
 <p>Se former autrement HONORIS UNITED UNIVERSITIES</p>	<p>Année Universitaire : 2023-2024</p> 
<p>Atelier n° 5:</p> <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 d'une liste de souhaits à notre plateforme de gestion d'événements.

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.

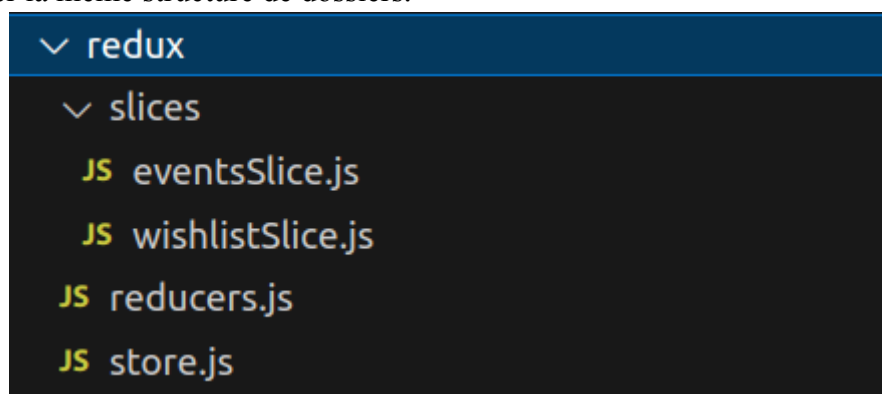


Figure 1 : La structure du boilerplate

- 3) Créez les fichiers js ci-dessus et laissez-les vides.
- 4) Modifier le fichier **eventsSlice.js** en ajoutant le code suivant :

```
import { createSlice } from "@reduxjs/toolkit";
import { getAllEvents } from "../../services/api";

let initialState = {
  events: [],
  selectedEvent: {},
  errors: "",
```

```

});
const eventsSlice = createSlice({
  name: "events",
  initialState,
  reducers: {
    populateEvents(state, action) {
      state.events = action.payload;
    },
    selectEvent(state, action) {
      state.selectedEvent = action.payload;
    },
    unselectEvent(state) {
      state.selectedEvent = null;
    },
    deleteEventReducer: (state, action) => {
      const payload = action.payload;
      state.events = state.events.filter(
        (eventItem) => eventItem.id !== payload
      );
    },
    updateEventReducer: (state, action) => {
      const payload = action.payload;
      const index = state.events.findIndex((item) =>
item.id === payload.id);
      if (index !== -1) {
        state.events[index] = payload;
      }
    },
    addEventReducer: (state, action) => {
      const payload = action.payload;
      state.events.push(payload);
    },
    setErrors(state, action) {
      state.errors = action.payload;
    },
  },
});

```

```

export const fetchEvents = () => async (dispatch) =>
{
  try {
    const eventsResult = await getAllEvents();
    dispatch(populateEvents(eventsResult.data));
    dispatch(setErrors(null));
  } catch (error) {
    dispatch(setErrors(error));
  }
};

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

export const selectSelectedEvent = (state) => {
  return state.events.selectedEvent;
};

export const {
  populateEvents,
  selectEvent,
  unselectEvent,
  setErrors,
  deleteEventReducer,
  updateEventReducer,
  addEventReducer,
} = eventsSlice.actions;

export default eventsSlice.reducer;

```

Figure 2 : Code du EventsSlice

- 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 dans le fichier **main.jsx** 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.jsx afin de faire l'appel de l'action de la sélection de la liste des événements.

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

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

- 9) Modifier le fichier **Events.jsx** 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 écouterait les changements, il va donc déclencher un nouveau rendu chaque fois que l'état sélectionné est modifié.

- 10) Modifier les fichiers **AddEvents.jsx**, **Event.jsx** et **UpdateEvent.jsx** en utilisant la logique du redux.
- 11) Modifier le fichier **wishlistSlice.js** afin d'implémenter une logique bien déterminée pour le système d'une liste de souhait dans notre boutique.
- 12) Ajouter la partie de la liste de souhait dans le **navbar** et le bouton "ADD TO WISHLIST" dans le composant **Event.jsx**.



Figure 3 : Navbar et le bouton "ADD TO WISHLIST"

- 13) Implémenter la méthode **addToWishlist** pour faire appel à l'action appropriée à l'ajout d'un événement dans la liste de souhait.
- 14) Implémenter l'interface de la liste de souhait dans un nouveau fichier **Wishlist.jsx**.

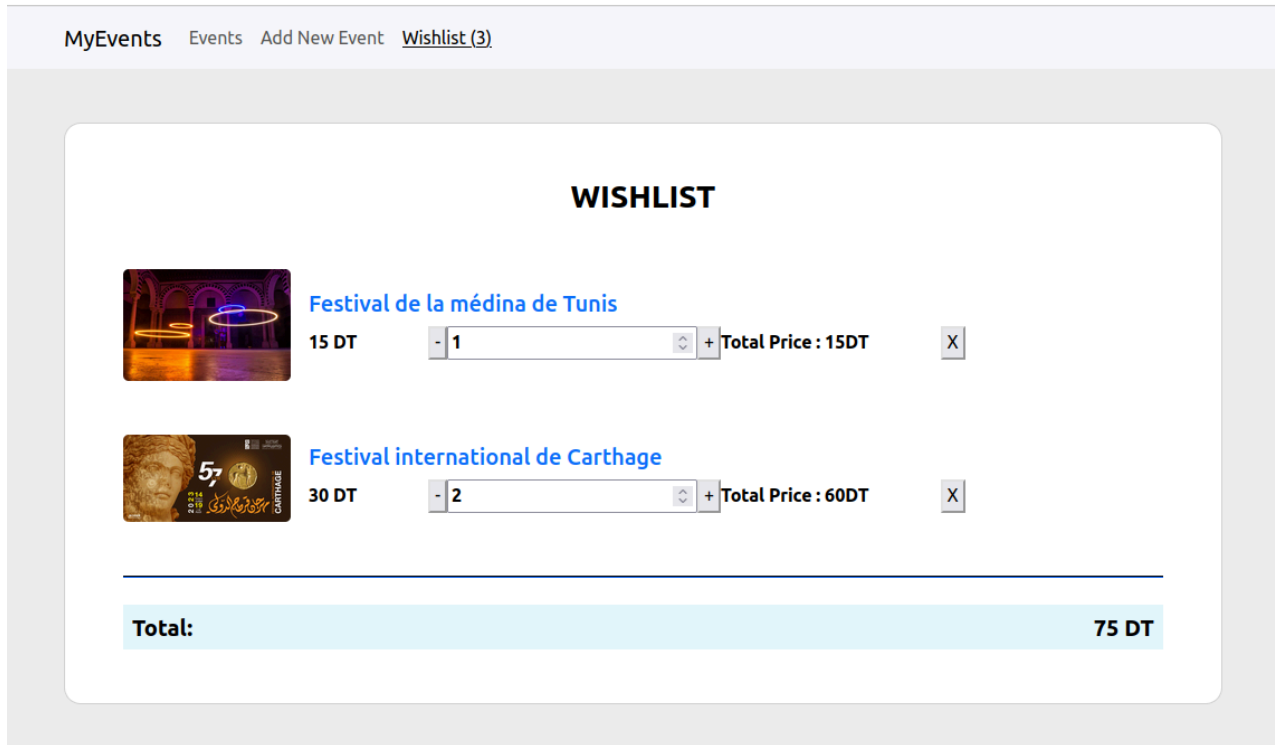


Figure 4 : Interface de la liste de souhait

- 15) Implémenter les méthodes suivantes :
- **addItemToWishlist** : pour faire appel à l'action appropriée à l'ajout d'un événement dans la liste de souhait.
 - **RemoveItemFromWishlist** : pour faire appel à l'action appropriée à la décrémentation du nombre de tickets d'un événement dans la liste de souhait.
 - **DeleteItem** : pour faire appel à l'action appropriée à la suppression d'un événement de la liste de souhait.