Chapitre 4 : Les composants en React

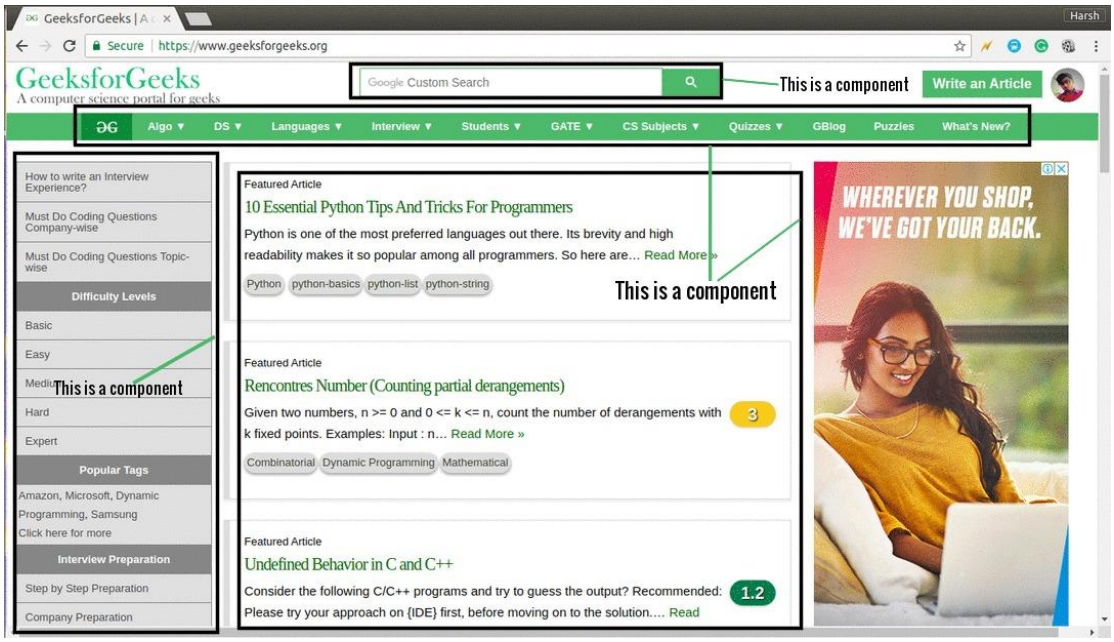
Table des matières

I. Définition

Un composant est l'un des principaux éléments constitutifs de React. En d'autres termes, on peut dire que chaque application que vous développerez avec React sera constituée de pièces appelées **composants**.

Les composants facilitent grandement la construction des interfaces utilisateur. Vous pouvez voir une interface utilisateur décomposée en plusieurs pièces individuelles appelées composants et travailler sur elles indépendamment et les fusionner toutes dans un composant parent qui sera votre interface finale.

Vous pouvez voir dans l'image ci-dessous que nous avons décomposé l'interface utilisateur de la page dans la figure suivante en composants individuels :



II. Les classes en JavaScript

a. ECMAScript et JavaScript

ECMAScript est un standard de langage de programmation, il définit:
– La syntaxe

– Les types de variable

et encore pas mal d'autres choses…

Le rapport entre ECMAScript et javascript tout simplement, JavaScript est un des langages qui respecte le standard ECMAScript. C'est tout ! Il y a d'autre langages qui respectent ce standard, comme ActionScript ou JScript.

Les versions de ECMAScript sont :

- ES5 : Il a été publié en 2009
- ES6 : Il a été publié en 2015, d'ailleurs vous pouvez aussi le voir aussi sous le nom ES2015. Il y a eu un saut particulièrement important entre ES5 et ES6. Beaucoup de choses ont évolué en ES6… Fonctions Fléchées, Classes, Modules et j'en passe… ça a été une vraie révolution du JavaScript
- ES7 : Il a été publié en 2016, d'ailleurs vous pouvez aussi le voir aussi sous le nom ES2016
- ES8 : Il a été publié en 2017, d'ailleurs vous pouvez aussi le voir aussi sous le nom ES2017

b. Déclaration d'une classe

Pour utiliser une déclaration de classe simple, on utilisera le mot-clé `class`, suivi par le nom de la classe que l'on déclare. L'exemple suivant montre la création de la classe Rectangle qui contient deux attributs :

```
class Rectangle {
        constructor(hauteur, largeur) {
          this.hauteur = hauteur;
          this.largeur = largeur;
        }
      }
```

Voici un exemple de création d'objets (ou instances) à partir de la classe précédente :

```
let r1 = new Rectangle(3,5)
let r2 = new Rectangle(4)
let r3 = new Rectangle()
```

c. Constructeur

La méthode **constructor** est une méthode spéciale qui permet de créer et d'initialiser les objets créés avec une classe. Il ne peut y avoir qu'une seule méthode avec le nom

"constructor" pour une classe donnée. **Si la classe contient plusieurs occurrences d'une méthode constructor, cela provoquera une exception SyntaxError**.

d. Les méthodes d'instance

Une méthode d'instance est une fonction faisant partie d'une classe, et qui agit sur une instance de cette classe. Voici un exemple de déclaration d'une classe pour représenter les comptes bancaires :

```
class BankAccount {

        constructor(proprietaire, solde) {
          this.proprietaire = proprietaire;
          this.solde = solde;
        }

        showSolde() {
          console.log("Solde: " + this.solde + " DH");
        }

        deposer(amount) {
          console.log("Dépôt de " + amount + " DH");
          this.solde += amount;
          this.showSolde();
        }

        retirer(amount) {
          if (amount > this.solde) {
            console.log("Retrait refusé !");
          } else {
            console.log("Retrait de " + amount + " DH");
            this.solde -= amount;
            this.showSolde();
          }
        }
   }
```

e. Héritage

L'héritage de classe permet de créer une nouvelle classe à partir d'une classe déjà existante. On dit que la nouvelle classe est dérivée (héritée) de la première. Il est ainsi possible d'enrichir (améliorer) une classe existante sans avoir à modifier son code interne. On peut

alors réutiliser des parties de code existants (c'est-à-dire utiliser d'autres classes qui ont déjà été écrites et qui fonctionnent correctement).

L'héritage en JavaScript se fait à l'aide du mot cle extends comme le montre l'exemple suivant :

```
class Vehicle {
  constructor(name) {
    this.name = name;
  }
  describe() {
    return `Car named ${this.name}`;
  }
}

class Car extends Vehicle {
  constructor(name, color) {
    super(name);
    this.color = color;
  }
  describe() {
    return `${super.describe()} (${this.color})`;
  }
}
```

f. Exercices d'application

Exercice 1 : Manipulation des classes et objets en javascript ES6

a) Créer la classe Document contenant les attributs id et dateEdition

b) Créer la classe Livre qui hérite de la classe Document contenant en plus les attributs titre et auteur.

c) Créer deux instances de Document

d) Créer trois instances de Livre

e) Créer la méthode infoLivre qui retourne les informations du livre exemple :

```
Livre: id:10 titre:POO Auteur: Rami date Edition:10/05/2020
```

Exercice 2 : Manipulation des classes

Créer une application appelée RestfulReading (Lecture reposante), qui permettra aux utilisateurs de suivre les livres qu'ils sont en train de lire ou qu'ils ont lus. Votre tâche est de créer la classe Book (Livre) et de remplir la base de données de développement de trois ou quatre livres.

1- La classe Book doit contenir les champs suivants :

– **title** - string - le titre du livre

– **author** - string - l'auteur du livre

– **description** - string - une description du livre

– **pages** - number - le nombre total de pages

– **currentPage** - number - la page où se trouve l'utilisateur actuellement (entre 1 et pages)

– **read** - boolean - si l'utilisateur a lu ou non le livre (par défaut: false)

La classe Book doit aussi contenir la méthode instance suivante : readBook(page). Cette instance permet à l'utilisateur de dire à quelle page il se trouve actuellement :

– Si l'argument page est inférieur à un ou supérieur au nombre total de pages du livre, readBook retourne dans la console 0

– Si l'argument page est supérieur ou égal à 1 et inférieur au nombre total de pages du livre, readBook modifie le champ currentPage de l'instance pour être égal à la valeur de l'argument passé, et retourne dans la console 1

– Si l'argument page est égal au nombre total de pages du livre, readBook modifie le champ currentPage de l'instance pour être égal à la valeur de l'argument passé, modifie le champ read de l'instance en true, et retourne dans la console 1

2- Créer trois instances de la classe book et Afficher vos livres dans la console.

3- Tester la fonction readBook.

III. Création d'un composant

Il existe deux types de composants que nous pouvons crée à l'aide de React : les composants de classe et les composants de fonction.

a. Composant Classe

Pour définir un composant définis par une classe, nous devons déclarer une classe qui hérite de la classe `React.Component`. comme le montre l'exemple suivant :

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

class Welcome extends React.Component {

  render(){
    return <h1>Welcome Message</h1>;
  }
}

function App() {
  return <Welcome />;
}

export default App;
```

b. Composant fonctionnel

Les composants fonctionnels font partie des composants les plus courants que l'on rencontre en travaillant avec React. **Il s'agit simplement de fonctions JavaScript.**

Nous pouvons créer un composant fonctionnel pour React en écrivant une fonction JavaScript. Ces fonctions peuvent recevoir ou non des données en tant que paramètres. Dans les composants fonctionnels, la valeur de retour est le code JSX à rendre à l'arbre DOM.

Exemple de déclaration et d'appel d'une composant fonctionnel :

```
import React from "react";

// This is a functional component
const Welcome= () =>
{
      return <h1>Hello World!</h1>
}
```

```
function App() {
      return <Welcome />;
}

export default App;
```

Voici un exemple de déclaration des composants imbriqués :

```
import React from "react";

const Filiere = () => {
  return <div>
      <Nom />
      <Module />
      <NbHeures />
     </div>;
}

const Nom = () => <h1>Developpement Web Full Stack</h1>
const Module = () => <h2>Developpement Front End avec React</h2>
const NbHeures = () => <h2>1300 heures</h2>

function App() {
  return <Filiere />;
}

export default App;
```

c. Utiliser une fonction ou une classe pour créer les composants en JSX ?

Cette question se pose car les deux manières vues précédemment sont similaires et aboutissent visiblement aux mêmes résultats.

– On utilisera une fonction si l'on n'a pas besoin de créer des propriétés ou des méthodes pour faciliter les traitements ;

– On utilisera plutôt une classe si des propriétés ou des méthodes sont nécessaires pour les traitements.