Activité - Moteur de recherche de jeux vidéo - Étape 2

Comme vous avez pu le constater, chaque composant React peut gérer ces données grâce à `useState`. Nous avons également vu que nous pouvions passer de courtes informations à l'aide du routeur. C'est ce que nous avons fait avec de `slug` vers la page de détails. Mais comment faire en sorte de centraliser des données et les rendre disponibles dans tous les composants ? Il existe plusieurs solutions, comme Redux, qui est très populaire. React fourni cependant de manière native un outil appelé "context", permettant de faire cela de manière assez simple.

Nous allons ajouter à notre application la possibilité de créer une liste de jeux favoris. Comme nous pourrons interagir avec les favoris à plusieurs endroits, nous aurons besoin d'un tableau, disponible dans tous les composants. Nous utiliserons donc un context.

Initialiser un "context"

Créer le context

Le context a besoin d'être placé dans un fichier que l'on pourra appeler dans les composants qui auront besoin de l'utiliser. Créons un fichier `/src/BookmarksContext.js`, déclarons un context, puis exportons le pour pouvoir l'utiliser dans d'autres fichiers.

```
import { createContext } from "react";
const BookmarksContext = createContext(null);
export default BookmarksContext;
```

Placer le context dans notre composant racine

Les applications React forment une cascade de composants. Pour faciliter le partage d'informations dans toute l'application, il est recommandé de créer notre context dans le composant racine (ici, App). Nous allons donc commencer à travailler dans le fichier `App.js`.

1. Importer le context

   ```
   import BookmarksContext from './BookmarksContext';
   ```

2. Préparer les données à partager

   Comme nos favoris aurons un impact sur l'affichage de l'application, commençons par créer un state local. Pour le moment, nous allons y placer de fausses valeurs pour tester. Ajoutez cela dans le composant `App` (au début de la fonction App).

   ```
   const [bookmarks, setBookmarks] = useState([
     {
       slug: "super-mario-bros-3",
       name: "Super Mario Bros. 3",
       background_image:
   "https://media.rawg.io/media/screenshots/092/092fc1910f067a95a07c0fbfd
     },
     {
       slug: "the-legend-of-zelda-the-wind-waker",
       name: "The Legend of Zelda: The Wind Waker",
       background_image:
   "https://media.rawg.io/media/games/45f/45f6d31b0fcefe029e33d258a7beb6a
     }
   ]);
   ```

3. Poser le context

   Finalement, il faut encadrer notre composant principal avec un celui fourni par notre `BookmarksContext`. On passe à ce composant les informations que l'on souhaite retrouver dans tous les sous composants (ici notre state `bookmarks`, ainsi que la fonction `setBookmarks` qui permet de le mettre à jour). Nous pourrons ajouter d'autres choses plus tard.

```
<BookmarksContext.Provider value={{bookmarks,setBookmarks}}>
  <RouterProvider router={router}></RouterProvider>
</BookmarksContext.Provider>
```

Utiliser le context dans les composants enfants

Travail préparatoire

Afin de tester, je vous propose de créer un nouveau composant `Bookmarks` (Bookmarks.js), dans le dossier `pages`.

```
import React from "react";

const Bookmarks = () => {

  return (
    <>
      <h1>Mes favoris</h1>

    </>
  )
}

export default Bookmarks;
```

Modifions ensuite la configuration du routeur (dans App.js), pour lui ajouter notre nouvelle page.

```
const router = createBrowserRouter([
  {
    path: "/",
```

```
    element: <Home />,
    errorElement: <ErrorMessage />,
  },
  {
    path: "/details/:slug",
    element: <Details />,
  },
  // On ajoute cette route
  {
    path: "/bookmarks",
    element: <Bookmarks />,
  },
], { basename: "/" })
```

Il ne nous reste plus qu'à ajouter un lien sur la page d'accueil (Home.js), pour pouvoir naviguer vers notre nouvelle page. Je vous conseille d'ajouter ce composant dans le JSX, juste au-dessus de la balise UL.

```
<Link to={'/bookmarks'}>Favoris</Link>
```

Si vous vous rendez sur la page d'accueil du site, vous devriez maintenant pouvoir naviguer vers notre nouvelle page de favoris.

Utilisation du context

Pour utiliser le context dans les composants enfants, il faut utiliser un nouveau `hook` React appelé `useContext`. Modifions le composant `Bookmarks` de manière à

- Récupérer le context
- Extraire les données souhaitées (grâce à useContext)
- Afficher la liste des faux favoris que nous avons listé dans `App.js`

Voici le code modifié du composant `Bookmarks` :

```
import React, { useContext } from "react";
import BookmarksContext from "src/BookmarksContext";

const Bookmarks = () => {
```

```
  const { bookmarks, setBookmarks } = useContext(BookmarksContext);

  return (
    <>
      <ul className="sm:w-full md:w-2/3 mx-auto px-2 text-2xl">
        {bookmarks.map((bookmark, index) => (
          <li className="py-2 px-4 border-b border-gray-500" key={index}>
{bookmark.name}</li>
        ))}
      </ul>
    </>
  )
}

export default Bookmarks;
```

En naviguant vers la page `Favoris`, vous devriez maintenant voir les deux jeux que nous avons ajoutés en dur dans le code au début de cette activité.

Prenez un instant pour améliorer l'apparence de la page.

Modifier les données du context

Si vous revenez sur le composant `App`, vous pourrez voir que lorsque nous avions placé le context à la racine de notre JSX, nous lui avions fourni la fonction `setBookmarks` qui permet de mettre à jour notre state `bookmarks`.

```
// nous avions passé des données au context dans App.js:
<BookmarksContext.Provider value={{ bookmarks, setBookmarks }}>
```

Nous pouvons donc utiliser cette fonction pour mettre à jour le state.

Supprimer un favoris

Je vous propose d'essayer de supprimer un élément de la liste des favoris. Pour cela, nous allons travailler dans le composant `Bookmarks` (Bookmarks.js).

1. Créer une fonction de suppression

Dans la fonction Bookmarks (notre composant), nous allons créer une fonction qui peut supprimer une entrée dans le tableau des `Bookmarks`, à partir de son index, et qui met à jour le state.

```
const deleteBookmarks = (index) => {
  const tmpBookmarks = [...bookmarks]; // On créé une copie de bookmarks
  tmpBookmarks.splice(index,1); // On supprime 1 entrée à partir de l'index
  setBookmarks(tmpBookmarks); // On met à jour le state avec le nouveau ttableau
}
```

Puis, sur chaque balise `LI`, nous allons ajouter un bouton `Supprimer`, qui appellera la fonction `deleteBookmarks` quand on clique dessus en prenant soin de bien lui passer en paramètre l'index du favori à supprimer.

```
<button onClick={() => { deleteBookmarks(index) }}>🗑️</button>
```

Votre mission

Faites en sorte que l'on puisse ajouter/supprimer des jeux aux favoris, à partir de la recherche. Une étoile doit être présente sur chaque ligne de la recherche. Cette icône est vide si le jeu n'est pas dans les favoris (☆), et pleine s'il n'y est pas (★). Appuyer sur cette étoile ajoute ou supprime le favori, en fonction de son état actuel.