriture de réducteurs, la création d'actions et la configuration du store.

Voici quelques-unes des fonctionnalités clés de Redux Toolkit :

1. Syntaxe simplifiée des réducteurs : Redux Toolkit inclut une fonction **createSlice** qui réduit la quantité de code boilerplate nécessaire pour créer un réducteur. Elle génère automatiquement des créateurs d'actions et utilise la bibliothèque Immer pour vous permettre d'écrire un code plus simple et immuable qui met à jour votre état.
2. Configuration par défaut : Redux Toolkit inclut une configuration par défaut qui configure votre store avec des middlewares couramment utilisés tels que redux-thunk et redux-logger.
3. Gestion immuable de l'état : Redux Toolkit utilise la bibliothèque Immer pour vous permettre d'écrire un code plus simple et mutable qui met à jour votre état. Cela signifie que vous pouvez écrire du code qui modifie l'état directement, sans vous soucier de l'immuabilité.
4. Simplification des thunks : Redux Toolkit fournit une syntaxe simplifiée pour créer des fonctions thunk qui facilitent la gestion des actions asynchrones.
5. Utilitaires supplémentaires : Redux Toolkit inclut plusieurs utilitaires supplémentaires, tels que **createAsyncThunk** pour gérer les actions asynchrones, et **createEntityAdapter** pour gérer les données normalisées.

Dans l'ensemble, Redux Toolkit peut vous aider à écrire un code Redux plus propre, plus concis et plus maintenable. C'est un excellent choix pour les développeurs qui souhaitent utiliser Redux, mais ne veulent pas passer beaucoup de temps à écrire du code boilerplate.

1. **ReduxToolkit**
2. **Store**

**configureStore** est une fonction fournie par Redux Toolkit, une bibliothèque utilitaire pour Redux, qui vous aide à configurer un store Redux avec des valeurs par défaut et des middlewares pertinents. C'est une surcouche de la fonction **createStore** de Redux, mais avec des fonctionnalités supplémentaires et des optimisations.

**Store.js**

import { configureStore } from '@reduxjs/toolkit'

import { cakeReducer } from '../features/cake/cakeSlice'

import { icecreamReducer } from "../features/icecream/icecreamSlice";

import { useReducer } from "../features/user/userSlice"

// const reduxLogger=require('redux-logger');

// const logger=reduxLogger.createLogger()

export const  store=configureStore({

    reducer:{

        cake:cakeReducer,

        icecream:icecreamReducer,

        user:useReducer

    },

    // middleware:(getDefaultMiddleware)=>getDefaultMiddleware().concat(logger)

})

1. **Reducer && Actions**

"reducer" (réducteur), une fonction pure qui spécifie comment l'état de l'application doit être modifié en réponse à chaque action.

**createSlice** est une fonction fournie par Redux Toolkit, une bibliothèque utilitaire pour Redux, qui simplifie la création de tranches de réducteurs (reducers) en combinant les définitions d'actions et de réducteurs dans un format simplifié.

**cakeSlice.js**

const reduxToolkit = require("@reduxjs/toolkit");

const initialState = {

  numOfCakes: 10,

};

const cakeSlice = reduxToolkit.createSlice({

  name: "cake",

  initialState,

  reducers: {

    ordered: (state) => {

      state.numOfCakes--;

    },

    restocked: (state, action) => {

      state.numOfCakes += action.payload;

    },

  },

});

module.exports.cakeReducer = cakeSlice.reducer;

module.exports.cakeActions = cakeSlice.actions;

**index.js**

const {store} = require('./app/store')

const { cakeActions } = require('./features/cake/cakeSlice')

console.log('initial state',store.getState())

const unsubscribe=store.subscribe(()=>{

    console.log("Update state",store.getState())

})

store.dispatch(cakeActions.ordered())

store.dispatch(cakeActions.ordered())

store.dispatch(cakeActions.ordered())

store.dispatch(cakeActions.restocked(3))

unsubscribe()

Une image contenant texte

Description générée automatiquement

1. **Logger Middleware**

**Store.js**

. . .

const reduxLogger=require('redux-logger')

const logger=reduxLogger.createLogger()

const store=reduxToolkit.configureStore({

    reducer:{

        cake:cakeReducer,

        icecream:icecreamReducer

    },

    middleware:(getDefaultMiddleware)=>getDefaultMiddleware().concat(logger)

})

. . .

Une image contenant texte

Description générée automatiquement

1. **ExtraReducers**

Dans Redux Toolkit, "extraReducers" est une option de configuration supplémentaire fournie par la fonction "createSlice". Elle vous permet de définir des réducteurs qui gèrent des actions provenant d'autres tranches de réducteurs ou d'actions externes. Ces réducteurs peuvent être ajoutés en plus des réducteurs définis dans l'objet "reducers" de la tranche de réducteur.

En utilisant "extraReducers", vous pouvez étendre les fonctionnalités d'une tranche de réducteur en traitant des actions qui ne sont pas directement liées à son propre état. Cela est particulièrement utile lorsque vous devez répondre à des actions envoyées par d'autres parties de l'application ou des actions définies dans des tranches de réducteurs différentes.

**icecreamSlice.js**

const reduxToolkit = require("@reduxjs/toolkit");

const initialState = {

  numOfIcecreams: 10,

};

const icecreamSlice = reduxToolkit.createSlice({

  name: "icecream",

  initialState,

  reducers:{

    ordered:(state)=>{

        state.numOfIcecreams--

    },

    restocked:(state,action)=>{

        state.numOfIcecreams+=action.payload

    }

  },

  extraReducers:{

    ['cake/ordered']:(state)=>{

        state.numOfIcecreams-=2

    }

  }

});

module.exports.icecreamReducer=icecreamSlice.reducer

module.exports.icecreamActions=icecreamSlice.actions

**index.js**

store.dispatch(cakeActions.ordered())

unsubscribe()

**it will return : cake: { numOfCakes: 9 }, icecream: { numOfIcecreams: 8 },**

1. **Async Thunks**

**createAsyncThunk** est une fonction fournie par Redux Toolkit, une bibliothèque qui simplifie la gestion de l'état dans les applications Redux. **createAsyncThunk** est une fonction utilitaire qui crée un créateur d'action asynchrone qui envoie trois actions différentes en fonction de l'état de l'opération asynchrone : en cours (**pending**), réussie (**fulfilled**), et rejetée (**rejected**).

**userSlice.js**

const axios = require('axios')

const createSlice = require('@reduxjs/toolkit').createSlice

const createAsyncThunk = require('@reduxjs/toolkit').createAsyncThunk

const initialState = {

  loading: false,

  users: [],

  error: ''

}

// Generates pending, fulfilled and rejected action types

const fetchUsers = createAsyncThunk('user/fetchUsers', () => {

  return axios

    .get('https://jsonplaceholder.typicode.com/users')

    .then(response => response.data.map(user => user.id))

})

const userSlice = createSlice({

  name: 'user',

  initialState,

  extraReducers: builder => {

    builder.addCase(fetchUsers.pending, state => {

      state.loading = true

    })

    builder.addCase(fetchUsers.fulfilled, (state, action) => {

      state.loading = false

      state.users = action.payload

      state.error = ''

    })

    builder.addCase(fetchUsers.rejected, (state, action) => {

      state.loading = false

      state.users = []

      state.error = action.error.message

    })

  }

})

module.exports.userRecer = userSlice.reducer

module.exports.fetchUsers = fetchUsers

**index.js**

const {store} = require('./app/store')

store.dispatch(fetchUsers())

Une image contenant texte

Description générée automatiquement

1. **ReactRedux**
2. **Definition**

React Redux est une bibliothèque qui permet d'intégrer le gestionnaire d'état Redux dans une application React. Elle facilite la connexion entre les composants React et l'état global géré par Redux.

1. **Provider**

En React Redux, le **Provider** est un composant de haut niveau qui permet de fournir le store Redux à tous les composants de votre application React. Il s'agit d'un élément essentiel de l'intégration de Redux avec React.

import { Provider } from 'react-redux';

import { store } from './app/store';

root.render(

  <React.StrictMode>

    <Provider store={store}>

    <App />

    </Provider>

  </React.StrictMode>

);

1. **useSelector**

**useSelector** est un hook fourni par la bibliothèque React Redux qui vous permet d'extraire des données du store Redux dans un composant React.

icecreamView.jsx

import React from "react";

import { useSelector } from "react-redux";

export const IcecreamView = () => {

  const numOfIcecrem = useSelector((state) => state.icecream.numOfIcecreams);

  return (

    <div>

      <div>Number of ice creams:{numOfIcecrem}</div>

      <button>Order ice cream</button>

      <button>Restock ice cream</button>

    </div>

  );

};

1. **useDispatch**

**useDispatch** est un hook fourni par la bibliothèque React Redux qui vous permet d'envoyer des actions au store Redux à partir d'un composant React.

IcecreamView.jsx

import React from "react";

import {useDispatch, useSelector} from 'react-redux'

import { ordered, restocked } from "./icecreamSlice";

export const IcecreamView = () => {

  const numOfIcecrem = useSelector((state) => state.icecream.numOfIcecreams);

  const dispatch=useDispatch()

  return (

    <div>

      <div>Number of ice creams:{numOfIcecrem}</div>

      <button onClick={()=>dispatch(ordered())}>Order ice cream</button>

      <button onClick={()=>dispatch(restocked(3))}>Restock ice cream</button>

    </div>

  );

};

1. **FetchingData**

**UserView.jsx**

import React, { useEffect } from "react";

import { useDispatch, useSelector } from "react-redux";

import { fetchUsers } from "./userSlice";

export const UserView = () => {

  const user = useSelector((state) => state.user);

  const dispatch = useDispatch();

  useEffect(() => {

    console.log(user)

    dispatch(fetchUsers());

  }, []);

  return( <div>

    <h2>List of Users</h2>

    {user.loading && <div>Loading...</div>}

    {!user.loading && user.error ?<div>Error:{user.error}</div> : null}

    {!user.loading && user.users.length ?(

<ul>{

  user.users.map(user=>{

  return  <li key={user.id}>{user.name}</li>

  })

  }</ul>

    ):null}

  </div>)

};

1. **ReduxDevTools**

Redux DevTools est une extension de navigateur pour le débogage et la visualisation de l'état et des actions dans une application Redux. Elle permet de suivre les modifications de l'état, de rejouer des actions et de déboguer des erreurs. Il suffit de l'installer, de la configurer dans le store Redux et de l'ouvrir dans le navigateur pour l'utiliser.

L'extension DevTools offre plusieurs fonctionnalités pour vous aider à déboguer et inspecter votre store Redux :

1. Débogage avec remontée dans le temps : Vous permet de rejouer des actions et de visualiser l'état de votre application à n'importe quel moment. Vous pouvez avancer et reculer à travers vos actions pour suivre les modifications de l'état.
2. Inspection de l'état : Vous permet de visualiser l'état actuel de votre store Redux, ainsi que les états précédents de votre application.
3. Journal des actions : Affiche un journal de toutes les actions qui ont été envoyées à votre store Redux, avec l'heure et les données pertinentes.
4. Envoi d'actions : Vous permet d'envoyer des actions directement à votre store Redux depuis la console DevTools, ce qui est utile pour tester et déboguer votre application.

[**https://www.youtube.com/watch?v=iECBiELxrNE&list=PLC3y8-rFHvwiaOAuTtVXittwybYIorRB3&index=29&ab\_channel=Codevolution**](https://www.youtube.com/watch?v=iECBiELxrNE&list=PLC3y8-rFHvwiaOAuTtVXittwybYIorRB3&index=29&ab_channel=Codevolution)