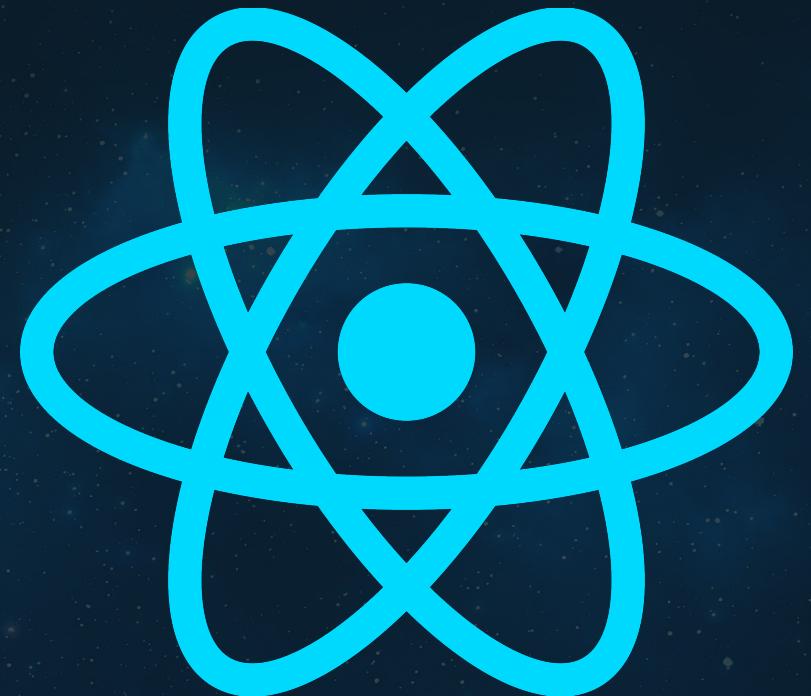


Introduction à React

Créer votre première application - Partie 2



Mathieu Jeanmougin - 2022

Previously on ... The Walking ... React

Créer un composant

```
import * as React from 'react';  8k (gzipped: 3.2k)
```

Imports

```
const DigitalLearningHub: React.FunctionComponent = () => {
  const onClick = () => {
    alert('onClick');
  };
}
```

Logique

```
return (
  <div className="training-list" onClick={onClick}>
    <h1>Training List</h1>
    <ul>
      <li>React - Les bases</li>
      <li>React - Intermédiaire</li>
      <li>React - Expérimenté</li>
    </ul>
  </div>
);
```

Rendu Visuel

```
export {DigitalLearningHub};
```

Composant

Export

Écrire du JSX

```
const DigitalLearningHub: React.FunctionComponent = () => {
    const onClick = () => {};
    const text = "Hello !";
    const boolean = true;
    const id = "id";

    return (
        <div
            onClick={onClick}
            style={{ display: "flex", backgroundColor: "red" }}
            id={id}
        >
            {text}
            {boolean && <div>true</div>}
            {boolean ? <div>true</div> : <span>false</span>}
        </div>
    );
};
```

Instancier un composant

```
import React from "react";  6.9k (gzipped: 2.7k)

import { Card } from "./Card";

const Galerie: React.FunctionComponent = () => {
  return <Card></Card>;
};

export { Galerie };
```

Passer des propriétés

```
const Galerie: React.FunctionComponent = () => {
  return <Card name="The Walking Dead"></Card>;
};
```

Recevoir des propriétés

```
const Card: React.FC<{ name: string }> = ({ name }) => {
  return <span>Voici la carte: {name}</span>;
};
```

Utiliser des propriétés par défaut

```
const CardWithOptionalProps: React.FC<{
  title?: string;
  srcImage?: string;
}> = ({  
  title = "Breaking Bad",  
  srcImage = "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcR_Xz7AhXy8_BzMJfMTSOIrPPhbprot1bg3_A&usqp=CAU",  
}) => {  
  return (  
    <div>  
      <span style={{ color: "#e5e5e5" }}>{title}</span>  
      <img  
        style={{ borderRadius: "0.2vw" }}  
        src={srcImage}  
        alt=""  
        width="300"  
        height="200"  
      />  
    </div>  
  );  
};
```

Créer un état local

```
import React, { useState } from "react";  6.9k (gzipped: 2.7k)

const SearchField: React.FC = () => {
  const [searchFieldValue, setSearchFieldValue] = useState("");
  const onChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    setSearchFieldValue(event.target.value);
  };

  return (
    <input
      value={searchFieldValue}
      onChange={onChange}
      className="search-field"
      type="text"
      placeholder="Search"
    />
  );
}

export { SearchField };
```

Créer une application



Appliquer tout ces principes dans cette application

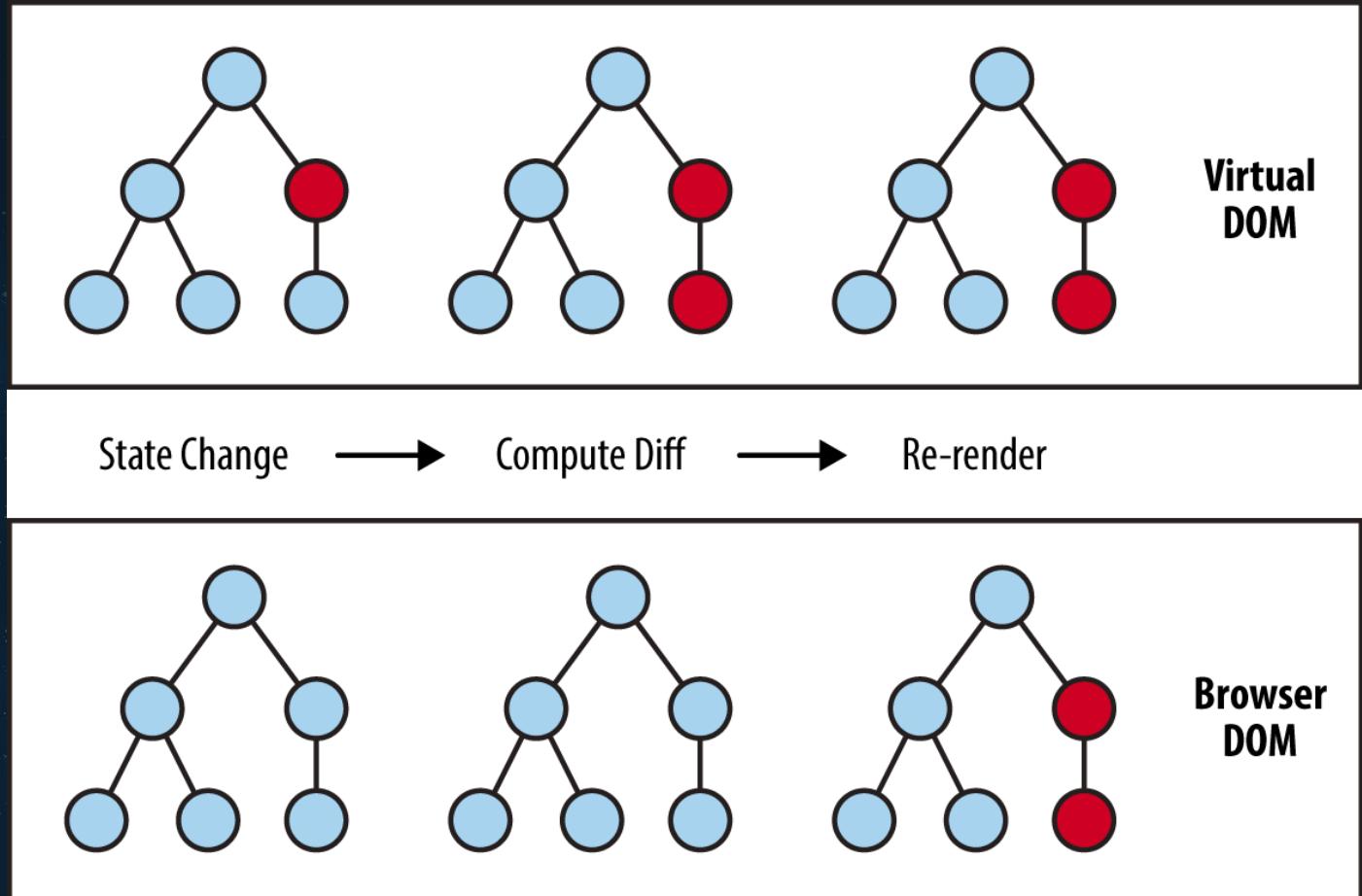


Beaucoup de principes

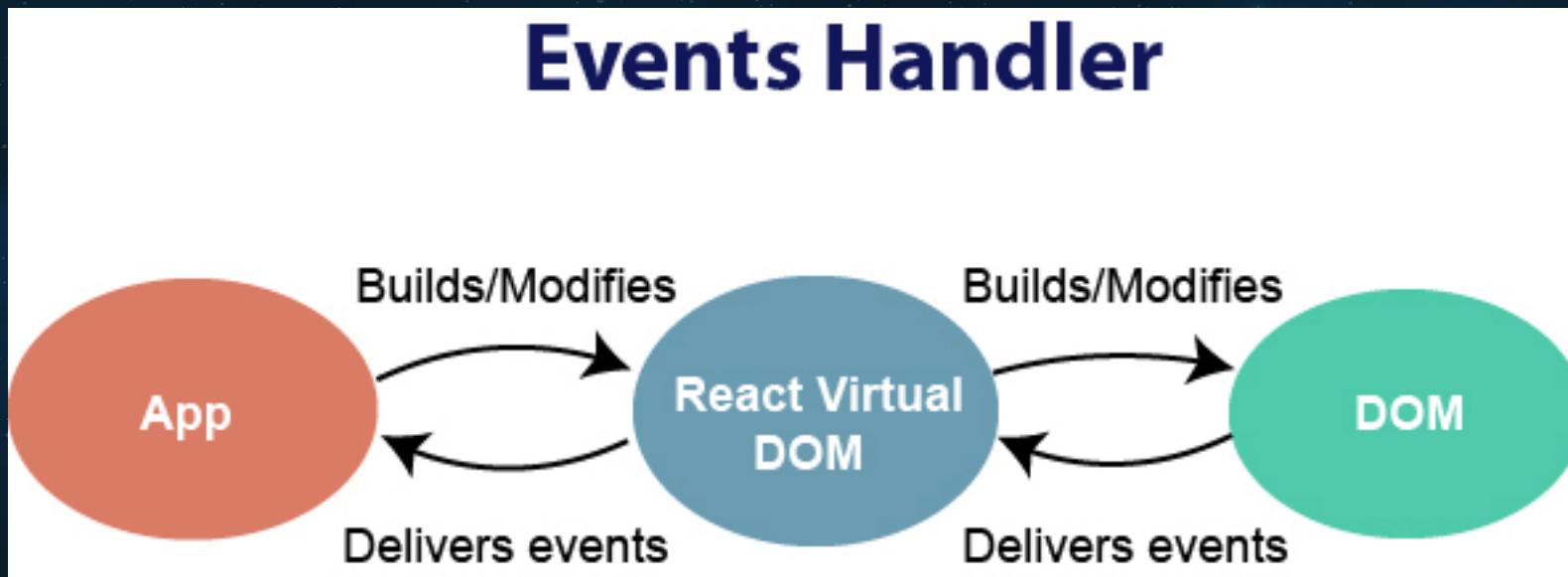


DOM virtuel

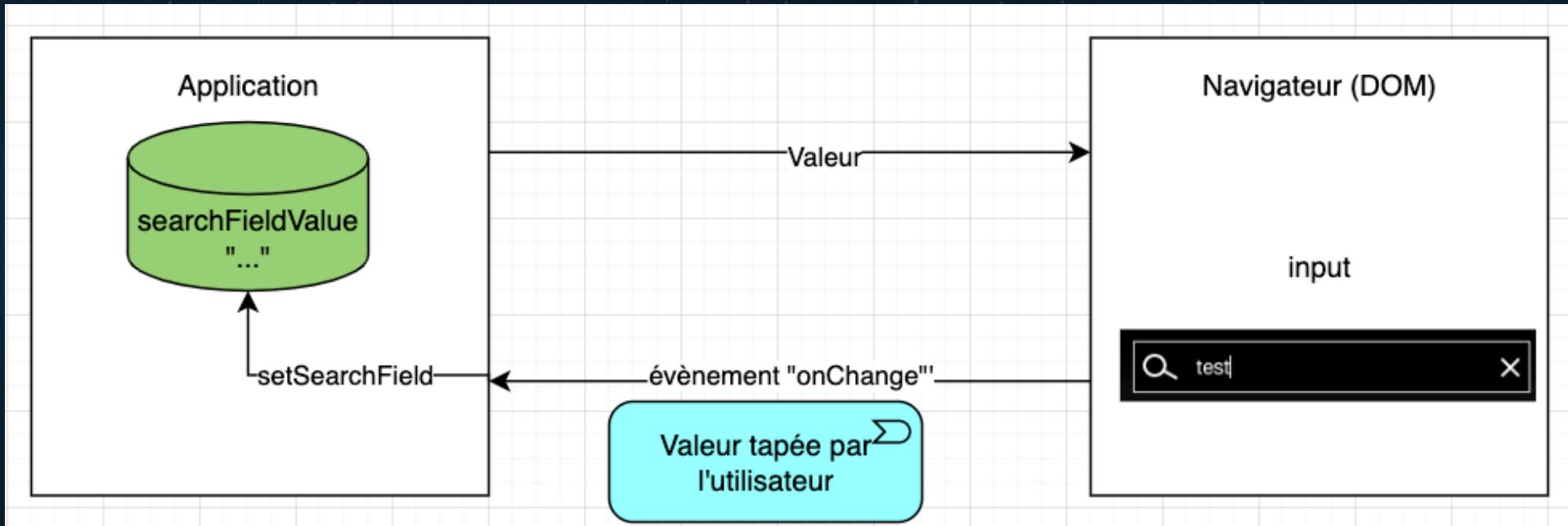
- React ne manipule pas directement le DOM
- Il est agnostique du context (navigateur, mobile, ...)
- Il optimise les éléments à rendre



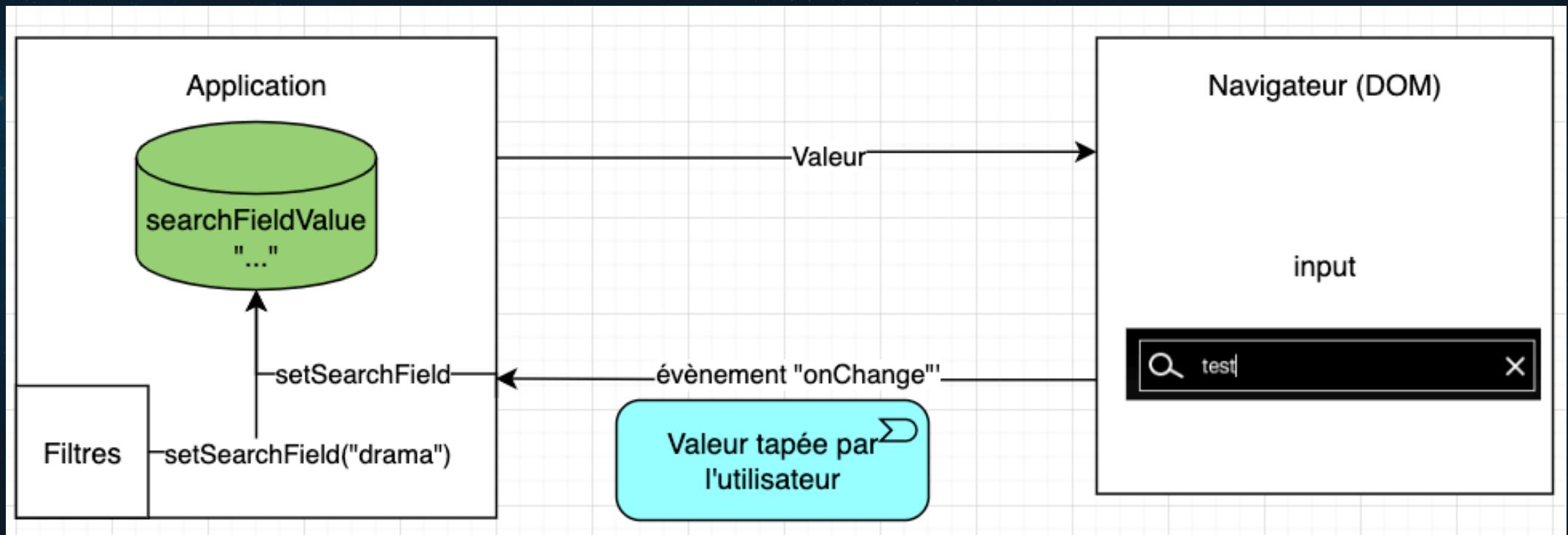
Comment connecter un composant React à un élément interactif du navigateur ?



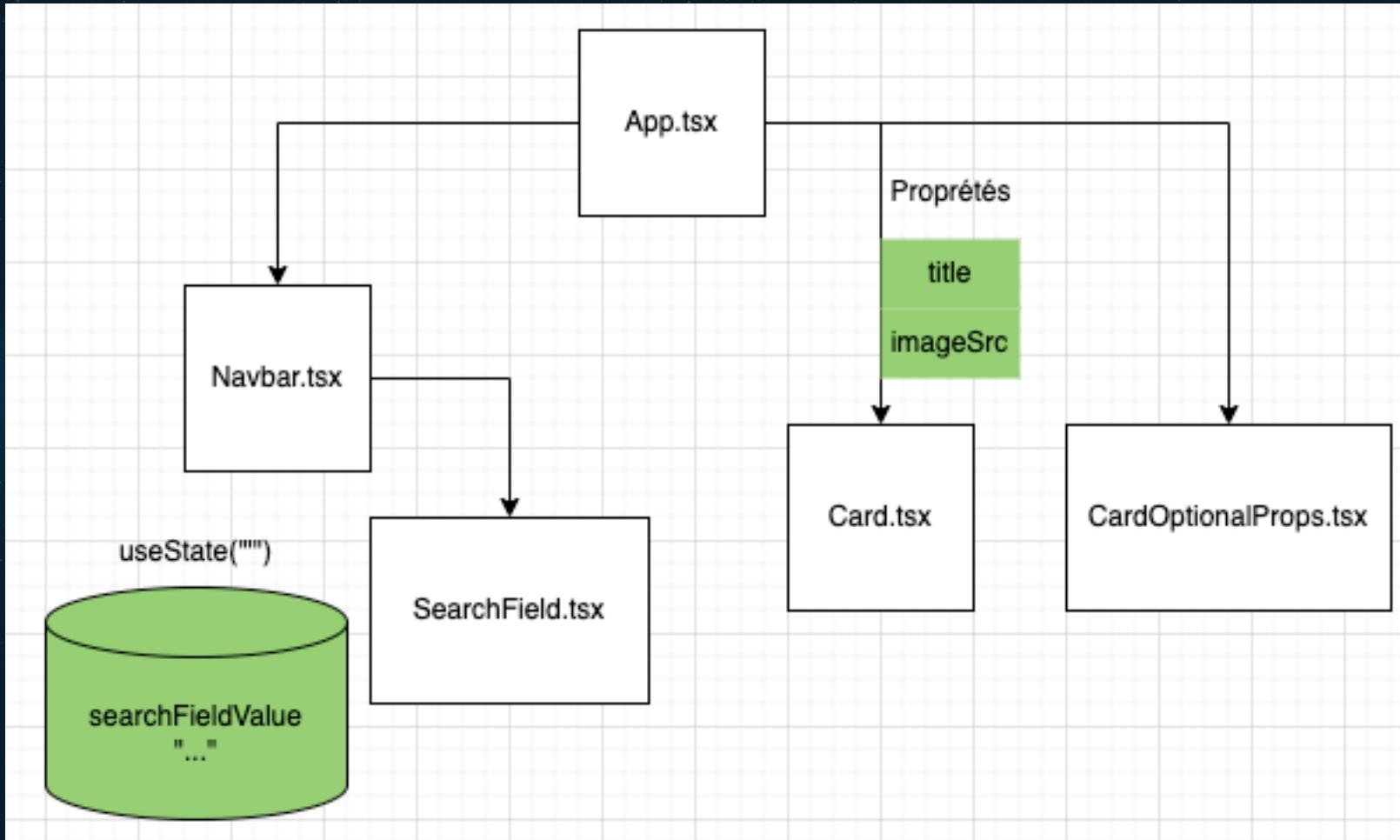
Comment connecter un composant React à un élément interactif du navigateur ?



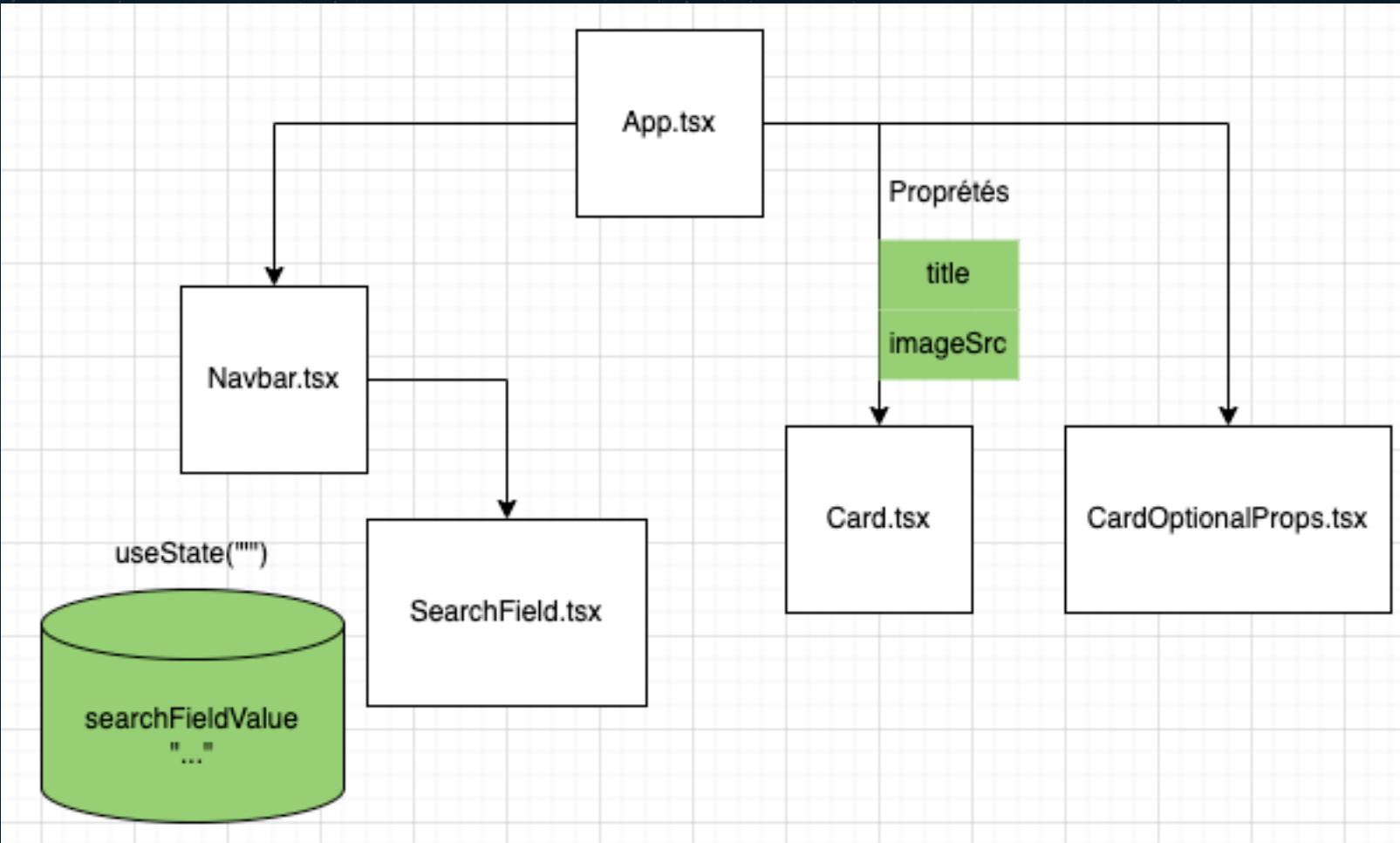
Comment connecter un composant React à un élément interactif du navigateur ?



État de l'application



Quelle est la prochaine étape?



L'état actuel du composant App:

```
const App: React.FunctionComponent = () => {
  return (
    <div>
      <NavBar></NavBar>
      <div className="cards">
        <Card title="The Walking Dead" srcImage="https://...2.jpg" />
        <CardWithOptionalProps />
      </div>
    </div>
  );
};
```

Qu'est ce qui se passe si je déplace le hook d'état dans ce composant ?

```
const App: React.FunctionComponent = () => {
  const [searchFieldValue, setSearchFieldValue] = useState("");
  const onSearchFieldChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    setSearchFieldValue(event.target.value);
  };

  return (
    <div>
      <NavBar></NavBar>
      <div className="cards">
        <Card title="The Walking Dead" srcImage="https://...2.jpg" />
        <CardWithOptionalProps />
      </div>
    </div>
  );
};
```

Je passe l'état de la recherche et sa fonction « onChange » associée au composant « NavBar »

```
const App: React.FunctionComponent = () => {
  const [searchFieldValue, setSearchFieldValue] = useState("");
  const onSearchFieldChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    setSearchFieldValue(event.target.value);
  };

  return (
    <div>
      <NavBar
        searchFieldValue={searchFieldValue}
        onSearchFieldChange={onSearchFieldChange}></NavBar>
      <div className="cards">
        <Card title="The Walking Dead" srcImage="https://...2.jpg" />
        <CardWithOptionalProps />
      </div>
    </div>
  );
};
```

Qu'allons nous devoir changer dans le composant « NavBar » ?

```
const NavBar: React.FunctionComponent = () => {
  return (
    <div className="navbar">
      
      <SearchField></SearchField>
    </div>
  );
};
```

```
const NavBar: React.FunctionComponent<{
  searchFieldValue: string;
  onSearchFieldChange: (event: React.ChangeEvent<HTMLInputElement>) => void;
}> = ({ searchFieldValue, onSearchFieldChange }) => {
```

Qu'allons nous devoir changer dans le JSX du composant « NavBar » ?

```
const NavBar: React.FC<{  
  searchFieldValue: string;  
  onSearchFieldChange: (event: React.ChangeEvent<HTMLInputElement>) => void;  
> = ({ searchFieldValue, onSearchFieldChange }) => {  
  return (  
    <div className="navbar">  
        
      <SearchField  
        ></SearchField>  
    </div>  
  );  
};
```

Qu'allons nous devoir changer dans le JSX du composant « NavBar » ?

```
const NavBar: React.FC<{  
    searchFieldValue: string;  
    onSearchFieldChange: (event: React.ChangeEvent<HTMLInputElement>) => void;  
}> = ({ searchFieldValue, onSearchFieldChange }) => {  
    return (  
        <div className="navbar">  
              
            <SearchField  
                searchFieldValue={searchFieldValue}  
                onSearchFieldChange={onSearchFieldChange}  
            ></SearchField>  
        </div>  
    );  
};
```

Etape-05 Instructions

- Déplacez le hook d'état du composant « SearchField » au composant « App »
- Déplacez la fonction « onSearchFieldChange » du composant « SearchField » au composant « App »
- Modifiez le JSX du composant « App » pour passer les propriétés souhaitées
- Modifiez les propriétés du composant « Navbar » et son JSX
- Modifiez les propriétés du composant « SearchField » et son JSX pour connecter à nouveau notre « input »

Etape-05 Nettoyage de l'environnement

Avant d'aller sur la branche de solution il faut que vous tapiez dans votre console:

- « git reset --hard && git clean -fdx —exclude="node_modules" »

Pour accéder à l'étape numéro 5 vous pouvez taper dans votre terminal:

- « git checkout step-05 »

Etape-05 Instructions détaillées

- Modifiez le composant « App »
 - Déplacez le hook d'état qui vient du composant « SearchField »
 - Déplacez la fonction « onSearchFieldValue » du composant « SearchField »
 - Modifiez le JSX pour passer la valeur « searchFieldValue » et la fonction qui met à jour l'état local au composant « NavBar »
- Modifiez le composant « NavBar »
 - Modifiez les propriétés du composant
 - Modifiez le JSX pour passer les bonnes propriétés au composant « SearchField »
- Modifiez le composant « SearchField »
 - Supprimez le hook d'état « useState » dans le composant « App.tsx »
 - Modifiez les propriétés du composant pour prendre « onSearchFieldChange » et « searchFieldValue »
 - Passez la fonction « onSearchFieldChange » ainsi que la valeur « « searchFieldValue » à votre « input »

Etape-05 Solution

- On va parcourir ensemble, comment j'ai déplacé l'état du composant « SearchField » vers le composant « App »

On va enfin pouvoir filtrer nos séries



Composant - JSX

```
const DigitalLearningHub: React.FunctionComponent = () => {
    const onClick = () => {};
    const text = "Hello !";
    const boolean = true;
    const id = "id";

    return (
        <div
            onClick={onClick}
            style={{ display: "flex", backgroundColor: "red" }}
            id={id}
        >
            {text}
            {boolean && <div>true</div>}
            {boolean ? <div>true</div> : <span>false</span>}
        </div>
    );
};
```

Rappel JS - Les listes (tableaux)

- Crédit d'une liste de texte

```
const fruits = ["Banane", "Pomme"];
```

- Crédit d'une liste d'objet

```
const fruits = [{ nom: "Banane" }, { nom: "Pomme" }];
```

- Accéder à un index de la liste

```
let premier = fruits[0];
```

- Comment parcourir une liste

```
fruits.forEach((item, index) => {
  console.log(item, index);
});
```

Rappel JS - Les listes (tableaux)

- Connaissez vous la méthode « map » ?

« Elle crée un nouveau tableau avec les résultats de l'appel d'une fonction fournie sur chaque élément du tableau appelant. »

« https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map »

Rappel JS - Les listes (tableaux)

```
const liste = [1, 4, 9, 16];

// pass a function to map
const liste2 = liste.map((item) => item * 2);

console.log(liste2);
// expected output: Array [2, 8, 18, 32]
```

JSX - parcours de liste

```
const App: React.FunctionComponent = () => {
  const series = [
    {
      title: "The Walking Dead",
    },
    {
      title: "Breaking Bad",
    },
  ];
  return (
    <div className="cards">
      {series.map((serie) => (
        <div key={serie.title}>{serie.title}</div>
      ))}
    </div>
  );
};
```

- Ne pas oublier d'assigner un attribut « key » unique pour que React optimise le rendu des éléments d'une liste

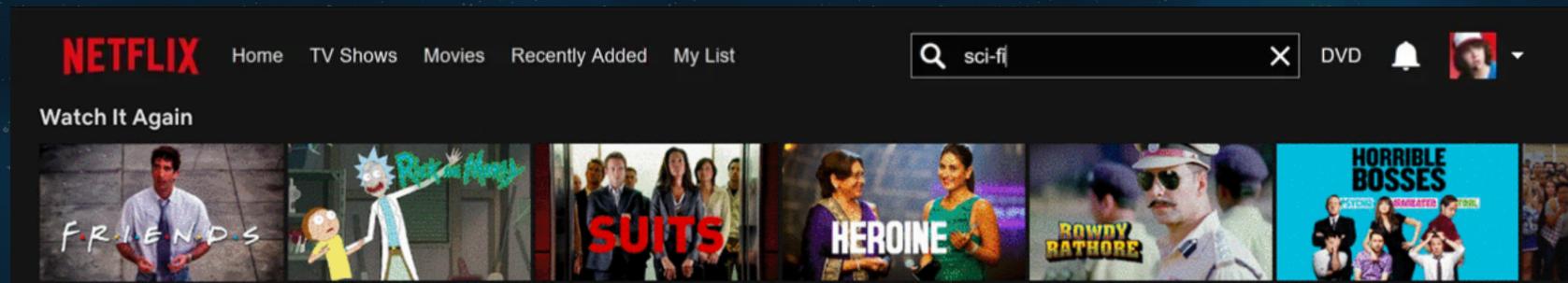
```
return (
  <div className="cards">
    <div key="The Walking Dead">The Walking Dead</div>
    <div key="Breaking Bad">Breaking Bad</div>
  </div>
);
```

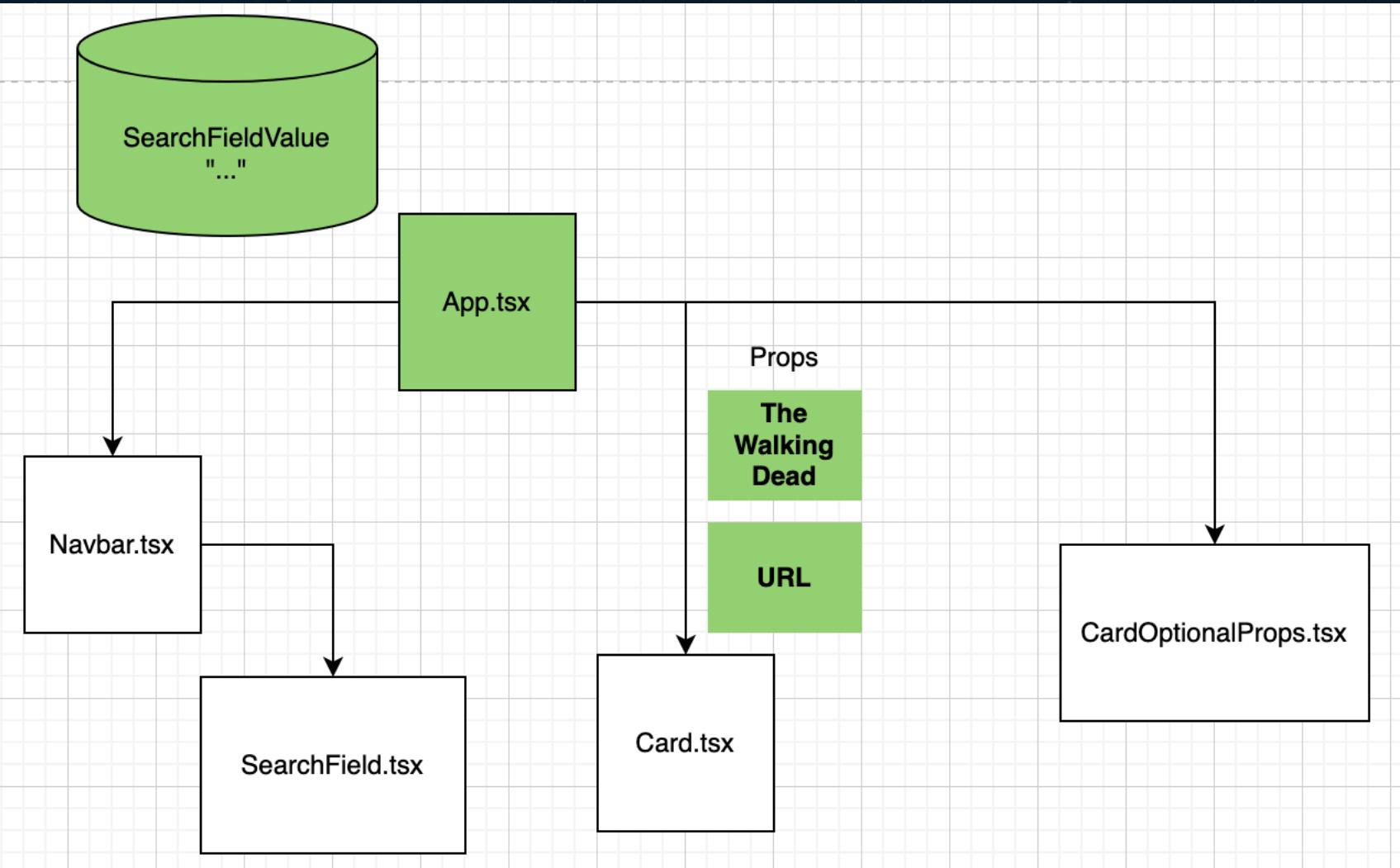
JSX - parcours de liste

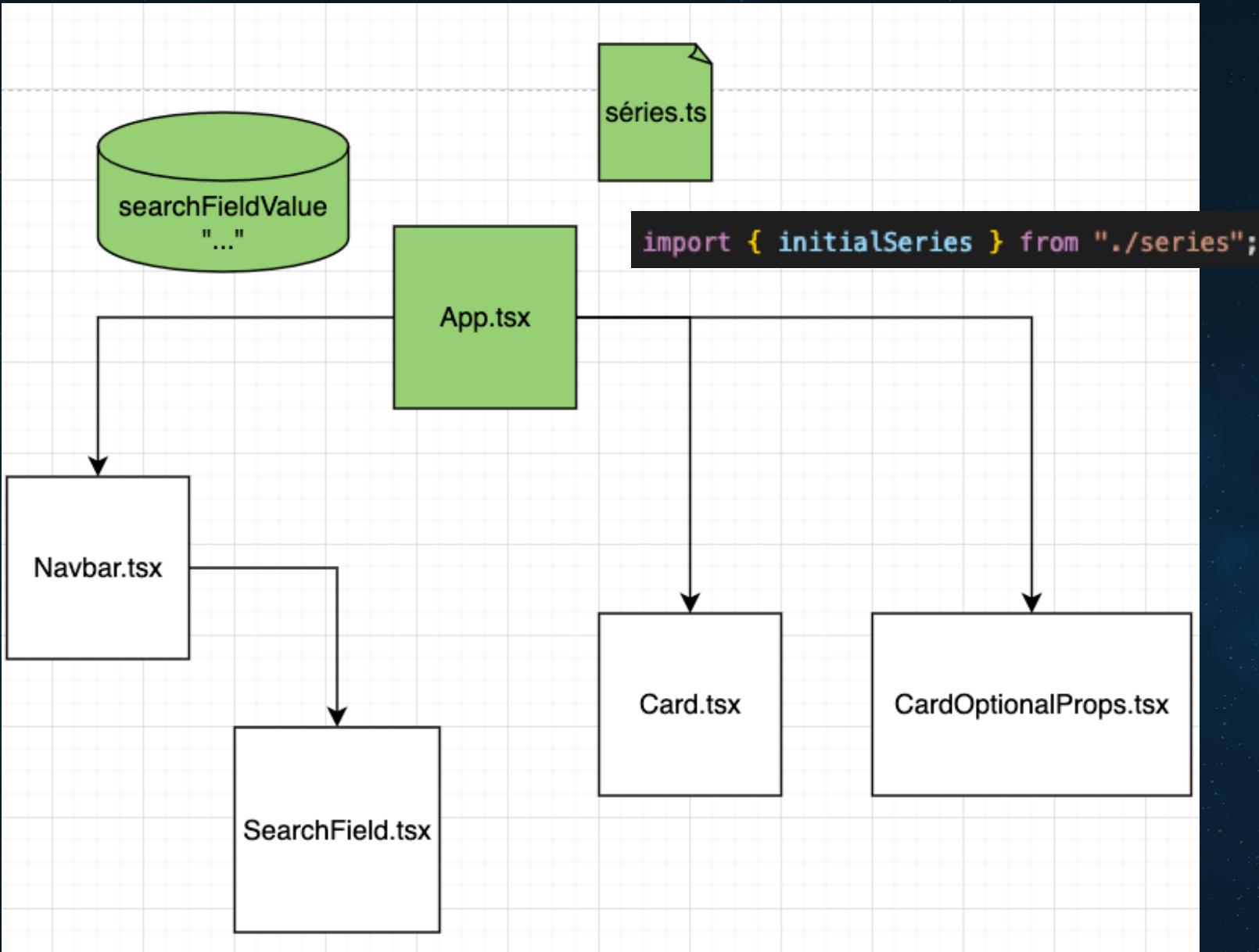
```
const App: React.FunctionComponent = () => {
  const series = [
    {
      title: "The Walking Dead",
    },
    {
      title: "Breaking Bad",
    },
  ];

  return (
    <div className="cards">
      {series.map((serie) => (
        <Card key={serie.title} title="..." srcImage="..."/>
      ))}
    </div>
  );
};
```

Comment filtrer nos séries en fonction de la recherche de l'utilisateur ?







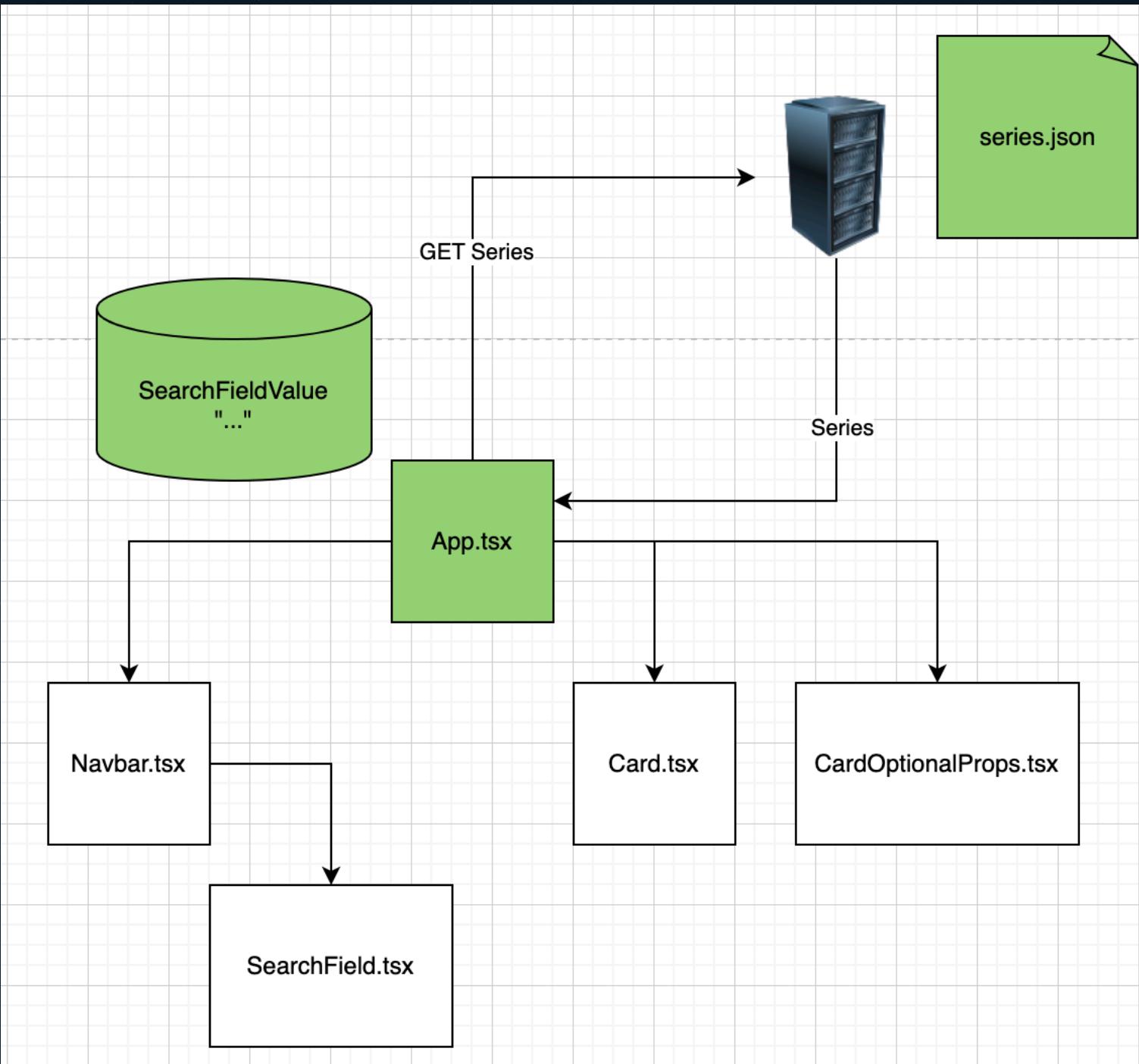
JSX - parcours de liste

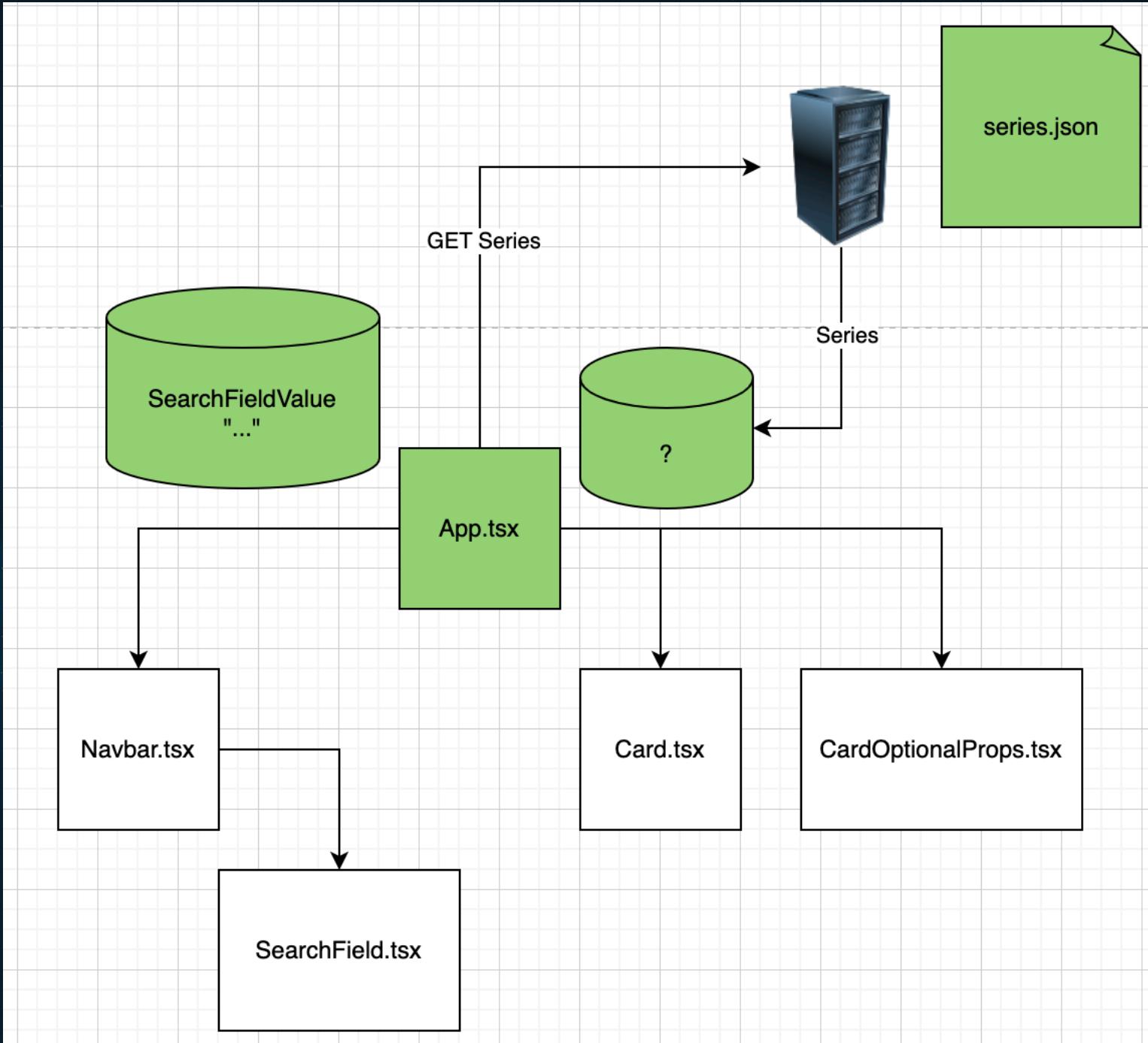
```
const App: React.FunctionComponent = () => {
  const series = [
    {
      title: "The Walking Dead",
    },
    {
      title: "Breaking Bad",
    },
  ];

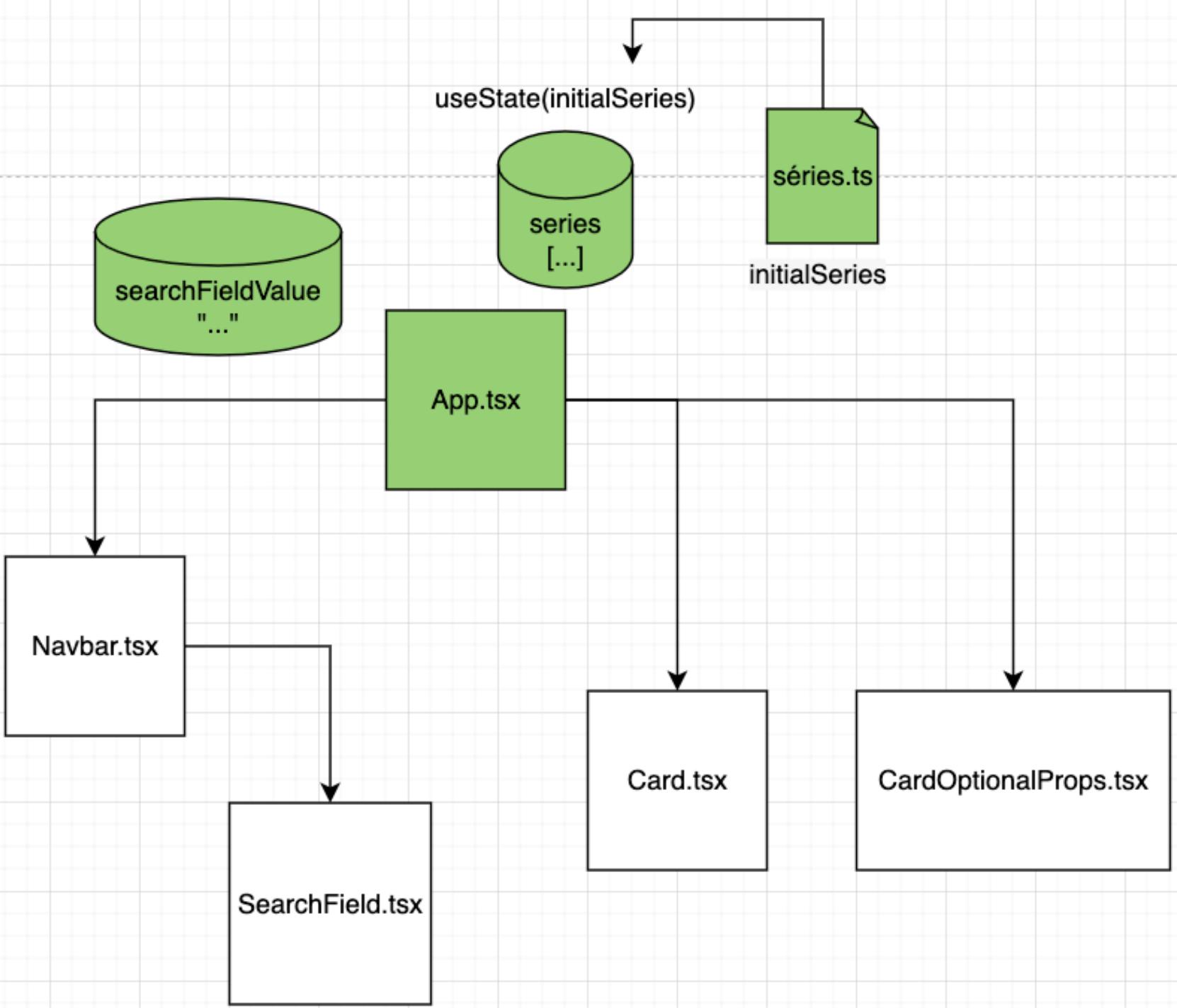
  return (
    <div className="cards">
      {series.map((serie) => (
        <div key={serie.title}>{serie.title}</div>
      ))}
    </div>
  );
};

import { initialSeries } from "./series";

{initialSeries.map((serie) => (
  <Card title="..." srcImage="..."/>
))}
```







Etape-06 Instructions

- Je vais vous montrer ce qui a changer entre l'étape 5 et 6

Etape-06 Instructions

- Modifier le composant « App »
 - Créer un hook d'état nommé « series » avec comme valeur initiale la constante « initialSeries » importé du fichier « series.ts »
 - Créer une constante nommé « filteredSeries » qui va filtrer les séries en fonction de la valeur de « searchFieldValue »
 - Modifier le JSX pour parcourir « filteredSeries » (grâce au map)
 - Pour chaque élément parcourus, vous devrez retourner un composant « Card », préciser sa « key » et ses propriétés « imageSrc » & « title »
 - [Bonus] Créer un composant « NoResult » à n'afficher que quand la mot recherché ne se trouve dans aucun titre de séries (pour éviter d'afficher une page vide)

Etape-06 Nettoyage de l'environnement

Avant d'aller sur la branche de solution il faut que vous tapiez dans votre console:

- « git reset --hard && git clean -fdx —exclude="node_modules" »

Pour accéder à l'étape numéro 6 vous pouvez taper dans votre terminal:

- « git checkout step-06 »

Etape-06 Solution



- Votre application n'est plus statique !
- Et si, plutôt que d'utiliser une liste qui se trouve dans l'application elle même, elle se trouvait sur un serveur ?

Les hooks

- Ils en existent beaucoup:
 - useState
 - useEffect
 - useContext
 - useCallback
 - useMemo
 - useReducer..

useEffect

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Vous avez cliqué ${count} fois`;
  });

  return (
    <div>
      <p>Vous avez cliqué {count} fois</p>
      <button onClick={() => setCount(count + 1)}>
        Cliquez ici
      </button>
    </div>
  );
}
```

- Il est utilisé pour indiquer à React que notre composant doit exécuter quelque chose après chaque affichage
- React enregistre la fonction passée en argument (« effet »), et l'appellera plus tard, après avoir mis à jour le DOM.

useEffect

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Vous avez cliqué ${count} fois`;
  });

  return (
    <div>
      <p>Vous avez cliqué {count} fois</p>
      <button onClick={() => setCount(count + 1)}>
        Cliquez ici
      </button>
    </div>
  );
}
```

- Cet effet met à jour le titre du document, mais il pourrait aussi bien charger des données distantes, ou appeler n'importe quelle autre API.
- Cet effet est appelé à chaque affichage (à chaque fois que le DOM est modifié)

useEffect

```
import React, { useState, useEffect } from 'react';

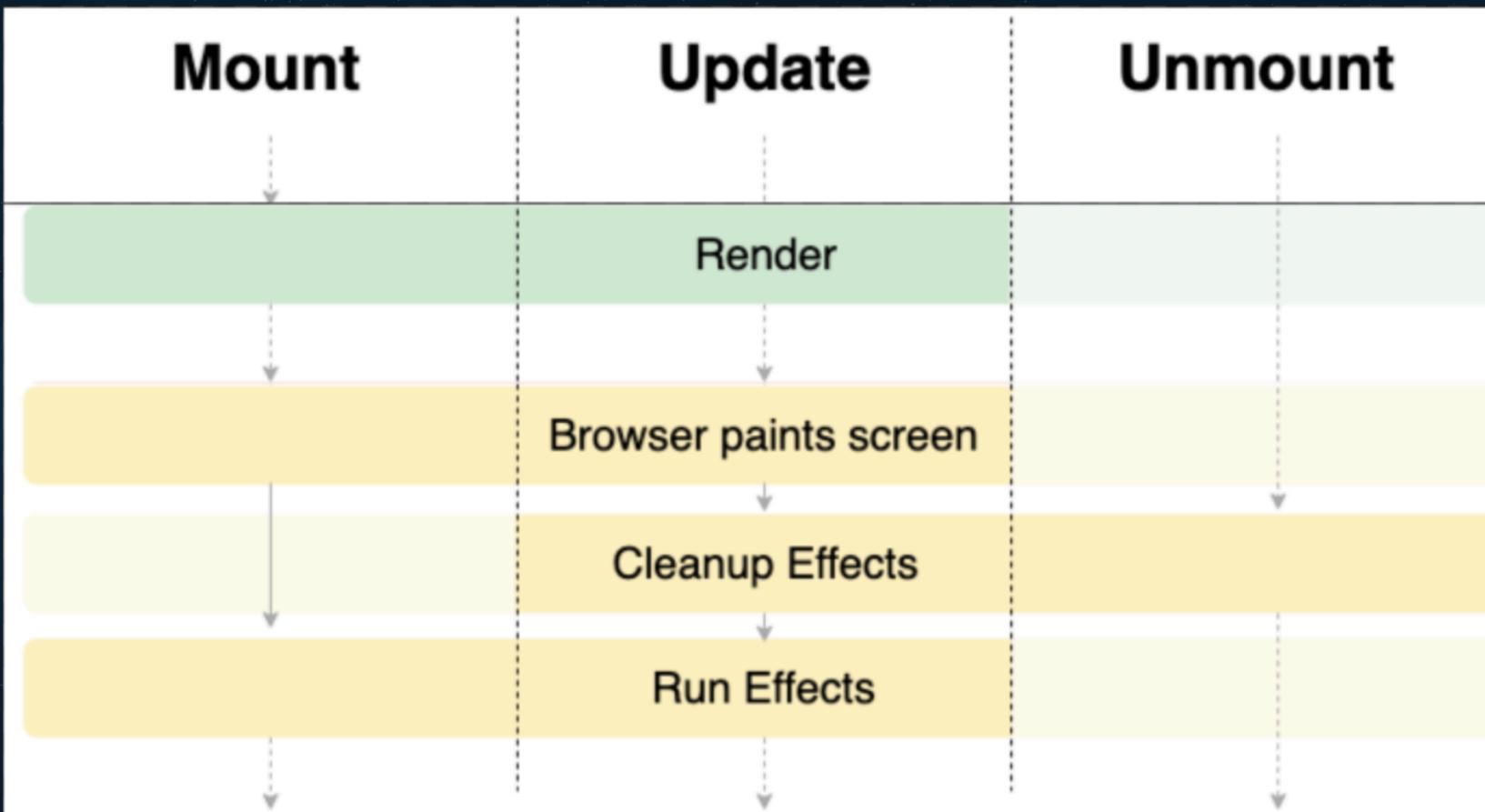
function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Vous avez cliqué ${count} fois`;
  });

  return (
    <div>
      <p>Vous avez cliqué {count} fois</p>
      <button onClick={() => setCount(count + 1)}>
        Cliquez ici
      </button>
    </div>
  );
}
```

- Cet effet met à jour le titre du document, mais il pourrait aussi bien charger des données distantes, ou appeler n'importe quelle autre API.
- Cet effet est appelé à chaque affichage (à chaque fois que le DOM est modifié)

React Hooks Flow



« <https://github.com/donavon/hook-flow> »

Rappel JS - L'API fetch

L'API fetch présente dans nos navigateurs rend accessible une fonction « fetch » qui permet de faire une requête HTTP.

Nous allons l'utiliser pour récupérer la liste des séries que nous allons afficher.

```
fetch('http://example.com/movies.json')
  .then((response) => response.json())
  .then((data) => console.log(data));
```

La fonction « fetch » retourne une promesse étant donné qu'on ne sait pas quand on récupérera le contenu de notre requête

Les méthodes « then » qu'on appelle sur cette promesse seront exécuté au moment on ou aura reçu la réponse de la requête HTTP.

useEffect

- Dans notre cas l'url de notre serveur sera « http://localhost:3001/series.json »

```
useEffect(() => {
  fetch("http://localhost:3001/series.json")
    .then((response) => response.json())
    .then((data) => {
      // Que fait-on ?
    });
});
```

useEffect

- On va modifier la valeur de notre état avec la donnée retournée par notre serveur

```
useEffect(() => {
  fetch("http://localhost:3001/series.json")
    .then((response) => response.json())
    .then((data) => {
      setSeries(data);
    });
});
```

- Il ne faudra pas oublier de changer la valeur initiale de l'état « series »

useEffect

- Quand est-ce qu'est exécuté mon effect ? Quel problème risque d'arriver ?

```
useEffect(() => {
  fetch("http://localhost:3001/series.json")
    .then((response) => response.json())
    .then((data) => {
      setSeries(data);
    });
});
```

useEffect

- On peut passer un deuxième argument à notre hook « useEffect »
- Une liste de dépendances

```
useEffect(() => {
  fetch("http://localhost:3001/series.json")
    .then((response) => response.json())
    .then((data) => {
      setSeries(data);
    });
}, []);
```

- Si cette liste est vide, l'effet ne sera joué qu'une fois (lors du premier rendu du composant)

useEffect

- On peut également passer des propriétés ou des états dans cette liste de dépendance

```
useEffect(() => {
  fetch("http://localhost:3001/series.json")
    .then((response) => response.json())
    .then((data) => {
      setSeries(data);
    });
}, [series, searchFieldValue]);
```

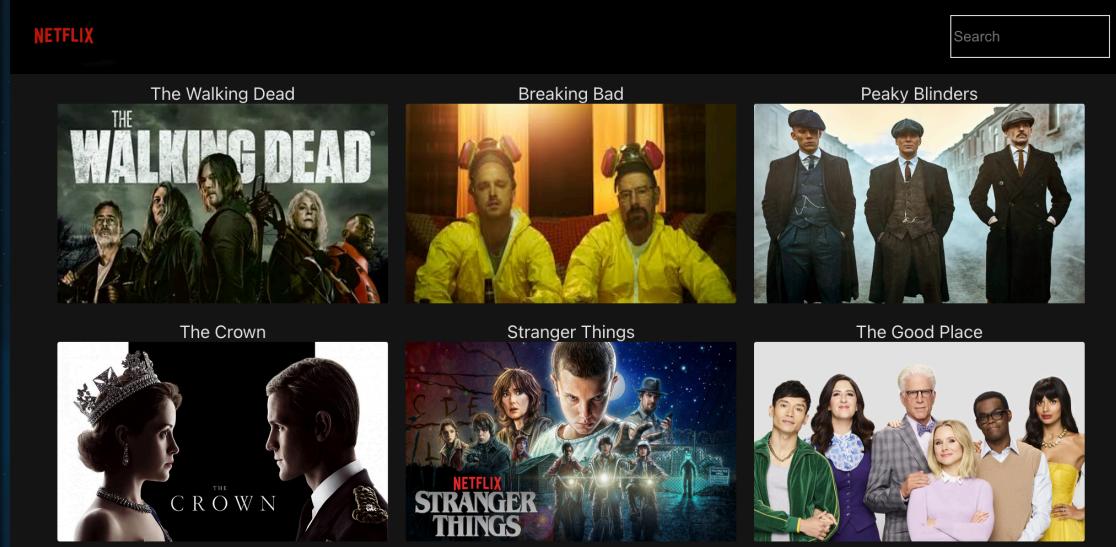
- A votre avis, quel est son intérêt ?

useEffect

- Imaginons que notre composant « Card » ai besoin de récupérer des informations sur un serveur

```
useEffect(() => {
  fetch("http://localhost:3001/series/" + title)
    .then((response) => response.json())
    .then((data) => {
      setCard(data);
    });
}, [title]);
```

- Ici on lui passera le « title » dans les dépendances au cas ou la propriété « title » change pour qu'on puisse changer l'état de la « Card » pour qu'il corresponde au nouveau titre.

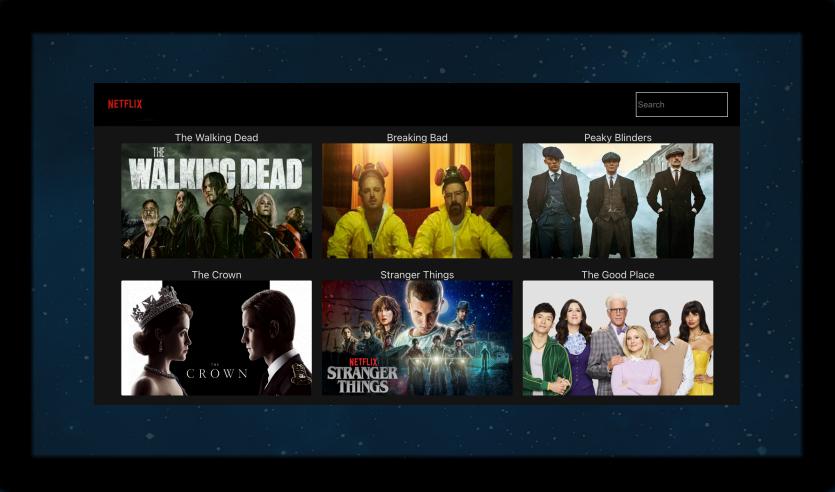


Séries

Notre application Netflix

Netflix

Application



Serveur



Séries

Etape-07 Modifications

- J'ai ajouté un script « http-server » que vous pouvez lancer avec « npm run http-server » pour servir la liste des séries.
- Vous pouvez accéder au fichier qui contient les séries en utilisant l'url "<http://localhost:3001/series.json>" une fois le serveur lancé.

Etape-07 Nettoyage de l'environnement

Avant d'aller sur la branche de solution il faut que vous tapiez dans votre console:

- \$ git reset --hard && git clean -fdx --exclude="node_modules"

Pour accéder à l'étape numéro 7 vous pouvez taper dans votre terminal:

- git checkout step-07

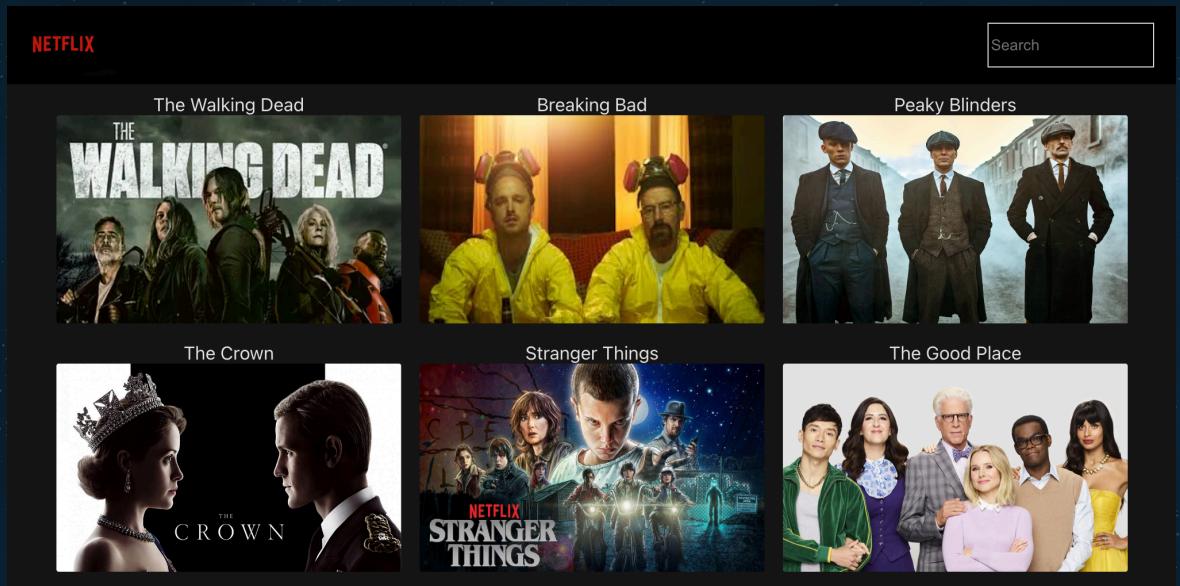
Etape-07 Instructions

- Pour faire fonctionner votre serveur local
 - Ouvrir un deuxième terminal et lancez la commande « npm i »
 - Dans ce même terminal, lancez la commande « npm run http-server »
 - Pour vérifier si votre serveur tourne bien, vous pouvez accéder au fichier qui contient vos séries depuis votre navigateur en allant sur l'url « <http://localhost:3001/series.json> »
- N'oubliez pas de lancer « npm run start » sur un autre terminal pour faire tourner l'application
- Modifiez le composant « App.tsx »
 - Modifiez la valeur par défaut de votre « useState » qui contient les séries
 - Importez et ajoutez le hook « useEffect » qui va récupérer les séries sur votre serveur local
 - Modifiez votre état « series » une fois la donnée récupérée via votre requête HTTP

Etape-07 Solution

- La prochaine étape ?

Félicitations !



Ne nettoyez pas votre branche !

- Si vous voulez continuer à travailler sur votre projet
- Si vous voulez le mettre en avant
- Ajouter des composants
- Ajouter du style
- Complexifier votre application, n'hésitez pas !

Vous pouvez retrouver le cours sur Github

- <https://github.com/sopretty/react-netflix-introduction>
- Avec le powerpoint et toutes les solutions

Ressources du cours

- Tutoriel de React « <https://reactjs.org/tutorial/tutorial.html> » qui est très bien fait (FR + EN)
- Tutoriel de W3School sur react « https://www.w3schools.com/REACT/react_getstarted.asp »
- Articles React intéressant « <https://react-facile.fr/> »
- Comprendre le cycle de vie des hooks « <https://github.com/donavon/hook-flow> »
- Documentation de l'API « fetch » « https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch »
- Documentation des listes en JS « https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array »



Please give us feedback about the course

<http://www.edulink.lu/rbkt>



Si vous voulez garder contact

- LinkedIn « <https://www.linkedin.com/in/mathieu-jeanmougin-11478412b/> »
- Github « <https://github.com/sopretty/> »
- Site web de Tadaweb « <https://www.tadaweb.com/> »

Merci de votre implication



tada^{web}

tadaweb.com

© 2022 Tadaweb S.A. All rights reserved. All other trademarks are the property of their respective owners.
Redistribution or public display not permitted without written permission from Tadaweb. • MKT070322x2