

React

Table des matières

- **Introduction**

- React, c'est quoi ?
- Le JavaScript & l'ECMAScript
- La JSX
- Le DOM virtuel
- Environnement de Dev

- **Développer avec React**

- Créer un nouveau projet
- Projet React avec Vite
- Le but des composants

- **Les composant React**

- Le paramètre d'entrée
- Gestion du style
- Rendu conditionnel
- Les collections
- Les événements du DOM

- **Les composants React avancé**

- Le mécanisme des Hooks
- Manipulation du state
- Le Hook d'état
- Les formulaires
- Interaction entre les composants React
- Le cycle de vie d'un composant
- Le Hook d'effet

- **Les requêtes AJAX**

- **Récapitulatif des fondamentaux**

- **Annexes**

- **Ressources**

Introduction

React, c'est quoi ?

Introduction

Qu'est-ce que React ?

- Une bibliothèque JavaScript qui a été initialement créée par Facebook en 2011.
 - Open-Source depuis Mai 2013.
 - Développé par : Meta (Facebook, Instagram) et la communauté.
- Permet de construire des interfaces pour des utilisateurs.
- Une structure basé sur des composants déclaratifs
 - Le rendu d'un composant est généré sur base des valeurs de l'état local.
 - Lorsqu'une valeur est modifiée, le rendu est automatiquement actualisé.
- L'interaction avec le DOM est extrêmement rapide via un DOM Virtuel.
- Intégration possible dans une application Web existante (*ASP.Net, PHP, Java, ...*)

Une bibliothèque ? Pas un Framework ?

React a été conçu comme étant une bibliothèque et non un framework, contrairement à ses concurrents (Angular, VueJS, ...).

Le fonctionnement de React est uniquement dédié à la gestion de l'interface de l'application. Il peut être considéré comme la vue dans le modèle MVC.

Par ailleurs, la bibliothèque de React n'intègre pas de mécanisme de routage, de gestionnaire de service, de générateur de code (CLI).

Flexibilité dans le développement

A part pour la génération des vues, React n'impose pas de technologie ou de pattern. Le développeur a donc le contrôle total de son code et de ses dépendances.

En fonction des besoins, il est possible d'ajouter des bibliothèques tierces pour enrichir les fonctionnalités de l'application lors de son développement.

Cette philosophie permet de bénéficier de plus de liberté qu'avec un framework.

Par exemple :

« react-router » ajoute un mécanisme de routage dans l'application.

« redux & react-redux » permet la mise en place d'un gestionnaire de données.

Langages de programmation utilisés

Pour développer des applications React, nous allons principalement utiliser les langages de programmation suivants :

- Le JavaScript (ECMAScript 2015+)
Langage qui nous permettra de coder la logique de nos composants.
- Le JSX
Une extension du Javascript qui permet réaliser le rendu des composants.

Il est également possible d'utiliser le langage TypeScript pour avoir du typage statique.
Cheatsheet TS pour React : <https://react-typescript-cheatsheet.netlify.app>

JavaScript & ECMAScript

Introduction

ECMAScript

L'ECMAScript (ECMA-262) est la standardisation du Javascript (depuis 1997).

La spécification du langage a stagné quelques années en version 5 (publié en 2009). En 2015, la nouvelle norme (ES6 / ES2015) a été publiée, celle-ci apportant énormément de nouveautés. Depuis celle-ci, la norme est mise à jour chaque année.

L'implémentation de l'ECMAScript dans le Javascript est mise en œuvre par la fondation Mozilla .

Plus d'info:

developer.mozilla.org/fr/docs/Web/JavaScript/JavaScript_technologies_overview

ECMAScript - Les fonctions fléchées

L'écriture fléchée (`=>`) offre une syntaxe raccourcie pour créer des fonctions.

En déclarant la fonction sous forme de constante, son unicité est assurée.

Les fonctions fléchées permettent également de créer des expressions lambda.

```
// ES5
var myFn = function(x) {
    return x + 1;
};

// ES2015+
const myFn = x => {
    return x + 1;
};

// Lambda expression
const myFn = (x) => x + 1;
```

ECMAScript - Le destructuring

Permet d'assigner des variables depuis les valeurs d'un objet ou d'un tableau.

- Objet

Le nom des variables à assigner doit être identique aux noms des propriétés.

- Tableau

L'assignation des variables est basé sur la position des éléments dans le tableau.

```
// Destructuring d'objet
const myObject = {
  number: 42,
  message: "Hello world!"
};

const {number, message} = myObject;

// Destructuring du tableau
const tab = ['A', 'B', 'C', 'D']

const [valA, valB] = tab;
```

ECMAScript - Le paramètre « Rest »

Permet de créer un tableau avec un nombre indéterminé d'argument.
Pour l'utiliser, il faut appliquer le préfixe « ... » sur le dernier argument.

```
const combine = function(separator, ...tabs) {  
    return tabs.join(separator);  
}  
  
const combinedString = combine("-", "Riri", "Fifi", "Loulou");
```

ECMAScript - Le paramètre « Rest »

★ Exemple d'utilisation du paramètre « Rest » avec du destructuring

```
//Destructurer un tableau
const myTeam = ["Donald", "Daisy", "Riri", "Fifi", "Loulou"];

// ES5
const first = myTeam[0];      //"Donald"
const second = myTeam[1];     //"Daisy"
const rest = myTeam.slice(2); //["Riri", "Fifi", "Loulou"]

// ES2015
const [first, second, ...rest] = myTeam; //Même résultat
```

ECMAScript - L'opérateur « Spread »

Également appelé *“Opérateur de décomposition”*. Il permet de décomposer un objet itérable. Pour l'utiliser, il faut appliquer le préfixe « ... » sur l'objet à décomposer.

```
// Exemple de décomposition
const myString = "Riri";

const myArray = [...myString];
// myArray => ["R", "i", "r", "i"]
```

ECMAScript - L'opérateur « Spread »

★ Exemple de l'utilisation de l'opérateur « Spread »

```
// Concaténer plusieurs éléments itérables
const neveux = ["Riri", "Fifi", "Loulou"];

// ES5
const team = ["Donald", "Daisy"].concat(neveux);

// ES2015+
const team = ["Donald", "Daisy", ...neveux];

// Dans les 2 cas le resultat sera :
// ["Donald", "Daisy", "Riri", "Fifi", "Loulou"]
```


Manipulation de tableau de données

Le langage JavaScript définit des méthodes qui permettent de manipuler les tableaux de manière simple.

Parmi celle-ci, nous allons souvent utiliser les méthodes suivantes en React :

- `Array.prototype.filter()`
- `Array.prototype.map()`

Array - Méthode « filter »

La méthode « filter » permet d'obtenir un nouveau tableau contenant tous les éléments du tableau remplissant la condition déterminée par le « callback ».

```
// Liste de données
const myTeam = ['Donald', 'Daisy', 'Riri', 'Fifi', 'Loulou'];

// Filtre via une fonction
const myFilter = function(world) {
  return world.includes("o");
}
const r1 = myTeam.filter(myFilter);      // ["Donald", "Loulou"]

// Filtre via une fonction Lambda
const r2 = myTeam.filter(p => p.length < 5); // ["Riri", "Fifi"]
```

Array - Méthode « map »

La méthode « map » permet d'obtenir un nouveau tableau contenant tous les éléments modifiés par l'exécution de la méthode « callback ».

```
// Liste de nombres
const myValues = [1, 2, 3, 4, 5];

// Utilisation à l'aide d'une fonction Lambda
const myDoubles = myValues.map(n => n * 2);
// myDoubles => [2, 4, 6, 8, 10]
```

Exemple d'utilisation avec React : Elle permet de transformer facilement un tableau d'objet (en JS) en tableau de Composant (en JSX), pour la création d'une liste.

Le JSX

Introduction

Le JSX

Le JSX est une syntaxe d'extension du Javascript permettant de faciliter l'écriture du rendu des composants React. Cette syntaxe est très similaire à la syntaxe du HTML.

```
const element = (  
  <div>  
    <p className="title-element">Exemple de JSX</p>  
    <img src={image.url} alt={image.description} />  
  </div>  
);
```

L'écriture JSX utilise le “*camelCase*” pour les attributs des éléments.

L'attribut HTML « **class** » devient « **className** », car le mot “class” est réservé en JS.

Exemple précédent en Javascript

Brève explication des paramètres de la méthode « createElement » :

- Premier paramètre (string) :
Le type de la balise
- Deuxième paramètre (objet) :
Les attributs de la balise
- Les paramètres suivants (...objet) :
Les éléments à placer dans la balise

Note: Depuis la version 17 de React, une nouvelle méthode transformation est utilisée.

```
const element = React.createElement(  
  "div",  
  null,  
  React.createElement(  
    "p",  
    { className: "title-element" },  
    "Exemple de JSX"  
  ),  
  React.createElement(  
    "img",  
    {  
      src: image.url,  
      alt: image.description  
    }  
  )  
);
```

Utilisation de JSX avec le DOM

- Exemple de l'utilisation de JSX à l'aide de ReactDOM

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Exemple JSX</title>
  <!-- Scripts nécessaires pour utiliser React -->
  <script defer src="index.js" type="text/babel"></script>
</head>
<body>
  <div id="container"></div>
</body>
</html>
```

```
const element = <h1>Hello from React!</h1>;

const container = document.getElementById('container');

const root = ReactDOM.createRoot(container);
root.render(element);
```

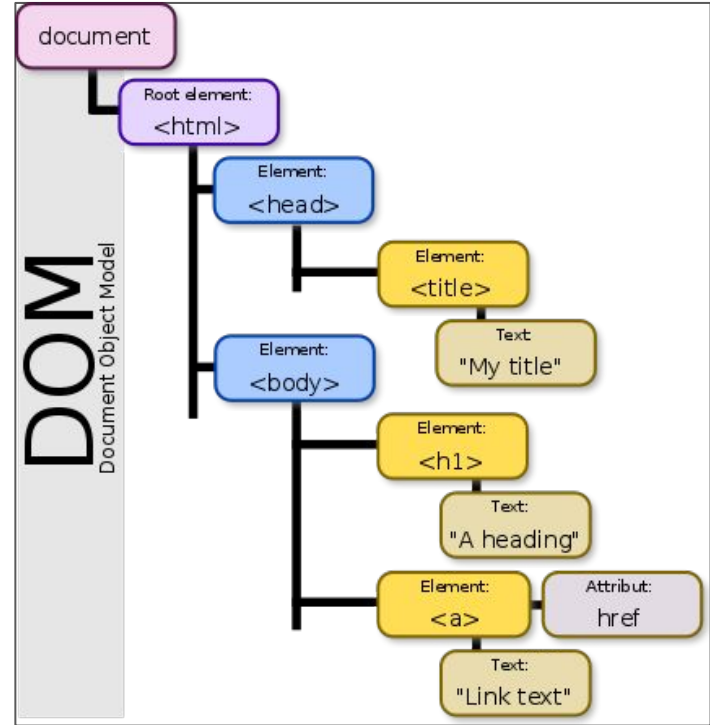
Le DOM Virtuel

Introduction

Document Object Model

Le DOM est une API qui permet d'interagir en JavaScript avec les documents HTML ou XML.

La représentation du document est un arbre nodal. Chaque nœud représente une partie du document.



Le DOM virtuel

Interagir avec le DOM devient vite lourd et complexe. Pour éviter cela, React utilise un DOM virtuel qui est une représentation du DOM sous forme d'objet JavaScript.

Le DOM virtuel est créé à partir des méthodes de rendu des composants et celui-ci est régénéré après chaque changement d'état de l'application.

Les avantages sont :

- La génération du DOM virtuel est extrêmement rapide.
- Analyse l'évolution du DOM virtuel pour définir les modifications à réaliser.
- React réalise le minimum de modifications sur le DOM à l'aide d'opérations simples.

L'environnement de développement

Introduction

L'environnement de développement

Pour développer en React, nous allons utiliser Node JS

- C'est une plateforme de développement JavaScript (en dehors des navigateurs).
- Permet de configurer les dépendances du projet à l'aide de « npm ».

Pour écrire notre code, nous pouvons utiliser par exemple :

- Visual Studio Code
- Webstorm (Payant)
- Sublime Text 3

L'environnement de développement

Le plugin "React Developer Tools" est un outils qui permet :

- Inspecter l'arborescence des composants React
- Voir l'état de « Props » et du « State » d'un composant.

Le plugins s'intègre dans les outils de développement du navigateur → Onglet React

"React Developer Tools" est disponible pour Firefox et Chrome

Développer avec React

Créer un nouveau projet

Développer avec React

Aperçu des différentes technologies

Il existe plusieurs solutions qui permettent de créer un projet avec React

- Web App Client
 - Vite
 - Create React App (CRA)
 - Parcel
- [Framework] Web App Server
 - Next.js
 - Remix
 - Gasby
- Intégration dans une page Web
 - CDN React
- [Framework] App Mobile
 - React Native
 - Expo
 - Ionic
- [Framework] App Desktop
 - Electron

Choix pour créer le projet

Avant de créer le projet, il faudra choisir la solution la plus adaptée à nos besoins :

- Application Web (Client ou Serveur)
- Application Mobile
- Application Desktop

Chacune de ces solutions ont bien entendu leurs avantages et leurs inconvénients.

Dans le cadre de l'apprentissage de React, nous allons créer un projet avec « Vite ».

Projet React avec Vite

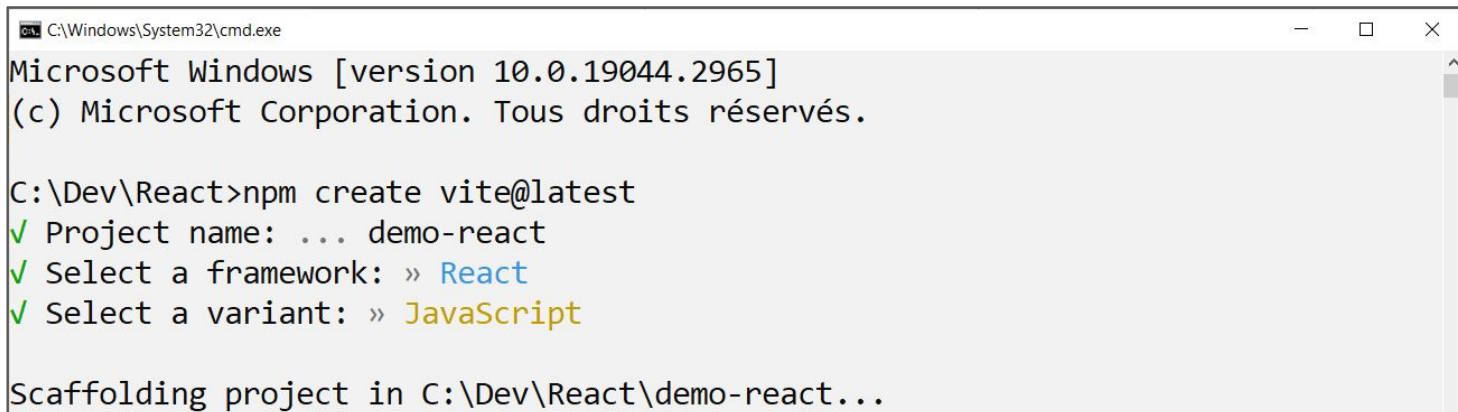
Développer avec React

Création de la base du projet

Ouvrir le terminal dans un répertoire.

Générer le projet en utilisant « Vite » avec le template « React + JavaScript »

⚠ Le nom du package doit être en minuscule et il ne doit pas contenir d'espace !



```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.19044.2965]
(c) Microsoft Corporation. Tous droits réservés.

C:\Dev\React>npm create vite@latest
✓ Project name: ... demo-react
✓ Select a framework: » React
✓ Select a variant: » JavaScript

Scaffolding project in C:\Dev\React\demo-react...
```

Remarque : Remplacer le nom du projet par un point, pour le créer dans le répertoire actuel.

Création de la base du projet

Une fois la génération de code terminée, nous obtenons cette structure.

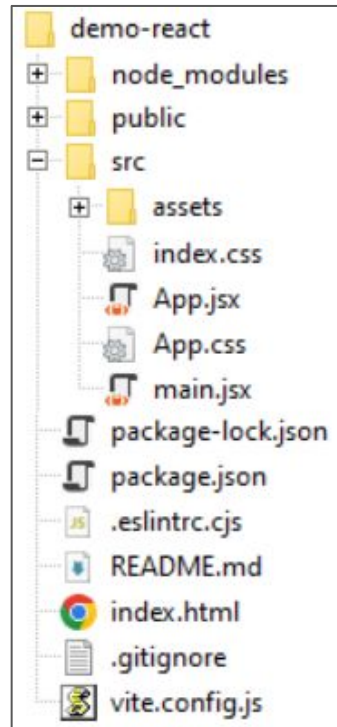
Pour tester le projet, nous allons démarrer le serveur de développement.

Pour cela, utiliser les commandes :

" cd <my-project> "

" npm install "

" npm run dev "



Explication des éléments générés

Quelques explications sur les éléments créés :

- Le dossier « public »
Ressources static (par exemple : une icône).
- Le dossier « src »
Code source de l'application React.
- Les fichiers « app.jsx »
Composant « **App** » de l'application
- Les fichiers « main.jsx »
Utilisation du **ReactDOM** pour injecter le composant « App » dans le code HTML.

Structure de fichier

React n'impose pas de structure pour vous ordonnez vos fichiers.

Il est conseillé d'en mettre une en place pour faciliter le développement et la maintenance des projets.

Des solutions populaires existent pour ordonner les fichiers d'un projet.

Par exemple :

- Par types d'éléments (components, store, api, page, ...)
- Par fonctionnalités

Structure de fichier – Choix pour le support

La structure utilisée dans le support est « Par types d'éléments », sous la forme :

- Un dossier par éléments.
- Les éléments seront séparé en catégories :
 - **Components** Composant “*simple*” et réutilisable.
Exemple → Une barre de recherche.
 - **Containers** Composant reprenant les différentes features.
Exemple → Le menu de navigation.

D'autres catégories de fichiers seront ajoutés par la suite (*page, store, api, ...*).

Le but des composants

Développer avec React

Les Composants React

Les composants React vont nous permettre de créer nos interfaces en blocs de code indépendant et réutilisable. Ceux-ci ont un paramètre d'entrée et un rendu.

Il existe deux familles de composant React :

- Les fonctions
 - Basique : limité à la génération de rendu via le JSX
 - Avancé : utilisation du mécanisme des Hooks
- Les classes héritant de « `React.Component` »

 Lors du développement, nous allons créer un fichier par composant React. Pour cela, nous utiliserons les instructions d'import et d'export du JavaScript.

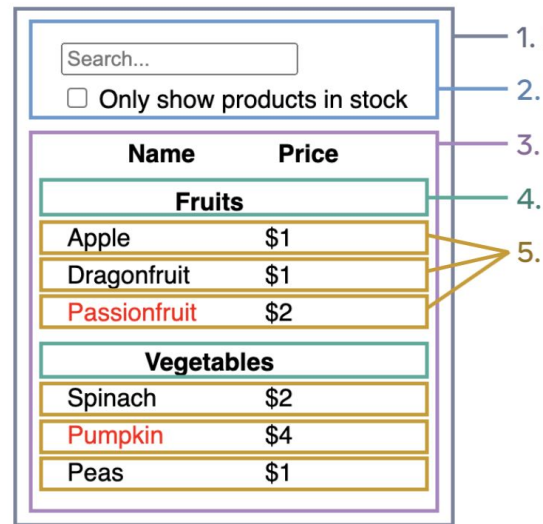
Découpe d'un rendu sous forme de composant

Pour concevoir une application Web en React.

Il est nécessaire de découper le rendu sous forme de différents composants.

Exemple de découpe :

- 1) **FilterableProductTable** : Le composant principal.
- 2) **SearchBar** : Interaction avec la saisie de l'utilisateur.
- 3) **ProductTable** : Affiche et filtre les données.
- 4) **ProductCategory** : Affiche le titre d'une catégorie.
- 5) **ProductRow** : Affiche de détail d'un produit.



Les composants React

Composant basé sur une fonction

Les composants React

Composant basé sur une fonction

Un composant React est une fonction qui renvoie un rendu "React" à l'aide du JSX .

```
const Welcome = function(props) {  
  return (  
    <h1>Hello {props.name}!</h1>  
  );  
};  
  
export default Welcome;
```

Utiliser un Composant dans JSX

En JSX, un Composant React est appelé sous forme de balise XML.

Les attributs permettent d'interagir avec le paramètre d'entrée du composant.

 Attention aux nom des éléments

→ React en **PascalCase**.

→ HTML en **lowercase**.

```
const element = (  
  <div>  
    <Welcome name='Zaza' />  
    <p>Lorem ipsum...</p>  
    <GoodBye />  
  </div>  
)
```

Le paramètre d'entrée

Les composants React

Paramètre d'entrée

Chaque composant possède un paramètre d'entrée appeler « props », ce paramètre est objet **immutable** qui contient les différentes valeurs reçues.

Pour manipuler les props du composant, il est conseillé d'utiliser le destructuring.

```
const DemoParametre = function ({ arg1, arg2 }) {  
  // ...  
};
```

Lors de l'utilisation d'un composant, pour envoyer les différentes valeurs de props nécessaire au composant, il faut utiliser des attributs dans l'écriture JSX

```
<DemoParametre arg1="Hello" arg2={42} />
```


Paramètre d'entrée - Valeur par défaut

Pour définir une valeur par défaut à une des props d'un composant, il faut d'utiliser la syntaxe du destructuring et définir la valeur à utiliser à l'aide de l'opérateur « = ».

```
const DemoDefault = function ({ name = 'Della', message }) {  
  return (  
    <p>{name} dit : {message}</p>  
  );  
};
```

NB : Cette valeur sera utilisé que si la valeur de props reçue est « undefined ».

Modifier l'apparence

Les composants React

Utilisation du CSS au sein d'un composant

Pour modifier l'apparence d'un composant, il faut lui appliquer des règles CSS.

Il existe 3 méthodes pour cela :

- L'importation de fichiers CSS.
- L'ajout de module CSS dédié au composant.
- Le CSS-in-JS : Création du style sous forme d'objet Javascript.

Ce cours n'aborde pas le « CSS-in-JS ». Pour plus d'information sur son utilisation, vous pouvez consulter le site : <https://fr.reactjs.org/docs/dom-elements.html#style>

Import de fichier CSS

Pour ajouter des fichiers « css » au composants , nous allons utiliser le mécanisme d'import des modules javascript étendu au fichier « css ».

Cela permet de lier les fichiers de style directement avec le composant.

Il est possible d'utiliser tous les sélecteurs CSS.

Lors de l'exécution, les styles sont appliqués :

- Dev : une balise « style » dans le header
- Build : un fichier css qui sera lié à l'application compilé

 Attention, il est possible qu'une règle CSS en écrase une autre !

Exemple d'import de fichier CSS

stylized-title.css

```
.title {  
  color: ■ orangered;  
  font-size: 2rem;  
}
```

stylized-title.jsx

```
import './stylized-title.css';  
  
const StylizedTitle = function ({ title }) {  
  return (  
    <h1 className='title'>{title}</h1>  
  );  
};  
  
export default StylizedTitle;
```

Module CSS


Pour utiliser les modules CSS, il est nécessaire de

- Respecter la convention « **[name].module.css** » pour les noms de fichier.
- Les règles CSS doivent être composées d'un sélecteur de type classe.
- D'importer le fichier à l'aide de « `import ... from './...'` »
- D'utiliser en JSX l'attribut « `className` »

L'avantage de cette méthode est que chaque nom de classe présent dans le fichier CSS sera automatiquement modifié pour éviter des conflits.

Exemple d'import de Module CSS

stylized-title.module.css

```
.title {  
  color:  orangered;  
  font-size: 2rem;  
}
```

stylized-title.jsx

```
import style from './stylized-title.module.css';  
  
const StylizedTitle = function ({ title }) {  
  return (  
    <h1 className={style.title}>{title}</h1>  
  );  
};  
  
export default StylizedTitle;
```

Exercice • Création d'un composant

Les composants React

Exercice

- Créer un composant React qui affiche un message de bienvenue.
 - Le composant reçoit en paramètre d'entrée un « nom » et un « âge ».
 - La valeur par défaut de l'âge est de 18.
 - Le rendu du composant doit être composé de deux balises « paragraphe »
 - Mockup du rendu à obtenir

Bienvenue [Nom] sur l'application React !

Vous avez [Age] ans!



Attention aux dépendances entre les différents fichiers.

Rendu conditionnel

Les composants React

Rendu dynamique

Il est possible de générer un rendu dynamique en fonction des valeurs :

- Paramètre d'entrée « props »
 - Du « State » (Abordé plus tard)
- A l'aide de structures conditionnels
- Création d'un élément React.
 - Renvoyer un rendu en JSX.
- Directement dans le rendu JSX
- Via l'opérateur ternaire
 - Via les opérateurs logiques

```
const Conditionnal = function ({msg, visible, error}) {  
  if(error) {  
    return (  
      <div className="example error">  
        <p>Une erreur est survenue !</p>  
      </div>  
    );  
  }  
  
  return (  
    <div className="example">  
      {visible ? (  
        <p>{msg || 'Aucun message'}</p>  
      ) : (  
        <p>Element masqué</p>  
      )}  
    </div>  
  );  
}  
  
export default Conditionnal;
```

Les collections

Les composants React

Utilisation de collection dans le JSX

Pour afficher une collection d'objets JS avec React, il est nécessaire de la convertir en collection d'éléments React.

Pour cela, il est possible d'utiliser la méthode « map » des arrays Javascript.

Une collection JSX peut être utilisée directement dans le JSX, React générera automatiquement le rendu de celle-ci.

Exemple d'une liste constituée d'une balise « ul » avec des éléments « li ».

```
const SimpleList = function ({ names }) {  
  const listItems = names.map(  
    (name) => <li>{name}</li>  
  );  
  
  return (  
    <ul>{listItems}</ul>  
  );  
};  
  
export default SimpleList;
```

Clefs nécessaire

Le rendu des éléments fonctionne, mais il y a une erreur dans la console!

✖ ▶ Warning: Each child in a list should have a unique "key" prop.

React a besoin d'une clef **unique** et **stable** pour chaque élément de la liste.

Les clefs aideront React à identifier les objets qui doivent être modifiés, ajoutés, supprimés dans la liste lors des rendus.

Ajouter l'identifiant de l'objet

L'idéal est d'utiliser une valeur présente au sein des données.

Lors de la création des éléments, utiliser l'attribut « key » sur l'élément JSX.

```
const listItems = objs.map(  
  (obj) => <li key={obj.key}>{obj.data}</li>  
);
```

⚠ La valeur de la « key » de l'objet doit être unique et ne doit pas être changée.

Cas particulier : Impossible d'avoir un identifiant stable

Dans le cas où il est **impossible** d'ajouter un identifiant stable.

Il est possible **en dernier recours** d'utiliser un index avec la fonction « map ».

```
const listItems = names.map(  
  (name, index) => <li key={index}>{name}</li>  
);
```

⚠ Attention ⚠

Cette méthode peut avoir un impact négatif sur les performances !

Des effets de bord peuvent apparaître lors de l'utilisation du composant.

Liste d'éléments complexe

Il est possible de créer un composant React qui représente un élément de liste.

Pour cela, il faut créer un composant qui a comme nœud principal la balise adaptée.

L'attribut « key » ne doit pas être ajouté dans ce composant. Celui-ci sera défini lors de la génération de la liste JSX dans le composant parent.

```
const DemoItem = function ({firstname, lastname, urlImage}) {  
  
  const altImage = `Photo de ${firstname} ${lastname}`;  
  
  return (  
    <li>  
      <h3>{firstname} <span>{lastname}</span></h3>  
      <img src={urlImage} alt={altImage} />  
    </li>  
  );  
};  
  
export default DemoItem;
```

Exemple complet de liste React - L'application

```
import './App.css';
import { nanoid } from 'nanoid'
import ElementList from './components/demo-liste/element-list';

const App = function () {
  // Demo avec une liste pré-définie
  const list = [
    {id: nanoid(), firstname: 'Riri', lastname: 'Duck'},
    {id: nanoid(), firstname: 'Balthazar', lastname: 'Picsou'},
    {id: nanoid(), firstname: 'Zaza', lastname: 'Vanderquack'}
  ];
  return (
    <div className="App">
      <h1>Demo d'utilisation de liste</h1>
      <ElementList list={list} />
    </div>
  );
}

export default App;
```

Exemple complet de liste React - La liste

```
const ElementListItem = function ({ firstname, lastname }) {  
  return (  
    <li>  
      <h2>{firstname} {lastname}</h2>  
    </li>  
  );  
};  
  
const ElementList = function ({ list = [] }) {  
  const listItems = list.map(  
    (elem) => <ElementListItem key={elem.id} {...elem} />  
  );  
  return (  
    <ul>  
      {listItems}  
    </ul>  
  );  
};  
  
export default ElementList;
```

Exercice • Création d'une liste

Les composants React

Exercice

- Créer une liste de produit dans le projet (une array dans l'app ou un fichier JSON)
 - Un produit possède les propriétés suivantes :
 - Un nom
 - Une description (Optionnel)
 - Un prix
 - Un booléen « En promo »
- Créer un composant qui affiche la liste des différents produits
- Afficher les prix en Euro dans le format suivant : « 1 234,79 € »
- (BONUS) Mettre le prix des produits en promotion en rouge.

Les événements du DOM

Les composants React

Les événements du DOM

Les balise JSX qui représente des balises de l'HTML possèdent les mêmes événements que ceux du langage HTML.

Par contre, la syntaxe des événements en JSX sont en **camelCase**.

Pour utiliser des événements dans nos composants, il faut transmettre la fonction JavaScript dans le code JSX.

```
const SimpleEvent = function() {  
  
  const handleAction = function() {  
    // Code exécuté lors de l'événement  
    console.log('Action déclenchée');  
  }  
  
  return (  
    <div>  
      <button onClick={handleAction}>  
        Cliquer ici  
      </button>  
    </div>  
  );  
}  
  
export default SimpleEvent;
```

Variable de l'événement

Pour manipuler la variable d'événement, la méthode à exécuter doit attendre un paramètre.

L'envoi de la variable d'événement vers la méthode est implicite dans le JSX.

Pour désactiver le comportement par défaut de l'événement, il est nécessaire de déclencher la méthode « `preventDefault()` » sur la variable.

```
const VariableEvent = function() {  
  
  const handleEvent = function(event) {  
    const value = event.target.value;  
    console.log(`La valeur est ${value}`);  
  }  
  
  return (  
    <div>  
      Entrer une valeur  
      <input type="text" onChange={handleEvent} />  
    </div>  
  );  
}  
  
export default VariableEvent;
```


Les composants React avancé

Le mécanisme des Hooks

Les composants React avancé

Raisons d'être des Hooks

Les Hooks sont disponible depuis la version 16.8.0 de React.

Voici quelques raisons qui ont données naissance au mécanisme des Hooks :

- Utiliser davantage de fonctionnalités de React sans recourir aux classes.
 - Permet d'utiliser un état local, d'avoir un cycle de vie.
- Extraire la logique d'un composant.
 - Permet de réutiliser une logique sans devoir modifier la hiérarchie de l'application.
- Simplifier les composants en les découpant en petites fonctions
 - Permet d'isoler les codes intrinsèquement liées (abonnement / désabonnement)

Règles d'utilisation des Hooks

- Appelez les Hooks uniquement depuis des fonctions React

Il faut toujours utiliser les hooks depuis :

- Des fonctions composants React
- Des Hooks personnalisés

- Appelez les Hooks uniquement au niveau racine

Il ne faut jamais utiliser des Hooks à l'intérieur de code conditionnel, de code itératif ou de fonctions imbriquées !

Manipulation du state

Les composants React avancé

Le State... C'est quoi?

Un composant peut posséder différentes variables qui sont nécessaires lors du rendu. L'ensemble de ses variables est appelé le « state » (ou « état local » en français)

Lorsqu'une valeur du state est modifiée dans un composant, l'application React régénère le DOM virtuel et actualise la page Web.

Avant les Hooks, le « state » n'était disponible que sur les composants de type classe.

Il faut uniquement stocker dans le State ce qui est nécessaire à la génération du rendu.



Stocker des variables "Business" provoquera des rendus inutiles dans l'application.

Le Hook d'état

Les composants React avancé

Le Hook d'état

Le Hook d'état permet de manipuler une variable de « State » dans un composant. Il permet d'obtenir la valeur actuelle du State et d'interagir avec à l'aide d'une méthode.

Pour utiliser ce Hook, il faut appeler la méthode « useState » dans notre composant.

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

const DemoStateHook = function() {
  const [value, setValue] = useState(initialValue);
  //...
}

export default DemoStateHook;
```


Le Hook d'état

- Le Hook « useState » a un argument (optionnel) pour la valeur initiale de l'état.
 - Celle-ci peut être de tout type : Nombre, Chaîne de caractère,...
- Le Hook « useState » renvoie une paire de valeurs sous forme de tableau.
 - La première est la valeur actuelle de l'état.
 - La seconde est une fonction qui permet de modifier la valeur de l'état.
- Lorsqu'un composant nécessite l'utilisation de plusieurs valeurs d'état, il est conseillé de créer des hooks d'état différents pour les stocker.

 L'utilisation d'objet est conseillée uniquement pour un ensemble de valeurs liées.

Le Hook d'état - Modifier l'état

La méthode « set...() » générée permet de modifier la valeur stockée du Hook.

Le paramètre de la méthode peut être :

- Une nouvelle valeur à affecter.
- Une valeur calculée sur base de la précédente valeur du Hook.



La valeur de l'état sera disponible au prochain rendu du composant.

```
import {useState} from 'react'

const DemoStateHook = () => {
  const [value, setValue] = useState(1);

  const onMulti2 = () => {
    setValue(prevValue => prevValue * 2);
  }

  const onReset = () => {
    setValue(1);
  }

  return (
    <div>
      <h1>La valeur est {value}</h1>
      <button onClick={onMulti2}>x2</button>
      <button onClick={onReset}>Reset</button>
    </div>
  );
}

export default DemoStateHook;
```

Exercice • Utilisation du State

Les composants React avancé

Exercice

- Créer un composant « Compteur »
 - Il se compose d'une balise paragraphe et de 2 buttons.
 - La valeur initiale du compteur est de 0.
 - Le premier bouton permet d'incrémenter la valeur.
 - La valeur d'incrémentation est définie via les props. (Par défaut, valeur de 1).
 - Le deuxième bouton permet de remettre à zéro la valeur.
 - Ce bouton n'apparaît pas tant que la valeur est à 0.
- Afficher le composant « Compteur » dans le composant « App ».

Les Formulaires

Les composants React avancé

Les balises de formulaire

Pour interagir avec la saisie utilisateur en React, il faut utiliser les balises de formulaire

- `<form> ... </form>`
- `<input ... />`
- `<textarea ... />`
- `<select> ... </select>`
- `<label> ... </label>`

Les balises de formulaire - HTML vs JSX

Quelques différences entre les balises formulaire HTML et JSX :

- L'événement « onChange » se déclenche dès qu'il y a un changement de valeur.
- La balise « textarea » utilise l'attribut "value" et s'écrit en mono-ligne.
- La balise « select » possède un attribut "value" pour sélectionner la valeur utiliser, cela permet d'éviter l'utilisation de l'attribut "selected" sur une des options.
- L'attribut "for" de la balise « label » est remplacé par "htmlFor".

Les balises de formulaire - Source de données

Un problème se pose avec les balises de formulaire :

- Les balises de formulaire possèdent leur propre valeur interne, qui est modifiée lorsque l'utilisateur interagit avec celle-ci.
- Un composant possède un « State » qui représente les valeurs de l'élément.

 Il y a donc 2 sources de données possibles !

Composants contrôlés

Pour résoudre ce problème, il faut synchroniser la valeur interne des balises avec les valeurs du « State », pour obtenir une seule source de vérité.

Comment réaliser cela :

- Utiliser l'événement « onChange » de la balise des formulaires pour modifier l'état d'une variable du « State » lors de la saisie de l'utilisateur.
- Définir la valeur de la balise de formulaire comme étant égale à la variable contenu dans le « State »

Exemple de composant contrôlé - Input

- La balise « Input » de type « text »

```
import { useState } from 'react';

const ExempleInput = function () {
  const [msg, setMsg] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    // Do something
  }

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor='msg'>Entrer un message</label>
      <input id='msg' type='text' value={msg} onChange={(e) => setMsg(e.target.value)} />
      <input type='submit' value='Envoyer' />
    </form>
  );
}

export default ExempleInput
```

Exemple de composant contrôlé - Textarea

- La balise « Textarea »

```
import { useState } from 'react';

const ExempleTextarea = function () {
  const [msg, setMsg] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    // Do something
  }

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor='msg'>Entrer un message</label>
      <textarea id='msg' value={msg} onChange={(e) => setMsg(e.target.value)} />
      <input type='submit' value='Envoyer' />
    </form>
  );
}

export default ExempleTextarea
```

Exemple de composant contrôlé - Select

- La balise « Select »

```
import { useState } from 'react';

const ExempleSelect = function () {
  const [choice, setChoice] = useState('firefox');
  const handleSubmit = (e) => {
    e.preventDefault();
    // Do something
  }

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor='nav'>Quel est votre navigateur</label>
      <select id='nav' value={choice} onChange={(e) => setChoice(e.target.value)}>
        <option value="chrome">Google Chrome</option>
        <option value="firefox">Mozilla Firefox</option>
        <option value="other">Autre</option>
      </select>
      <input type='submit' value='Envoyer' />
    </form>
  );
}

export default ExempleSelect
```

Exemple de composant contrôlé - Checkbox

- La balise « Input » de type « checkbox »

```
import { useState } from 'react';

const ExempleCheckbox = function () {
  const [isChecked, setCheck] = useState(false);

  const handleSubmit = (e) => {
    e.preventDefault();
    // Do something
  }

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor='case'>Case à cocher</label>
      <input id='case' type='checkbox' checked={isChecked} onChange={(e) => setCheck(e.target.checked)} />

      <input type='submit' value='Envoyer' />
    </form>
  );
}

export default ExempleCheckbox
```

Formulaire complexe

Pour les formulaires complexe, il y a deux approche pour réaliser la synchronisation :

- Utiliser un « state » pour chaque valeurs du formulaire
 - Utiliser un hook d'état pour créer une valeur pour chacun des éléments.
 - Synchroniser chaque balise individuellement.
- Utiliser un « state global » pour stocker toutes les valeurs du formulaire
 - Utiliser un hook d'état pour créer une valeur de type objet.
 - Ajouter une propriété dans l'objet pour chaque élément.
 - Ajouter un nom (*identique à la propriété du state*) dans chaque balise du formulaire.
 - Créer une méthode qui synchronise la balise et la valeur de state sur base du nom.

Utiliser la méthode la plus adéquate par rapport au formulaire de votre composant.

Formulaire complexe - Plusieurs hook d'état

```
import { useState } from 'react';

const MultiField = function () {
  const [name, setName] = useState('');
  const [nbGuest, setNbGuest] = useState('1');
  const [isPresent, setPresent] = useState(false);

  const handleSubmit = (e) => {
    e.preventDefault();
    // Do something
  }

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor='name'> Nom :</label>
      <input id="name" type="text" value={name} onChange={(e) => setName(e.target.value)} />

      <label htmlFor='nbGuest'> Nombre de personne :</label>
      <input id="nbGuest" type="number" min={1} value={nbGuest} onChange={(e) => setNbGuest(e.target.value)} />

      <label htmlFor='isPresent'> Soirée uniquement :</label>
      <input id="isPresent" type="checkbox" checked={isPresent} onChange={(e) => setPresent(e.target.checked)} />

      <input type="submit" value="Valider" />
    </form>
  );
}

export default MultiField
```

Formulaire complexe - Hook d'état global

```
import { useState } from 'react';
const MultiField = function () {
  const [inputs, setInputs] = useState({ name: '', nbGuest: '1', isPresent: false });

  const handleSubmit = (e) => {
    e.preventDefault();
    // Do something
  }

  const handleInput = (e) => {
    const { name, type, checked, value } = e.target;
    setInputs({
      ...inputs,
      [name]: type === 'checkbox' ? checked : value
    });
  }

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor='name'> Nom :</label>
      <input id="name" name="name" type="text" value={inputs.name} onChange={handleInput} />
      <label htmlFor='nbGuest'> Nombre de personne :</label>
      <input id="nbGuest" name="nbGuest" type="number" min={1} value={inputs.nbGuest} onChange={handleInput} />
      <label htmlFor='isPresent'> Soirée uniquement :</label>
      <input id="isPresent" name="isPresent" type="checkbox" checked={inputs.isPresent} onChange={handleInput} />
      <input type="submit" value="Valider" />
    </form>
  );
}

export default MultiField
```


Valeur en lecture seul

Pour créer des balises en lecture seul, il faut utiliser l'attribut « readOnly ».

```
import { useState } from 'react';

const ExempleInput = function () {
  const [number, setNumber] = useState('0');
  const [result, setResult] = useState('0');

  const handleNumberInput = (e) => {
    const nb = e.target.value;
    setNumber(nb);
    setResult(parseInt(nb) * 2);
  }

  return (
    <form>
      <label htmlFor='number'>Entrer un nombre</label>
      <input id='number' type='number' value={number} onChange={handleNumberInput} />

      <label htmlFor='result'>Le double du nombre est</label>
      <input id='result' type='number' value={result} readOnly />
    </form>
  );
}

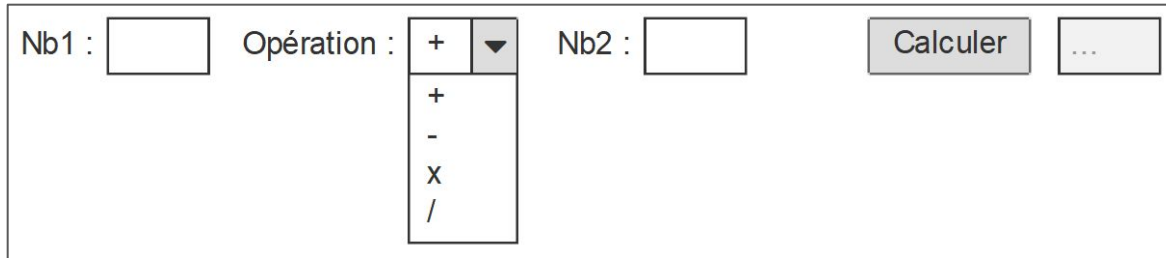
export default ExempleInput
```

Exercice • Manipulation des formulaires

Les composants React avancé

Exercice

- Créer un composant « Calculatrice »
 - Il se compose de :
 - Deux champs de type "texte", où la saisie est limitée à des valeurs numériques.
 - Un sélecteur pour l'opération
 - Un bouton de validation
 - Un champs texte en lecture qui affichera le résultat de l'opération
 - Mockup du rendu à obtenir



The mockup shows a calculator interface within a rectangular frame. On the left, the label "Nb1 :" is followed by a text input field. To its right is the label "Opération :" followed by a dropdown menu. The dropdown menu is open, showing a list of arithmetic operators: "+", "-", "x", and "/". To the right of the dropdown is the label "Nb2 :" followed by another text input field. Further right is a button labeled "Calculer". To the right of the "Calculer" button is a text input field containing three dots "...".

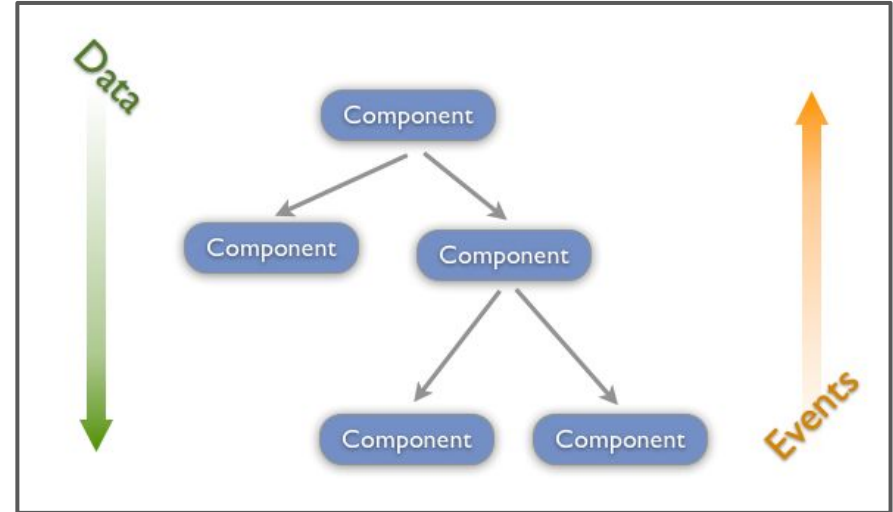
Interaction entre les composants React

Les composants React avancé

Communication entre composant

Pour rendre possible la communication entre composants, il est possible de :

- Envoyer des données au composant pour **descendre** dans l'arborescence.
- Utiliser un callback sur le composant (*mécanisme d'événement*) pour **monter** dans l'arborescence.



Ce chapitre n'aborde pas les concepts avancés tels que : Redux, le Context, ...

Depuis le Parent vers l'Enfant

L'envoi de données vers un composant enfant est réalisable en utilisant uniquement le système de « props » de React.

- Dans le parent : En JSX, utiliser les attributs du composant.

```
<Demo data={donneeEnvoyee} />
```

- Dans l'enfant : Récupérer les données via les « props » du composant.

```
const Demo = function ({ data }) {  
  return <p>Les données reçues sont : {data}</p>;  
};
```

Depuis l'Enfant vers le Parent

L'envoi d'événement (avec ou sans données) vers un composant parent est possible en définissant un callback dans les « props » du composant enfant.

- Dans le parent : En JSX, ajouter un callback sur l'attribut utilisé en tant qu'event.

```
<Demo onResult={handleFunction} />
```

- Dans l'enfant : Déclencher le "callback" depuis les « props » du composant.

```
onResult(donneeEnvoye)
```

Pour éviter une erreur quand un événement est déclenché et qu'aucun callback n'a été défini pour cet événement lors de l'utilisation du composant. Il est possible d'ajouter une fonction vide (NOOP) en tant que valeur par défaut via les props.

Exemple

avec un événement

```
const ReponseEvent = function ({ question, onReponse = () => { } }) {  
  
  const sendResponse = () => {  
    onReponse(42);  
  };  
  
  return (  
    <div>  
      <label htmlFor="event-demo">{question}</label>  
      <button id="event-demo" onClick={sendResponse}>Envoyer une réponse...</button>  
    </div>  
  );  
};  
  
export default ReponseEvent;
```


Entre deux composants frères

Pour permettre une communication bidirectionnelle entre 2 composants frères, il est possible d'utiliser le composant parent en tant qu'intermédiaire.

Cette stratégie consiste à réaliser une liaison "Enfant > Parent" et "Parent > Enfant" vers chaque composant, et leur permettre de communiquer à l'aide de fonctions.

```
<div>
  <SiblingA
    data={valueStateA}
    onEvent={eventSiblingA} />
  <SiblingB
    data={valueStateB}
    onEvent={eventSiblingB} />
</div>
```

Exercice • Interaction entre composant

Les composants React avancé

Exercice

1. Créer une application « Todo list » constitué des éléments suivants :
 - Un formulaire pour ajouter une nouvelle tâche à effectuer.
 - Une liste de tâches, qui permet les actions suivantes :
 - Terminer une tâche.
 - Supprimer une tâche
2. Une tâche contient les informations suivantes :
 - Un nom (*Obligatoire*) et une description (*Optionnel*)
 - Une priorité (*Basse / Normal / Urgent*)
 - Une complétion (*Une valeur booléen*)

Exercice

3. Ajouter une validation au formulaire
 - Champs manquants en rouge.
4. Ajouter de la couleur aux tâches
 - Urgente en rouge
 - Terminée en gris (Rayure diagonal)
5. (BONUS) Ajouter des filtres à la liste
 - En cours
 - Urgentes
 - Terminées

Ajouter une nouvelle tâche

Nom

Description

Priorité

Normal ▼

Ajouter

Liste des tâches

Acheter du café (Urgent)

Terminer

Supprimer

Réaliser l'exercice
Créer l'application « Todo List »

Terminer

Supprimer

Tâche terminée
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Maecenas scelerisque nisi sed.

Terminer

Supprimer

Cycle de vie des composants

Les composants React avancé

But du cycle de vie d'un composant

C'est un mécanisme qui permet d'exécuter du code lors d'une actualisation du rendu.

Elles permettent, par exemple de :

- Activer / désactiver les abonnements (méthode `setInterval`, requête Ajax,...)
- Éviter les erreurs provoqués par la modification du state d'un élément inexistant.

Exemple d'erreur lors du déclenchement d'un « `setState` » d'un composant retiré du DOM.

```
Horloge affiché
5 tick
Horloge masqué
✖ ▶ Warning: Can't perform a React state update on an unmounted component. This is
  a no-op, but it indicates a memory leak in your application. To fix, cancel all
  subscriptions and asynchronous tasks in the componentWillUnmount method.
    at Horloge (http://localhost:3000/static/js/main.chunk.js:2574:5)
7 tick
```

Le Hook d'effet

Les composants React avancé

Le Hook d'effet

Le Hook d'effet permet au composant React d'exécuter un bloc de code lors de l'ajout, la mise à jour et le retrait d'un composant du DOM.

Il est conseillé de créer un Hook d'effet pour chaque effet de bord du composant.

Il existe deux grands types d'effets de bord dans les composants React :

- Ceux qui ne nécessitent pas de traitement après leurs exécutions.
- Ceux qui nécessitent d'un nettoyage.

Le Hook d'effet - Effets sans nettoyage

Le Hook d'effet sans nettoyage permet d'exécuter du code uniquement lors de l'ajout et la mise à jour du composant.

Celui-ci peut être utilisé par exemple pour effectuer :

- Une requête réseau
- Des modifications du DOM

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

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

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

  return (
    <div>
      <button onClick={() => setCount(c => c + 1)}>
        Compteur : {count}
      </button>
    </div>
  );
}

export default DemoEventHook;
```

Le Hook d'effet - Effets avec nettoyage

Le Hook d'effet avec nettoyage permet d'ajouter du code à exécuter lors du retrait du composant.

Pour cela, il faut renseigner la fonction de nettoyage dans le retour du hook.

Le nettoyage permet de :

- D'annuler une/des requête(s)
- Retirer un abonnement

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

const DemoEventHookCleanup = function() {

  useEffect(() => {
    console.log("Lancement de l'effet");

    return () => {
      console.log("Nettoyage de l'effet");
    }
  });

  return (
    <div>Demo d'un effet avec nettoyage</div>
  );
}

export default DemoEventHookCleanup;
```

Le Hook d'effet - Optimiser les performances

Il est possible d'indiquer à React de sauter l'exécution d'un effet.

Pour cela, il faut donner une liste en deuxième argument du Hook d'effet.

L'effet sera déclenché, uniquement si au moins une des valeurs du tableau a changé.

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

Ce mécanisme permet également de déclencher un effet uniquement lors du premier rendu. Pour cela, il suffit de lui donner un tableau vide « [] » comme argument.

Exercice • Cycle de vie des composants

Les composants React avancé

Exercice

- Créer un composant « Horloge »
 - Une balise "p" affiche le temps du chrono au format : « **hh : mm : ss** ».
 - L'horloge s'actualise automatiquement toutes les 200 millisecondes.
- Créer un composant « DateDuJour »
 - Une balise "p" affiche la date du jour en français au format : « **dd mois yyyy** ».
- Ajouter les 2 composants dans l'application
 - Permettre d'alterner entre l'affichage des 2 composants, à l'aide d'un bouton.

Les requêtes AJAX

Réaliser des requêtes en React

Les requêtes AJAX

Comment faire un appel AJAX ?

React n'impose pas de librairie pour réaliser des requêtes Ajax.

Vous pouvez utiliser, au choix :

- L'objet XMLHttpRequest
- L'API Fetch
- La bibliothèque Axios
- La bibliothèque Superagent
- Etc...

<https://axios-http.com/docs/intro>

<https://visionmedia.github.io/superagent/>

Comment faire un appel AJAX ?

Il y a deux cas d'utilisation typique d'appel ajax :

- Un composant présent dans le DOM qui réalise une requête.
 - La requête est appelée suite un événement ou une interaction de l'utilisateur.
- Un composant dédié à l'exécution d'une requête et l'affichage de son résultat.
 - Tant qu'il n'y a pas d'appel à réaliser, le composant n'est pas présent dans le DOM.
 - Lorsque le composant est instancié, la requête Ajax est envoyée.
 - La requête doit toujours être réalisée dans un Hook d'effet.

Une fois le résultat de la requête obtenu, le composant met à jour les valeurs de "state". Cela provoquera l'actualisation du composant, pour afficher les données reçues.

Exemple d'une requête API dans un composant dédié

Consommation de l'API « agify.io » (*Prédiction de l'âge d'un nom*) avec la librairie Axios.

```
useEffect(() => {  
  const url = `https://api.agify.io/?name=${name}&country_id=BE`;   
  
  setLoading(true);  
  setError(null);  
  setData(null);  
  
  axios.get(url)  
    .then(({ data }) => {  
      setData({  
        name: data.name,  
        age: data.age  
      });  
    })  
    .catch(e => {  
      setError(e.error);  
    })  
    .finally(() => {  
      setLoading(false);  
    });  
}, [name]);
```

Exercice • App météo

Les requêtes AJAX

Exercice

- Créer le composant « Recherche ».
 - Contient une input de type *“text”* avec un placeholder (à définir via un argument).
 - Contient une input de type *“submit”* pour valider la recherche.
 - La valeur est renvoyée au composant parent.
- Créer le composant « Meteo » qui affiche la météo d’une région.
- Utiliser l’API « openweathermap.org » pour obtenir les données météo.
- BONUS : Permettre à l’utilisateur de sauvegarder le résultat obtenu dans une liste.

Récapitulatif des fondamentaux

Penser une application avec React

Lors de la création d'un projet, vous devez penser votre application pour React.

Dans les grandes lignes :

- Décomposer l'interface utilisateur en une hiérarchie de composants.
- Déterminer les états nécessaires à placer dans le « State » de l'application.
- Identifier où les différents états devront être définis dans la hiérarchie.
- Identifier les différents flux de données au sein de l'application.

Exercice récapitulatif

- Dans un nouveau projet créer une application de recherche de vidéo « Youtube »
 - L'utilisateur a la possibilité de réaliser une recherche à l'aide d'un input texte.
 - Les résultats de la recherche sont affichés sous forme de liste d'éléments.
 - Si aucune vidéo n'est sélectionnée, la première vidéo trouvée doit être sélectionnée.
 - L'utilisateur peut changer la vidéo sélectionnée en cliquant sur la liste.
 - La vidéo sélectionnée est affichée dans le lecteur Youtube en lecture automatique.

API Youtube « Search » : <https://developers.google.com/youtube/v3/docs/search/list>

Player Youtube en React : <https://github.com/troybetz/react-youtube>

Exercice récapitulatif - Exemple de rendu


 Youtube App

Rechercher



La bande à Picsou (générique entier)

En suivant Fifi, Riri et Loulou Nous entrerons dans la bande à Picsou !



La bande à Picsou (générique entier)
Génériques de dessins animés



Générique LA BANDE A PICSOU (HD)
Absolument Dorothée



La Bande à Picsou et l'Atlantide ! La Bande à
Disney Channel Belgique



La Bande à Picsou - Générique
Disney Série & Film



La Bande à Picsou | L'île de rêve inhabitée | Disney
Disney Channel Belgique



Disneyphile - 160 - La Bande à Picsou (2017) Saison 11
Mayo-Lek Disneyphile

Formation de React JS ♥ Made in Belgium

Annexes

Les Fragments

Annexes

Les fragments

Le rendu d'un composant React doit toujours être composé d'un seul nœud racine. Pour respecter cela, il est possible d'ajouter une balise « div » pour regrouper nos différents éléments sous un même nœud.

Cela peut provoquer des problèmes :

- Un excès de « div » dû à de multiples composants imbriqués.
- Casser la sémantique du HTML

Exemple : Une balise « div » dans la structure de table

Les fragments

Les fragments permettent de regrouper plusieurs éléments sans ajouter de nœud supplémentaire au sein du DOM.

```
import React, {Fragment} from "react";

export default function DemoFragment(props) {
  return (
    <Fragment>
      <MyComponent />
      <MyComponent />
    </Fragment>
  );
}
```

Les fragments

Il est possible d'utiliser les fragments avec la syntaxe raccourci « `<> ... </>` »

```
import React from "react";

export default function DemoFragment(props) {
  return (
    <>
      <MyComponent />
      <MyComponent />
    </>
  );
}
```

Attention, cette syntaxe n'est pas supportée par tous les IDE

Construire un Hook personnalisé

Annexes

Création d'un Hook personnalisé

Lorsque nous souhaitons rendre réutilisable une logique au sein de nos composants. Il est possible d'extraire cette logique dans un Hook personnalisé.

Règles de création d'un Hook :

- C'est une fonction JavaScript.
- Son nom doit commencer par « use ».
- Il peut appeler d'autres Hooks.

Création d'un Hook personnalisé - Exemple

Hook permettant d'obtenir une valeur booléen et d'inverser sa valeur.

```
const { useState, useCallback } = require('react');

// Définition du Hook
export const useToggle = (initialValue = false) => {

  // Initialisation de la valeur booléen
  const [isActive, setActive] = useState(initialValue);

  // Définition d'une méthode (mémoisé) pour inverser le booléen
  const toggle = useCallback(() => {
    setActive(isActive => !isActive);
  }, []);

  // Renvoi du booléen et de la méthode
  return [isActive, toggle];
}
```


Délégation de contenu

Annexes

Délégation de contenu

Lors de la définition de composants réutilisables, il est possible de les construire de manière générique sans connaître leurs contenues à l'avance.

C'est par exemple le cas pour un composant de type « DialogBox », « Sidebar », ...

Dans ce cas, le composant est utilisé sous la forme d'une balise ouvrante et fermante. Afin de pouvoir définir le contenu de celui-ci (en JSX) entre ses balises.

Le contenu est récupérable en utilisant l'attribut réservé « children » sur le paramètre d'entrée (props) du composant.

Délégation de contenu - Exemple

```
import React from 'react';
import PropTypes from 'prop-types';
import style from './fancy-border.module.css';

const FancyBorder = (props) => {
  return (
    <div className={` ${props.className} ${style.fancyBorder}`} >
      {props.children}
    </div>
  );
}

FancyBorder.defaultProps = {
  className: ''
}

FancyBorder.propTypes = {
  children: PropTypes.node.isRequired,
  className: PropTypes.string
}

export default FancyBorder;
```

Utilisation de l'objet « ref »

Annexes

Utilité de l'objet « ref »

L'objet « ref » peut être utilisé pour deux types de scénario :

- Accéder de manière impérative à un nœud du DOM ou à un composant React.

Quelques cas d'usages possible :

- Gérer le focus, la sélection du texte, ou la lecture de média.
- Lancer des animations impératives.
- S'interfacer avec des bibliothèques DOM tierces.

 **Evitez d'utiliser les refs pour tout ce qui peut être fait déclarativement.**

- Générer un objet persistant au sein d'un composant de type « fonction ».

Mise en place d'un acces impératif sur un élément

Pour commencer, il est faut de créer un objet de type « ref » dans notre composant :

- Utiliser le hook de référence à l'aide de la méthode « useRef ».

Ensuite, il faut associer la "ref" créé et l'élément React à l'aide de l'attribut « ref={...} ».


```
<input ref={inputRef} type="text" />
```

Mise en place d'un acces impératif sur un élément

La référence de l'élément devient accessible via l'attribut « current » de l'objet "ref".

Le type de référence récupéré change en fonction du type d'élément ciblé :

- Balise HTML : On obtient la référence du DOM de l'élément.
- Composant de classe : On a accès à l'instance du composant.

 Il ne faut pas utiliser l'attribut "ref" sur un composant de type « fonction », car ceux-ci ne possèdent pas d'instance !

Acces impératif - Exemple via le hook de référence

```
import React, { useRef } from 'react'

const DemoRefHook = () => {

  const inputRef = useRef();

  const handleFocus = () => {
    inputRef.current.focus();
  }

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={handleFocus}>Get Focus</button>
    </div>
  );
}

export default DemoRefHook;
```


Objet persistant à l'aide du hook de référence

Contrairement au composant classe qui possède des champs d'instance, les composants de type « fonction » n'ont pas de persistance de donnée pour stocker des valeurs business, car ceux-ci ne possèdent pas d'instance.

L'utilisation du Hook « useRef » permet de solutionner cette problématique.

L'objet renvoyé par le hook possède un attribut « current » qui est modifiable. Cette objet persistera pendant toute la durée de vie composant, cela permet donc de stocker une valeur business dans le composant.

 La modification du contenu de l'objet "ref" n'entraîne pas de rafraîchissement.

Objet persistant - Exemple d'utilisation

Dans cet exemple, une horloge qui s'actualise à l'aide d'un « setInterval ».

La valeur de l'id généré est placé dans l'objet "ref" obtenu grâce au « useRef ».

Cela permet d'arrêter l'horloge dans un événement (*en dehors du useEffect*), où l'id n'aurait pas été disponible.

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

const DemoPersistence = () => {
  const [time, setTime] = useState(new Date());
  const timerId = useRef(null);

  useEffect(() => {
    timerId.current = setInterval(() => setTime(new Date()), 500);
    return () => {
      clearInterval(timerId.current)
    }
  }, [])

  const handleStop = () => {
    clearInterval(timerId.current)
  }

  return (
    <div>
      <p>{time.toLocaleTimeString('fr-FR')}</p>
      <button onClick={handleStop}>Stop !</button>
    </div>
  )
}

export default DemoPersistence
```

Ressources

Bibliothèques utiles

Ressources

Bibliothèques de Composant UI

- **Material UI** - Composant basé sur « Material Design »
<https://mui.com>
- **React Bootstrap** - Composant basé sur « Bootstrap »
<https://react-bootstrap.github.io/>
- **Shadcn/ui** - Composant basé sur « Radix UI » et « Tailwind CSS »
<https://ui.shadcn.com/docs>
- **Styled components** - Bibliothèque pour créer des composant basé sur du CSS
<https://styled-components.com/>

Bibliothèques de Composant UI

- **Mantine** - Ensemble de composants et hooks avec support natif du dark theme
<https://mantine.dev>
- **Blueprint** - Optimisé pour des interfaces complexes à forte densité de données
<https://blueprintjs.com>
- **Chakra UI** - Bibliothèque de composants modulaire et accessible
<https://chakra-ui.com/>
- **Fluent UI** - Bibliothèque de composant créer par Microsoft
<https://react.fluentui.dev/>

Bibliothèques pour les listes et les data tables

- **React Window** - Permet la création de liste
<https://react-window.now.sh>
- **React Virtualized** - Permet la création de liste, de tableur et de liste infini
<https://bvaughn.github.io/react-virtualized>
- **React Data Grid** - Permet la création de data table
<https://github.com/adazzle/react-data-grid>

Bibliothèques pour les formulaires

- **React Hook Form** - Gestion des formulaires à l'aide de Hook
<https://react-hook-form.com>
- **React JsonSchema Form** - Génération de formulaire sur base de schema JSON
<https://react-jsonschema-form.readthedocs.io/>
- **Formik** - Gestion des formulaires simplifiés
<https://formik.org>
- **React Select** - Implémentation du select2 pour React
<https://react-select.com/home>

Autres Bibliothèques pour React

- **React Router** - Gestion de routing pour créer des applications SPA
<https://reactrouter.com>
- **SWR** - Hook pour réaliser des requête AJAX avec un mécanisme de "cache"
<https://swr.vercel.app/>
- **React Use** - Collection de Hooks essentiels
<https://github.com/streamich/react-use>
- **React Beautiful DnD** - Création d'interface « Drag and Drop »
<https://react-beautiful-dnd.netlify.app>

Autres Bibliothèques pour React

- **Sonner** - Bibliothèque pour générer des Toasts
<https://sonner.emilkowal.ski/>
- **React Hot Toast**- Bibliothèque pour générer des Toasts
<https://react-hot-toast.com/>
- **React-i18next** - Bibliothèque pour internationaliser l'application
<https://react.i18next.com/>

Bibliothèques divers pour le JavaScript

- **Pnpm** - Gestionnaire de packages (Alternative à *npm* et *yarn*)
<https://pnpm.io/>
- **Clsx** - Bibliothèque pour générer les classnames
<https://github.com/lukeed/clsx>
- **Nanoid** - Bibliothèque pour générer des identifiants uniques « URL-friendly »
<https://github.com/ai/nanoid>
- **Uuid** - Bibliothèque pour générer des identifiants « RFC4122 UUIDs »
<https://github.com/uuidjs/uuid>

Sites utiles

Ressources

Sites utiles

- **La documentation officiel**
<https://react.dev/> (Ancienne doc : <https://fr.reactjs.org>)
- **React Resources** - Ensemble d'articles et tutoriels par rapport à React.
<https://reactresources.com>
- **Absolutely Awesome React Components & Libraries**
<https://github.com/brillout/awesome-react-components>
- **Awesome React** - Collection de bibliothèque concernant l'écosystème React
<https://github.com/enaqx/awesome-react>

Merci pour votre attention.

