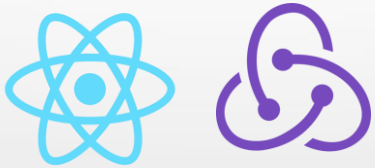
esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

CSA2
Application Côté Client 2

Le conteneur d'état REDUX

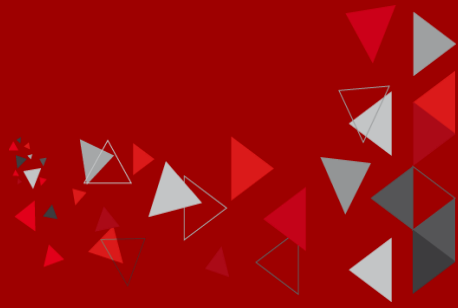


United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
EUR-ACE® Délivrée par la Commission des Titres d'Ingénieur
CONFERENCE DES GRANDES ÉCOLES
cdio CONCEIVE DESIGN IMPLEMENT OPERATE

Année universitaire : 2022/2023

- Problématique
- Context API vs Redux
- Pourquoi Redux ?
- Introduction à Redux
- Les notions de base Redux
- Les concepts fondamentaux de Redux
- L’architecture Redux

Exemples



Problématique

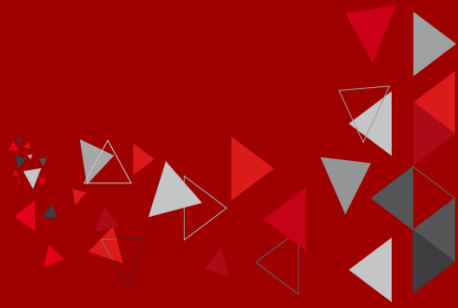
Problématique

- Le moyen le plus simple de transmettre des données d'un **composant parent** à un **composant fils** dans une application **React** consiste à les transmettre aux **props** de l'enfant.

❖ Mais un problème survient lorsqu'un enfant **profondément imbriqué** nécessite des données d'un composant plus haut dans l'arborescence.

Problème : Prop drilling

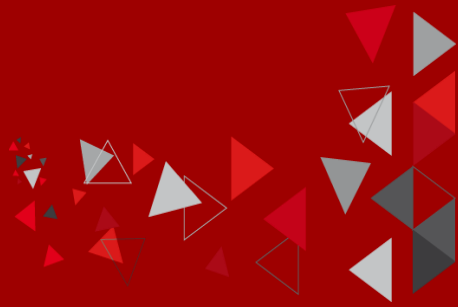
❖ Pour résoudre ce problème, nous avons des solutions de gestion d'état comme **Context API** et **Redux**.



Context API vs Redux

Context API vs Redux

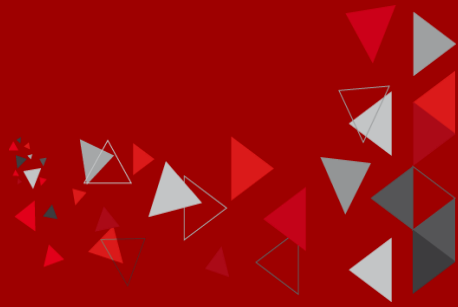
Context API	*Redux*
Outil intégré livré avec React	Installation supplémentaire requise
configuration minimale	Nécessite une configuration étendue pour l'intégrer à une application React
Spécialement conçu pour les données statiques, qui ne sont pas souvent actualisées ou mises à jour.	Fonctionne comme un charme avec des données statiques et dynamiques
L'ajout de nouveaux contextes nécessite une création à partir de zéro	Facilement extensible en raison de la facilité d'ajout de nouvelles données/actions après la configuration initiale.



Pourquoi Redux ?

Pourquoi Redux ?

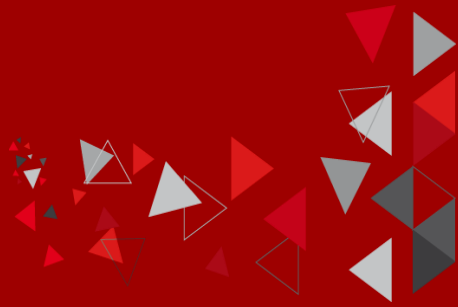
- En production, aujourd'hui, dès qu'un projet atteint une taille moyenne, il devient très pratique d'utiliser un système de **centralisation des données** et des **actions**, afin de simplifier le développement de votre application.
  - ❖ Redux est dédié pour les grands projets et l'API Context pour les petits projets.
- Redux **permet** une **gestion** de **"states globaux"** de l'application.
- Redux est une **librairie très légère**, et il n'y a que **3 notions de base** à comprendre.



Introduction à Redux

Introduction à Redux

- Peu de temps après la création de React Facebook a introduit une **bibliothèque** JavaScript nommé **Flux** pour gérer l'état des application.
- En **mai 2015** un certain **Dan Abramov** a annoncé une nouvelle **bibliothèque** appelée **Redux** : basée sur la même architecture de Flux mais qui est moins complexe.
- Redux et Flux existent en parallèle mais **Redux** est plus **populaire** et plus **utilisé.**

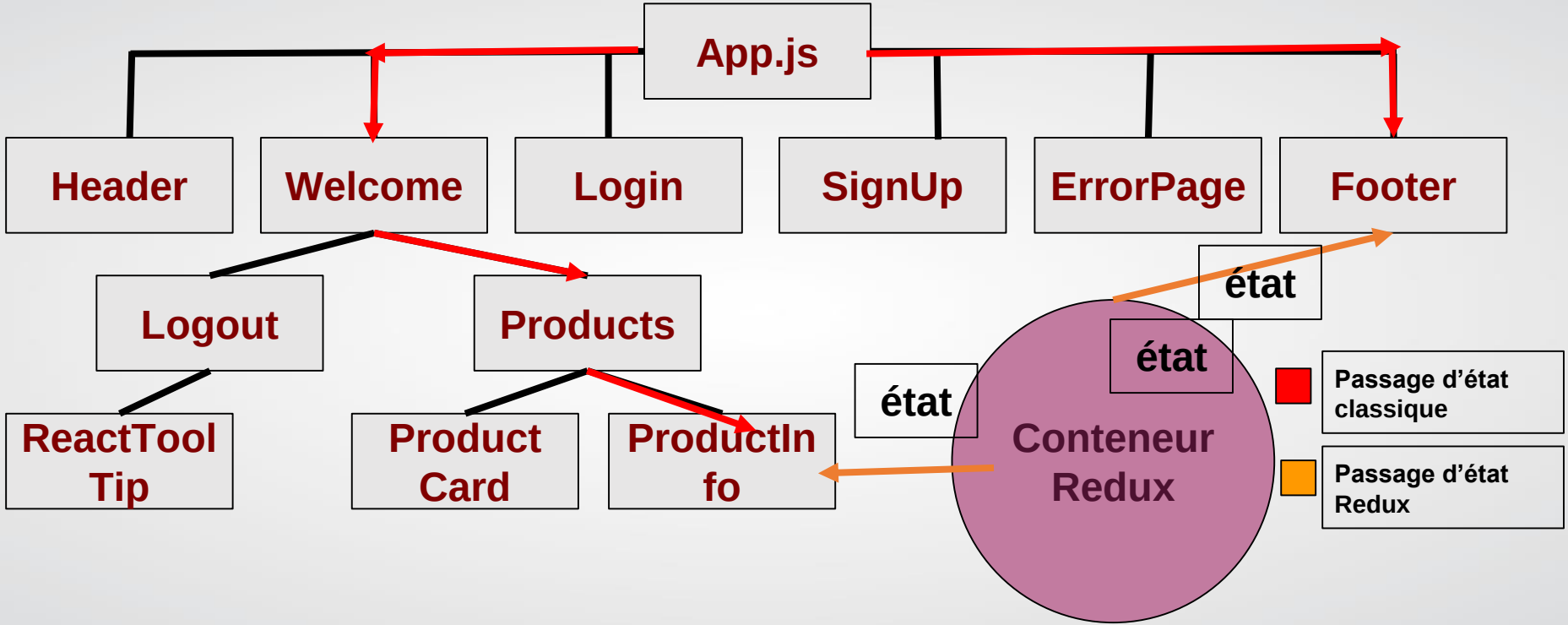


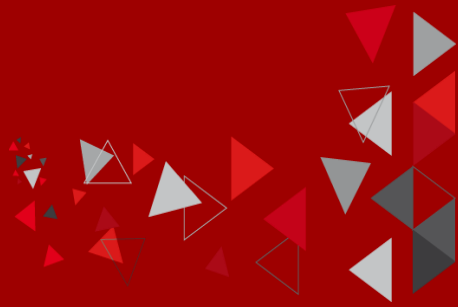
Les notions de base Redux

Notions de base Redux

“ Redux est un **conteneur d’état** **prévisible** pour les **applications JS**”

Conteneur d’état





Les concepts fondamentaux de Redux

Les concepts fondamentaux de Redux

STORE

```
state = {
    numItems: 10
}
```

ACTION

```
{
    type: 'BUY_ITEM'
}
```

REDUCER

(prevState, action) => **newState**

Les concepts fondamentaux de Redux

- **Les actions**

  - Servent à modifier l'état global de l'application.
  - Se sont des objets.
  - Représentent une interaction avec le state global.
  - N'apportent rien de fonctionnel.
  - Ils apportent un type qui les représente.
  - Peuvent contenir plusieurs arguments.

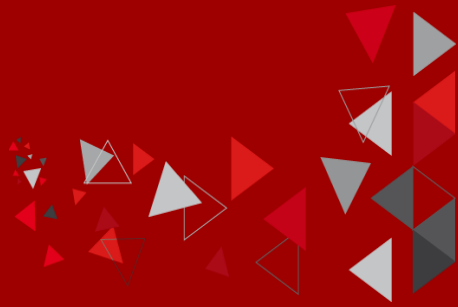
Les concepts fondamentaux de Redux

- **Les reducers**

  - Des fonctions qui prennent comme argument : 1- Un state 2- Une action et retournent un nouvel état (state) après que l'action soit effectuée.

- **Le store**

  - Avec Redux, tout état d'une application est décrit comme un objet.
  - Cet objet est enregistré dans ce qu'on appelle **un store**.



Avec redux

Les actions

```
export const CartActionTypes = {
  CART_ADD_ITEM: 'CART_ADD_ITEM',
  ...
};
```

```
import { CartActionTypes } from './cart.actionTypes';

export const cartAddItem = (item) => ({
  type: CartActionTypes.CART_ADD_ITEM,
  payload: item,
});
```

Les reducers

```
import { CartActionTypes } from './cart.actionTypes';

const INITIAL_STATE = {
  cartItems: [],
};

const cartReducer = (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case CartActionTypes.CART_ADD_ITEM:
      return {
        //...
      };
      //...
```

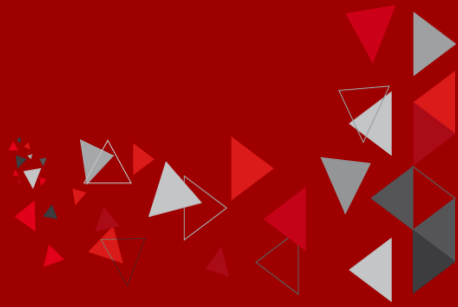
Le store

Créer **un magasin Redux store** qui contient l'arborescence d'état complète de votre application. Il ne devrait y avoir qu'un seul magasin dans votre application.

```
import { createStore } from 'redux';
import cartReducer from './cart/cart.reducer';

const store = createStore(cartReducer);

export { store };
```



Avec Redux Toolkit

@reduxjs/toolkit

Les reducers

```
import { createSlice } from "@reduxjs/toolkit";

export const slice = createSlice({
    name: "Nom_Slice",
    initialState: {
        // ...
    },
    reducers: {
        function: (state,action) => {
            // ...
        },
    }
});

export const { function } = slice.actions;
export default slice.reducer;
```

Une fonction qui accepte un **état initial**, **un objet de fonctions de réducteur** et **un "nom de tranche"**. Elle génère automatiquement des **créateurs d'action** et **des types d'action** qui correspondent aux réducteurs et à l'état.

- En interne, elle utilise **createAction** et **createReducer.**

Le store

```
import { configureStore } from "@reduxjs/toolkit";
import reducer from "./reducer";

export default configureStore({
    reducer: {
        reducer: reducer
    }
});
```

Mise en place

```
<Provider store={store}>
    <App />
</Provider>
```

- Pour donner accès au magasin à notre application, nous devons envelopper notre application à l'intérieur du **<Provider/>** qui est importé de **react-redux**.
- Il prend comme **props** le store.

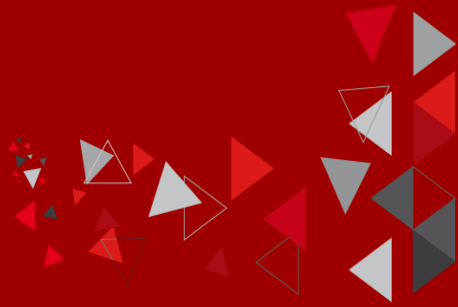
Mise en place

useSelector() : Vous permet d'extraire des données de l'état du magasin Redux, à l'aide d'une fonction de sélection.

```
import { useSelector, useDispatch } from 'react-redux';
import { function } from './redux/reducer';

const Composant = () => {
    const reducerState = useSelector((state) => state.reducer);
    const dispatch = useDispatch();
    const doSomething = () = > dispatch(function)
    return (
        <>
            {/* ... */}
        </>
    );
}
export default Composant;
```

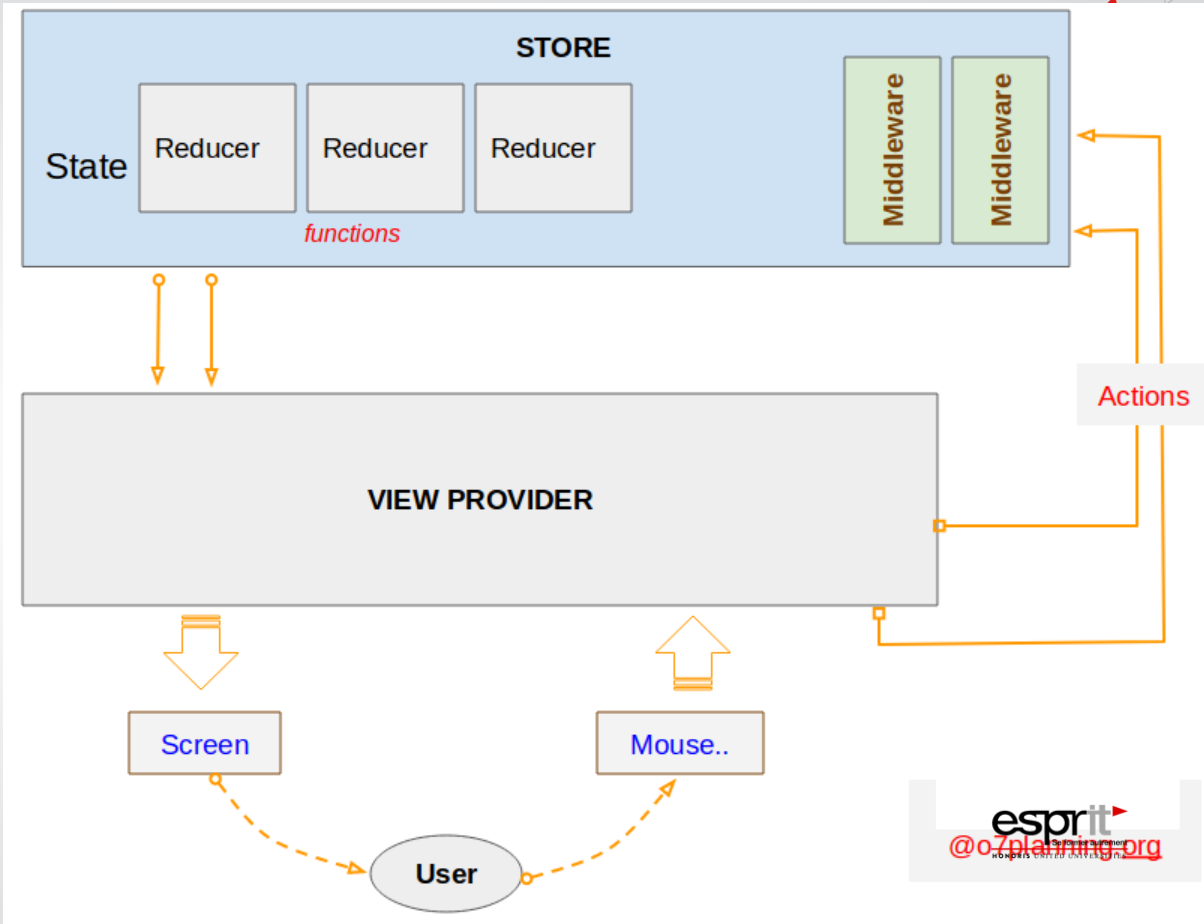
useDispatch(): Ce hook renvoie une référence à la fonction dispatch du magasin Redux. Vous pouvez l'utiliser pour dispatcher les actions nécessaires.



L’architecture Redux

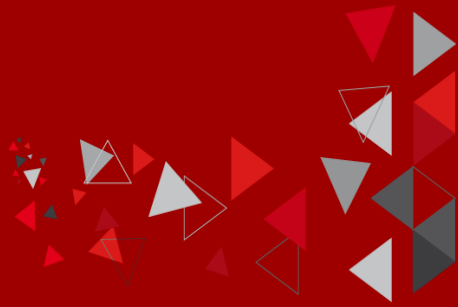
L’architecture Redux 1/2

- Redux n’a pas de concept dispatcher comme Flux (il a plutôt une fonction dispatch)
- Redux n’a qu’un seul store.
- Les objets Actions seront reçus et gérés directement par le store.



L’architecture Redux 2/2

- **VIEW PROVIDER**: Représente une application coté client qui peut être React ou Angular ou n’importe quelle application JavaScript
- **ACTION**: un objet pur créé pour stocker les informations relatives à l'événement d'un utilisateur (click sur un bouton, click sur une interface, drag and drop ...). Il inclut les informations telles que: le type d'action, l'heure de l'événement, l'emplacement de l'événement, ses coordonnées et quel est le state qu'il vise à modifier.
- **STORE**: Gère l'état de l'application et contient une fonction dispatch (action) qui distribue les actions pour chaque état.
- **MIDDLEWARE**: (Logiciel intermédiaire) Fournit un moyen d'interagir avec les objets Action envoyés au **STORE** avant leur envoi au **REDUCER**
- **REDUCER:** (Modificateur) Une fonction pure pour renvoyer un nouvel état à partir de l'état initial.



Exemples d’utilisation Redux

Exemple : Compteur

```
import { createSlice, configureStore } from '@reduxjs/toolkit'

const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0
  },
  reducers: {
    incremented: state => {
      // Redux Toolkit allows us to write "mutating" logic in reducers. It
      // doesn't actually mutate the state because it uses the Immer library,
      // which detects changes to a "draft state" and produces a brand new
      // immutable state based off those changes
      state.value += 1
    },
    decremented: state => {
      state.value -= 1
    }
  }
})

export const { incremented, decremented } = counterSlice.actions

const store = configureStore({
  reducer: counterSlice.reducer
})

// Can still subscribe to the store
store.subscribe(() => console.log(store.getState()))

// Still pass action objects to `dispatch`, but they're created for us
store.dispatch(incremented())
// {value: 1}
store.dispatch(incremented())
// {value: 2}
store.dispatch(decremented())
// {value: 1}
```

Merci de votre attention