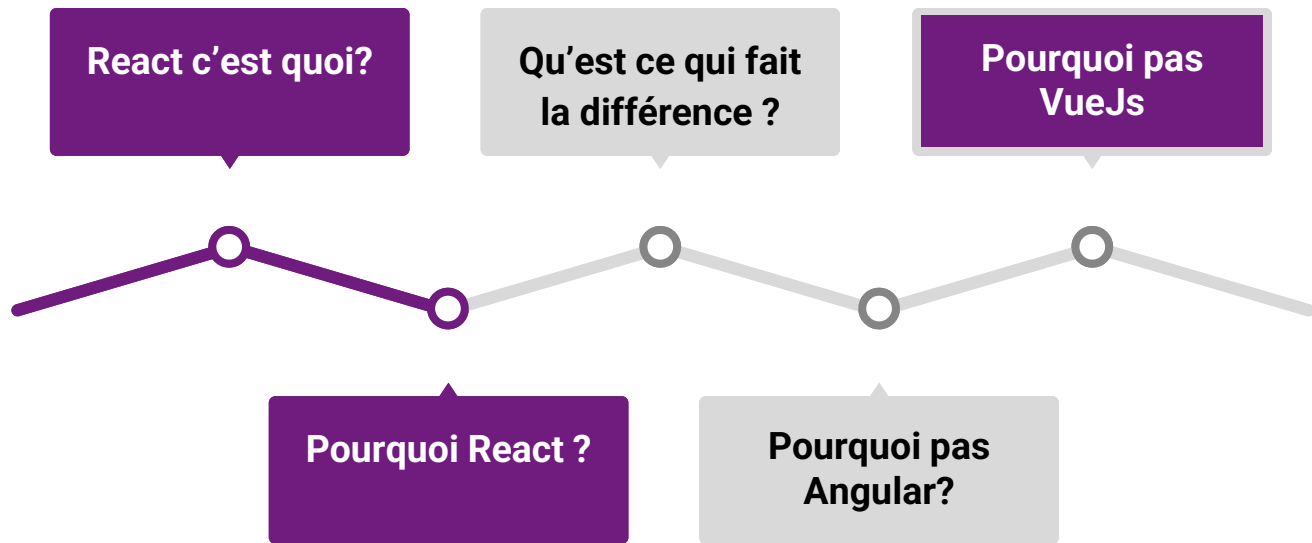


Introduction



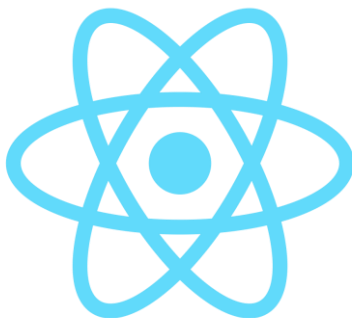
Introduction - React c'est quoi

- React ou ReactJs
- Une **bibliothèque** JavaScript qui ne gère que les interfaces utilisateurs
- **Ce n'est pas un framework!!!!!!!**
- Développée par Facebook depuis 2013.

A base de composants

Déclaratif

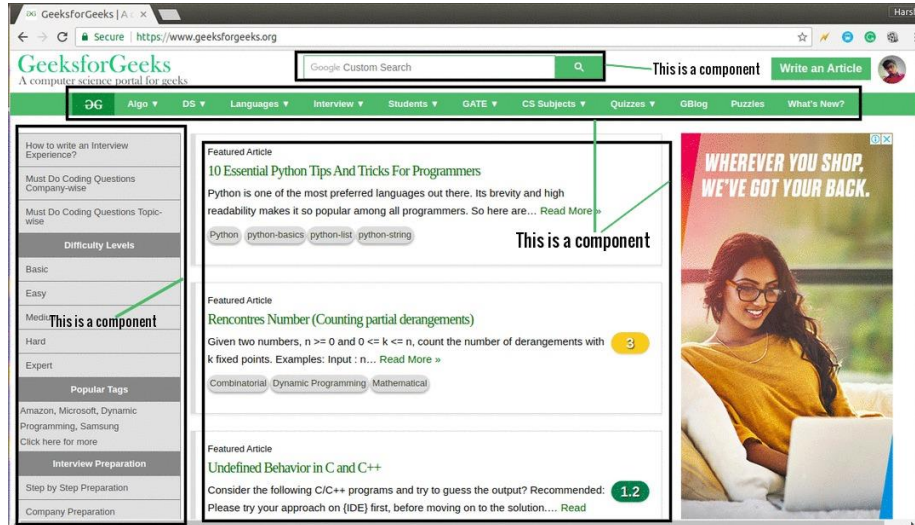
Virtual Dom



One way data binding

Introduction - React c'est quoi

- Connue pour créer des “single page application”
- React permet de composer des UI complexes à partir de petits morceaux de code isolés appelés « **composants** »



Introduction - React c'est quoi

- Utilisée par : Facebook, Twitter, Whatsapp, Instagram Paypal, Netflix, Airbnb



Introduction - Pourquoi React

- **Seuls pré-requis** : html, css, javascript et un peu de commandes npm
- Courbe **d'apprentissage** assez **rapide**



Introduction - Pourquoi pas Angular

- C'est plus libre : yessssssssss!!!!



Introduction - Pourquoi pas Angular => Liberté

- React n'est pas capricieux...
- Il n'impose pas non plus de bibliothèque spécifique pour la data, il n'est que la partie « Vue » de l'interface
- Il ne fonctionne qu'avec du **JavaScript** grâce à sa syntaxe JSX
- Avec React le développeur est libre de choisir la structure de son application
- Avec React le développeur a la liberté de choisir les outils de programmation de son choix
- Si vous embauchez les meilleurs développeurs de logiciels expérimentés, ils profiteront beaucoup de cette flexibilité

Elle n'est pas belle la vie ?



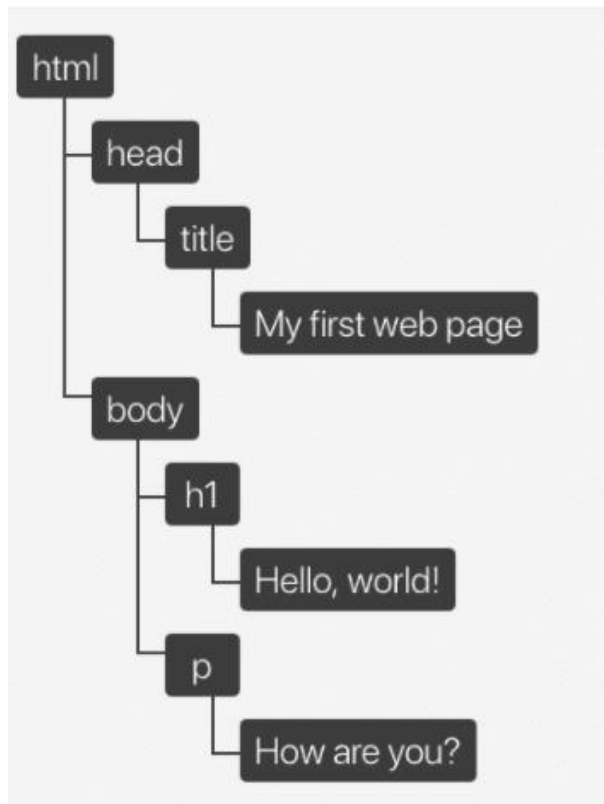
Introduction - Pourquoi pas Angular => Performance

- React est plus performant grace au DOM virtuel

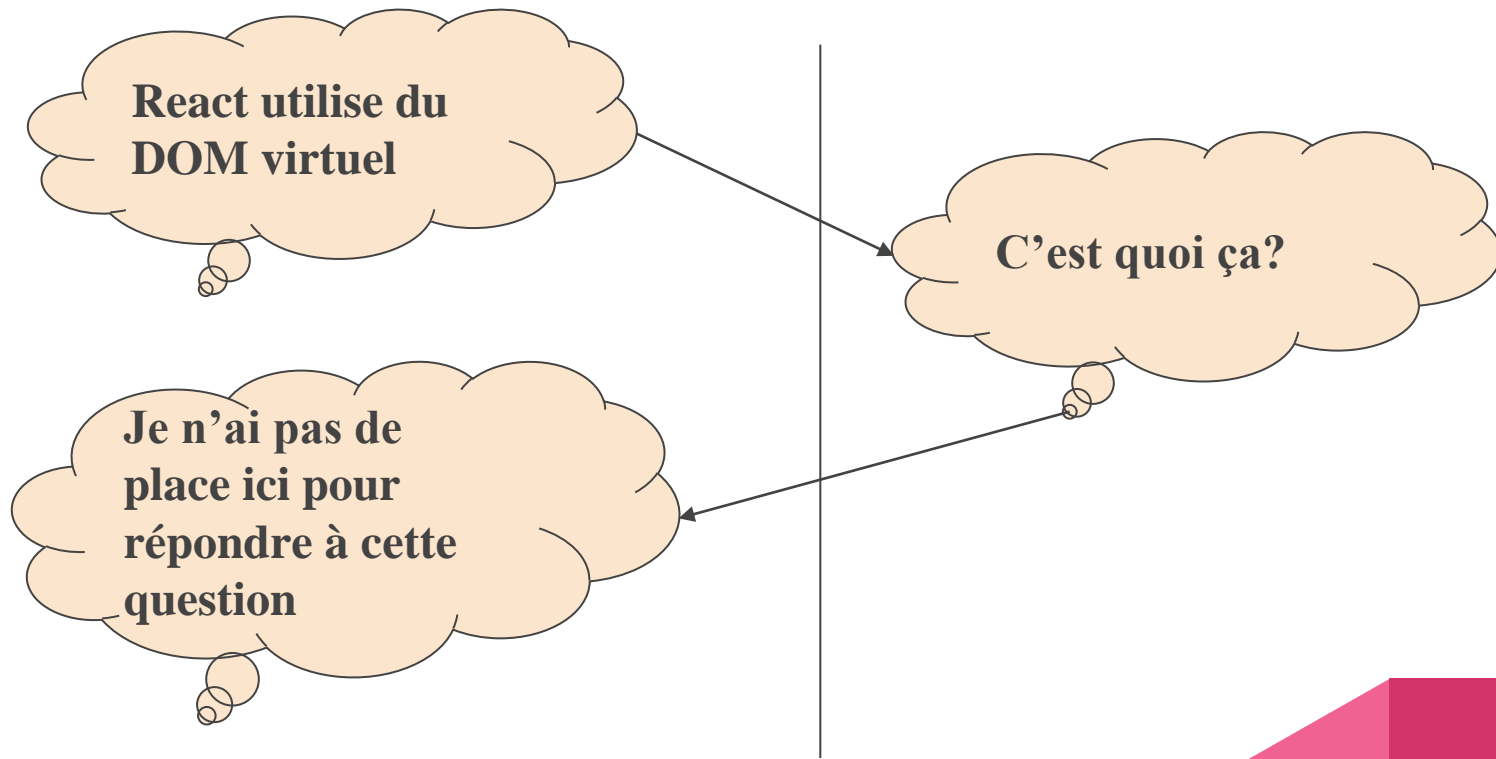


Introduction - Pourquoi pas Angular => Performance


- **C'est quoi le DOM** : c'est une représentation du document html sous forme d'une arborescence de noeud



Introduction - Pourquoi pas Angular => Performance



Introduction - Pourquoi pas Angular => Performance

- Comme les navigateurs sont plutôt lents à réagir face aux changements du DOM, React a l'avantage (et l'intelligence) de limiter les interactions avec ce dernier.
 - Il effectue donc les opérations sur le DOM virtuel et le compare au vrai DOM pour réaliser les changements à effectuer.
 - Le DOM virtuel n'est qu'une arborescence d'objets JS, qui permet d'identifier rapidement les nœuds à actualiser.
 - On réduit donc les dialogues avec les API des navigateurs pour construire le DOM, et gagne en performance.
- 

Introduction - Pourquoi pas VueJs

React et Vue ont beaucoup en commun. Tous les deux :

- utilisent un DOM virtuel,
- fournissent des composants de vue réactifs et composables,
- restent concentrés sur le cœur de la bibliothèque, en déléguant le routage et la gestion d'état à des bibliothèques connexes.

Mais, mais mais

- React brille par rapport à Vue, par exemple dans la richesse de son écosystème et l'abondance de ses moteurs de rendu personnalisés.



Introduction : aperçu sur React

Prenons un exemple fait avec du html et
css et transformons le en React

Add a task

Learn Html

delete update

Learn Html

delete update

Learn Html

delete update

Introduction : aperçu sur React

```
<div class="tasks-list">
  <div class="task-form">
    <input type="text" name="task" id="" />
    <button class="button">Add a task</button>
  </div>
  <div class="task">
    <div class="title">Learn html</div>
    <div class="actions">
      <span>delete</span>
      <span>update</span>
    </div>
  </div>
  <div class="task">
    <div class="title">Learn html</div>
    <div class="actions">
      <span>delete</span>
      <span>update</span>
    </div>
  </div>
  <div class="task">
    <div class="title">Learn html</div>
    <div class="actions">
      <span>delete</span>
      <span>update</span>
    </div>
  </div>
</div>
```

Transformation du
code html/css en
code react

```
function App() {
  return (
    <div className="tasks-list">
      <TaskForm />
      <Task />
      <Task />
      <Task />
    </div>
  )
}

function TaskForm() {
  return (
    <div className="task-form">
      <input type="text" name="task" id="" />
      <button className="button">Add a task</button>
    </div>
  )
}

function Task() {
  return (
    <div className="task">
      <div className="title">Learn html</div>
      <div className="actions">
        <span>delete</span>
        <span>update</span>
      </div>
    </div>
  )
}
```

Introduction - Qu'avez vous besoin pour commencer

1 Installer un éditeur : VSCode

2 Installer Node Js pour utiliser npm

3 INTERNET !!!!!!!

Rappel sur java script et ES6

- Les structures de contrôles et les boucles
- Les types et leur manipulation
- export/import
- Destructuring
- L'opérateur spread



Rappel sur java script - if ...else et les boucles

- if ... else
- for
- do...while
- while
- for...in ⇒ pour itérer sur les propriétés d'un objet
- for...of => pour parcourir un tableau

```
const letters = ['a', 'b', 'c']  
for (const letter of letters) {  
  console.log(letter)  
}
```

Rappel sur java script - les types

Il existe 6 types primitifs en Java script

- String
- Number
- Null (no value)
- Undefined (une variable déclarée mais sans aucune initialisation)
- Boolean
- Symbol

Le reste est objet : objet classique, function, Array, Date

- Object
- 

Rappel sur java script - les types - String

```
let lastName="Smith"  
let firstName='Will'
```

⇒ Vous pouvez initialiser les string en utilisant les " " ou les ' '



Si vous utilisez un module de **linting**, il va formater le code en utiliser soit l'un soit l'autre

Rappel sur java script - les types - String

- Template literals

```
let a = 5  
let b = 10  
let c=a+b  
console.log("La somme de " + a + " et " +b + " est: " +c)
```



```
console.log(`La somme de ${a} et ${b} est: ${c}`)
```

Rappel sur java script - les types - String

- Il y a beaucoup de propriétés et de méthodes sur la manipulation de chaînes de caractères

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/String

- Exemples

- `length`
- `includes()`
- `indexOf()`
- `slice()`
- `split()`
- `substring()`
- `toLowerCase()` / `toUpperCase()`



Rappel sur java script - les types - Number

Fonctions sur les nombres

- `Number()`
- `Number.isNaN()`
- `Number.parseInt()`
- `Number.parseFloat()`



Rappel sur java script - les types - Boolean

- true/false
- Chaque chose sans valeur est évaluée à **false**

0 / "" / undefined / null

- Chaque chose avec valeur est évaluée à **true**

100 / 3.14 / -15 / "Hello" / "false" / 7 + 1 + 3.14



Rappel sur java script - les types - Boolean

```
> 0 == false;
```

```
< true
```

```
> 0 === false;
```

```
< false
```

```
> 5 == '5';
```

```
< true
```

```
> 5 === '5';
```

```
< false
```

```
> 0 == ''; // Both are falsy
```

```
< true
```


```
> 0 === '';
```

```
< false
```


Rappel sur java script - les types - Boolean

Attention!!! **condition1 && condition2 && ... && condN** : ne retourne pas forcément true ou false

⇒ **Voici le processus**

- Evaluer les conditions de gauche à droite
 - Pour chaque condition, si son type original n'est pas boolean le convertir en booléen. Si le résultat est **false**, retourner la valeur originale de cette condition avant conversion
 - Si toutes les conditions sont évaluées à **true** , retourner la valeur originale de la dernière condition
- 

Rappel sur java script - les types - Boolean

Testez ce code

```
console.log(true && true)
console.log(true && false)
console.log(true && 0)
console.log(true && "")
console.log(true && "aaa")
```

Rappel sur java script - les types - Boolean

Attention!!! **condition1 || condition2 || ... || condN** : ne retourne pas forcément true ou false

⇒ **Voici le processus**

- Evaluer les conditions de gauche à droite
- Pour chaque condition, si son type original n'est pas boolean le convertir en booléen. Si le résultat est **true**, retourner la valeur originale de cette condition avant conversion
- Si toutes les conditions sont évaluées à **false** , retourner la valeur originale de la dernière condition



Rappel sur java script - les types - Boolean

Testez ce code

```
console.log(true || false)
console.log(true || true)
console.log(false || 0)
console.log(false || "")
console.log(false || "aaa")
```

Rappel sur java script - les types - Object

- let person={} => Objet vide

```
let person = {  
  lastName: "Smith",  
  age: 32,  
  sports: ['tennis', 'soccer'],  
  salary() {  
    console.log('My salary is high')  
  },  
  hello : function () {  
    console.log("Hello!!!")  
  }  
}
```

Rappel sur java script - les types - Object

- Essayez ces instructions dans la console. N'oubliez de créer l'objet person avant
 - `console.log(person.lastName)`
 - `person.salary()`
 - `person.firstName="Will"`
 - `console.log(person)`
 - `delete person.age`



Rappel sur java script - les types - Function

Syntaxe

// syntaxe classique

```
function myFunction (param1, param2){  
  ...  
}
```

// arrow function (ES6)

```
const myFunction= (param1, param2)=>{  
  ...  
}
```

// fonction anonyme

```
function(param1, param2){  
  ....  
}
```

Rappel sur java script - les types - Function

- les arrow function sont utilisées depuis ES6

// arrow function (ES6)

```
const myFunction= (param1, param2)=>{  
  ...  
}
```

// arrow function (ES6)

```
const myFunction= ()=>{  
  ...  
}
```

// arrow function (ES6)

```
const myFunction= (param1)=>{  
  ...  
}
```

// arrow function (ES6)

```
const myFunction= param1=>{  
  ...  
}
```


Rappel sur java script - les types - Function

- Les fonctions anonymes : sont généralement utilisées comme des callbacks

```
// fonction anonyme  
[1,2,3].forEach(function(element){  
    console.log(element)  
})
```

```
// fonction anonyme  
[1,2,3].forEach((element)=>{  
    console.log(element)  
})
```

Rappel sur java script - les types - Function

Important

- Une fonction peut contenir une déclaration d'une autre fonction

```
function displaySum(a, b) {  
  const display = value => {  
    console.log(value)  
  }  
  const sum = a + b  
  display(sum)  
  return sum  
}
```

- Est-ce que la fonction display est connue à l'extérieur?

Rappel sur java script - les types - Array

```
Array.prototype.concat()  
Array.prototype.filter()  
Array.prototype.find()  
Array.prototype.findIndex()  
Array.prototype.forEach()  
Array.prototype.join()  
Array.prototype.map()  
Array.prototype.pop()
```

```
Array.prototype.push()  
Array.prototype.reduce()  
Array.prototype.reverse()  
Array.prototype.shift()  
Array.prototype.slice()  
Array.prototype.sort()  
Array.prototype.splice()  
Array.prototype.unshift()
```



Es6 - Export/Import

Il y a 3 types d'exports

1. ***Named exports*** (plusieurs par module)
2. ***Default exports*** (un par module)
3. ***Mixed named & default exports***



Es6 - Export/Import - Named exports

```
//----- lib.js -----  
export const sqrt = Math.sqrt;  
export function square(x) {  
  return x * x;  
}  
export function diag(x, y) {  
  return sqrt(square(x) + square(y));  
}  
  
//----- main.js -----  
import { square, diag } from 'lib';  
console.log(square(11)); // 121  
console.log(diag(4, 3)); // 5  
or  
//----- main.js -----  
import * as lib from 'lib';  
console.log(lib.square(11)); // 121  
console.log(lib.diag(4, 3)); // 5
```

Es6 - Export/Import - Default exports

```
//----- myFunc.js -----  
export default function () { ... };  
  
//----- main1.js -----  
import myFunc from 'myFunc';  
myFunc();  
|
```

Es6 - Export/Import - Mixed named & default exports

```
//----- underscore.js -----  
export default function (obj) {  
    ...  
};  
export function each(obj, iterator, context) {  
    ...  
}  
  
//----- main.js -----  
import _, { each } from 'underscore';  
...
```

Es6 - Export/Import

Attention aux dépendances cycliques

```
// lib.js
import Main from 'main';
var lib = {message: "This Is A Lib"};
export { lib as Lib };
```

```
// main.js
import { Lib } from 'lib';
export default class Main {
  // ....
}
```


Es6 - Destructuring (object or array)

Le destructuring consiste à assigner des variables provenant d'un objet ou tableau en reposant sur leur structure.

Exemple avec Object

```
// ES5
```

```
var width = settings.width;
```

```
var height = settings.height;
```

```
// ES6
```

```
var { width, height } = settings;
```

Es6 - Destructuring (object or array)

Exemple avec Array

```
let introduction= ["Bonjour", "Will", "Smith"]  
let [greeting, firstName]=introduction  
console.log(greeting) // "Bonjour"  
console.log(firstName) // "Will"
```

spread

Le spread operator sert à **éclater un tableau/un objet en une liste finie de valeurs/ de propriétés**.

```
let args = ['var 1', 'var 2', 'var 1']  
  
console.log(...args) // => console.log('var 1', 'var 2', 'var 1')
```

Cela permet aussi de créer une copie d'un tableau

```
let args = ['var 1', 'var 2', 'var 1']  
  
let argsCopy=[...args]
```

spread

Exemple de copie d'un objet ou de concaténation de 2 objets sans changer les objets d'origine

```
const obj1 = { a: 1, b: 2 }  
const objCopy={...obj1}  
  
const obj2 = { a: 2, c: 3, d: 4 }  
const obj3 = { ...obj1, ...obj2 }
```

Fondamentaux de React

Commençons maintenant à faire du React proprement dit



Fondamentaux de React - Component

- Un composant est une fonction prenant en paramètre un objet appelé props et retournant du JSX
- Les composants vous permettent de découper l'interface utilisateur en éléments indépendants et réutilisables,

Combien de composants a-t-on?

```
import React from 'react'
import './Hello.css'
function Hello() {
  return (
    <p>Hello</p>
  )
}
export default Hello
```

Input field: Add a task

Task 1: Learn Html delete update

Task 2: Learn Html delete update

Task 3: Learn Html delete update

Fondamentaux de React - Installation



Créer un projet

Pour commencer à travailler :

1. Installer node pour pouvoir utiliser les commandes npm et npx
2. Créer un projet en se basant sur une structure prédéfinie (boilerplate) appelée "create-react-app"

```
C:\Users\Yosr\Desktop\reactjs\tp1>npx create-react-app MyProject
```

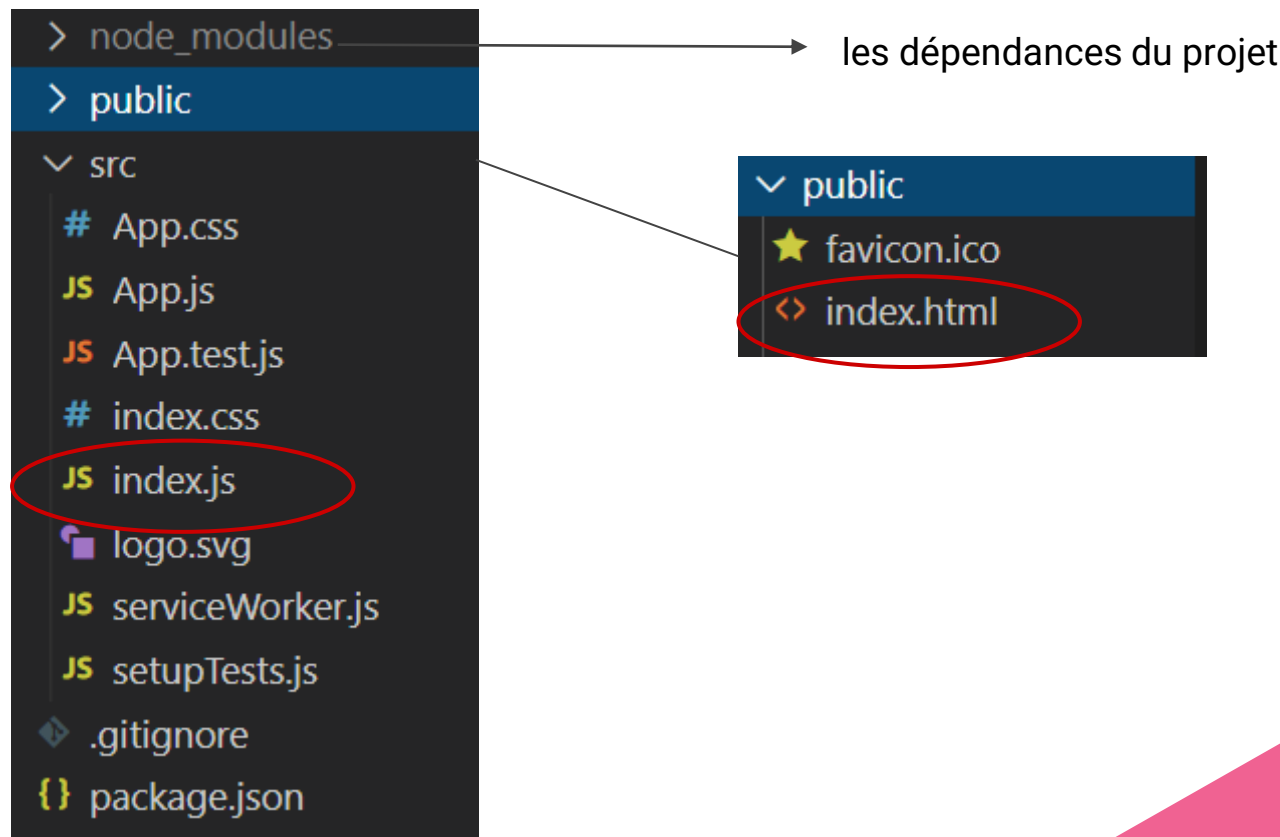
⇒ Ceci crée un dossier MyProject sous tp1.

=> vous pouvez appeler votre projet comme vous voulez

1. Allez au répertoire "MyProject"
2. Taper code . pour ouvrir vscode

```
C:\Users\Yosr\Desktop\reactjs\tp1>cd MyProject
```

Fondamentaux de React - Structure du projet



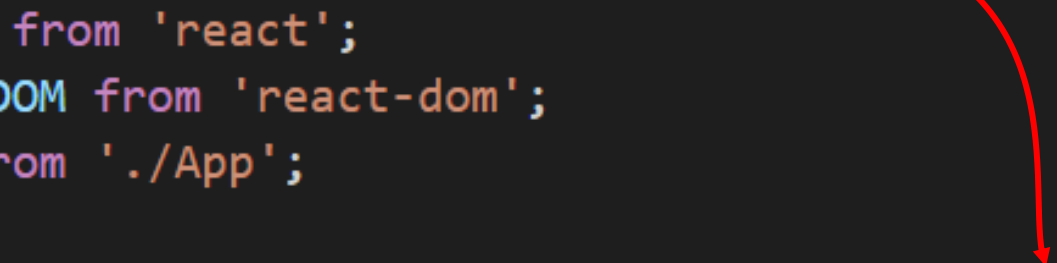
Fondamentaux de React - Structure du projet

Que contient le fichier index.html?

```
<body>  
  <noscript>You need to enable JavaScript to run this app.</noscript>  
  <div id="root"></div>
```

Que contient le fichier index.js

```
import React from 'react';  
import ReactDOM from 'react-dom';  
import App from './App';  
  
ReactDOM.render(<App />, document.getElementById('root'));
```



Fondamentaux de React - Structure du projet

- App est le composant root.
- C'est dans ce composant qu'on va appeler d'autres composants
- On peut bien sur changer le composant root `<App />` par un nouveau composant dans `ReactDOM.render(<App />, document.getElementById('root'));`



Fondamentaux de React - Premier composant statique

- **Comment créer un composant**

- Ecrire une fonction qui retourne du html
- La première lettre de la fonction est TOUJOURS en MAJUSCULE
- Il faut toujours importer le package React
- Le composant est exporté généralement en défaut

```
import React from 'react'
import './Hello.css'
function Hello() {
  return (
    <p>Hello</p>
  )
}
export default Hello
```

- **Comment utiliser un composant?**

- L'importer dans App ou dans n'importe quel composant
- L'utiliser comme une balise

```
import Hello from './Hello'
import './App.css'

function App() {
  return (
    <div className="App">
      <Hello />
    </div>
  )
}
```

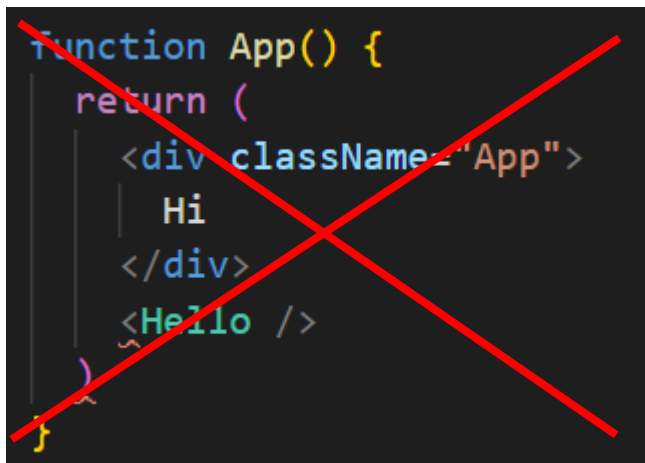
Fondamentaux de React - Premier composant statique

Attention :

- Il y a une erreur connue en React :

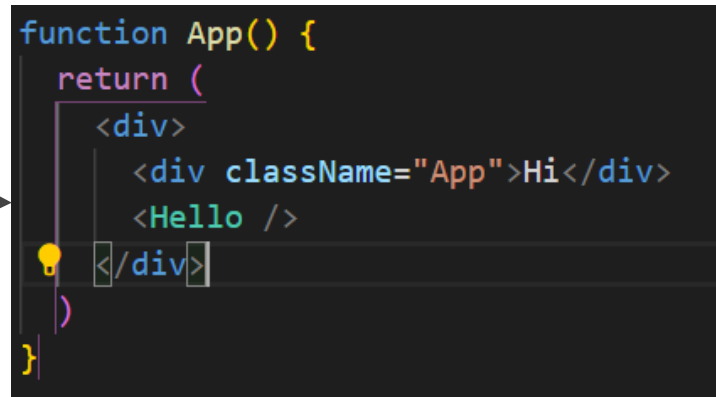
SyntaxError: Adjacent JSX elements must be wrapped in an enclosing tag

- Exemple



```
function App() {  
  return (  
    <div className="App">  
      Hi  
    </div>  
    <Hello />  
  )  
}
```

The code on the left is crossed out with a large red 'X', indicating it is incorrect. It shows two adjacent JSX elements, `<div>` and `<Hello />`, not wrapped in a single enclosing tag.



```
function App() {  
  return (  
    <div>  
      <div className="App">Hi</div>  
      <Hello />  
    </div>  
  )  
}
```

The code on the right is the correct version, wrapped in a single `<div>` tag. A lightbulb icon is placed next to the opening `<div>` tag, indicating a tip or a correct solution.

⇒ On peut utiliser aussi les fragments : `<> ...</>` ou `<React.Fragment>...</React.Fragment>`

Fondamentaux de React - JSX

- Vous remarquerez bien qu'on est en train d'écrire une fonction en javascript et que cette fonction retourne du html.

⇒ **Cette combinaison est le JSX**

- JSX est :
 - c'est une extension syntaxique de JavaScript
 - Il ressemble à un langage de balisage mais avec toute la puissance javascript=> c'est *"du javascript dans du html"* ou *"du html dans du javascript"*

Fondamentaux de React - JSX - className

- Il a des points communs avec XML :
 - La sensibilité à la casse : les majuscules et minuscules ne sont pas interchangeables,
 - L'exigence de fermeture des éléments :
 - même pour les éléments « seuls », comme par exemple `<input />` (et non pas `<input>`),
 - dans le bon ordre (on ferme dans l'ordre inverse de l'ouverture) : `<p>Bien</p>` mais pas `<p>Pas bien</p>`

Apprenons un peu plus JSX



Fondamentaux de React - JSX - className et string

- Use className instead of class for css class

```
<div className="App">Hi</div>
```



Premier exercice react

Reproduire cet exemple en react

- source html et css:

https://drive.google.com/drive/folders/1t_jLIY_OVyksm2WhoESCy8KALRIMisHg?usp=sharing

<input type="text"/>	Add a task
Learn Html	delete update
Learn Html	delete update
Learn Html	delete update



branche htmlToReact

Fondamentaux de React - JSX - variables

- JSX permet l'utilisation de variables en utilisant les {}

```
export default function TaskForm() {  
  const addTask = "Add a task"  
  return (  
    <div className="task-form">  
      <input type="text" name="task" id="" />  
      <button className="button">{addTask}</button>  
    </div>  
  )  
}
```



Appliquez sur votre exemple de todo list

Retour sur les fondamentaux de React - JSX - les fonctions

- JSX permet aussi l'appel de fonctions à l'intérieur de votre code html en utilisant les {}



Appliquez sur votre exemple de todo

list

```
import React from 'react'
import './Task.css'
export default function Task() {
  const renderActions = ()=>{
    return (
      <div className="actions">
        <span>delete</span>
        <span>update</span>
      </div>
    )
  }
  return (
    <div className="task">
      <div className="title">Learn html</div>
      {renderActions()}
    </div>
  )
}
```

Fondamentaux de React - JSX - les tableaux

- JSX permet d'afficher un tableau sans faire de boucle
- ⇒ ceci affiche

Enter the task title click on add task

```
function App() {  
  const steps = ["Enter the task title", "click on add task"]  
  
  return (  
    <div className="tasks-list">  
      {steps}  
    </div>  
  )  
}
```

Fondamentaux de React - JSX - les tableaux



Appliquez sur votre exemple de todo

- JSX ne permet pas d'utiliser des boucles à l'intérieur du code html (à l'intérieur du return)
- Or on veut retourner un tableau un peu formaté?
- ⇒ utiliser la fonction map

```
function App() {  
  const steps = ["Enter the task title", "click on add task"]  
  return (  
    <div className="tasks-list"  
      >  
        To add a Task  
        <ul>  
          {steps.map(step => (  
            <li>{step}</li>  
          ))}  
        </ul>  
      )  
    )  
}
```

Fondamentaux de React - JSX - les conditions

- JSX ne permet pas d'utiliser des if...else
- Solution : utiliser *ternary operation*

function getFee(isMember) {if (isMember) return '\$2.00' else '\$10.00'}

function getFee(isMember) {return (isMember ? '\$2.00' : '\$10.00')}

```
{loading ? <div>Loading...</div> :  
  <div>  
    <TaskForm />  
    <Task />  
    <Task />  
    <Task />  
  </div>  
}
```

Fondamentaux de React - JSX - les conditions

Une autre solution est d'utiliser les trully et falsy values

```
{loading && <div>Loading ... </div>}  
{!loading && (  
  <div>  
    <TaskForm />  
    <Task />  
    <Task />  
    <Task />  
  </div>  
)}  
}
```

Fondamentaux de React - JSX - les styles



Appliquez sur votre exemple de todo

list

- L'attribut style en react est un peu spécial
- Il accepte comme paramètre un objet



branche jsx

```
<div style={{ "background-color": "red" }}>
```

ou

```
<div style={{ backgroundColor: "red" }}>
```

ou

```
const myStyle = { backgroundColor: "red" }  
return (  
  <div style={myStyle}>  
    ...  
  </div>  
)
```

Fondamentaux de React

React et les props



Fondamentaux de React - Props - Introduction

- Conceptuellement, les composants sont comme des **fonctions JavaScript**. Ils acceptent des entrées quelconques appelées « **props** »

```
export default function Task(props) {  
  return (  
    <div className="task" >  
      <div className="title">  
        {props.title}  
      </div>  
    </div>  
  )  
}
```



```
function App() {  
  return (  
    <div className="app">  
      <Task title="learn Html" />  
      <Task title="learn React" />  
      <Task title="learn Angular" />  
    </div>  
  )  
}
```



Appliquez sur votre exemple de todo

list

Fondamentaux de React - Props - Introduction

Comment cela fonctionne?

```
<Task title="Learn Html" subTitle="for beginner" />
```

⇒ on est en train d'envoyer au composant Task 2 propriétés : **title** et **subTitle**

=> L'envoi de paramètres ressemble à l'utilisation d'attributs pour les balises html.

Rappelez vous la balise !!!!

Fondamentaux de React - Props - Introduction

Comment cela fonctionne?

```
<Task title="Learn Html" subTitle="for beginner" />
```

⇒ De l'autre côté, Task va recevoir un objet appelé **props**

⇒ Dans notre cas cet objet contiendra 2 propriétés. Comment être sûr?

=> Ajoutons un `console.log(props)` à notre code

```
export default function Task(props) {  
  console.log(props)  
  
  return (  
    <div className="task" >  
      <div className="title">  
        {props.title}  
      </div>  
      <div className="title">  
        {props.subTitle}  
      </div>  
    </div>  
  )  
}
```

Fondamentaux de React - Props - Type de props

On peut envoyer n'importe quel type de propriétés

- string
- number
- object
- function
- array

=> A part le type string, pour envoyer ces propriétés, on doit utiliser les {}

⇒ pour string on peut aussi utiliser les {} mais c'est facultatif



Fondamentaux de React - Props - Type de props

Exemple avec number

```
<Task duration={60} />
```

Exemple avec object

```
<Task details={{type:"IT", date:"2020-01-02"}} />
```



Installez l'extension react dev tools pour mieux voir comment ces props sont envoyées



Appliquez sur votre exemple de todo



branche props

Fondamentaux de React - Props - Type de props

Exemple avec array

```
<TasksList tasks=[  
  {title:"Learn Html", duration:30},  
  {title:"Learn React", duration:60},  
  {title:"Learn Node", duration:70},  
]  
]  
</TasksList>
```



Mise à jour de l'exemple

- Reprenez votre exemple de todo lists :
 - Déclarer dans App un tableau todos qui contiendra des objets. Chaque objet représentant une tâche décrite par id, title, duration, type, date)
 - Créer un composant TasksList : ce composant aura comme props un tableau de tâches et retournera une liste de composants Task
 - Appeler le composant TasksList dans App pour avoir le résultat suivant



branche advancedProps

To add a task

- Enter the task title
- Click on add task

learn Html basic (2020-01-02) [delete](#) [update](#)

learn React IT (2020-02-10) [delete](#) [update](#)

learn Angular IT (2020-03-15) [delete](#) [update](#)

Fondamentaux de React - Props - Type de props

Exemple avec function

```
function App() {  
  const sayHello = () =>{alert("Hello")}  
  return (  
    <div className="tasks-list" style={{ backgroundColor: "white" }}>  
      ...  
      <TaskForm sayHello={sayHello} />  
      ....  
    </div>  
  )  
}
```

De l'autre côté l'utilisation de la fonction se fera comme n'importe quelle propriété

```
export default function TaskForm(props) {  
  props.sayHello()  
}
```


Fondamentaux de React - Props - children props

- Cette propriété est particulière: elle n'est pas envoyée comme un attribut, mais en imbriquant des composants à l'intérieur du composant concerné.
- Le composant Task recevra `<div>Hello</div>` comme **`props.children`**



Appliquons le ensemble avec un champ description qu'on ajoutera à notre tableau de tasks

```
<Task title="Learn React" >  
  <div>Hello</div>  
</Task>
```

```
export default function Task(props) {  
  
  return (  
    <div className="task" >  
      <div className="title">  
        {title}  
      </div>  
      {props.children}  
    </div>  
  )  
}
```

Fondamentaux de React - Props - destructuring

- On a déjà vu le destructuring dans ce cours
- On va l'appliquer à l'objet props
- Ainsi on peut utiliser directement des variables/constantes
- Dans l'exemple à droite : on peut directement utiliser **title** et **subTitle** au lieu de **props.title** et **props.subTitle**

```
export default function Task(props) {  
  
  const {title, subTitle}=props  
  
  return (  
    <div className="task" >  
      <div className="title">  
        {title}  
      </div>  
      <div className="title">  
        {subTitle}  
      </div>  
    </div>  
  )  
}
```

Fondamentaux de React - Props - destructuring

- On peut aussi faire le destructuring directement au niveau de la déclaration du composant

```
export default function Task(props) {  
  const {title, subTitle}=props  
  
  return (  
    <div className="task" >  
      <div className="title">  
        {title}  
      </div>  
      <div className="title">  
        {subTitle}  
      </div>  
    </div>  
  )  
}
```

Task({ title, subtitle }) {



Appliquons le ensemble
sur TasksList

Fondamentaux de React - Props - exemple avec conditional rendering

- Maintenant qu'on a des données dans notre composant, on peut faire des conditions pour afficher ou non des parties html selon la valeur des props
- Voici des exemples :

```
{props.type && (  
  <div className="sub-title">  
    {props.type} - {props.date}  
    {props.children}  
  </div>  
)}
```



Appliquons le ensemble

```
<div  
  className={`title ${props.type === "beginner" ? "customTask" : ""}`}  
>
```

Fondamentaux de React

React et le state



branche newLookBeforeState

Fondamentaux de React - State - Introduction

- Les variables de **state** en français “**état**” permettent au composant de créer et de gérer ses propres données
- Jusque là un composant recevait seulement des **props**. Or les props sont en **lecture seulement**
- Les variables d'état peuvent changer de valeurs mais restent propres au composant lui même.



Fondamentaux de React - State - Introduction

- Motivation

“**Declarative** programming in **React**

In **react**, You make interactive UIs by changing the state of the component and **React** takes care of updating the DOM according to it. ... In **react** the DOM is **declarative**. This means we never interact with DOM, the UI is updated when we change the state”



Fondamentaux de React - State - Syntaxe

- Pour déclarer une variable de state, on utilise useState

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

function App() {
  const [isVisible, setIsVisible] = useState(true)
```

hook : une fonction
de la bibliothèque
React

Variable de state

Fonction qui permet de
modifier la variable
isVisible

- Les *Hooks* sont arrivés avec React 16.8.
- Par convention un hook commence par **use**

Fondamentaux de React - State - Exemple

- Qu'est ce qui se passe quand on appuie sur le bouton?

Rerendering

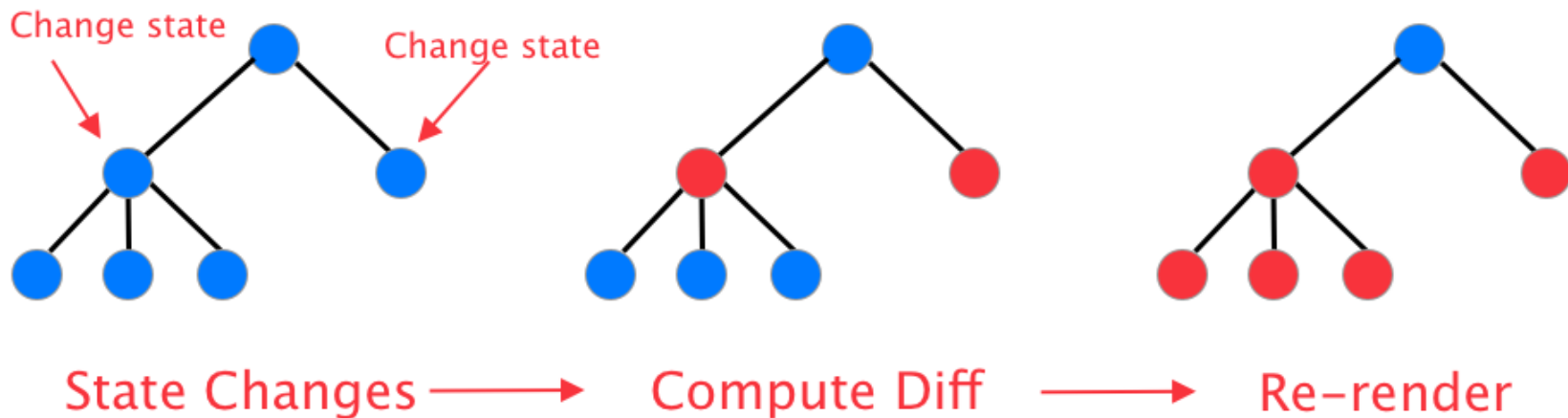


branche state

```
function App() {  
  const [isVisible, setIsVisible] = useState(true)  
  
  const toggleVisibility = () => {  
    setIsVisible(!isVisible)  
  }  
  
  return (  
    <div className="app">  
      <div className="toggle">  
        /* 1ère solution */  
        <button onClick={() => toggleVisibility()}>Toggle visibility</button>  
        /* 2ème solution */  
        /* <button onClick={toggleVisibility}>Toggle visibility</button> */  
      </div>  
      <div>  
        {isVisible && (  
          <div>  
            <TaskForm />  
          </div>  
        )}  
      </div>  
    </div>  
  )  
}
```

Fondamentaux de React - State - Exemple

- Les **composants React** font un **re-rendering automatique** à chaque fois qu'une de leurs variables de props ou de state change.
- On peut parfois assister à un re-rendering en cascade



Fondamentaux de React

Formulaires et gestion des événements



Formulaires et gestion des événements

- Les événements de React sont nommés en `camelCase` plutôt qu'en minuscules.
 - `onClick`, `onChange`, `onSubmit`, ...
- En JSX on passe **une fonction** comme gestionnaire d'événements **plutôt qu'une chaîne de caractères**.

```
<button onclick="activateLasers()">  
  Activer les lasers  
</button>
```

html

```
<button onClick={activateLasers}>  
  Activer les lasers  
</button>
```

React

Formulaires et gestion des événements

Soit le formulaire suivant (extrait de la doc de reactjs)

```
<form>
  <label>
    Nom :
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Envoyer" />
</form>
```

=> Formulaire classique en html mais non souhaité en react : « **composant non contrôlé** »

=> On voudrait gérer la soumission du formulaire nous-même => « **composant contrôlé** ».

Formulaires et gestion des événements

```
function MyForm() {  
  const [value, setValue] = useState("")  
  const handleChange = (e) => {  
    setValue(e.target.value)  
  }  
  
  const handleSubmit = (e) => {  
    alert("Le nom a été soumis : " + value)  
    e.preventDefault()  
  }  
  
  return (  
    <form onSubmit={handleSubmit}>  
      <label>  
        Nom :  
        <input type="text" value={value} onChange={handleChange} />  
      </label>  
      <input type="submit" value="Envoyer" />  
    </form>  
  )  
}
```



Mise à jour de l'exemple

- Reprenez votre exemple de todo lists :
 - Implémenter le code du bouton add task delete task, update task



branche addDeleteTask



branche updateTask

To add a task

- Enter the task title
- Click on add task

learn Html basic (2020-01-02) [delete](#) [update](#)

learn React IT (2020-02-10) [delete](#) [update](#)

learn Angular IT (2020-03-15) [delete](#) [update](#)

Fondamentaux de React

le hook useEffect



services

Hook important : useEffect

Que fait `useEffect` ?

- `useEffect` indique à React que notre composant doit exécuter quelque chose **après chaque affichage**.
- React enregistre la fonction passée en argument (que nous appellerons « effet »), **et l'appellera plus tard, après avoir mis à jour le DOM**.
- L'effet ci-dessous met à jour le titre du document, mais il pourrait aussi bien appeler n'importe quelle API.

```
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>
  );
}
```

Hook important : useEffect

- Quatre formes de useEffect

```
useEffect(  
  () => { /* write here the effect code*/ }  
)
```

le code de l'effet est toujours exécuté après chaque affichage/rerendering du composant

```
useEffect(  
  () => { /* write here the effect code*/ },  
  []  
)
```

le code de l'effet est exécuté une seule fois après le premier affichage du composant (lors du **"mounting"** du composant dans le DOM)

```
useEffect(  
  () => { /* write here the effect code*/ },  
  [ /*dependency variables*/ ]  
)
```

le code de l'effet est exécuté après chaque affichage/rerendering du composant **Si Si une des variables de dépendance a changé**

```
useEffect(  
  () => {  
    /* write here the effect code*/  
    return () => {  
      //cleanup  
    }  
  }, [ /*dependency variables*/ ]  
)
```

- le code de l'effet est exécuté après chaque affichage/rerendering du composant **Si Si une des variables de dépendance a changé**
- **Le code de cleanup** est exécuté après changement de la variable

Hook important : useEffect

- 1ère forme de useEffect

```
useEffect(  
  () => { /* write here the effect code*/ }  
)
```

le code de l'effet est toujours exécuté après chaque affichage/rendering du composant

- Exemple

```
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>  
  );  
}
```

- **Attention** : il ne faut jamais changer le state dans cette forme de useEffect **Sinon** boucle infinie

Hook important : useEffect

- 2ème forme de useEffect

```
useEffect(  
  () => {/* write here the effect code*/},  
  []  
)
```

le code de l'effet est exécuté une seule fois après le premier affichage du composant (lors du **"mounting"** du composant dans le DOM)

- Exemple

```
useEffect(() => {  
  const fetchData = async () => {  
    setLoading(true)  
    const result = await fetchTasks()  
    setTasks(result)  
    setLoading(false)  
  }  
  console.log("useEffect")  
  
  fetchData()  
}, [])
```

- **Remarque:** on peut changer le state dans cet type de useEffect

Hook important : useEffect

- 3ème forme de useEffect

```
useEffect(  
  () => { /* write here the effect code*/ },  
  [ /*dependency variables*/ ]  
)
```

le code de l'effet est exécuté après chaque affichage/rerendering du composant **SiSi une des variables de dépendance a changé**

- Exemple

```
useEffect(() => {  
  const fetchData = async () => {  
    setLoading(true)  
    if (searchValue.length === 0) {  
      console.log("tasks empty")  
      setTasks([])  
      setLoading(false)  
    } else {  
      const result = await fetchTasksByFilter(searchValue)  
      console.log("tasks form api")  
      setTasks(result)  
      setLoading(false)  
    }  
  }  
  console.log("searchValue", searchValue)  
  fetchData()  
}, [searchValue])
```

- **Attention** : il ne faut jamais changer le state d'une des dépendances dans cette forme de useEffect **Sinon** **boucle infinie**

Hook important : useEffect

- 4ème forme de useEffect

```
useEffect(  
  () => {  
    /* write here the effect code*/  
    return () => {  
      //cleanup  
    }  
  }, /*dependency variables*/  
)
```

- Exemple

```
useEffect(() => {  
  const interval = setInterval(function () {  
    setCount((prev) => prev + 1);  
  }, 1000);  
  // return optional function for cleanup  
  // in this case acts like componentWillUnmount  
  return () => clearInterval(interval);  
}, []);
```


- le code de l'effet est exécuté après chaque affichage/rerendering du composant **SiSi une des variables de dépendance a changé**
- **Le code de cleanup** est exécuté après changement de la variable

- **Attention** : il ne faut jamais changer le state d'une des dépendances dans cette forme de useEffect **Sinon boucle infinie**

Règles sur les hooks

- Ne jamais appeler les hooks à l'intérieur de boucles ou à l'intérieur de conditions
- Ne jamais appeler un hook à l'intérieur d'une fonction classique

Custom hooks

- Vous pouvez créer vous même un hook
 - Par convention, tous les hooks commencent par "use"
- 

React

Communication avec les API



Communication avec une api

- Lors de la création d'une application Web, vous souhaitez parfois utiliser et afficher les données provenant d'une API.
- Il existe plusieurs manières de le faire, mais une approche très populaire consiste à utiliser **axios**, un client HTTP basé sur les Promises
- **Solution:**
 - installer **axios** : \Rightarrow **npm i axios**
- **Documentation**

<https://github.com/axios/axios>

React

Les routes avec React router




React - les Routes

Principe des routes :

- Quand on a besoin de naviguer entre plusieurs interfaces d'une application React, nous avons besoin d'un routeur ("router") qui va gérer les différents urls
- C'est le package `react-router-dom` qui permet gérer ces routes
- A chaque route est associé un composant

Etapas :

- installer **react-router-dom**
 - Créer des routes : faire correspondre à chaque vue, une url
- 

React - les Routes

permet de choisir la première route dont le chemin vérifie l'url

La route et son chemin

Composant à afficher quand l'url est vérifié

```
import {
  BrowserRouter as Router,
  Switch,
  Route,
} from "react-router-dom"
function App() {
  return (
    <div className="app">
      <Router>
        <Switch>
          <Route exact path="/hello">
            <Hello value="Welcome to my web site" />
          </Route>
          <Route exact path="/tasks">
            <ListPage />
          </Route>
          <Route exact path="/tasks/:taskId">
            <TaskDetails />
          </Route>
        </Switch>
      </Router>
    </div>
  )
}
```

React - les Routes

Comment utiliser ces routes

- **Solution intuitive** : saisir l'url correspondante dans le navigateur
- **Solution avec un lien** : **créer des liens qui ressemblent à des `<a href>` de html**

⇒ utiliser **`<Link/>`** ou **`<NavLink/>`** de **react-router-dom**

La destination du lien

La classe css à appliquer quand le lien est actif (l'url courante correspond au chemin de celui)

```
<Link to="/tasks" activeClassName="active">  
  My tasks  
</Link>
```

```
<NavLink to="/tasks" activeClassName="active">  
  My tasks  
</NavLink>
```

React - les Routes

Comment utiliser ces routes

- **Solution avec un événement click:** déclencher une action de redirection quand on clique sur un bloc

```
import { useHistory } from "react-router-dom"

export default function Task({ id, title, duration, deleteTask, updateTask }) {

  const history = useHistory()
  const handleDetails = () => {
    history.push(`/tasks/${id}`)
  }

  return (
    <div className="task">
      <div onClick={handleDetails} className="link">
        <div className="title">
          {title} ({duration} mn)
        </div>
      </div>
    </div>
  )
}
```

React - les Routes

Route avec des Paramètres d'url

- Parfois on a besoin d'avoir une url ou seul un id change.Ex
 - <http://localhost:3000/tasks/1>
 - <http://localhost:3000/tasks/2>
- Comment faire :

```
<Route exact path="/tasks/:taskId">  
  <TaskDetails />  
</Route>
```

```
<Link to={`/${tasks/${id}}`}>  
  <div className="title">  
    {title} ({duration})  
  </div>  
</Link>
```

React - les Routes


- Comment récupérer la valeur du paramètre taskId dans le composant TaskDetails:

```
import { useParams } from "react-router-dom"
function TaskDetails() {
  const { taskId } = useParams()
```

- Comment récupérer le chemin

```
import { useLocation } from "react-router-dom"
function TaskDetails() {
  const location = useLocation()

  console.log('useLocation(): ', location);
```



```
useLocation(): Object {
  hash: "",
  key: "r6x2bo",
  pathname: "/tasks/1",
  search: "",
  state: undefined,
  __proto__: Object
}
```