
 Se former autrement HONORIS UNITED UNIVERSITIES	Année Universitaire : 2022-2023 
Atelier n° 5: <h2 style="text-align: center;">Redux</h2>	

Objectifs :

- Comprendre le concept de Redux.
- Créer un Redux Boilerplate avec redux-toolkit.
- Créer un exemple d'utilisation de redux.
- Implémenter une gestion de panier dans notre boutique en ligne.

### Travail à faire :

Au cours de ce workshop, nous allons ré-implémenter certaines de nos fonctionnalités en utilisant Redux.

- 1) Installer les dépendances redux, react-redux, redux-persist, redux-thunk et @reduxjs/toolkit .

Commençons par créer notre boilerplate, il ressemblera à ceci :

- 2) Créer la même structure de dossiers.



Figure 1 : La structure du boilerplate

- 3) Créez les fichiers js ci-dessus et laissez-les vides.

- 4) Modifier le fichier **productsSlice.js** en ajoutant le code suivant :

```

import { createSlice } from "@reduxjs/toolkit";
import { getAllProducts } from "../../service/api";
let initialState = {
  products: [],
  selectedProduct: {},
  errors: "",

```

```

};
const productsSlice = createSlice({
  name: "products",
  initialState,
  reducers: {
    populateProducts(state, action) {
      state.products = action.payload;
    },
    selectProduct(state, action) {
      state.selectedProduct = action.payload;
    },
    unselectProduct(state) {
      state.selectedProduct = null;
    },
    deleteProductReducer: (state, action) => {
      const payload = action.payload;
      const index = state.products.findIndex((item) => item.id ===
payload);
      if (index !== -1) {
        state.products.splice(index, 1);
      }
    },
    updateProductReducer: (state, action) => {
      const payload = action.payload;
      const index = state.products.findIndex(
        (item) => item.id === payload.id
      );
      if (index !== -1) {
        state.products[index] = payload;
      }
    },
    addProductReducer: (state, action) => {
      const payload = action.payload;
      state.products.push(payload);
    },
    setErrors(state, action) {
      state.errors = action.payload;
    },
  },
});
export const fetchProducts = () => async (dispatch) => {
  getAllProducts()

```

```

.then((response)=>{dispatch(populateProducts(response.data));dispatch(setErrors(null))})
    .catch((error)=>dispatch(setErrors(error)));

};

export const selectProducts = (state) => {
    return [state.products.products, state.products.errors];
};

export const selectSelectedProduct = (state) => {
    return state.products.selectedProduct;
};

export const {
    populateProducts,
    selectProduct,
    unselectProduct,
    setErrors,
    deleteProductReducer,
    updateProductReducer,
    addProductReducer,
} = productsSlice.actions;
export default productsSlice.reducer;

```

Figure 2 : Pseudo-code du ProductsSlice

- 5) Mettre à jour le fichier **reducers.js** afin de combiner nos réducteurs en une seule entité.
- 6) Mettre à jour le fichier **store.js**.
- 7) Modifiez le code **index.js** afin d'appliquer le store à notre application.

**Remarque** : vous allez utiliser le **Provider** de la librairie **react-redux** et le **persistStore** de la librairie **redux-persist**.

- 8) Modifier le App.js afin de faire l'appel de l'action de la sélection de la liste des produits.

**Remarque** : Vous allez utiliser le prop **loader** et le hook **useDispatch** pour faire appel à l'action de sélection de la liste des produits.

**useDispatch** est un hook qui nous permettra d'appeler nos actions que nous avons définies dans nos tranches.

- 9) Modifier le fichier **Products.js** en utilisant la logique du redux.

**Remarque** : Vous allez utiliser le hook **useSelector** pour accéder à l'état du magasin redux.

**useSelector** est un hook pour accéder à l'état du magasin redux. Ce hook prend une fonction de sélecteur comme argument. Le sélecteur est appelé avec l'état du magasin. Comme il s'agit d'un hook, il écouter les changements, il va donc déclencher un nouveau rendu chaque fois que l'état sélectionné est modifié.

- 10) Modifier les fichiers **AddProducts.js**, **Product.js** et **UpdateProduct.js** en utilisant la logique du redux.
- 11) Modifier le fichier **cartSlice.js** afin d'implémenter une logique bien déterminée pour le système d'un panier dans notre boutique.
- 12) Ajouter la partie du panier dans le **navbar** et le bouton "ADD TO CART" dans le composant **Product.js**.

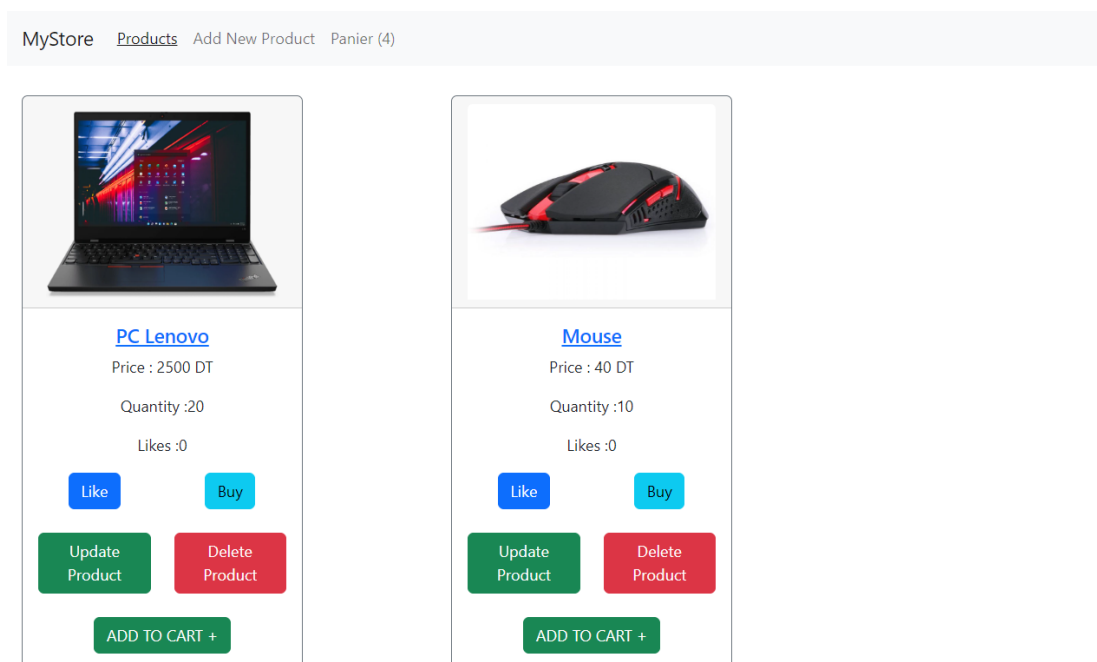


Figure 3: Interface de la liste des produits avec la nouvelle fonctionnalité du panier

- 13) Implémenter la méthode **addToCart** pour faire appel à l'action appropriée à l'ajout d'un produit dans le panier.
- 14) Implémenter l'interface du panier dans un nouveau fichier **Cart.js**.

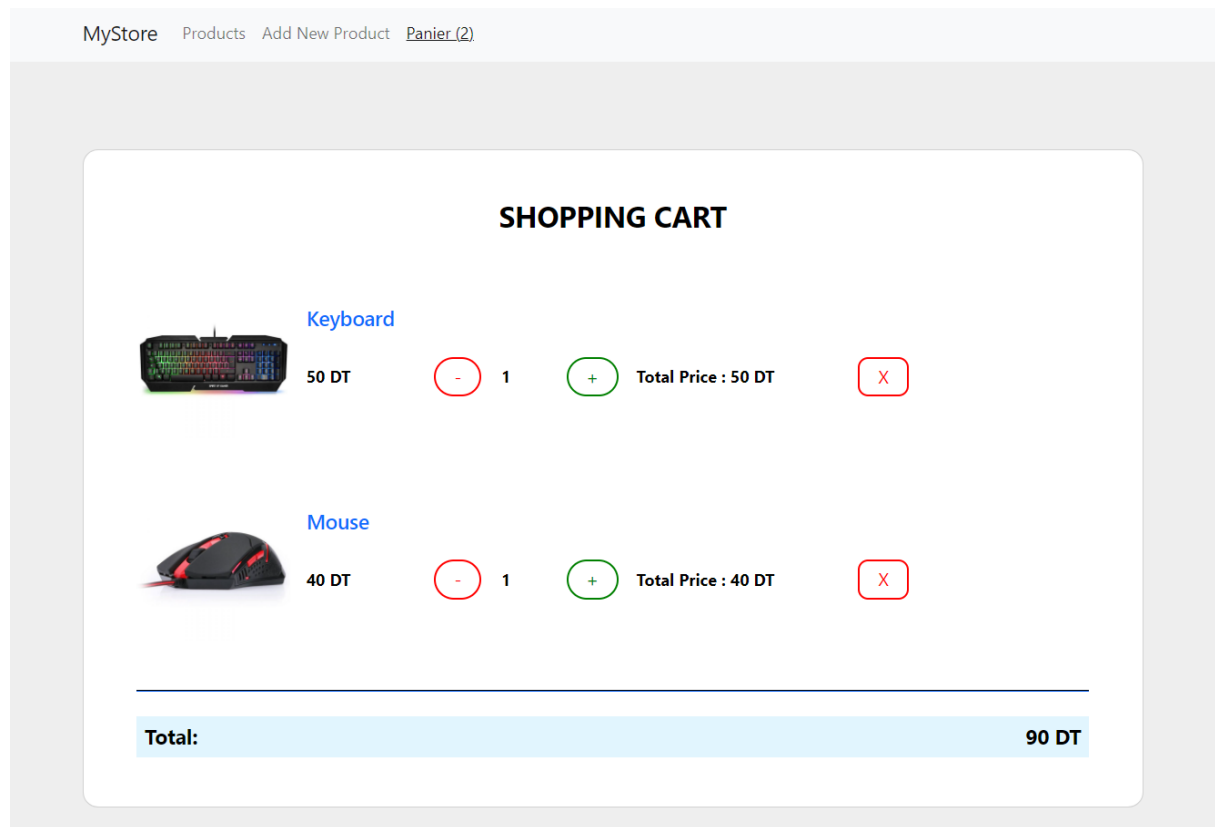


Figure 4 : Interface du panier

15) Implémenter les méthodes suivantes :

- **addItemToCart** : pour faire appel à l'action appropriée à l'ajout d'un produit dans le panier.
- **RemoveItemFromCart** : pour faire appel à l'action appropriée à la décrémentation de la quantité d'un produit dans le panier.
- **DeleteItem** : pour faire appel à l'action appropriée à la suppression d'un produit du panier.