Vous venez de terminer le cours [Créez une application complète avec HYPERLINK "https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete"React](https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete), ou vous maîtrisez déjà les bases et souhaitez vous rapprocher d’un exercice de terrain ? Ce cours est fait pour vous !

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. **Le plus populaire est Redux** – il y a en moyenne [6 téléchargements de HYPERLINK "https://npm-stat.com/charts.html?package=redux&package=react&from=2017-07-30&to=2021-06-15"Redux HYPERLINK "https://npm-stat.com/charts.html?package=redux&package=react&from=2017-07-30&to=2021-06-15" pour 10 de HYPERLINK "https://npm-stat.com/charts.html?package=redux&package=react&from=2017-07-30&to=2021-06-15"React](https://npm-stat.com/charts.html?package=redux&package=react&from=2017-07-30&to=2021-06-15) ; c’est celui-ci que l’on découvrira dans ce cours.

Mais à quoi sert donc Redux ?

**Redux permet une** **gestion de "states globaux"**. **Grâce à Redux, vous palliez les faiblesses des props qui ne sont accessibles qu’en lecture seule, ou des states qui sont liés à un unique composant, et donc seulement localement.**

Redux permet donc de faciliter le développement de toute application dès que celle-ci implique l’existence de composants dépendant les uns des autres.

Dans ce cours, nous travaillerons sur trois cas pratiques où Redux se montrera très utile. Nous verrons d’abord Tennis Score, pour mettre en place Redux sur une petite application et comprendre son fonctionnement. Dans la partie 2, on utilisera une version de Tennis Score construite avec React pour voir comment utiliser les deux ensemble.

Puis, nous ajouterons Redux à une application plus conséquente. Il s’agit de Shiny, dont vous devez vous souvenir si vous avez suivi le prérequis de ce cours !

Enfin, je vous partagerai quelques outils et astuces pour évoluer dans cet écosystème de React, et continuer votre apprentissage.

Vous êtes prêt à passer au niveau supérieur avec Redux ? Alors rejoignez-moi dans ce cours !

**À la fin de ce cours, vous serez capable de :**

Identifier les concepts clés du state management et de Redux.

* Manipuler des states complexes d’une application React avec Redux.
* Déployer Redux dans un projet à grande échelle.
* Optimiser le state management avec des outils de l’écosystème React-Redux.

**Prérequis**

Pour réussir à suivre ce cours, vous devez être en mesure de créer une application React dans son intégralité. Vous pouvez acquérir ces connaissances de React en suivant le cours :

* [**Créez HYPERLINK "https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete" HYPERLINK "https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete"une HYPERLINK "https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete" application React HYPERLINK "https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete"complète**](https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete)

**Outils nécessaires**

* Un IDE, tel que VSCode
* React et create-react-app
* Redux
* GitHub

Enfin, ce cours a été conçu autour de deux cas fil rouge, pour vous aider à **mettre en pratique le state management**. Nous verrons d’abord **Tennis Score**, que nous utiliserons pour découvrir le state management avec Redux et puis avec React. Dans un second temps, nous utiliserons **Shiny**, une application React complète à laquelle nous ajouterons Redux et quelques nouvelles fonctionnalités ! Si vous avez suivi le cours [Créez une application HYPERLINK "https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete"React HYPERLINK "https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete" complète](https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete), vous devez déjà connaître Shiny. 😉

Vous verrez que la plupart des ressources dans ce cours seront en anglais, sauf s’il s’agit d’un cours d’OpenClassrooms ou si c’est autrement indiqué.

**Reprenez les cours sur React**

Un petit doute sur le fonctionnement du state ou des props ? Ce sont des concepts fondamentaux que nous allons utiliser dans ce cours. Il est donc important de bien les avoir en tête avant de vous lancer.

N'hésitez pas à [reprendre les cours sur HYPERLINK "https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete"React](https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete), notamment les chapitres sur le state, le passage des props et le contexte.

**Testez vos acquis**

Chaque partie se termine par un quiz. Ces derniers vous permettent de vérifier que vous avez bien compris les principaux concepts de chaque chapitre.

Si vous faites des erreurs, ne baissez pas les bras ! C’est normal de ne pas tout assimiler à la première lecture. Reprenez le ou les chapitres qui posent problème, puis retentez votre chance ! Je vous encourage également à mettre en pratique ce que je vous montre, avec les fils rouges et les quiz, cela vous aidera dans votre compréhension du state management et de Redux.

**Découvrez le state management**

Vous avez peut-être déjà entendu le terme “State Manager”, mais qu’est-ce que cela veut dire exactement ? Et puis quel est le lien avec le state de React ?

Nous allons voir tout cela ensemble, mais d’abord un petit rappel sur **le fonctionnement du state dans React**.

**Découvrez comment fonctionne le state dans React**

**Dans React, le state nous permet de** **contrôler l’affichage et la logique de l’application**.

Chaque composant peut contenir un ou plusieurs states, grâce au hook  useState.

Dès qu'un des states change, le composant est **automatiquement mis à jour**.

Voici un petit exemple :

*function* MonComposant() {

// on utilise le hook useState de React

*const* [count, setCount] = React.useState(0);

return (

// on affiche un bouton

<button

// au clique, on change le state avec setCount

onClick={() => setCount(count + 1)}

>

{/\* on utilise le state dans l'affichage \*/}

{count}

</button>

);

}

[Ouvrir l’exemple sur HYPERLINK "https://codepen.io/nicolaspatschkowski/pen/yLMzPad?editors=0010"CodePen](https://codepen.io/nicolaspatschkowski/pen/yLMzPad?editors=0010)

Il arrive souvent qu’un composant ait besoin d’un state stocké dans un autre composant. Dans ce cas, on peut utiliser les props pour transmettre les données nécessaires. Mais attention, car **les props ne vont que dans un sens**: des composants parents vers les composants enfants. Il faut donc bien choisir où l’on met son state. Quand ce choix devient difficile, on peut faire appel aux **state managers**.

**Comprenez ce qu’est un State Manager**

Le terme “State Manager” désigne **un outil qui permet de manipuler le state d’une application**.

Dans ce cas, React est un State Manager, puisque le  useState  permet de manipuler le state ?

Exactement ! Cependant, dans l'écosystème React, lorsque l’on fait référence à la notion de state manager, on désigne en réalité un outil **autre que React** pour manipuler le state.

Quel intérêt d’utiliser un outil externe alors que React permet déjà la gestion de state ?

Si useState  fonctionne très bien pour les petites et moyennes applications, lorsqu'une application atteint une certaine taille avec plusieurs dizaines, voire centaines de composants, useState  ne suffit plus. Il devient alors intéressant d’utiliser un state manager externe pour gérer **les states globaux**.

Un state global ? Qu’est-ce que c’est ?

C’est ce que nous allons voir ensemble dans la suite du cours !

**Identifiez les states globaux**

Dans une application, tous les states ne sont pas égaux, et on distingue deux grandes catégories de state : **local** et **global**.

*Les states locaux*

On parle de state local pour désigner **un state utilisé dans une seule et unique partie de l’application** (souvent un seul composant). Ce sont souvent des states qui contrôlent des éléments d’interface, par exemple :

* Un booléen qui détermine si un composant Dropdown  est ouvert ou fermé.
* Une chaîne de caractères qui détermine si un menu est caché, visible ou minifié.

Ce qui est bien avec les states locaux, c’est qu’une fois qu’ils sont en place, on peut oublier qu’ils existent.

Par exemple, récemment, dans une application, j’ai créé un composant NumberInput. Il était assez complexe avec de la validation, des valeurs minimum et maximum, etc. Finalement, le composant fait plus de **200 lignes et contient 5 ou 6 states**! Mais comme ce sont des states locaux, une fois le composant fini, je ne m’en préoccupe plus, j’ai juste à utiliser mon composant  <NumberInput />  , et ça fonctionne !

*Les states globaux*

À l’inverse des states locaux, les states globaux sont **des states utilisés à de multiples endroits de l'application**. Vous connaissez probablement quelques exemples :

* Le thème light/dark sélectionné par l’utilisateur.
* La langue d’affichage de l’interface.
* L’utilisateur actuellement connecté.

Dans une application React, les states globaux sont **plus complexes à gérer**. Il faut bien choisir dans quel composant on appelle useState, afin de pouvoir passer le state via les props à tous les composants qui en ont besoin. Dans de petites applications, nous pouvons gérer les states globaux sans besoin d’un state manager ; mais dès qu’une application devient un peu grosse, on se heurte à certains problèmes.

**Les problématiques liées aux states globaux**

Prenez la structure d’application suivante :



Wireframe et structure d’une application d’actualités

Où faut-il stocker le state correspondant aux informations de l’utilisateur actuellement connecté (le prénom “Paul”) ?

Dans le composant Header, peut-être ?



Le composant Welcome n’a pas accès au state

Cela semble fonctionner pour le composant UserMenu  , sauf que le composant Welcome  a lui aussi besoin du prénom de l’utilisateur (“Bonjour **Paul**”). Là nous avons un problème : comme on ne peut pas passer les props vers le haut, **il n’y a aucun moyen pour Welcome d'accéder au state**. Il faut donc déplacer le state dans App.



On déplace le state dans App

Dans cet exemple, on constate deux choses :

* Le composant dans lequel on déclare un state est important car il conditionne quel composant peut avoir accès au state.
* Plus un composant est haut dans la hiérarchie de composants, plus il y a d’intermédiaires. Dans l’exemple ci-dessus, il a fallu passer par le composant NewsFeed  pour que Welcome  ait accès au state déclaré dans App.

Quand on passe des props à un composant uniquement pour qu’un composant enfant y ait accès (comme c’est le cas de NewsFeed  ci-dessus), on appelle cela le **Props Drilling**. Un nom plutôt bien trouvé puisque les props “transpercent” nos composants.

L’exemple utilisé ici est très simple, mais plus une application est grosse et plus ces problématiques sont amplifiées.

J’ai aussi une anecdote sur les states globaux, mais elle est beaucoup moins glorieuse que celle sur les states locaux… À mes débuts avec React, pour être sûr que tous mes composants aient accès à tous les states, je mettais tous les states dans App  . 😬 Mais du coup, mon composant App  était **immense** et j’avais du props drilling dans tous les sens ! 🤦‍♂️

J’ai rapidement abandonné l’idée, et c’est en cherchant une autre solution que j’ai découvert la star de ce cours : **Redux**.

**Utilisez Redux : le plus populaire des state managers**

Redux est donc **une librairie de state management**, c’est même un des premiers state managers à avoir été massivement utilisé. Nous verrons le fonctionnement de Redux en détail dans le prochain chapitre, mais voici un premier aperçu :



Avec Redux le state est à l'extérieur de React

La création de Redux remonte à 2015 (soit l'âge de pierre à l'échelle du Web), et au début c’était surtout pensé comme un outil de développement. On peut d'ailleurs trouver [la vidéo de présentation sur YouTube](https://www.youtube.com/watch?v=xsSnOQynTHs), attention cependant car le code qui y est montré ne correspond pas aux bonnes pratiques actuelles !

La vidéo ci-dessus est en anglais. Vous verrez que la plupart des ressources dans ce cours seront en anglais, sauf s’il s’agit d’un cours d’OpenClassrooms ou si c’est autrement indiqué.

Nous verrons dans cette première partie que même si Redux est issu de l'écosystème React, il peut fonctionner seul, voire avec d’autres frameworks.

Aujourd’hui Redux totalise plus de 5 millions de téléchargements **par semaine** comme nous pouvons voir [sur sa fiche NPM](https://www.npmjs.com/package/redux), et 55 000 étoiles sur [son projet GitHub](https://github.com/reduxjs/redux), ce qui en fait **le** state manager le plus populaire. Pourtant, si on regarde le nombre de téléchargements de Redux par rapport à ceux de React, on constate une perte de vitesse ces dernières années. Cela peut s'expliquer par l'évolution de React et l'émergence de nouveaux outils.

Vous vous demandez peut-être si apprendre Redux aujourd’hui, alors qu’il est en perte de vitesse, est pertinent ? Et la réponse est **oui**, pour plusieurs raisons. Tout d’abord Redux reste très utilisé en entreprise, et donc beaucoup demandé sur le marché du travail. Avoir Redux dans son CV est un avantage certain pour un développeur de nos jours.

De plus, Redux est une très bonne introduction au monde des state managers, et même sans utiliser Redux, les concepts étudiés dans ce cours vous seront très utiles !

**En résumé**

* Nous utilisons le state pour contrôler l’interface utilisateur et partager des données entre composants.
* Le state local est restreint à un composant, alors que les states globaux sont partagés entre plusieurs composants.
* Partager le state global peut devenir compliqué – pour résoudre ce problème, nous avons des state managers.
* Redux est le plus populaire des state managers et sera la star de ce cours.