*Formation React Avec Mike*

Dans une page HTML, copier 3 balises <script> dans le head:

<script src="https://unpkg.com/react@16/umd/react.development.js"></script>

    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

Mettre en suite dans le body une <div> d’entrée avec l’identifiant "root", par convention, qui sera notre point d’entrée pour React.

Placer aussi en bas du body une dernier balise <script> avec le type="text/babel" comme attribut.

***§§§§ Seulement pour de petits tests en Réact §§§§***

Module 7 : ES6

Le wrapper

    console.log('toto');

Il se peut que le navigateur ne reconnaisse pas console, donc fait planter ton code

     Faire une fonction

        function log(text){

            if (console){

                console.log('toto');

            }else{

                //code alternatif

            }

        }

        log("toto");

       Le polyfill

       Librairie/utilitaire, qui génère un code plus ancien, qui simule ton code récent pour qu'il soit compatible à certain naviguateur plus ancien.

        Les transplieurs (ex : babel)

        Tu écrire ton code normalement avec la derniere mise à jour puis le transpileur génère un code lisible par tous les navigateurs.

Par convention :

Le nom de la variable initialisée avec const s'écrit en majuscule.

Ex : const NOM = "Toto" ;

Les noms de classes, la première lettre en majuscule.

Ex : class Voiture{

constructeur()

}

Un Componant (leçon Mike : 10-c):

1. C'est une classe qui extends Componant, qui a un constructeur dans lequel on peut initialiser des states par défaut.
2. Des composants avec des props qui sont figés. Pour y accédé, on fait : "props.nomDuProps", dans le constructeur , ou dans le render, on fait : {this.props. nomDuProps }.
3. Pour lire un state on fait: {this.state.nomDuState}, mais si on veux changer la valeur du state, on fait un setState({propriété : e.taget.value}); (e.taget.value =la valeur de la propriété).
4. On ne peut pas modifier un props dans un composant.

Les imports et les exports :

Dans un fichier math.js, qui sera un module, on crée une constante, une fonction, une classe… dont on met le mot clé export devant. Celui-ci sera accessible depuis un autre fichier avec le mot clé import.

Ex : Dans le fichier de composants.

export default function Mulitplication ( x , y ){

return x\*y;

}

export const pi = 3.14;

Lors de la déclaration d'une fonction, on à 2 choix, on place les mots clé export default devant la fonction (exemple ci-dessus) ou bien en bas de page.

Ex:

export default Multiplication;

!!! attention !!!

En cas de plusieurs composant déclarer dans un fichier, le mot clé default ne fonctionne plus. Il faut exporter les composant un par un.

Ex:

export function Multiplication(){

( le code)

}

Dans le fichier de base, au niveau de l'import, il faut entre accolade, mettre toutes les compos séparées par des virgules.

import {Multiplication, Soustraction, Addition} from "lieu du fichier/nom du fichier"

Dans le fichier de base :

1)

import \* as math from "math";

console.log(math.multiplication( 5 , 4 ));

2)

Import {multiplication, pi} from "math";

console.log(multiplication( 5 , 4 ));

console.log("pi vaut :" + pi);

Les props :

On utilise le props pour des valeurs figées, comme une variable qui durera dans le code.

On déclare des props de 2 façons :

1. function Famille (props){

<div>

<h3> Pure compo {props.nomVariable}</h3>

</div>

}

On déclare le props dans les paramètres, et props.nomVariable entre accolades dans le code.

1. Function Famille ({nomVariable}){

<div>

<h3>Pure compo {nomVariable}</div>

</div>

Les states :

Les states sont utilisé pour des évènements, des animations, là ou il y de l'interaction, c'est dynamique.

Lorsque l'on crée un composant avec state, on peut utiliser 2 choses : des propriétés (props) et des states.

Les props :

<h2 className="state-h2">Compo state {this.props.label}</h2>

Donc => this.props.nom de la propriété

Les states : <input type="text" value={this.state.nom} onChange={this.handelChange}/>

Donc => this.state.nom du state

Pour les states, il faut soit importer component, ou bien dans la création d'une classe en extends.

Ex:

1. Avec import :

import React, {Componant} from "react";

class MaClasse extends React.Componant{

}

1. Sans import (squelette de base d'une classe) :

class MaClasse extends Componant{

constructeur(props){

super(props);

}

}

Le state est un objet qui est défini avec le mot clé "this".

this.state = {nom : ""};

Le bind() :

Le bind, c'est faire reconnaitre un mot, ici on dans le state on va faire reconnaitre le this.

Pour cela il y a trois façons :

1. Dans le constructeur :

this.handelChange = this.handelChange.bind(this);

Ici on utilise le this dans une méthode (handelchange()), il faut le déclaré dans le constructeur.

1. Dans la déclaration dans le render (non conventionnelle) :

<input type="text" value={this.state.nom} onChange={this.handelChange.bind(this)}/>

1. Faire une fonction fléchée, et là le bind sera sous-entendu.

*Formation React Avec Antho Welc*

Dans le dossier de projet, avec VSCode et après avoir installé node.js, je créer de l’espace de travail npm :

npx create-react-app (puis le nom de l’application)

cd nom de l’application

npm start pour lancer le serveur ctrl+c pour l’arrêter

Détails de l’installation de npx creat-react-app : leçon 12 sur react UDEMY

node\_modules => toutes les dépendances de nodes.

package-lock.json => lies les dépendances au fichiers du projet, et bloques les versions pour ne pas avoir des différences de versions lors de travaux en groupe.

Read-me.md => donnes des infos sur react et npm.

package.json => Qui contient les versions de react, de react-dom react-scripts, pour pouvoir les utiliser, puis des actions de script, qui agit avec npm (start, build, test, eject), c’est pour, dans l’ordre démarrer le serveur, construire l’application avec des fichiers HTML, CSS, JS, puis si l’on souhaite faire des tests et enfin eject pour personnaliser un peu plus l’application.

.gitignore => pour utiliser Git

Dans le dossier public :

Index.html => Page d’entrée du projet, possède un link qui donne un lien sur un manifest.json, une balise <noscript> pour avertir l’utilisateur que le site fonctionne avec react. Une div avec l’id root dans laquelle le code react va être injecter

manifest.json => pour créer des progressive web-app(pour mettre en cache certains fichiers).

Dans le fichier source (src) :

Index.js => fichier qui va lancer react, charger react-DOM, un fichier CSS, un componant app, aussi un serviceworker (application plus rapide, ou et accessible hors ligne), non activé par défaut (serviceWorker.unregister()).

Logo.svg => logo qui tourne sur le serveur.

index.css => le css qui tourne sur le sur la page de présentation du serveur.

App.test.js => test pour tester son application, si on utilise des framework de test.

App.js => (ce fichier est un conponent) App.js est écris en JSX et est importer dans index.js. Ce fichier peut et être réécris entre la 1ère balise <div>.

Point important concernant les conventions pour React :

Les composants (componants) commencent toujours par une majuscule.

*Formation React Avec Simon Dieny*

Mise en place du socle de notre application, ou mise en place de l’architecture de notre application.

Il y aura plusieurs fichiers différents pour mieux découper notre code et de mettre en place notre application de la bonne manière, pour ne pas mettre des fichiers qui ont des rôles différent ensemble.

**Création du dossier qui va accueillir le projet**

1. **Pour la coloration syntaxique dans Vscode**

TypeScript : npm install -g typescript

1. **Création du fichier package.json**,

Info officielle de json : <https://docs.npmjs.com/creating-a-package-json-file>

Info officielle de TypeScript : <https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>

Installation de Node.Js pour le npm (node package manager) : [https://nodejs.org/fr/download](https://nodejs.org/fr/download/)

Info de contenu de la formation : <https://www.alexandria-library.co/ressources-react/>

{

  "name": "react-pokemons-app",

  "version": "1.0.0",

  "description": "An awesome application to handle some pokemons.",

  "dependencies": { <= *les dépendances*

     "@types/node": "12.11.1",

     "@types/react": "16.9.9",

    "@types/react-dom": "16.9.2",

     "@types/react-router-dom": "^5.1.2",

     "react": "^16.10.2",

     "react-dom": "^16.10.2",

     "react-router-dom": "^5.1.2",

     "react-scripts": "3.2.0",

     "typescript": "3.6.4"

  },

  "scripts": { <= script

    "start": "react-scripts start", <= mise en place de l’environnement

    "build": "react-scripts build" <= construction du livrable final

  },

  "eslintConfig": { <= analyse la syntaxe es6

    "extends": "react-app"

  },

  "browserslist": { <= vérifie le comportement de certains script en fonction des navigateurs

    "production": [

      ">0.2%",

      "not dead",

      "not op\_mini all"

    ],

    "development": [

      "last 1 chrome version",

      "last 1 firefox version",

      "last 1 safari version"

    ]

  }

}

1. **Création du fichier tsconfig.json**

{

  "compilerOptions": {

"target": "es5", <= indique que notre code sera compilé sur de es5, on peut le remplacé par es6.

    "lib": [

      "dom",

      "dom.iterable",

      "esnext"

    ],

    "allowJs": true,

    "skipLibCheck": true,

    "esModuleInterop": true,

    "allowSyntheticDefaultImports": true,

    "strict": true,

    "forceConsistentCasingInFileNames": true,

    "module": "esnext",

    "moduleResolution": "node",

    "resolveJsonModule": true,

    "isolatedModules": true,

    "noEmit": true,

    "jsx": "react" <= indique à TypeScript que nous allons utiliser du jsx pour notre code, et qu’il le compile en JS

  },

  "include": [ <= indique au compilateur ou se trouve nos fichiers

    "src"

  ]

}

Nos composant doivent avoir l’extension .tsx ( tsx = TypeScript + jsx). Comme ça, le compilateur TypeScript saura qu’il doit traiter du code jsx.

1. **Installation de npm** :

Ouvrir la console terminale de VScode dans le dossier racine du projet.

Taper : npm install

Créer un dossier node\_modules qui contient toutes les dépendances de node.

1. **Organisation du dossier**

Création d’un dossier src à la racine du projet, à l’intérieur créer un fichier app.tsx qui contiendra notre premier composant.

Dans ce fichier nous allons mettre notre 1er code.

import React from 'react'; <= importation du module React

const App: React.FC = () => { <= appel de la fonction FC de React

const name: String = 'React'; <= déclaration d’une variable de type String qui contient ‘react’.

 return (

  <h1>Hello, {name} !</h1> <= la fonction retourne en jsx une balise h1 avec hello et notre variable.

 )

}

export default App; <= permet d’exporter le fichier dans le reste de notre application

Fc => Fonction importé de React (FunctionComponent).

Peut être mis dans le code ci-dessus, façon destructuring :

const App: React.FC = () => { <= remplacer par

const App: FunctionComponent = () => {

et rajouter dans l’import :

import React, { functionComponent } from ‘react’ ;

Création d’un second fichier, se sera le point d’entrer de notre SPA (Single Page Application)

Index.tsx.

// Les importations

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

// la fonction ReactDom.render prend 2 parametres

ReactDom.render(

    <App />,

    document.getElementById('root')

);

A la racine du projet, on créer ensuite un dossier public qui contiendra un fichier index.html.

<!DOCTYPE html>

<html lang="fr">

  <head>

    <meta charset="utf-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <title>Pokédex</title>

  </head>

  <body>

    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root">L'application est en cours de chargement...</div>

</body>

</html>

Pour voir le rendu du code actuel, on peut faire dans le terminal :

Npm start

Un serveur local est créer puis ouvre notre page Html qui affiche le code renvoyé par la fonction de notre fichier App.tsx.

Sur la page HTML, au niveau de la div avec l’id root, c’est ici que notre application sera injectée.

Pour arrêter le serveur faire ctrl+c.

En résumé

* On a besoin au minimum d’un composant par application.
* Le fichier *index.tsx* fait le lien entre notre composant racine, et le fichier *index.html*.
* L’ordre de chargement de l’application est le suivant : *index.html* > *index.tsx* > *App.tsx*.
* Dans une application React, l’extension des fichiers TypeScript est *tsx*, car nous utilisons JSX.
* Le fichier de configuration du compilateur de TypeScript se nomme *tsconfig.json*par convention.
* Le fichier *package.json* initial est fourni avec des commandes prêtes à l’emploi comme la commande *npm start*, qui nous permet de démarrer notre application web.

Il existe deux types de composant dans React, les composants déclaré avec une fonction comme on a fait, et les composant déclaré avec des classes

Modification du fichier App.tsx, on lui rajoute un Hook :

Code de base :

import React {FunctionComponant} from 'react';

const App: FunctionComponant = () => {

 const name: String = 'Yan';

 return (

  <h1>Hello, {name} !</h1>

 )

}

export default App;

Modification :

Importation du hook.

import React {FunctionComponant, useState} from 'react';

Modification de la variable de la fonction avec du destructuring [ ].

const [name, setName] = useState("React");

Ajouter un type TypeScript à la fonction pour s’assuré que ce qui est passé en paramètre soit bien une chaine de caractère.

const [name, setName] = useState<String>("Yan");