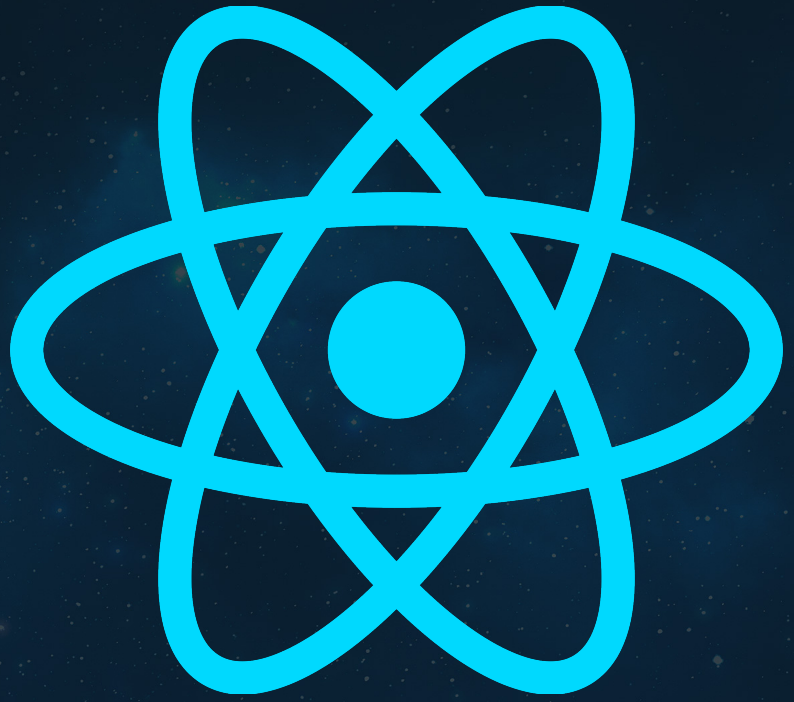
tadaweb

Introduction à React

Créer votre première application



Mathieu Jeanmougin - 2023

© 2023 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.

Présentation

Mathieu Jeanmougin

But du cours

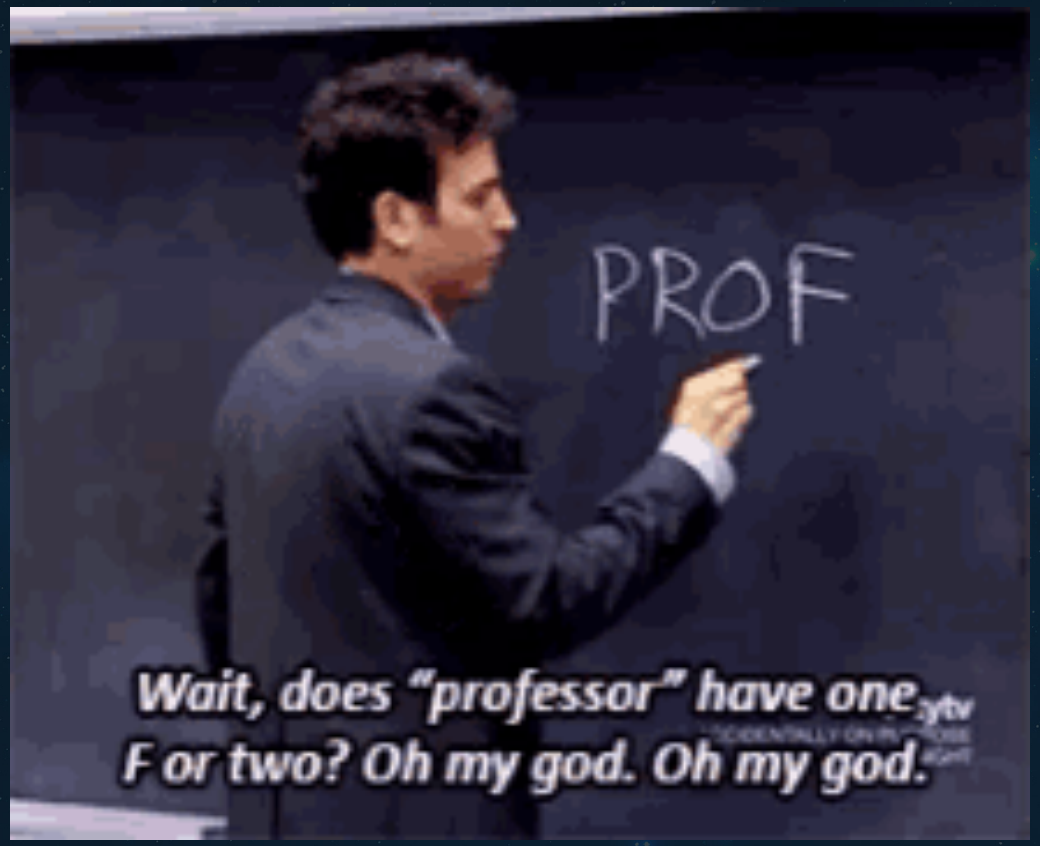
- Créer votre première application React
- Comprendre les bases du fonctionnement de React
- Comprendre les bonnes pratiques à suivre
- Rendre votre application interactives
- Récupérer de l’information via une API pour enrichir votre application

Quel genre d’application web ?

tada web

Comment ça va se passer ?

- Partie formation



tada web

Comment ça va se passer ?

- Partie travaux pratiques

tada web

Comment ça va se passer ?

- Le cours est constitué de 7 étapes
- Pour passer d’une étape à une autre, vous allez utilisez « git »
- Pour chaque partie pratique, vous aurez:
  - Une branche d’exercice: « step-XX »
  - Une branche de solution: « step-XX-solution »
- Vous pouvez utiliser vos ordinateurs personnels si vous le désirez
- Vous pouvez travailler par paire

React - c’est quoi ?

- Créé en 2011 par Facebook mais rendu accessible en 2013
- Fonctionne uniquement avec du Javascript
- Une librairie Javascript pour construire des interfaces utilisateurs à partir de petits morceaux de code isolés appelés « composants »

Pourquoi React ?

- Libre et accessible (~200k stars sur GitHub)
- Développé par Facebook (~1k5 contributeurs)
- Flexible et performant
- Utilisé par Netflix, Uber, Airbnb, Atlassian, The New York Times, …

Rappels - HTML

tada web

HTML

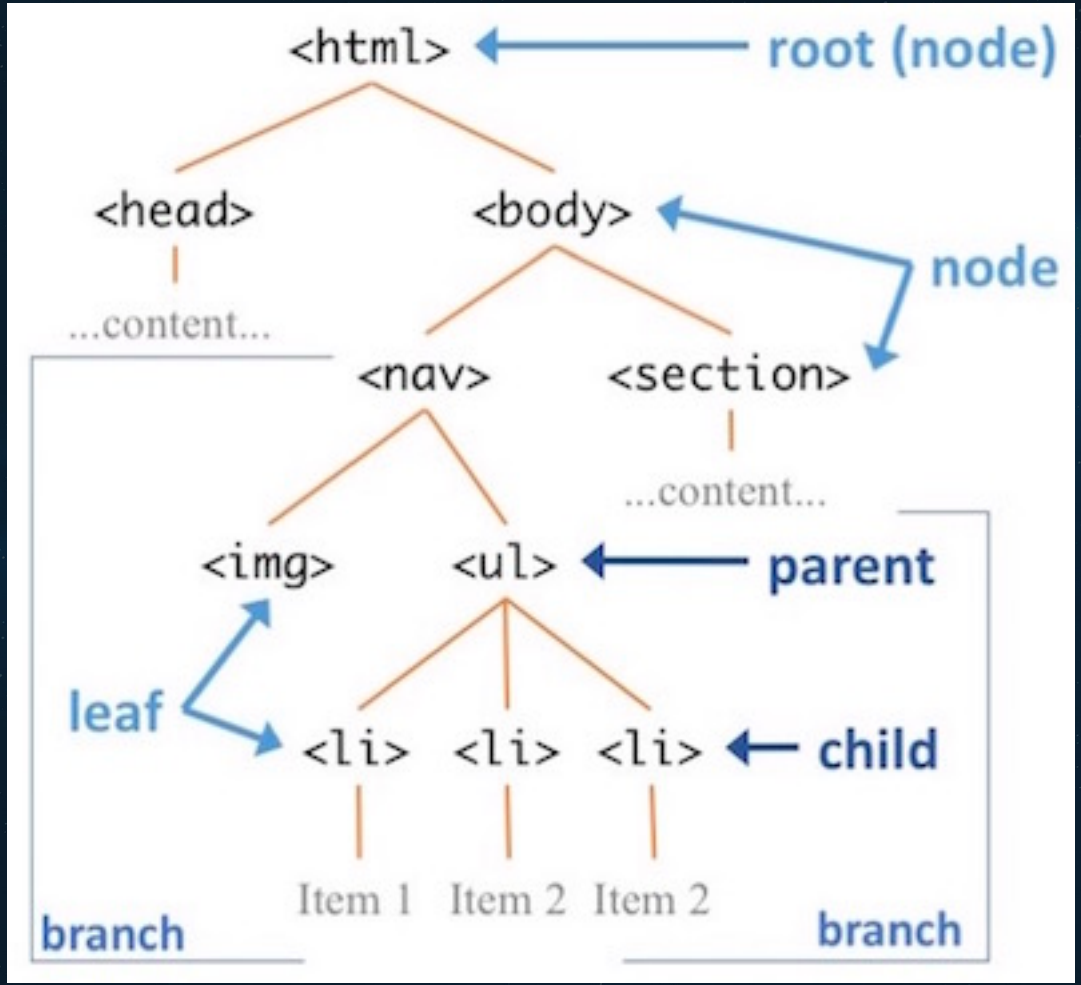
index.html

Raw

```html
<!doctype html>
<html>
  <head>
    <title>This is the title of the webpage!</title>
  </head>
  <body>
    <p>This is an example paragraph. Anything in the <strong>body</strong> tag will appear on the page, just like this <strong>
  </body>
</html>
```

tada web

DOM (Document Object Model)



tada web

DOM (Document Object Model)

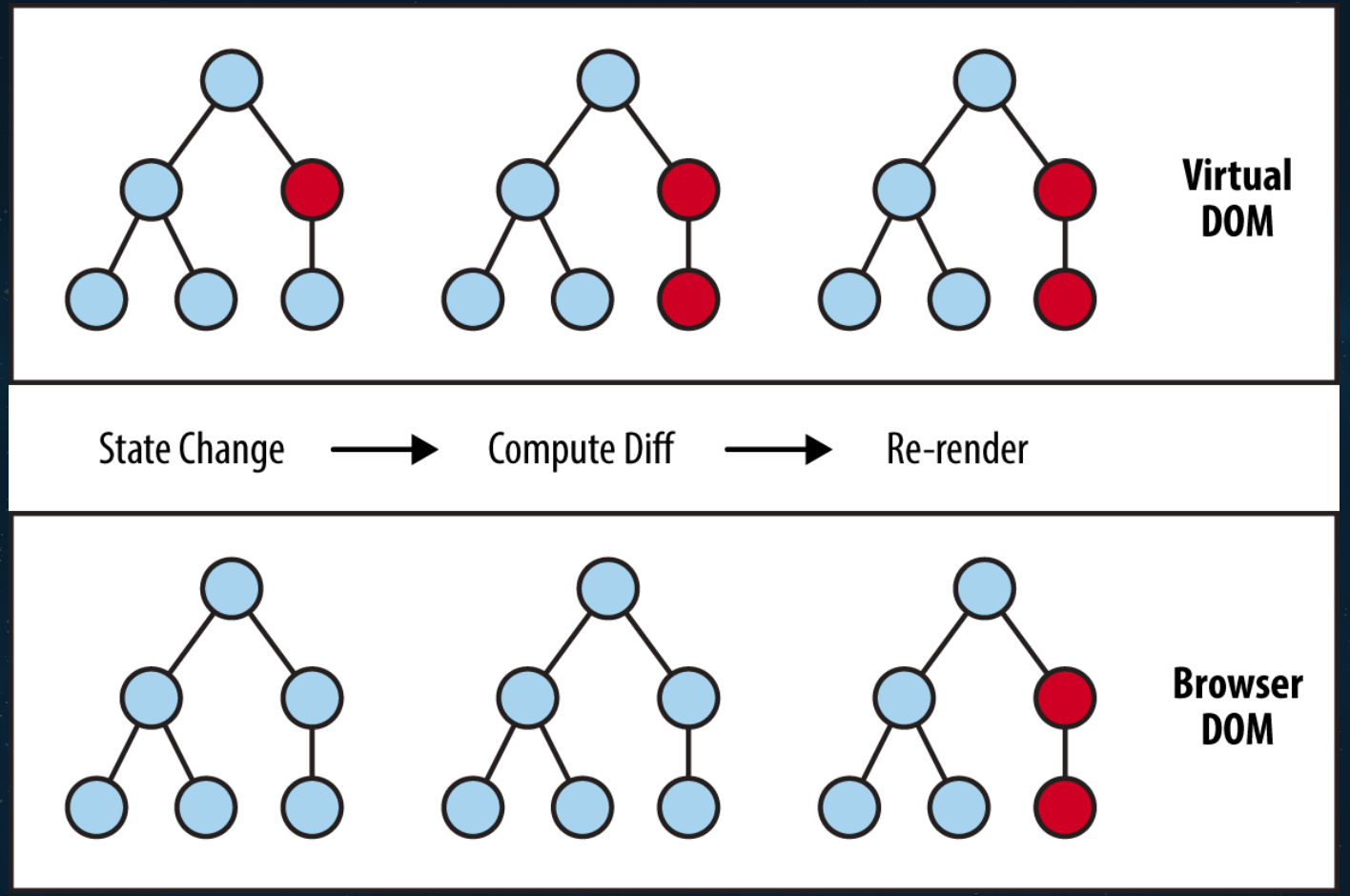
- C’est une représentation d’un fichier HTML
- C’est un arbre ou chaque élément peut avoir 0 à N enfants
- Il est généré par votre navigateur
- Vous pouvez interagir avec lui grâce à l’outil « Developers Tools » de votre navigateur

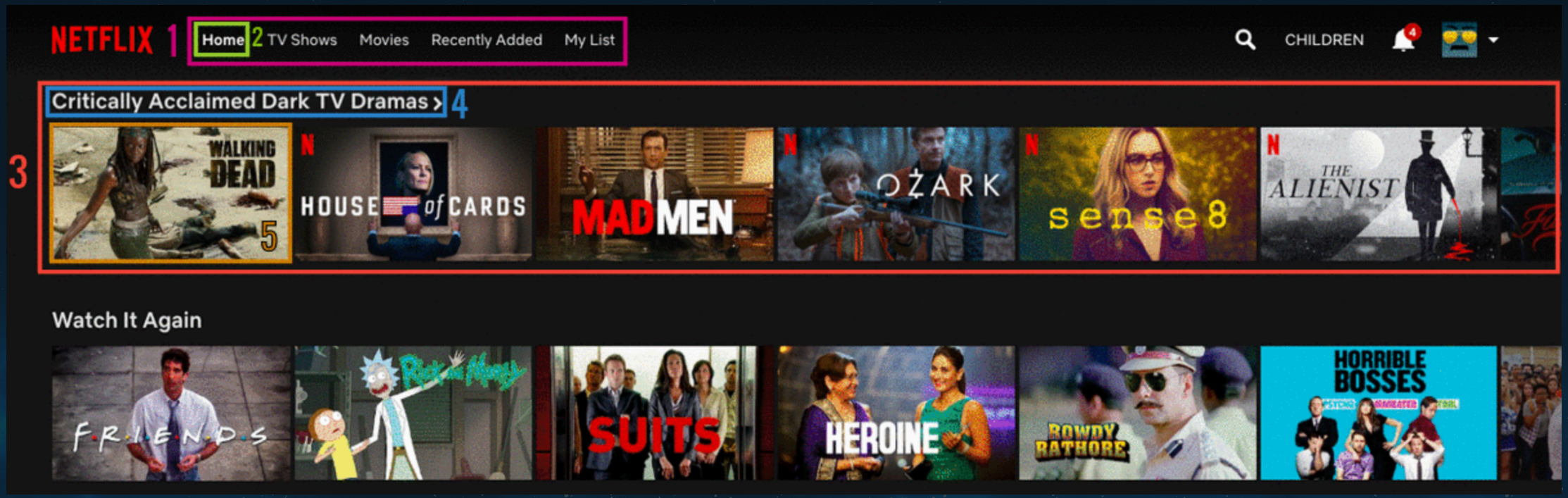
La philosophie

- Créer des interfaces utilisateurs avec un outil rapide et modulaire
- Construire une application à partir de composants

DOM virtuel

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





- 1 - Menu de navigation
- 2 - Onglet de navigation
- 3 - Galerie
- 4 - Entête de galerie
- 5 - Carte

Rappels - JS

tada web

Fonctions Javascript

```
// Function classique en Javascript
function MaFonction() {

}

// Fonction fléchée en Javascript (ES6)
const MaDeuxiemeFonction = () => {

}
```

tada web

Exports Javascript

Syntax

```
// Exporting declarations
export let name1, name2/*, … */; // also var
export const name1 = 1, name2 = 2/*, … */; // also var, let
export function functionName() { /* … */ }
export class ClassName { /* … */ }
export function* generatorFunctionName() { /* … */ }
export const { name1, name2: bar } = o;
export const [ name1, name2 ] = array;

// Export list
export { name1, /* …, */ nameN };
export { variable1 as name1, variable2 as name2, /* …, */ nameN };
export { variable1 as "string name" };
export { name1 as default /*, … */ };

// Default exports
export default expression;
export default function functionName() { /* … */ }
export default class ClassName { /* … */ }
export default function* generatorFunctionName() { /* … */ }
export default function () { /* … */ }
export default class { /* … */ }
export default function* () { /* … */ }
```

tada web

Imports Javascript

Syntax

```
import defaultExport from "module-name";
import * as name from "module-name";
import { export1 } from "module-name";
import { export1 as alias1 } from "module-name";
import { default as alias } from "module-name";
import { export1, export2 } from "module-name";
import { export1, export2 as alias2, /* … */ } from "module-name";
import { "string name" as alias } from "module-name";
import defaultExport, { export1, /* … */ } from "module-name";
import defaultExport, * as name from "module-name";
import "module-name";
```

tada web

Composant

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

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

  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>
  );
};

export {DigitalLearningHub};
```

Imports

Logique

Composant

Rendu Visuel

Export

tada web

Composant - JSX

- C’est une extension syntaxique de Javascript créée par React
- Très proche de la syntaxe HTML
- Son utilisation n’est pas obligatoire mais fortement conseillé
- Vous pouvez mettre n’importe quelle expression JavaScript entre accolades dans du JSX

Composant - JSX

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

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

  return (
    <div onClick={onClick} style={{ display: "flex", backgroundColor: "red" }}>
      {text}
    </div>
  );
};

export { DigitalLearningHub };
```

tada web

Rappels - JS

tada web

Ternaire JS

```
// Condition if ... else classique
let resultat;
const x = 3;
const y = 4;
if(x > y) {
  resultat = 'Supérieur';
} else {
  resultat = 'Inférieur';
}

// Utilisation d'une ternaire
const resultat2 = x > y ? 'Supérieur' : 'Inférieur';
```

tada web

Composant - JSX

```
const DigitalLearningHub: React.FunctionComponent = () => {

  const boolean = true;

  return <div>{boolean ? <div>Supérieur</div>: <div>Inférieur</div>}</div>
}
```

tada web

Composant - JSX

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

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

  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>
  );
};

export {DigitalLearningHub};
```

Imports

Logique

Composant

Rendu Visuel

Export

tada web

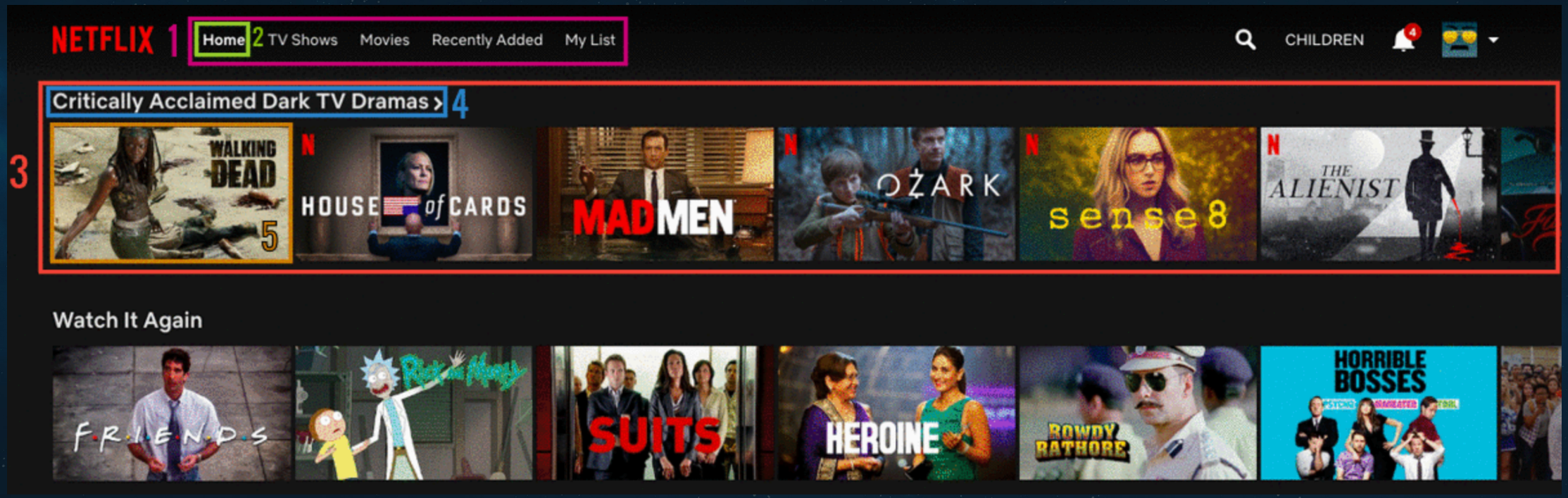
JSX - build

Au build, le JSX va être transformé en:

```
return React.createElement(
  'div',
  {className: 'training-list' /* ... onClick ... */},
  React.createElement('h1' /* ... h1 children ... */),
  React.createElement('ul' /* ... ul children ... */)
);
```

tada web

Composant - Comment instancier un composant ?



tada web

Composant - Comment instancier un composant ?

```
import * as React from "react";

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

export {Card}
```

tada web

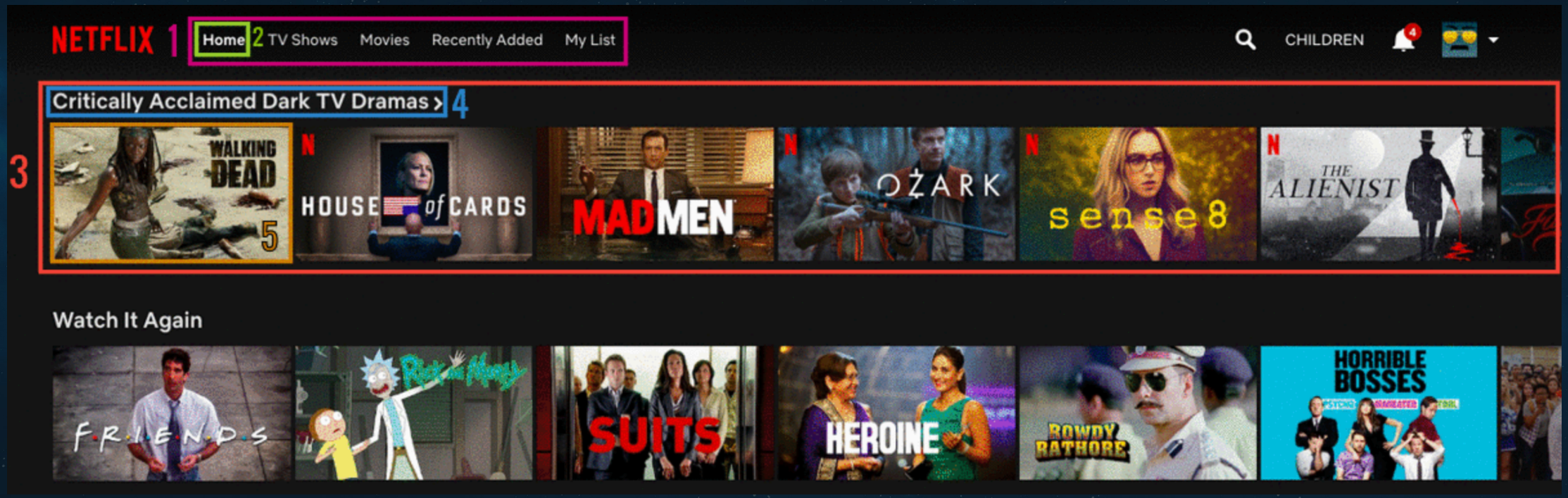
Composant - Comment instancier un composant ?

Pour instancier un composant, il suffit de l’appeler dans du JSX

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

import { Card } from "./Card";

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

export { Galerie };
```

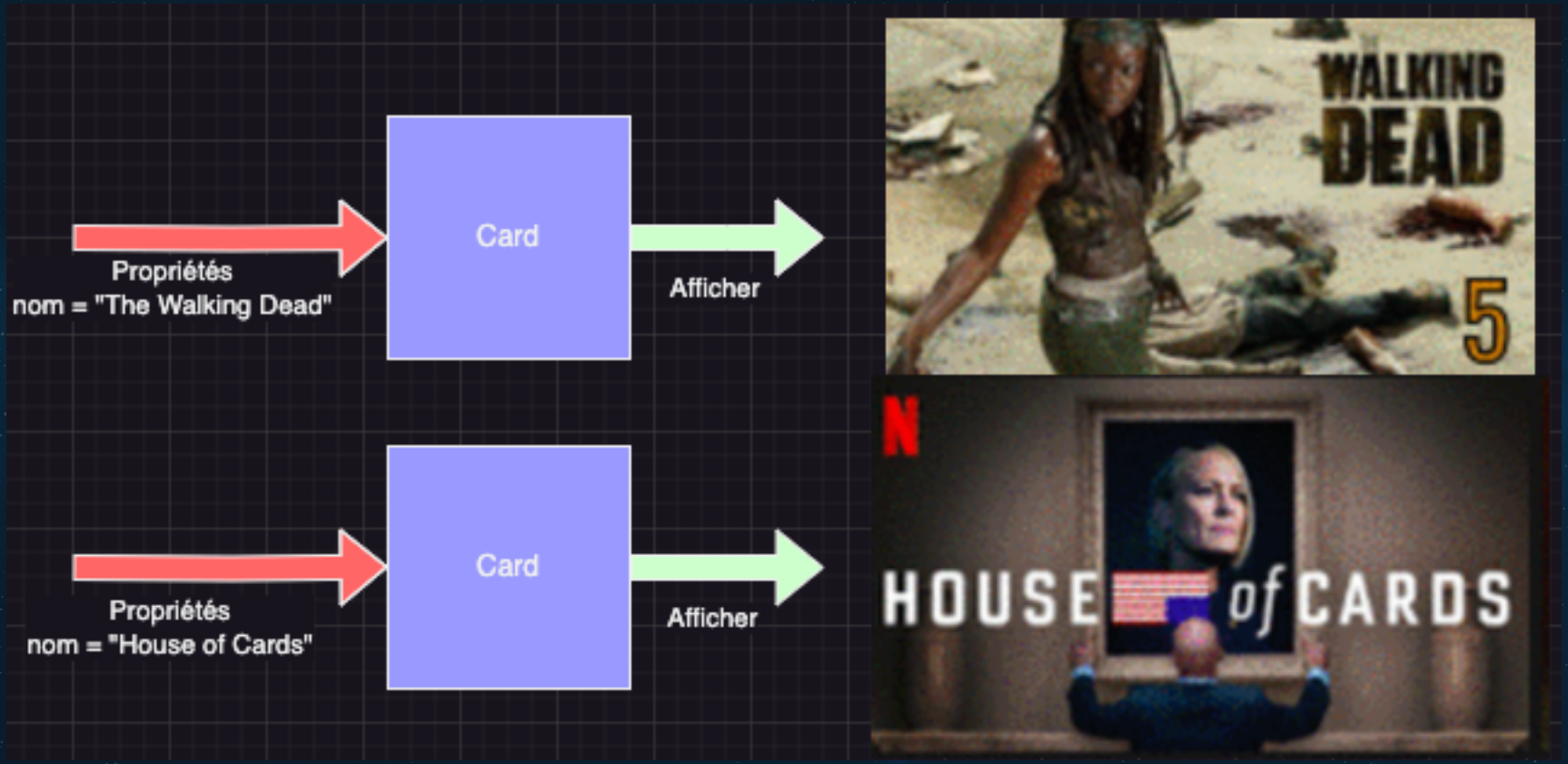
tada web

Comment configurer un composant ?



tada web

Injecter des propriétés dans nos composants



tada web

Composant - Propriétés

Il est possible de passer des propriétés à vos composants:

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

tada web

Composant

Vos composants peuvent recevoir des propriétés:

```
import * as React from "react";

const Card: React.FunctionComponent<{name: string}> = (propriétés) => {
  return <div>{propriétés.name}</div>
}

export {Card}
```

tada web

Travaux pratiques

tada web

Déroulement de la formation

- Je vais vous demander de vous connecter à vos ordinateurs
- On va lancer la VM
- Ensuite on va ouvrir Visual Studio Code et un terminal

tada web

Déroulement de la formation

tada web

Déroulement de la formation

- Ouvrir un terminal et lancer la commande suivante qui permet de cloner le projet sur votre machine
- « git clone https://github.com/sopretty/react-netflix-introduction »
- Une fois le projet cloné dans votre répertoire de travail, vous pouvez récupérer le PDF de la formation dans le dossier « react-netflix-introduction » et le copier en dehors du dossier « react-netflix-introduction ».

Déroulement de la formation

- Nous allons passer à l’étape numéro 1 du TP, pour cela il vous faut:
  - Allez dans le dossier « react-netflix-introduction »
  - En utilisant la commande « cd react-netflix-introduction »
  - Une fois dans le dossier, utilisez cette commande « git checkout step-01 »

Etape-01 Create-React-App

- Créé par Facebook
- Fait partie des meilleures options pour démarrer une nouvelle application web en React
- Configure votre environnement de développement de façon à proposer une expérience développeur agréable
- Optimise votre application pour la production.

Etape-01 Instructions

1. Créez votre environnement React grâce à la commande:

```
npx create-react-app . --template typescript
```

2. Lancez l'application via la commande: **npm run start**
3. Accéder à l’application via votre navigateur sur

```
http://localhost:3000/
```

4. Ouvrir Visual Studio Code, allez le menu « file » (en haut a gauche), choisissez "open folder" et choisissez le dossier « react-netflix-introduction »
5. Analysez la structure du projet et plus particulièrement le fichier « App.tsx »
6. N'hésitez pas à le modifier et à voir les résultats dans votre navigateur

Etape-01 Solution

- Bravo, vous avez créé votre première application React !

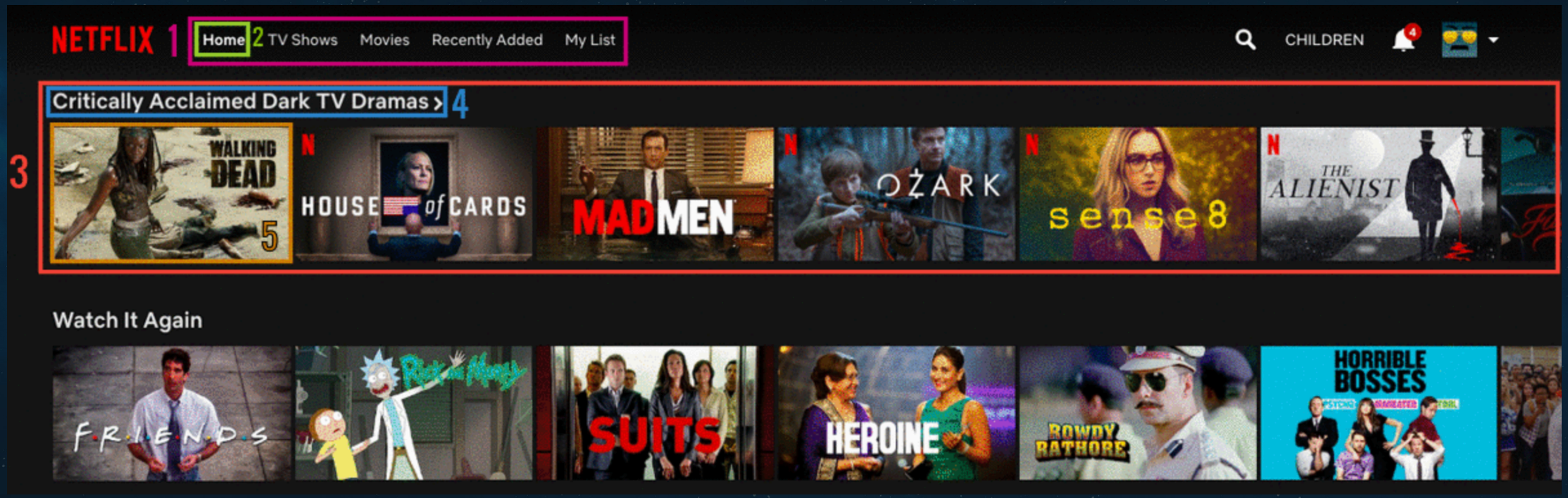
tada web

Etape-01 Solution

- On va parcourir ensemble, ce que create-react-app a généré.

tada web

Quelle est la prochaine étape ?



tada web

Etape-02 Instructions

- Créez votre premier composant appelé « Card »
- Affichez le dans votre application

tada web

Etape-02 Nettoyage de l’environnement

Avant d’aller sur la branche de l’étape suivante, il faut que vous tapiez dans votre console:

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

Pour accéder à l’étape numéro 2, vous pouvez taper dans votre terminal:

- « git checkout step-02 »

Etape-02 Instructions détaillées

1. Créez un composant « Card »
    - Créez un fichier « Card.tsx »
    - Créez un composant nommé « Card » dans ce fichier
    - [Aide]: Vous retrouvez un exemple de JSX pour votre « Card » dans le fichier « Instructions.md » à la racine de votre projet, n'hésitez pas à le changer si vous le voulez.
    - Exportez votre composant « Card »

2. Dans le fichier « App.tsx »
    - Importez le composant Card
    - Remplacez le JSX existant et utilisez votre composant « Card »

tada web

Etape-02 Solution

- On va parcourir ensemble, comment j’ai créé ce composant.

tada web

Etape-02 Solution

- Vous avez créé votre premier composant !
- Si demain Netflix ajoute un nouvelle série, on va devoir créer un composant « Card » pour chaque série ajoutée.
- Comment éviter ça ?

Etape-03 Instructions

Objectif: Rendre ce composant un peu plus générique

- Modifiez votre composant Card pour y ajouter deux propriétés:
  - Un titre (title)
  - L’url d’une image (imageSrc)
- Utilisez le composant « Card » dans votre application et affichez le

tada web

Etape-03 Nettoyage de l’environnement

Avant d’aller sur la branche de l’étape suivante, il faut que vous tapiez dans votre console:

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

Pour accéder à l’étape numéro 3, vous pouvez taper dans votre terminal:

- « git checkout step-03 »

tada web

Etape-03 Instructions détaillées

1. Modifiez le composant « Card.tsx »
    - Ajoutez des propriétés les propriétés « title » et « imageSrc » au composant « Card »
    - [Typescript]: ces deux propriétés sont de type « string »
    - Modifiez le JSX pour que ces propriétés soient prises en compte pour créer votre « Card »
    - [Aide]: vous pouvez trouver un exemple de JSX dans le fichier « instructions.md »

2. Modifiez le JSX du composant « App.tsx » pour ajouter les propriétés désirées
    - Les propriétés passées peuvent être écrites directement dans le JSX
    - Pour la propriété « imageSrc », n'hésitez pas à trouver en ligne une url d'image de votre film ou de votre série préférée

Etape-03 Solution

- On va parcourir ensemble, comment j’ai modifié ce composant.

tada web

Etape-03 solution

- Votre composant est devenu générique
- Vous avez compris comment passer des propriétés à un composant enfant et vous l'avez mis en pratique
- On peut également rendre des propriétés optionnels (exemple)

tada web

Malheureusement pour l’instant tout est statique

tada web

Les hooks

- Sorties en 2018
- Ce sont des fonctions Javascript utilitaires
- Ils sont très utiles pour rendre nos applications dynamiques
- Ils permettent de gérer des états, des effets secondaires au sein d’un composant
- Ils permettent d’accéder facilement à des éléments HTML (référence) du DOM
- Ils sont tous préfixés par « use… »
- Il est aussi possible de créer ses propres hooks

Les hooks

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

tada web

Gérer un état dans un composant

```
const Example: React.FunctionComponent = () => {
  let count = 0;

  const onClick = () => {
    count++;
  };

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

Vous avez cliqué 0 fois

Cliquez ici

tada web

Rappels - JS

tada web

Affecter par décomposition - Javascript

Syntaxe

```javascript
let a, b, rest;
[a, b] = [10, 20];
console.log(a); // 10
console.log(b); // 20

[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(a); // 10
console.log(b); // 20
console.log(rest); // [30, 40, 50]

({a, b} = {a: 10, b: 20});
console.log(a); // 10
console.log(b); // 20

// Proposition de syntaxe (niveau 4)
({a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40});
console.log(a); // 10
console.log(b); // 20
console.log(rest); // {c: 30, d: 40}
```

tada web

1. Hook d’état

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

const Example: React.FunctionComponent = () => {
  // Déclaration d'une nouvelle variable d'état, que l'on appellera “count”
  const [count, setCount] = useState(0);

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

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

- C’est un état local
- Il prend en paramètre une valeur par défaut
- Cet état est préservé d’un affichage à l’autre
- Il retourne une paire
  - l’état courant
  - Une fonction qui permet de le changer
- Chaque appelle au setter déclenche un re-rendu du composant

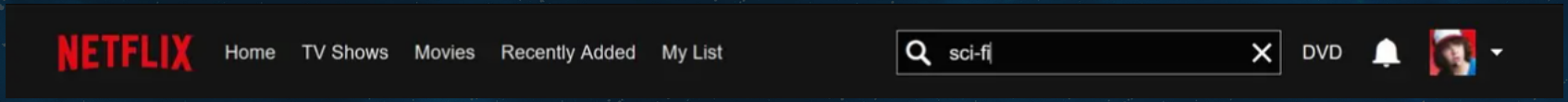
tada web

1. Hook d’état

```
import React, { ChangeEvent, useState } from "react";  6.9k (gzipped: 2.8k)

const LoginForm: React.FunctionComponent = () => {
  const [username, setUsername] = useState('')
  const [password, setPassword] = useState('')

  const handleUsernameChange =  (event: ChangeEvent<HTMLInputElement>) => {
    setUsername(event.target.value)
  }

  const handlePasswordChange = (event: ChangeEvent<HTMLInputElement>) =>  {
    setPassword(event.target.value)
  }

  return (
    <div>
      <input
        id='username'
        placeholder='username'
        value={username}
        onChange={handleUsernameChange}
      />
      <input
        id='password'
        type='password'
        placeholder='password'
        value={password}
        onChange={handlePasswordChange}
      />
    </div>
  );
};

export { LoginForm };
```

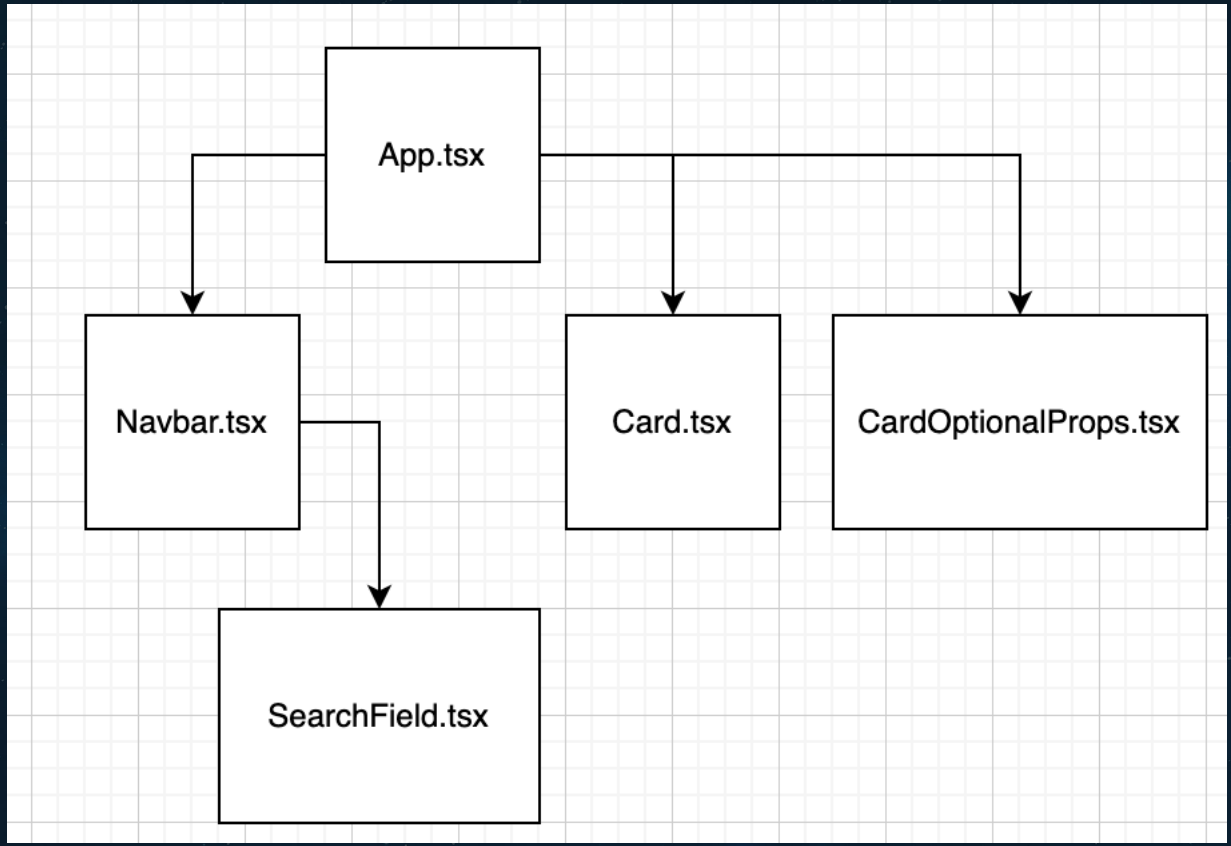
tada web

Ou pourrait-on utiliser ce hook dans notre application ?



tada web

Etape-04 Modifications



tada web

Etape-04 Instructions

- Modifiez le composant « SearchField »
  - Utilisez le hook d’état « useState » avec une valeur par défaut
  - Passez en propriété de votre « input » la valeur de l’état local
  - Créez une fonction qui va changer la valeur de votre état local dès que l’utilisateur écrit quelque chose
  - Passez en propriété de votre « input » la fonction qui va être appelée à chaque fois que l’utilisateur écrit quelque chose

Etape-04 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 -fxd —exclude="node_modules" »

Pour accéder à l’étape numéro 4 vous pouvez taper dans votre terminal:

- « git checkout step-04 »

tada web

Etape-04 Instructions détaillées

- Modifiez le composant « SearchField »
  - Utilisez le hook d’état « useState » avec une valeur par défaut
  - Connectez la valeur de l’état à votre input

```
<input
  value={maValeur}
  className="search-field"
  type="text"
  placeholder="Search"
/>
```

tada web

Etape-04 Instructions détaillées

- Modifiez le composant « SearchField »
  - Créez une fonction qui va changer la valeur de votre état dès que l'utilisateur écrit quelque chose

```
const onChange = (event: React.ChangeEvent<HTMLInputElement>) => {
  // changer la valeur de l'état local
  // vous récupérez la valeur de ce que tape l'utilisateur via event.target.value
};

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

tada web

Etape-04 Solution

- On va parcourir ensemble, comment j’ai modifié ce composant.

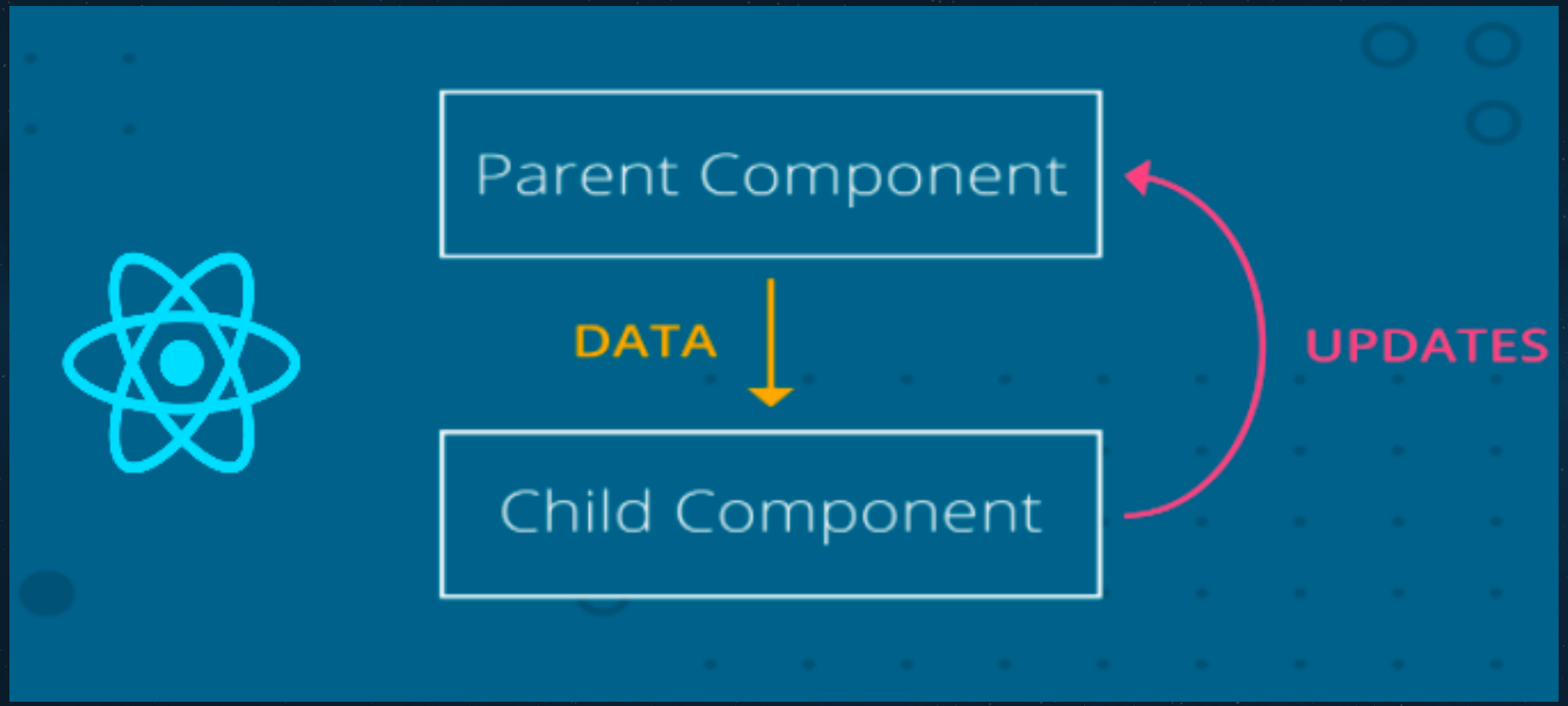
tada web

Etape-04 Solution

- Vous avez créé votre premier composant qui contient un état local
- On aimerait que les séries se filtrent lorsque l'on écrit quelque chose dans le champ de recherche, comment peut-on faire ?
- Comment faire remonter l'information du composant « SearchField » vers l'App ?

tada web

Une liaison unidirectionnel (One way biding)



https://labs.tadigital.com/index.php/2020/03/31/unidirectional-data-flow-in-react/

tada web

Composant - Propriétés

Il est possible de passer des propriétés à vos composants:

```
const App: React.FunctionComponent = () => {
  return (
    <div>
      <Card
        title="The Walking Dead"
        imageSrc="https://..."
      />
    </div>
  );
};

export { App };
```

tada web

Composant - Propriétés

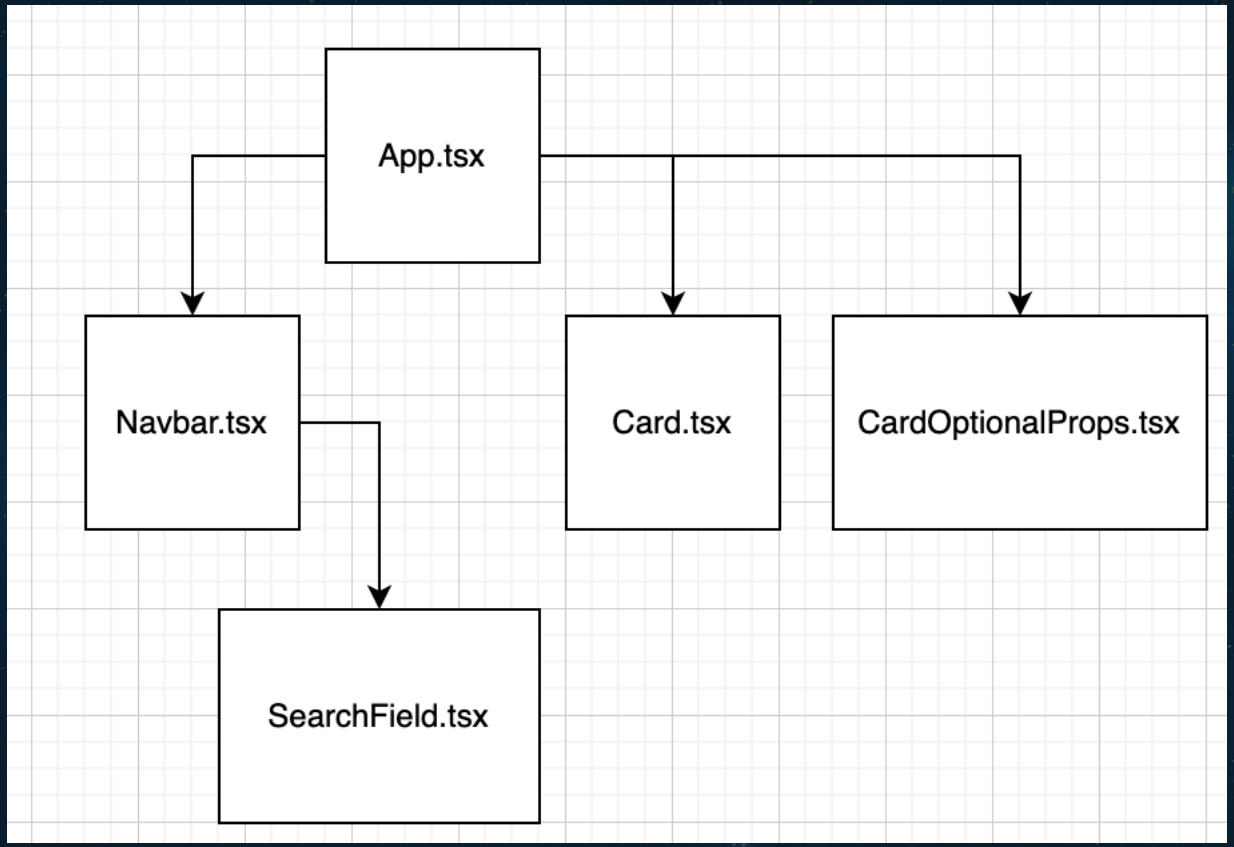
Il est possible de passer des propriétés à vos composants:

```
const App: React.FunctionComponent = () => {

  const onClick = () => {
    console.log('click')
  }

  return (
    <div>
      <Card
        onClick={onClick}
        title="The Walking Dead"
        imageSrc="https://..."
      />
    </div>
  );
};

export { App };
```

```
const Card: React.FunctionComponent<{ title: string; imageSrc: string; onClick: () => void }> = ({
  title,
  imageSrc,
  onClick
}) => {
  return (
    <div
      onClick={onClick}
    >
      <span>{title}</span>
      <img
        src={imageSrc}
        alt={title}
      />
    </div>
  );
};
```

tada web

Etape-05 Instructions



tada web

Etape-05 Instructions

- Déplacez le hook d’état du composant « SearchField » au composant « App »
- Déplacez la fonction « onChange » du composant « SearchField » au composant « App »
- Modifiez le JSX du composant « App »
- Modifiez les propriétés du composant « Navbar » et son JSX
- Modifiez les propriétés du composant « SearchField » (et son JSX)

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 -fxd —exclude="node_modules" »

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

- « git checkout step-05 »

tada web

- [Typescript]: lisez le fichier « instructions.md » pour récupérer le type de la fonction passé à vos composants

Etape-05 Instructions détaillées

- Modifiez le composant « App »
  - Déplacez le hook d'état du composant « SearchField » (n'hésitez pas à la renommer)
  - Créez une fonction qui va mettre à jour l'état local (n'hésitez pas à la renommer)
  - Modifiez le JSX pour passer la valeur de l'état local et la fonction qui met à jour l'état local au composant « NavBar »

- Modifiez le composant « Navbar »
  - Modifiez les propriétés du composant pour prendre en compte la fonction qui lui est passée et la valeur de l'état
  - Modifiez son JSX pour passer les bonnes propriétés au composant « SearchField »

- Modifiez le composant « SearchField »
  - Supprimez le hook d'état
  - Modifiez les propriétés du composant pour prendre en compte la fonction qui lui passée et la valeur de l'état
  - Modifiez le JSX pour passer cette fonction et la valeur de l'état

Etape-05 Solution

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

tada web

Etape-05 Solution

- Vous avez déplacé l’état « Searchfield » dans le composant « App »
- Ce qui va nous permettre de filtrer les séries dans le composant « App »

tada web

Composant - JSX

```
const DigitalLearningHub: React.FunctionComponent = () => {

  const boolean = true;

  return <div>{boolean ? <div>Supérieur</div>: <div>Inférieur</div>}</div>
}
```

tada web

Rappels - JS

tada web

Fonction map - Javascript

En JS, Il existe une méthode « map » qu’on peut appeler sur des listes

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

```
const array1 = [1, 4, 9, 16];

const map1 = array1.map(x => x * 2);

console.log(map1);
// expected output: Array [2, 8, 18, 32]
```

tada web

JSX - parcours de liste

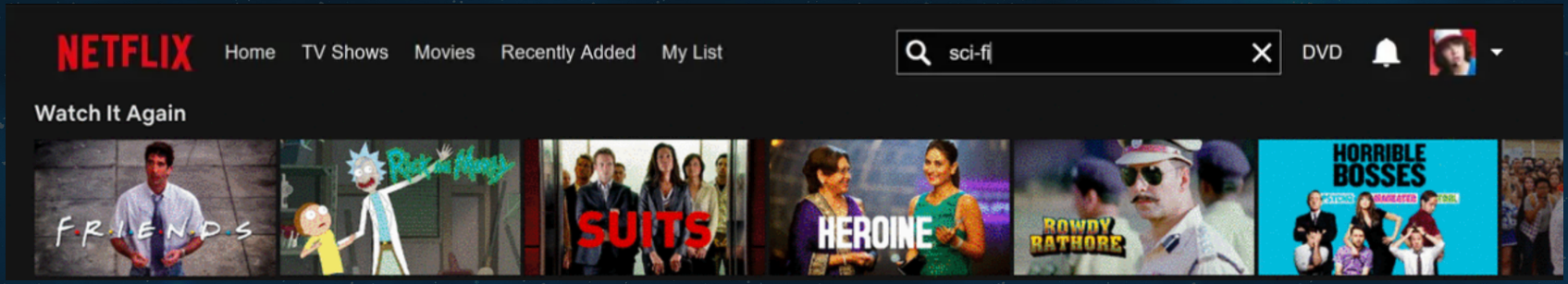
```
const DigitalLearningHub: React.FunctionComponent = () => {
  const courses = [
    { title: "React - Introduction" },
    { title: "React - Intermediate" },
    { title: "Deep dive into React" },
  ];

  return (
    <div className="courses">
      {courses.map((course) => (
        <div key={course.title}>{course.title}</div>
      ))}
    </div>
  );
};

export { DigitalLearningHub };
```

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

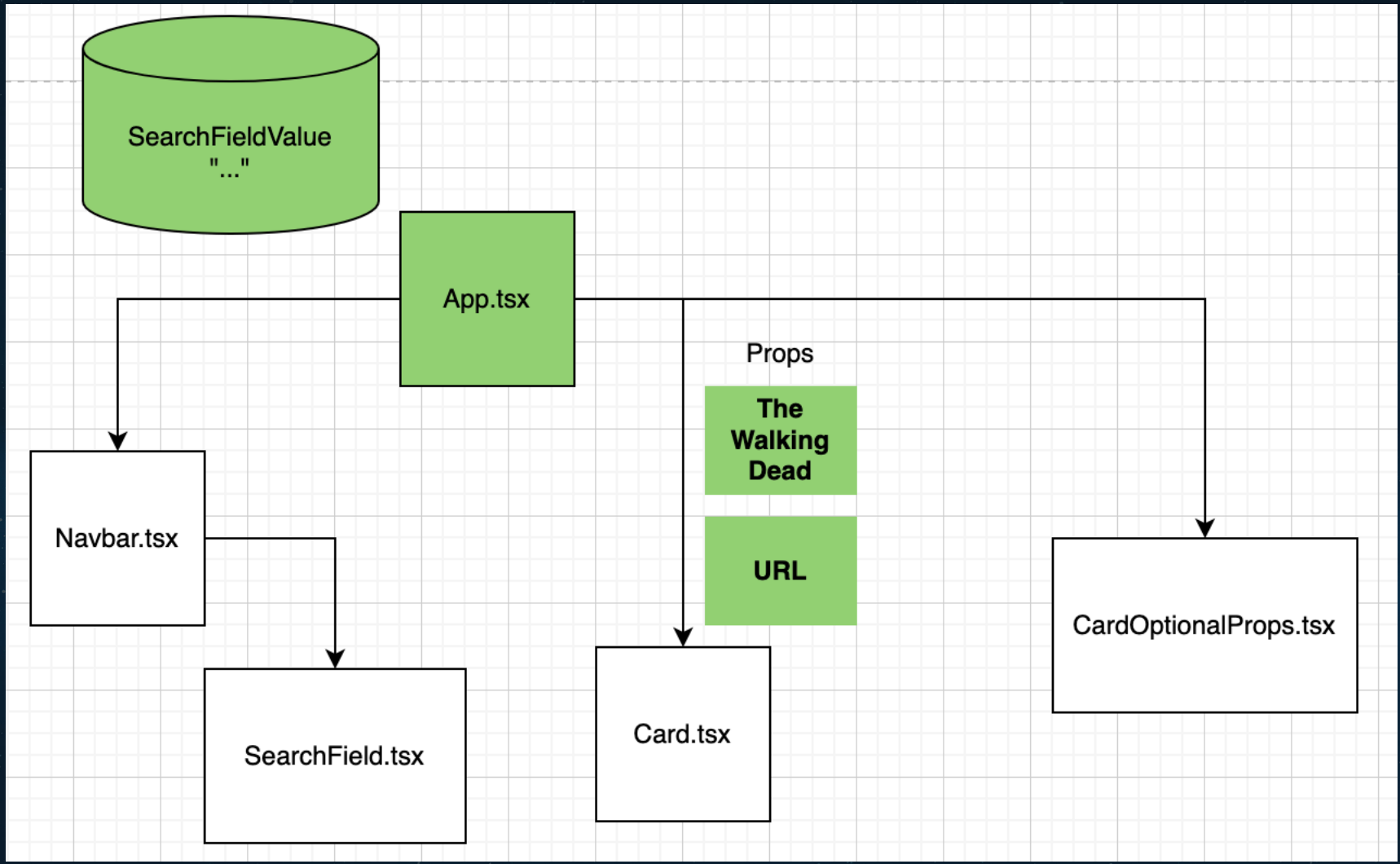
```
return (
  <div className="courses">
    <div key="React - Introduction">React - Introduction</div>
    <div key="React - Intermediate">React - Intermediate</div>
    <div key="Deep dive into React">Deep dive into React</div>
  </div>
);
```

tada web

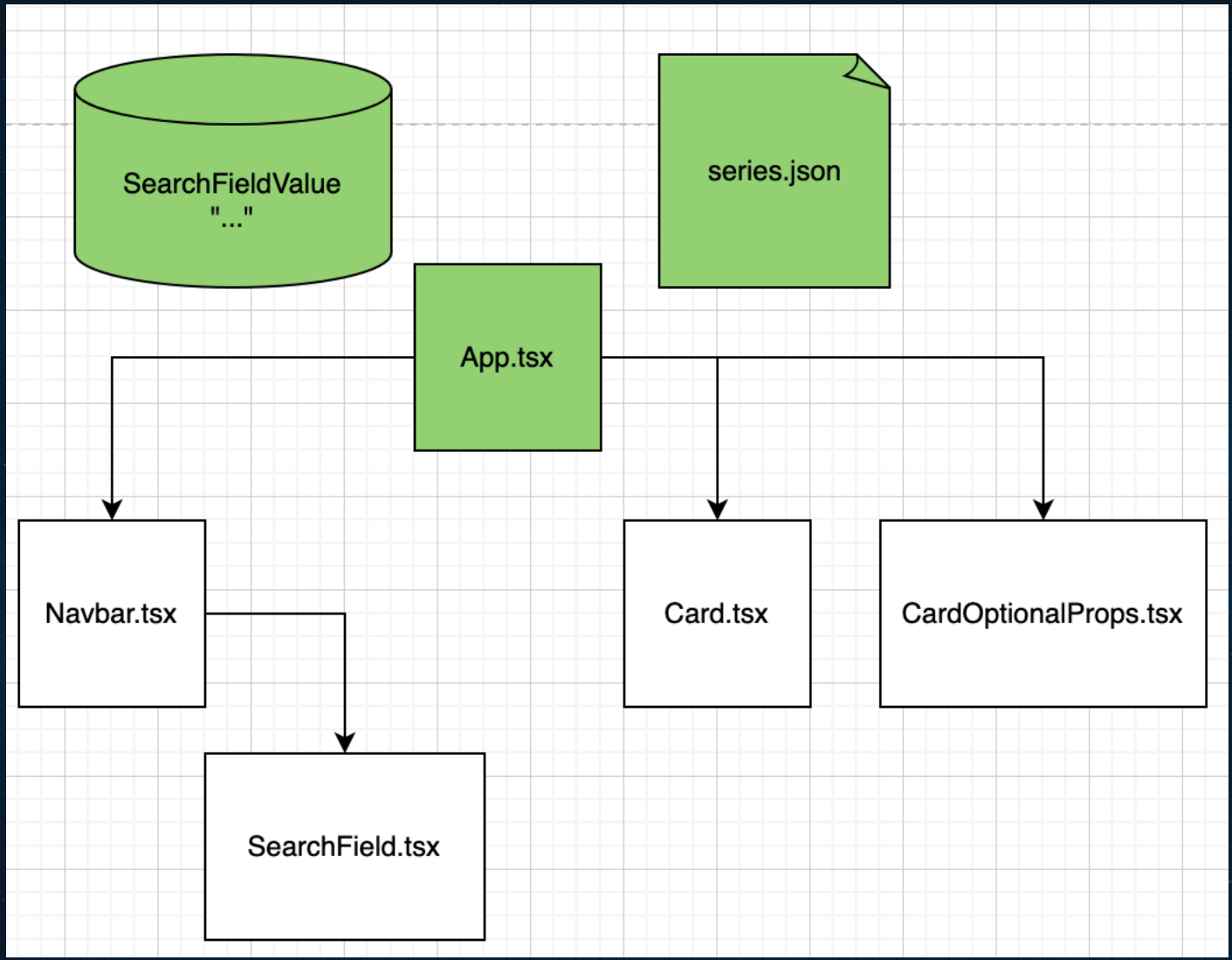
Comment filtrer nos séries en fonction de la recherche de l’utilisateur ?



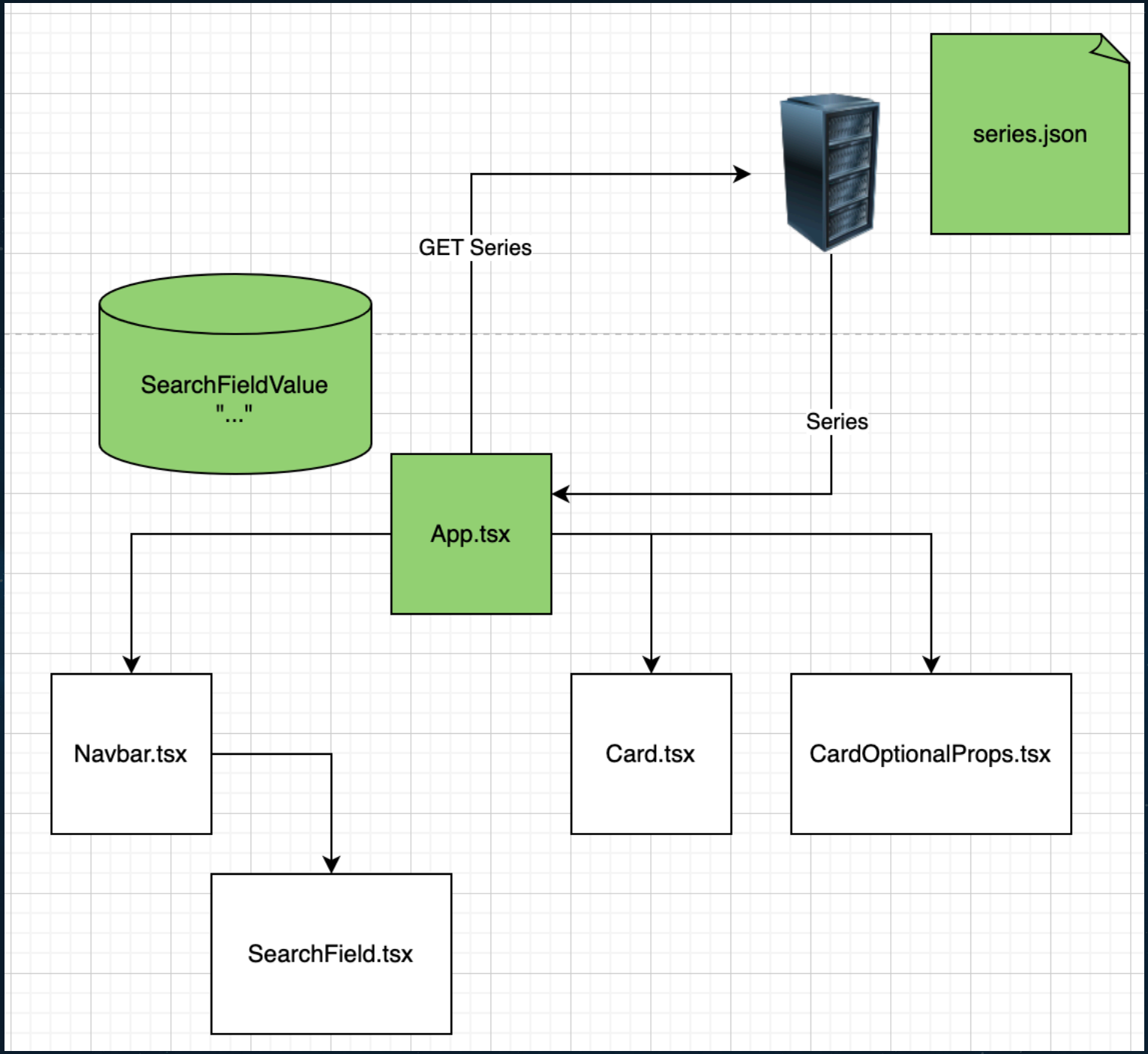
tada web


SearchFieldValue
"..."
App.tsx
Navbar.tsx
SearchField.tsx
Card.tsx
Props
The Walking Dead
URL
CardOptionalProps.tsx

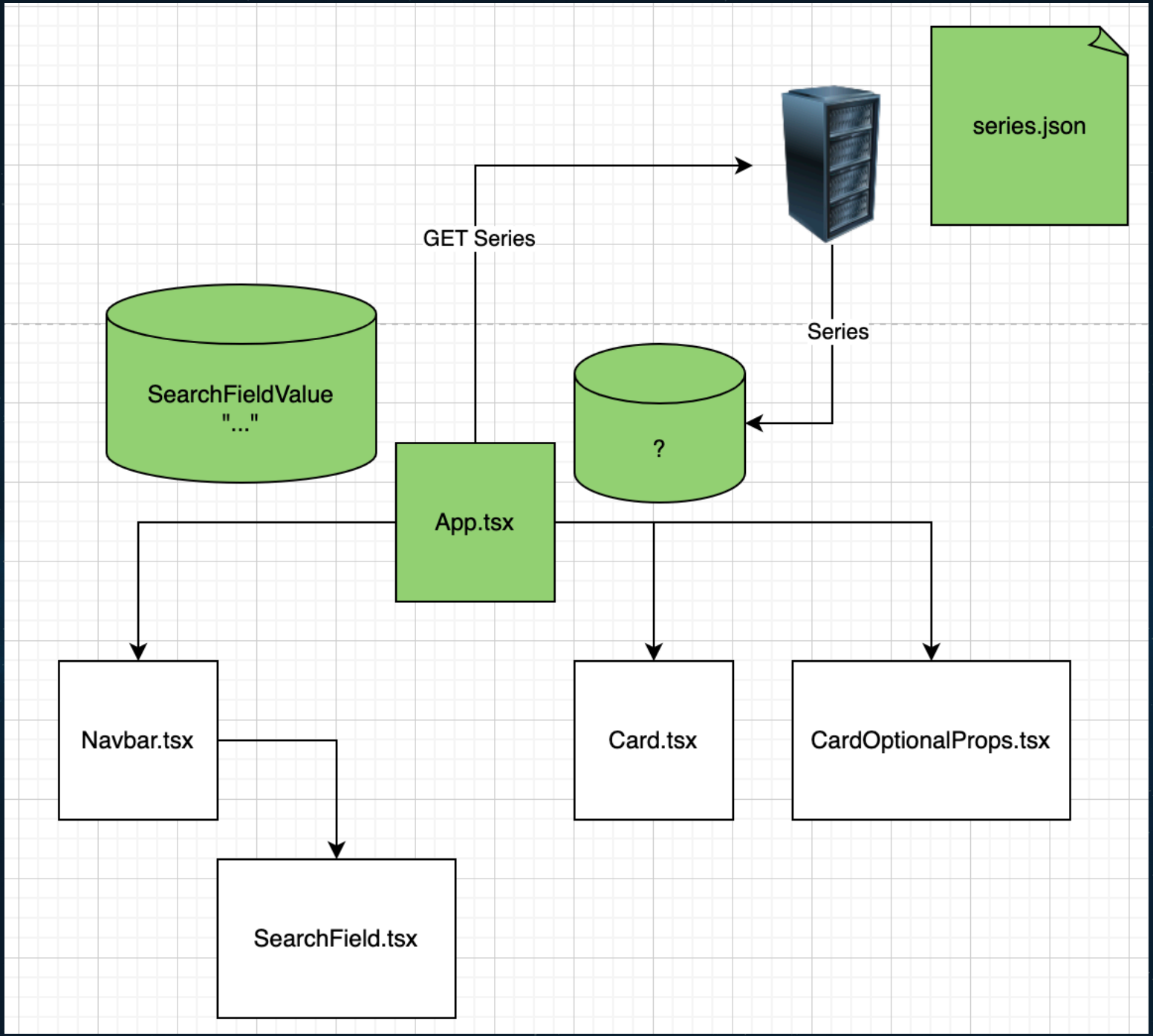
tada web


SearchFieldValue
"..."
series.json
App.tsx
Navbar.tsx
Card.tsx
CardOptionalProps.tsx
SearchField.tsx

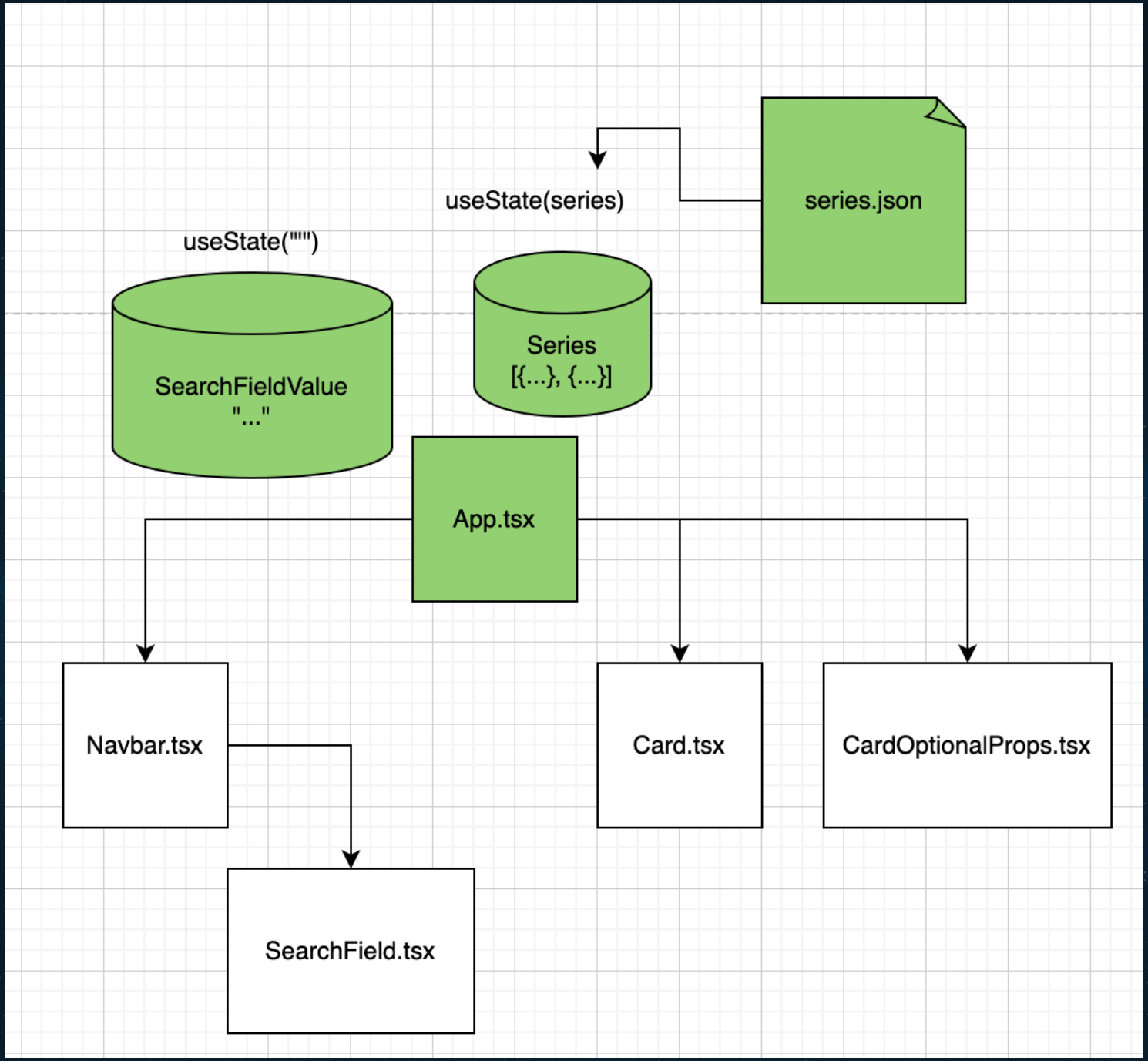
tada web


series.json
GET Series
SearchFieldValue
"..."
Series
App.tsx
Navbar.tsx
Card.tsx
CardOptionalProps.tsx
SearchField.tsx

tada web


series.json
GET Series
Series
SearchFieldValue
"..."
?
App.tsx
Navbar.tsx
Card.tsx
CardOptionalProps.tsx
SearchField.tsx

tada web


useState(series)
series.json
useState("")
SearchFieldValue
"..."
Series
[{...}, {...}]
App.tsx
Navbar.tsx
Card.tsx
CardOptionalProps.tsx
SearchField.tsx

tada web

Etape-06 Instructions

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

tada web

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 -fxd —exclude="node_modules" »

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

- « git checkout step-06 »

tada web

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 ?

tada web

Ressources du cours

- Documentation sur les imports JS: « https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import »
- Documentation sur les exports JS: « https://developer.mozilla.org/en-US/docs/web/javascript/reference/statements/export »
- Documentation sur les ternaires JS « https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_operator »
- Documentation sur les affectations par décomposition « https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment
- Documentation sur la fonction Map « https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map »