



18.1 Redux DevTools

Les Redux Devtools permettent d'explorer le state et les actions qui sont envoyées en temps réel, mais aussi de revenir en arrière pour comprendre l'évolution de state pas à pas

Installation Redux DevTools sur navigateur

Google chrome

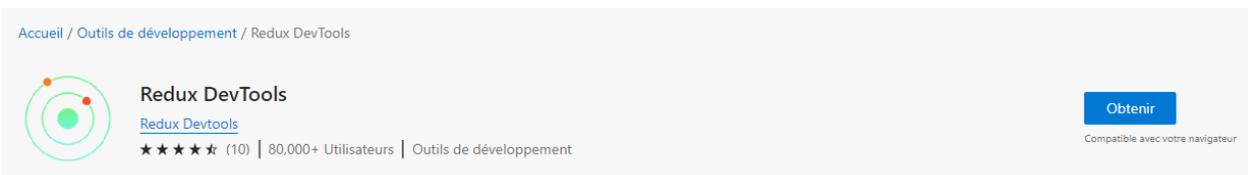
Pour installer devTools sur la navigateur :

[Lien Installation](#)



Edge :

[Lien Installation](#)



Au niveau de l'application :

Au niveau de la création de store ajouter le middleware suivant

```
import {legacy_createStore as createStore} from 'redux'
```

```
const store=createStore(reducer,  
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()  
)
```

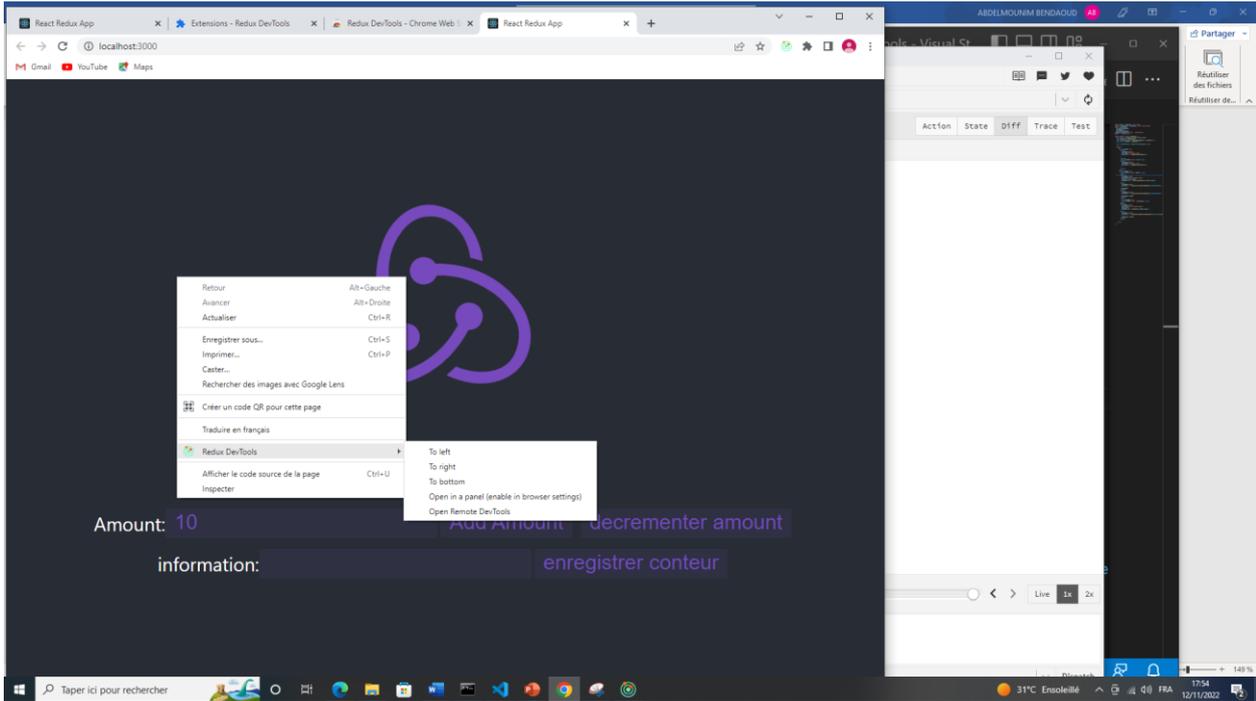
Utiliser le projet si joint :

Le zip : [projet-dev-tools](#) (WinRar)

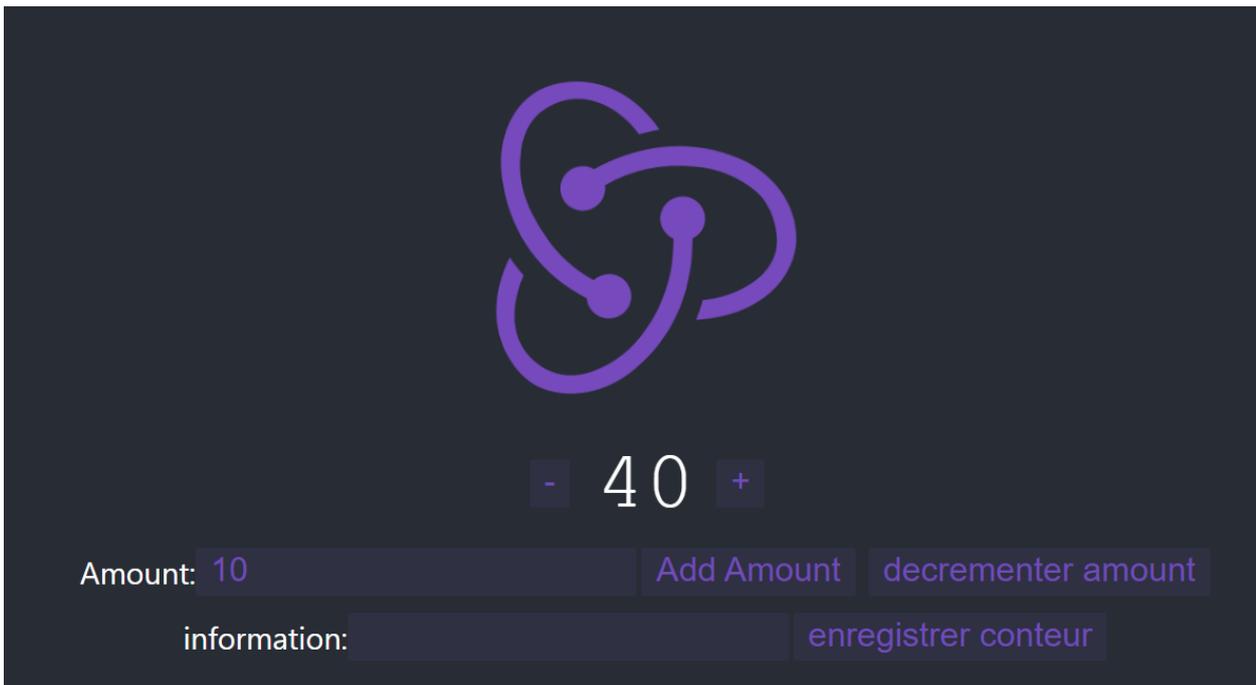
Lancer l'application après avoir

installer les dépendances par : `npm install`

lancer l'application : `npm start`



Bouton droite puis sélectionner ReduxDevTools puis to right

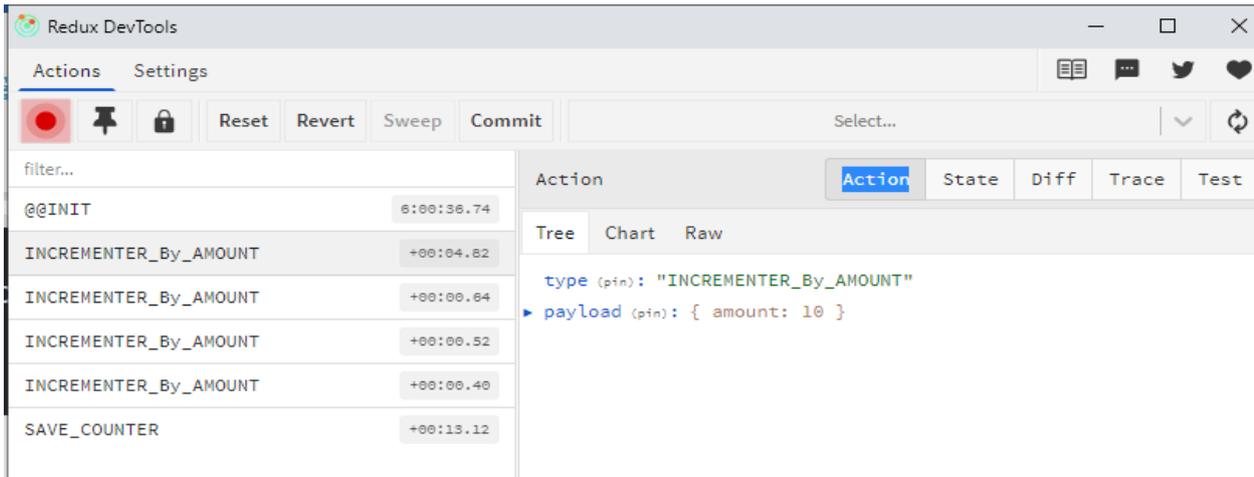


La fenêtre Redux DevTools affiche à gauche les actions

Puis à droite :

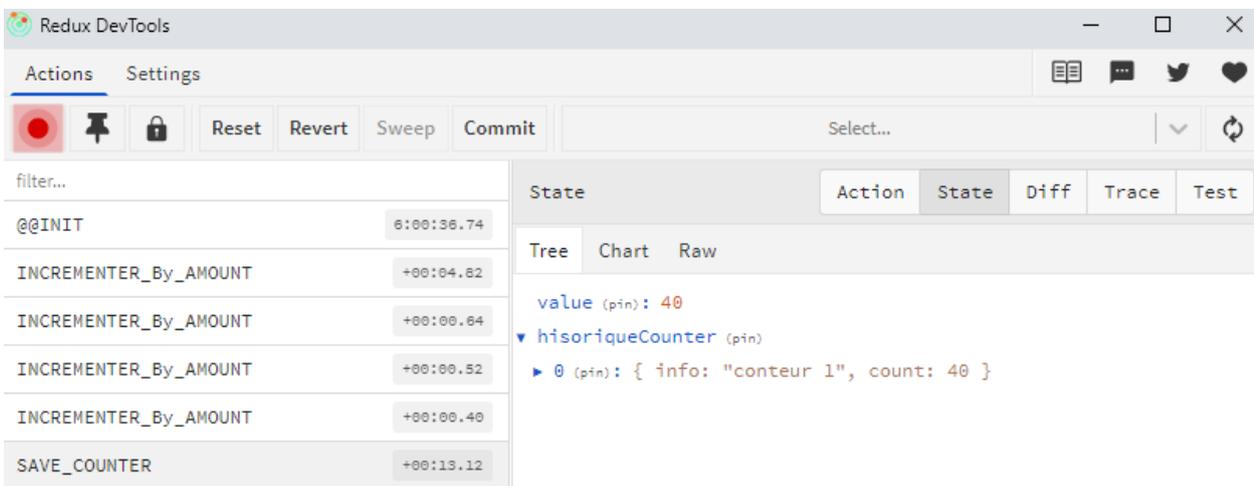
Pour le volet Action :

il s'affiche les détails de l'action sélectionnée



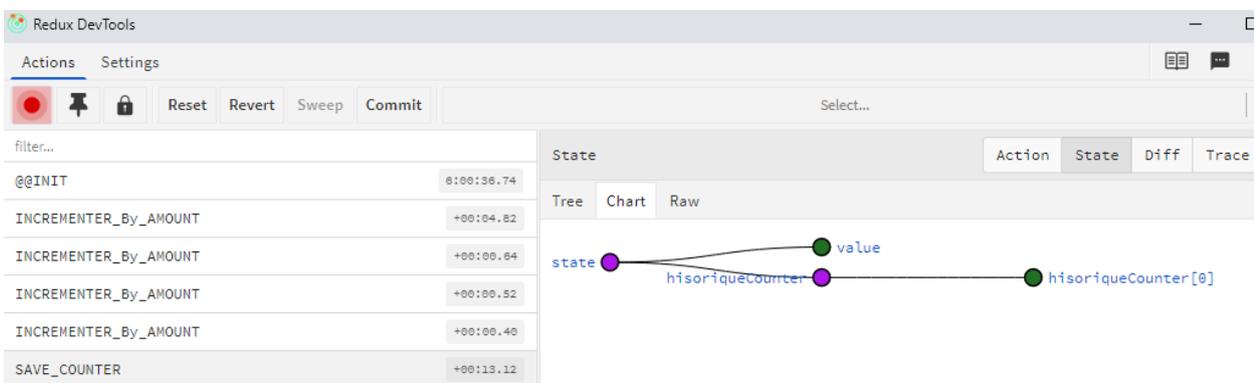
Pour le volet state :

Il s'affiche l'état state :



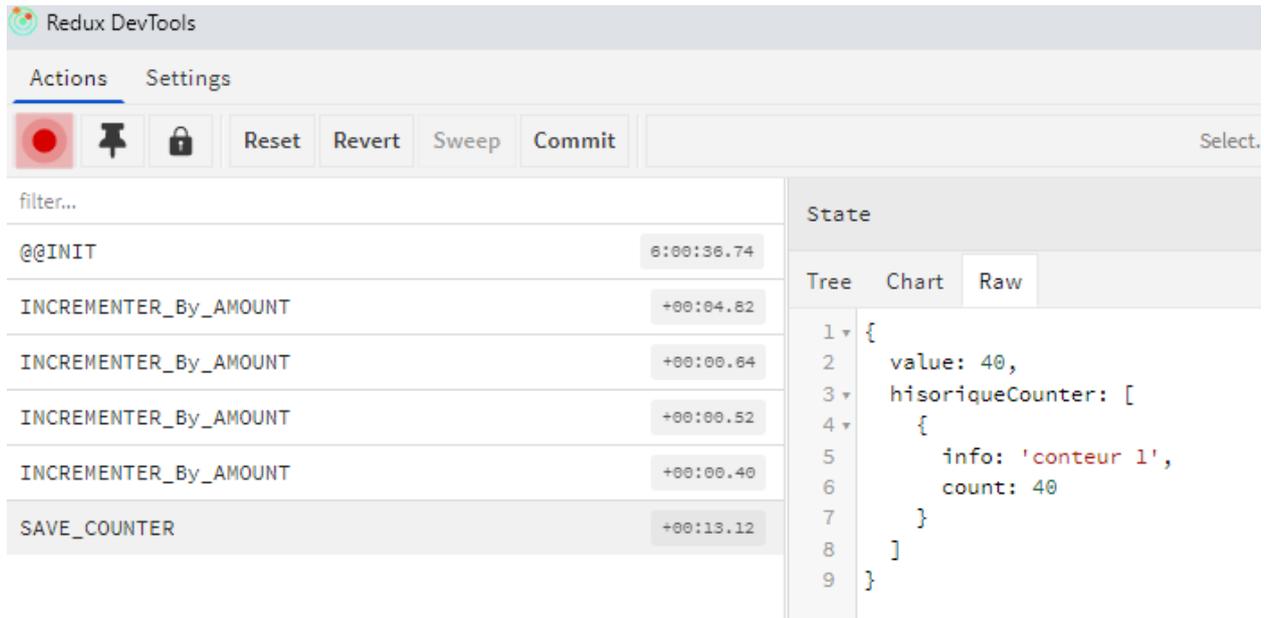
On peut visualiser les information sous forme de Tree, Chart, Raw

Chart





Raw

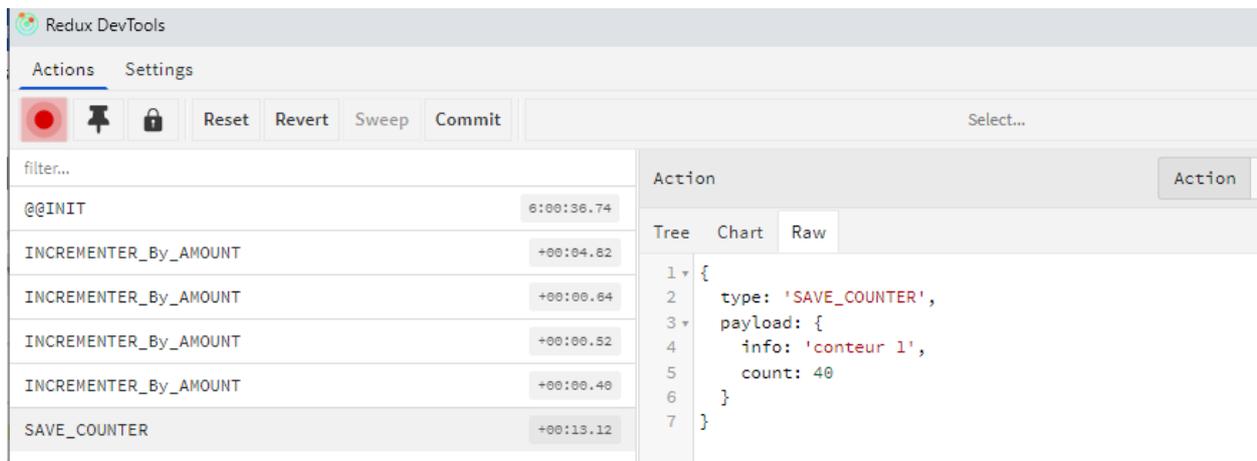


The screenshot shows the Redux DevTools interface with the 'Raw' view selected in the State panel. The left panel shows a list of actions, with the last one, 'SAVE_COUNTER', highlighted. The right panel displays the state as a JSON object:

```
1 {  
2   value: 40,  
3   historiqueCounter: [  
4     {  
5       info: 'conteur 1',  
6       count: 40  
7     }  
8   ]  
9 }
```

Pour le volet Action

Il s'affiche les détails de l'action sélectionnée

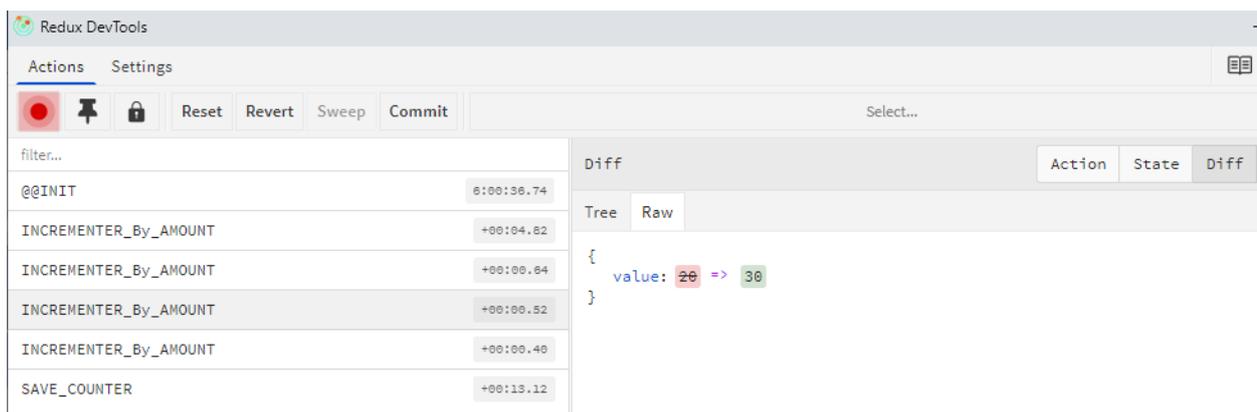


The screenshot shows the Redux DevTools interface with the 'Raw' view selected in the Action panel. The left panel shows the same list of actions, with 'SAVE_COUNTER' highlighted. The right panel displays the details of the selected action as a JSON object:

```
1 {  
2   type: 'SAVE_COUNTER',  
3   payload: {  
4     info: 'conteur 1',  
5     count: 40  
6   }  
7 }
```

Pour le volet diff :

Il s'affiche les valeurs de state avant et après l'exécution de l'action sélectionnée

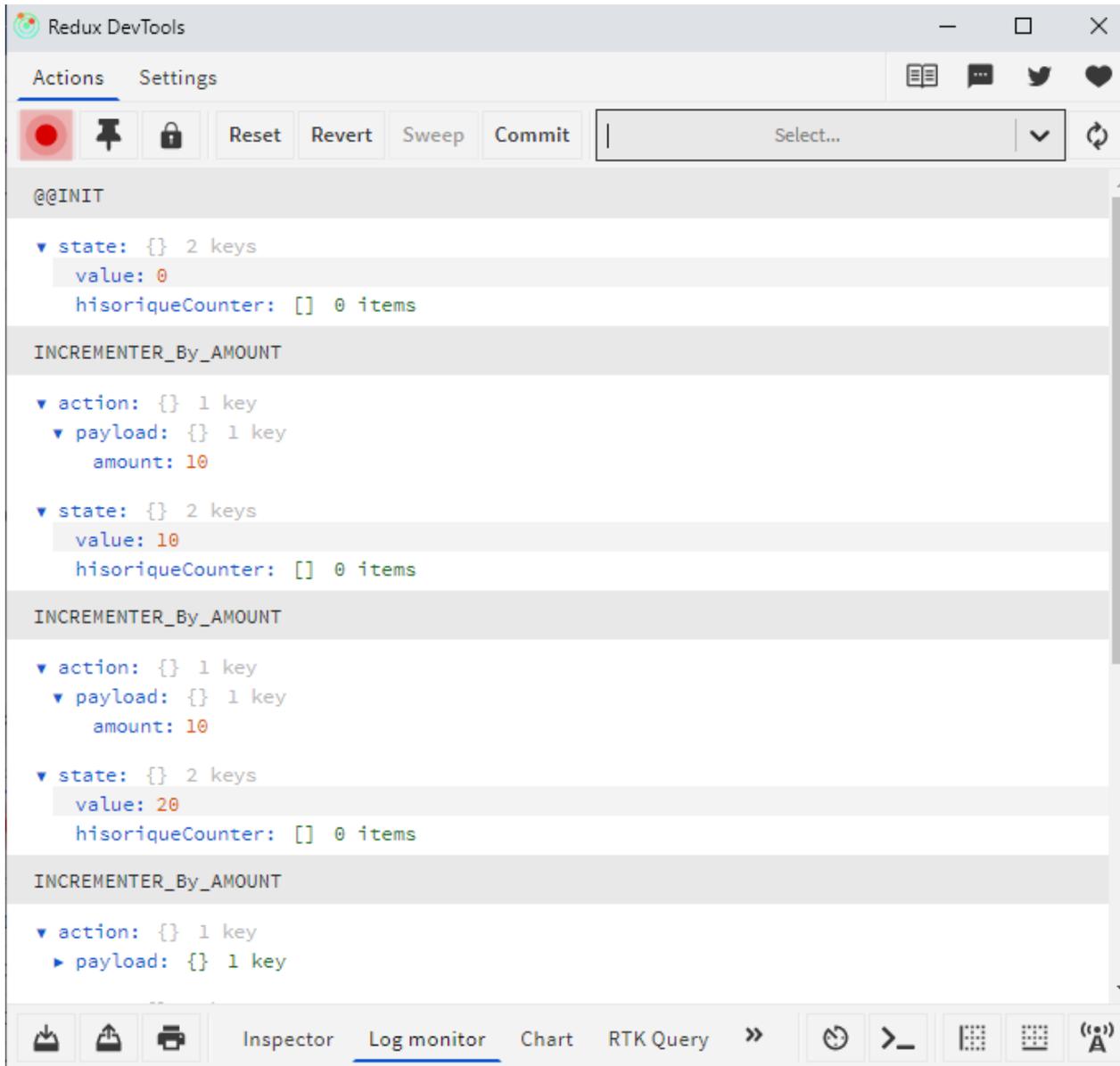


The screenshot shows the Redux DevTools interface with the 'Diff' view selected in the State panel. The left panel shows the same list of actions, with 'SAVE_COUNTER' highlighted. The right panel displays the difference between the state before and after the action:

```
{  
  value: 20 => 30  
}
```

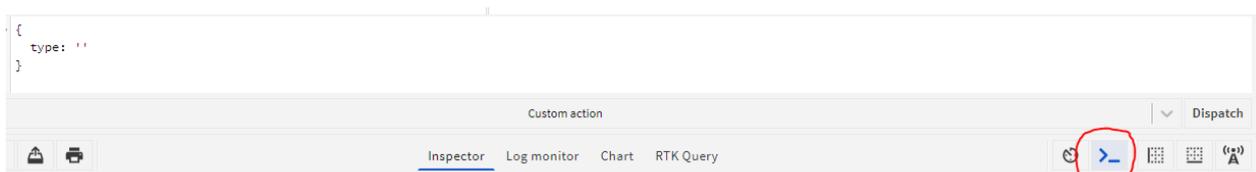
Log monitor :

Affiche toutes les états state de l'application



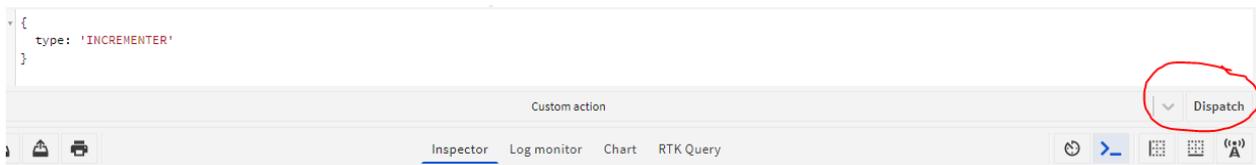
Lancer une Action via Redux DevTools

Afficher le volet dispatch





Puis saisir l'action puis cliquer sur Dispatch



Pour afficher le slider et défiler les actions



Pour persister les actions même si on actualise la page



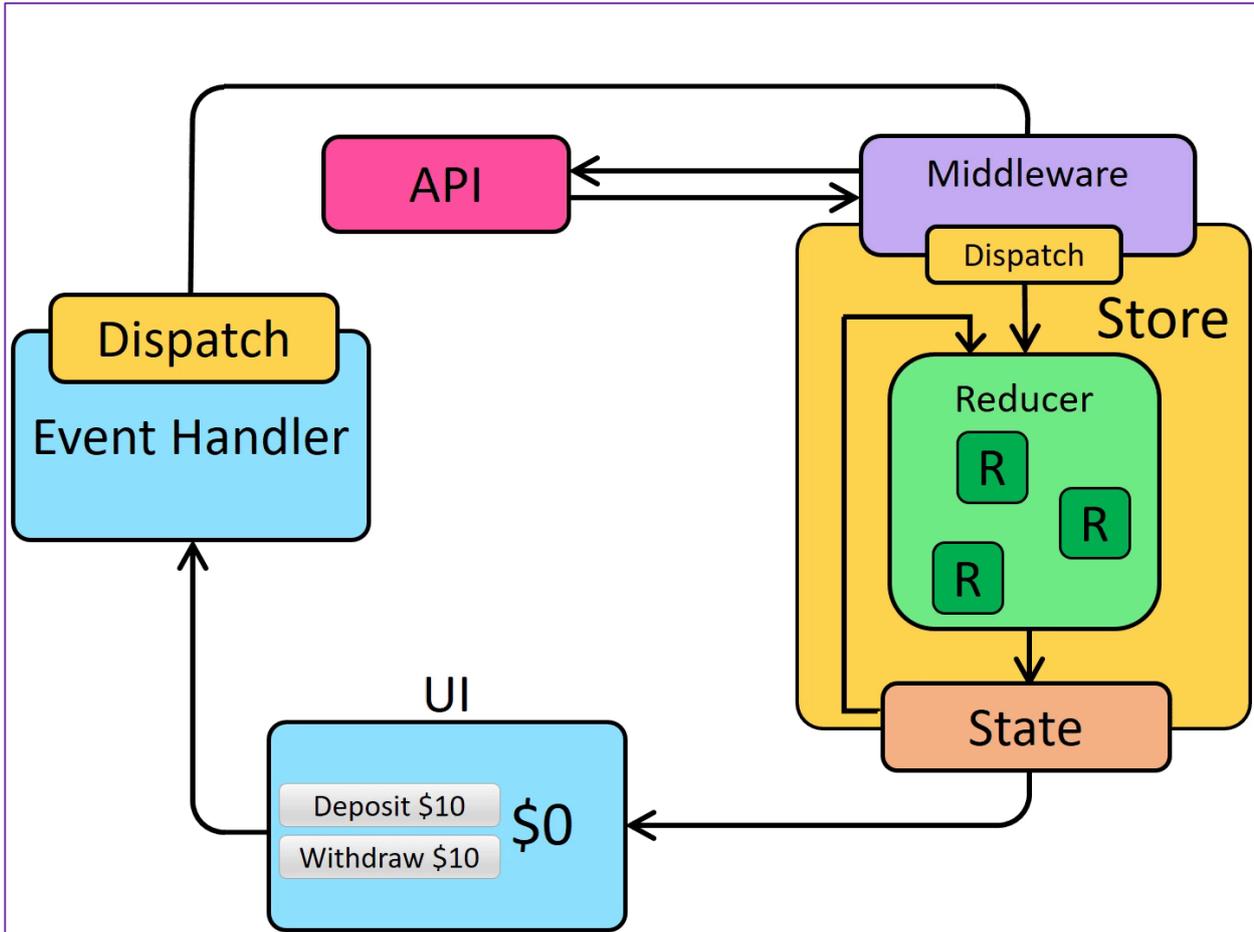
Pour bloquer les changements



18.2 Interaction avec les API

Introduction

Plus tôt, nous avons vu à quoi ressemble le flux de données synchrone pour Redux. Lorsque nous introduisons la logique asynchrone, nous ajoutons une étape supplémentaire où le middleware peut exécuter une logique comme les requêtes AJAX, puis répartir les actions. Cela donne au flux de données asynchrones l'aspect suivant :



18.2.1 Thunk function

Une fois que le middleware thunk a été ajouté au store Redux, il vous permet de transmettre les fonctions thunk directement à `store.dispatch`. Une fonction thunk sera toujours appelée avec `(dispatch, getState)` comme arguments, et vous pouvez les utiliser à l'intérieur du thunk si nécessaire.

Les fonctions thunks envoient généralement des actions simples à l'aide de créateurs d'action, Exemple de fonction thunk utilisée dans l'exemple ci-dessous :

```
//fonction thunk fetchPosts
export const fetchPosts=function(){
  return function(dispatch,getState){
    dispatch(fetchPostsRequest())

    axios.get('https://jsonplaceholder.typicode.com/posts').
    then(response=> dispatch(fetchPostsSuccess(response.data))).
    catch(err=>dispatch(fetchPostsFailure(err.message)))
  }
}
```

Comme vous remarquez la fonction thunk `fetchPosts` retourne une fonction qui a pour premier argument `dispatch` et deuxième argument `getState`.



L'argument `dispatch` est un callback fonction, c'est le middleware `Thunk` qui lui fait référence avec `store.dispatch`.

De même `getState` est un callback fonction, c'est le middleware `Thunk` qui lui fait référence avec `store.getState`.

On remarque de `fetchPosts` envoi les actions simple

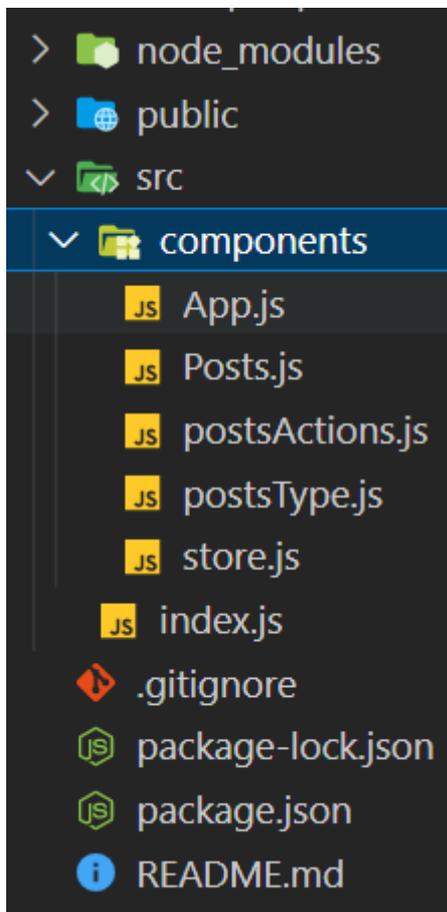
`fetchPostsRequest` et `fetchPostsSuccess` si succès

ou bien

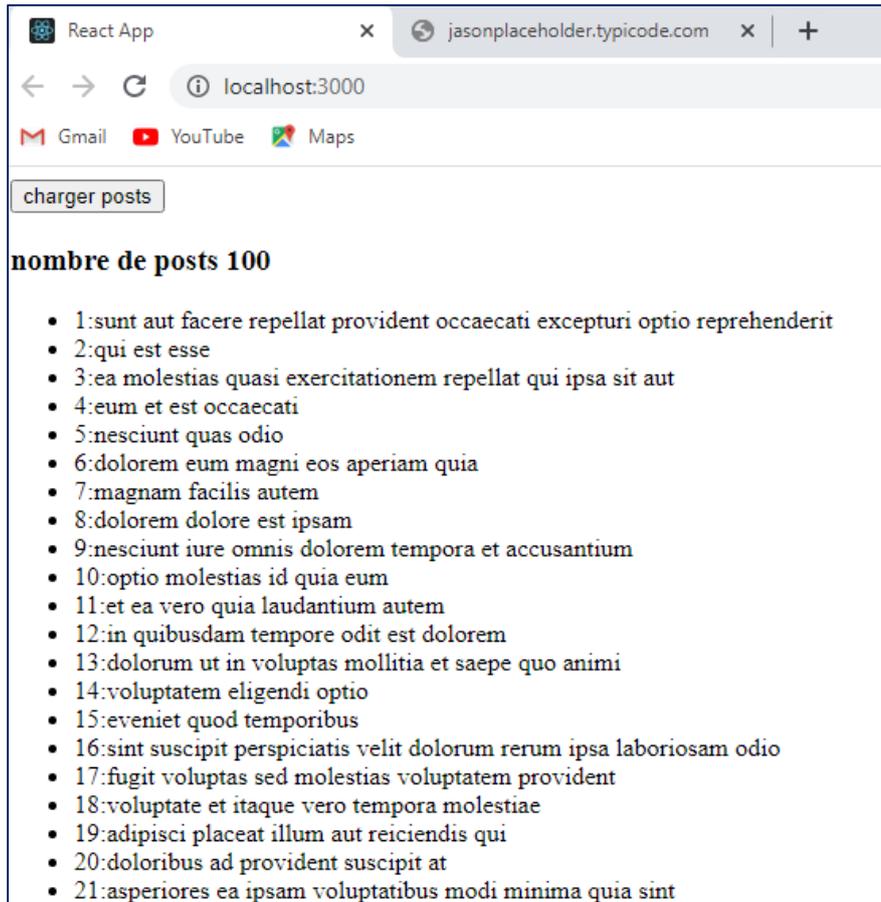
`fetchPostsRequest` et `fetchPostsFailure` si échec

18.2.2 Consommation de l'API posts

Appel de l'API end point : <https://jsonplaceholder.typicode.com/posts>



Le bouton `charger posts` permet de charger les posts en utilisant `Redux Axios` et `middleware Thunk`



Pour ce faire il faut ajouter les dépendances suivantes :

- `npm install react-redux redux`
- `npm install redux-thunk`
- `npm install axios`
- `npm install --save-dev redux-devtools-extension`

Préparation des actions

postsType.js

```
export const FETCH_POSTS_REQUEST='FETCH_POSTS_REQUEST';  
export const FETCH_POSTS_SUCCESS='FETCH_POSTS_SUCCESS';  
export const FETCH_POSTS_FAILURE='FETCH_POSTS_FAILURE';
```

postsActions.js

```
import * as type from './postsType'  
import axios from 'axios'  
  
//actions createurs  
export const fetchPostsRequest={()=>{  
  return{  
    type:type.FETCH_POSTS_REQUEST  
  }  
}}
```



```
export const fetchPostsSuccess=(posts)=>{
  return {
    type:type.FETCH_POSTS_SUCCESS,
    payload:{posts:posts}
  }
}

export const fetchPostsFailure=(err)=>{
  return{
    type:type.FETCH_POSTS_FAILURE,
    payload:{error:err}
  }
}

//fonction thunk fetchPosts
export const fetchPosts=function(){
  return function(dispatch,getState){
    dispatch(fetchPostsRequest())

    axios.get('https://jsonplaceholder.typicode.com/posts').
    then(response=> dispatch(fetchPostsSuccess(response.data))).
    catch(err=>dispatch(fetchPostsFailure(err.message)))
  }
}
```

Préparation de store :

Store.js

```
import {applyMiddleware} from 'redux'
import thunk from "redux-thunk";
import {composeWithDevTools } from 'redux-devtools-extension'
import { legacy_createStore as createStore } from 'redux';
import * as type from './postsType'

const initialState={
  loading:false,
  data:[],
  error:''
}

const reducer=(state=initialState,action)=>{
  switch(action.type){
    case type.FETCH_POSTS_REQUEST:
      return {loading:true,data:[],error:''}
    case type.FETCH_POSTS_SUCCESS:
      return{ loading:false,
        data:action.payload.posts,
```



```
        error:''}
    case type.FETCH_POSTS_FAILURE:
        return {loading:false,
                data:[],
                error:action.payload.error}
    default:
        return state
}}

export default createStore(reducer,composeWithDevTools(applyMiddleware(thunk)));
```

L'écriture

applyMiddleware(thunk) : permet d'ajouter le middleware thunk

composeWithDevTools(applyMiddleware(thunk)) : permet d'ajouter le Redux devTools

et en fin cette écriture permet de créer le store avec thunk et devTools
createStore(reducer,composeWithDevTools(applyMiddleware(thunk)));

index.js

```
import ReactDOM from 'react-dom/client';
import App from './components/App';
import store from './components/store';
import {Provider} from 'react-redux';
const root=ReactDOM.createRoot(document.getElementById('root'));

root.render(<Provider store={store}>
    <App/>
</Provider>)
```

Préparation des composants user interface Posts et App

Posts.js

```
import {useSelector} from 'react-redux'
export default function Posts(){

const posts=useSelector(state=>state.data)
    return(<ul>
        {posts.map((post,index)=><li key={index}><post.id>:{post.title}</li>)}
        </ul>)
}
```



App.js

```
import {useSelector,useDispatch} from 'react-redux'  
import Posts from './Posts';  
import { fetchPosts } from './postsActions'  
export default function App(){  
  
  const posts=useSelector(state=>state)  
  
  const dispatch=useDispatch();  
  function charger(){  
    dispatch(fetchPosts())  
  }  
  return(<>  
    <button onClick={()=>charger()}>charger posts</button>  
    {posts.loading && <h1>chargement</h1>}  
    {(!posts.loading && !posts.error) && <div><h3>nombre de posts  
{posts.data.length}</h3>  
    <Posts/>  
    </div>}  
    {(!posts.loading && posts.error) && <h3>erreur {posts.error}</h3>}  
  
    </>  
  )  
}
```

Vous trouvez ci-joint le projet de cours :

cours-redux-api.rar qui contient déjà les dépendances nécessaires

IL suffit de télécharger les dépendances par

lancez :npm install

Exercice d'application :

Appel de l'API end point : <https://jsonplaceholder.typicode.com/users>

En utilisant Redux créer l'application qui affiche la liste des users

Allez bon courage.