 Orange Digital Center	Formation Développement web Full Stack : MERN
Période : Les 15, 16 et 17 Aout 2022 (Formation en ligne)	
Formateur : Anis ASSAS	

Atelier 1 : Mise en place de l'environnement du travail : Création d'une application React & Initiation à JSX

1. Installation de Node JS :

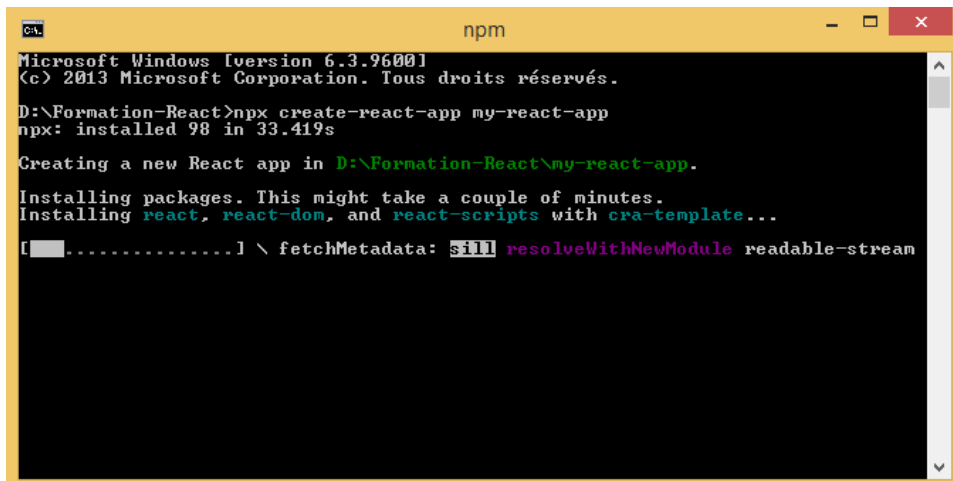
- Site : <https://nodejs.org/en/download/>
- Node JS installe l'outil npm (Node Package Manager) qui permet de télécharger et installer des bibliothèques Java Script : gestionnaire de modules de Node.
- Pour vérifier la version installée :

> **node --version** ou **node -v**

2. Création d'une application react :

- En adoptant la même arborescence que l'activité précédente, créer un nouveau dossier :
D:/formation-fullStack/ateliers/atelier1
- Sur votre ligne de commande et afin de créer l'application, taper :

> **npx create-react-app my-react-app**



```

C:\> npm
Microsoft Windows [version 6.3.9600]
(c) 2013 Microsoft Corporation. Tous droits réservés.

D:\Formation-React> npx create-react-app my-react-app
npx: installed 98 in 33.419s

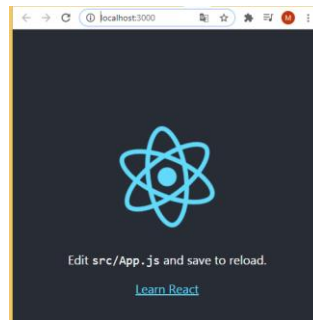
Creating a new React app in D:\Formation-React\my-react-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[ ] ..... \ fetchMetadata: sill resolveWithNewModule readable-stream

```

- Pour pouvoir lancer l'application, taper alors les deux instructions suivantes :
 - > **cd my-react-app**
 - > **npm start**
- Vérifier alors le lancement de l'application par défaut sur :
http://localhost :3000



- Ouvrir le projet qui vient d'être créé à partir de votre éditeur et vérifier la structure du code en repérant les principaux fichiers React.
- Changer l'allure de la page d'accueil par défaut en affichant à sa place un simple message de bienvenue.
- Réorganiser le rendu de votre page afin de vous présenter en insérant à titre d'exemple : votre nom et prénom, votre photo, vos contacts, des liens vers vos pages sur les réseaux sociaux, ...

3. Initiation à JSX

- Créer de nouveau(x) composant(s) ou simplement mettre à jour à chaque fois **index.js** et/ou **app.js** afin d'exécuter les exemples ci-après.

a. Le rendu des éléments : ReactDOM.render

On peut passer une expression JSX au niveau du premier paramètre de ReactDOM.render() décrivant ce que vous voulez voir à l'écran. React DOM se charge de mettre à jour le DOM afin qu'il corresponde aux éléments React.

Cette expression pourrait avoir plusieurs formes à savoir :

Exemple 1 :

```
ReactDOM.render(<h1>Bonjour </h1>, document.getElementById('root'));
```

Exemple 2 :

```
const todo = (
  <ol>
    <li>Working</li>
    <li>Go Shopping</li>
    <li>Dinner</li>
  </ol>
);
ReactDOM.render(todo, document.getElementById('root'));
```

Exemple 3 :

```
function tick(){
  const element = (
    <div>
      <h1>Bonjour !</h1>
      <h2>Aujourd'hui : {new Date().toLocaleDateString()}</h2>
      <h2>Il est {new Date().toLocaleTimeString()}</h2>
    </div>
  );
}
```

```

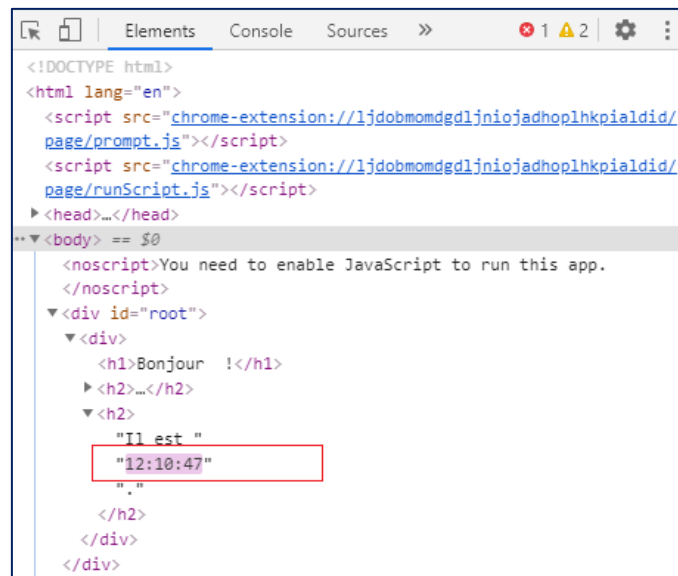
    ReactDOM.render(element, document.getElementById('root'));
  }
  setInterval(tick, 1000);
}

```

À chaque seconde, nous appelons ReactDOM.render() depuis une fonction de rappel passée à setInterval().

React met à jour le strict nécessaire. En effet, React DOM compare l'élément et ses enfants avec la version précédente, et applique uniquement les mises à jour DOM nécessaires pour refléter l'état voulu.

Vous pouvez vérifier ce comportement en inspectant le dernier exemple avec les outils de développement du navigateur :



b. Les variables et expressions

- Déclarer les variables suivantes et les afficher

```

let x="Bonjour"
var y="tout"
const z="le monde !"

```

- Que donne cette expression ?

```
const expression = <p> 1 + 1 = { 1 + 1 } </p>
```

- Déclarer des nouvelles variables : (nom, prénom, photo), les initialiser afin d'afficher leurs contenus au niveau de la page.

```

var nom = <h2> Anis </h2>
var prenom = <h2> Assas </h2>
const photo = ;

```

- Changer maintenant la structure des variables comme suit :

```
const user = {
  prenom: 'Anis',
  nom: 'Assas'
};
```

- Réafficher de nouveau la page en faisant référence à cette nouvelle variable :

```
render() {
  return (
    <div>
      Bonjour : {user.prenom}--{user.nom}
    </div>);
}
```

- Utiliser cette fois-ci une fonction formatName(user) afin de retourner la valeur à afficher.

Le code devrait normalement avoir cette allure :

```
const user = {
  prenom: 'Anis',
  nom: 'Assas'
};

function formatName(user){
  return user.prenom + '--' + user.nom;
}

render() {
  return (
    <div> Bonjour : {formatName(user)} </div>);
}
```

- Définir un nouveau style pour un affichage par l'intermédiaire d'une variable

```
var myStyle = {
  fontSize: 100,
  color: '#FF0000'
}

return (
  <div>
    <h1 style = {myStyle}>Bonjour</h1>
  </div>);
```

c. Traitement conditionnel

- Créer la fonction disBonjour(user) qui permet de tester si le prénom de l'utilisateur est le même que celui sauvegardé au niveau de la variable **user** et d'afficher par conséquent le message bonjour.

```
disBonjour(user) {
  if (user.prenom==='Anis') {
    return <h1>Bonjour, {formatName(user)} !</h1>; }
  return <h1>Bonjour Inconnu.</h1>;
}
```

Mettre à jour le code afin de tester cette fois-ci les deux attributs nom et prénom.

- Tester la deuxième forme d'écriture conditionnelle :

```
return (user.prenom==='Anis') ? (<h1>Bonjour, {formatName(user)} !</h1>) :
    (<h1>Bonjour Inconnu.</h1>);
```

d. Manipulation des listes

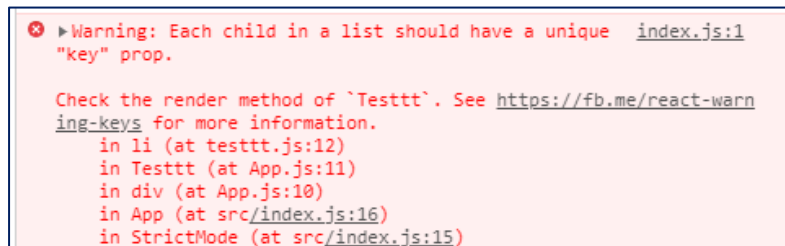
La fonction `map()` permet de parcourir les éléments d'un tableau.

- Exemple 1 : doubler les valeurs d'un tableau.

```
const numbers = [7, 12, 25, 4, 5];
const doubled = numbers.map((number) => number * 2);
```
- Exemple 2 : afficher les valeurs du tableau sous forme de liste

```
const listItems = numbers.map((number) => <li>{number}</li>);
```

Remarque : Si vous vérifiez la console, vous allez trouver un message d'avertissement (warning) :



Vous devez donner alors une clé à chaque élément dans un tableau afin d'apporter aux éléments une identité stable. Les clés aident React à identifier quels éléments d'une liste ont été ajoutés, modifiés ou supprimés.

Le meilleur moyen de choisir une clé est d'utiliser quelque chose qui identifie de façon unique un élément d'une liste parmi ses voisins. Le plus souvent on utilise ceci.

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

e. Autres manipulations :

- Écrire le code utilisant la fonction **reduce** permettant de calculer et afficher la valeur maximale présente dans ce tableau.

Solution :

```
// (version 1)
const max = numbers.reduce((a, b) => b > a ? b : a);
// (version 2)
const max = numbers.reduce((a, b) => Math.max(a, b));
```

- Construire un nouveau tableau en filtrant toutes les valeurs paires du tableau, et en transformant chaque valeur restante en son carré.

Solution :

```
const tab = numbers.filter(v => v % 2 !== 0).map(v => <li>{v * v}</li>);
```