# Chapitre 2 : Le langage JSX

# Elément React

- Les éléments sont les blocs élémentaires d'une application React.
- Un élément décrit ce qu'on veut voir à l'écran :

```
const element = <h1>Bonjour tout le monde</h1>;
```

- Contrairement aux éléments DOM d'un navigateur, les éléments React sont de simples objets peu coûteux à créer.
- React DOM se charge de mettre à jour le DOM afin qu'il corresponde aux éléments React.

# Afficher un élément dans le DOM 1/3

- On met en place une balise  dans le fichier « public/index.html ».
- C'est l'élément de la page dans lequel on va injecter l'application React.
- Il n'y a aucune contrainte sur cet élément : ce peut être n'importe quel élément HTML tant qu'on peut le retrouver en JavaScript.
- En quelque sorte, ce fichier HTML est le point d'entrée de l'application puisque c'est lui qui est affiché lorsque l'utilisateur ouvre l'application dans son navigateur :

```
<div id="root"></div>
```

- On parle de nœud DOM « racine » car tout ce qu'il contient sera géré par React DOM.

# Afficher un élément dans le DOM 2/3

- Les applications développées uniquement avec React ont généralement un seul nœud DOM racine.
- Si on intègre React dans une application existante, on peut avoir autant de nœuds DOM racines isolés qu'on le souhaite.
- « src/index.js » :

```
import React from 'react'
import ReactDOM from 'react-dom '
```

- Tout d'abord, on importe react. Puis on importe ensuite react-dom, qui va nous donner accès à la méthode render.

# Afficher un élément dans le DOM 3/3

- Ensuite, nous récupérons la div principale de l'application grâce à son ID.
- Puis nous demandons à ReactDOM.render de générer le rendu de l'application dans cette div.

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

*Cet exemple de code affichera « Bonjour » sur la page.*

- Une autre écriture de code donne également le même résultat :

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

# Récupérer les composants 1/3

- Quand vous créez un projet React avec create-react-app, il y a deux fichiers principaux, le fichier public/index.html et le fichier src/index.js.

- Dans le fichier public/index.html c'est là qu'est définie la structure de base d'une page HTML, plus important il y est défini l'id root :

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

- C'est en effet dans cette div que notre application sera affichée.

# Récupérer les composants 2/3

- Le fichier src/index.js va permettre de récupérer nos composants et de les afficher dans la div#root de public/index.html.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

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

- import App from './App' : nous importons le ficher src/App.js
- ReactDOM.render(<App />, document.getElementById('root')) : le composant <App /> est affiché dans la div#root de public/index.html.

# Récupérer les composants 3/3

- Nous allons définir l'interface de notre application dans le fichier src/App.js.
- Nous allons y dire comment les éléments de notre page vont être affichés, où est-ce que la barre de navigation sera affichée, les couleurs des boutons, etc.
- Puis le fichier src/index.js va récupérer ces informations et les renvoyer au fichier public/index.html pour que celui-ci puisse les afficher dans la div#root.

# Mettre à jour un élément affiché 1/4

- Les éléments React sont immuables.
- Une fois votre élément créé, vous ne pouvez plus modifier ses enfants ou ses attributs.
- Un élément est comme une image d'un film à un instant T : il représente l'interface utilisateur à un point précis dans le temps.
- La seule façon de mettre à jour l'interface utilisateur est de créer un nouvel élément et de le passer à ReactDOM.render().

# Mettre à jour un élément affiché 2/4

**Exemple** :

- *index.js*

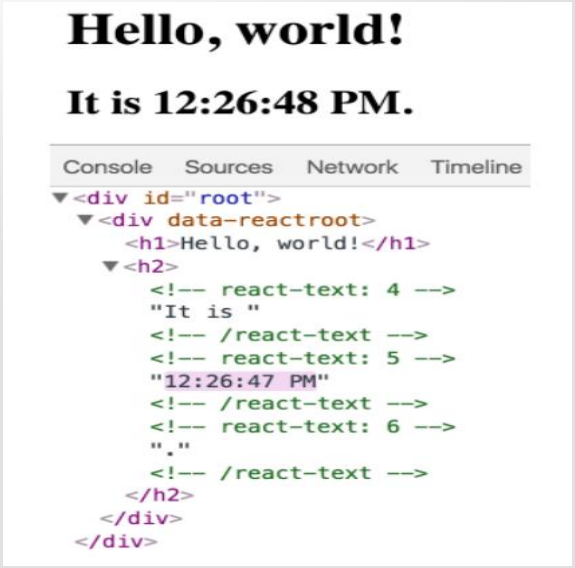
```
import React from 'react';
import ReactDOM from 'react-dom';
function tick() {
  const element = (
    <div>
      <h1>Hello, world !</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));}
  setInterval(tick, 1000);
```

- *Index.html*

```
<body>  <div id="root"></div>  </body>
```

# Mettre à jour un élément affiché 3/4

Si on voit le résultat dans le navigateur, *dans localhost:3000* à chaque seconde, ReactDOM.render() est appelé depuis une fonction de rappel passée à setInterval().




# Mettre à jour un élément affiché 4/4

- À chaque seconde, nous appellons ReactDOM.render() depuis une fonction de rappel passée à setInterval().
- 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.
- Même si nous créons à chaque seconde un élément décrivant l'arborescence complète de l'interface utilisateur, seul le nœud texte dont le contenu a été modifié est mis à jour par React DOM.

# Définition JSX

- JSX est une extension de syntaxe de JavaScript standard qui est utilisée pour créer des éléments React.
- Ces éléments sont ensuite rendus dans le DOM React.
- JSX est une étape de préprocesseur qui ajoute la syntaxe XML à JavaScript.
- Tout comme XML, les balises JSX ont un nom de balise, des attributs et des enfants.
- Au lieu de séparer artificiellement les technologies en mettant le balisage et la logique dans des fichiers séparés, React sépare les préoccupations via des unités faiblement couplées appelées « composants », qui contiennent les deux.

# Avantages de JSX

- Il est plus rapide que JavaScript car il effectue des optimisations lors de la conversion en JavaScript standard.
- Il est plus facile de créer des modèles.
- Au lieu de séparer le balisage et la logique dans des fichiers séparés, React utilise des composants à cet effet.

# Premier exemple JSX

- JSX fait sûrement penser à un langage de balisage, mais il recèle toute la puissance de JavaScript.
- En effet, JSX produit des « éléments » React qui seront retranscrits dans le DOM.
- Dans l'exemple suivant, nous déclarons une constante appelée element ayant comme contenu la balise titre h1,

- *Index.js*

```
import React from 'react';
import ReactDOM from 'react-dom';
const element = <h1>Bonjour tout le monde !</h1>;
ReactDOM.render(myelement, document.getElementById('root'));
```

- *Index.html*

```
<body>
    <div id="root"></div>
</body>
```

- *Affiche dans localhost:3000* **Bonjour tout le monde !**

# Éléments imbriqués dans JSX 1/3

- Si on souhaite rendre plusieurs balises à la fois, on doit envelopper toute cette balise sous une balise parente, puis rendre cet élément parent au format HTML.
- Toutes les sous-étiquettes sont appelées balises enfant ou enfant de cet élément parent.
- Ici, nous utilisons div comme élément conteneur contenant trois éléments imbriqués à l'intérieur.

```
const element =
    <div>
            <h1>Informatique</h1>
            <h2>Gestion</h2>
            <p>Génie civil.</p>
     </div>
```

- On peut mettre des parenthèses :

```
const element = (
  <div>  <h1>Bonjour !</h1>  <h2>Bienvenue à notre page.</h2>
  </div>  );
```

# Éléments imbriqués dans JSX 2/3

**Exemple :**

- *index.js*

```
import React from 'react';
import ReactDOM from 'react-dom';
const elt = (
  <ul>
    <li>DSI</li>
    <li>RSI</li>
    <li>SEM</li>
  </ul>
);
ReactDOM.render(elt, document.getElementById('root'));
```

# Éléments imbriqués dans JSX 3/3

- *index.html*

```
<body>
    <div id="root"></div>
</body>
```

*Donne le résultat dans loclahost:3000*

- **DSI**
- **RSI**
- **SEM**

# Les commentaires

- JSX nous permet d'utiliser des commentaires car il nous permet d'utiliser des expressions JavaScript.
- Les commentaires dans JSX commencent par / * et se terminent par * /.
- On peut ajouter des commentaires dans JSX en les enveloppant entre accolades {} comme on le fait dans le cas des expressions.

```
const element = <div><h1>Hello World !</h1>
            {/ * Il s’agit d’un commentaire * /}
               </div>;

ReactDOM.render(element, document.getElementById("root"));
```

# Balise vide

- Si une balise est vide, on peut la fermer immédiatement avec />, comme en XML :

```
const element = <img src={user.avatarUrl} />;
```

# Utiliser des expressions dans JSX 1/5

- Dans l'exemple suivant, nous déclarons une variable appelée name et nous l'utilisons ensuite dans JSX en l'encadrant avec des accolades :

```
const name = 'Mohamed Tounsi';
const element = <h1>Bonjour, {name}</h1>;
ReactDOM.render(element,document.getElementById('root'));
```

*Affiche dans localhost:3000* **Bonjour, Mohamed Tounsi**

# Utiliser des expressions dans JSX 2/5

- On peut utiliser n'importe quelle expression JavaScript valide dans des accolades en JSX.
- Par exemple, 2 + 2, user.firstName, ou formatName(user) sont toutes des expressions JavaScript valides.

```
import React from 'react';
import ReactDOM from 'react-dom';
const myelement = <h1>la somme de 5+5 est {5 + 5} </h1>;
ReactDOM.render(myelement, document.getElementById('root'));
```

*Affiche dans localhost:3000* **la somme de 5+5 est 10**

# Utiliser des expressions dans JSX 3/5

- Dans l'exemple suivant, on intègre le résultat de l'appel d'une fonction JavaScript : formatName(user), dans un élément <h1>.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}
const user = {
  firstName: 'Mohamed',
  lastName: 'Tounsi'
};
const element = (
  <h1> Bonjour, {formatName(user)} !  </h1>
);
ReactDOM.render(element,document.getElementById('root'));
```

*Affiche dans localhost:3000* **Bonjour, Mohamed Tounsi**

# Utiliser des expressions dans JSX 4/5

- Après la compilation, les expressions JSX deviennent de simples appels de fonctions JavaScript dont l'évaluation renvoie des objets JavaScript.
- Ça signifie qu'on peut utiliser JSX à l'intérieur d'instructions if ou de boucles for, l'affecter à des variables, l'accepter en tant qu'argument, et le renvoyer depuis des fonctions :

```
function getGreeting(user) {
  if (user) {
    return <h1>Bonjour, {formatName(user)} !</h1>;
        }
  return <h1> Utilisateur Inconnu.</h1>;
}
```

# Utiliser des expressions dans JSX 5/5

- On incorpore n'importe quelle expression JavaScript dans JSX en les enveloppant entre accolades à l'exception des instructions if-else.
- Mais on peut utiliser des instructions conditionnelles au lieu des instructions if-else dans JSX.
- Voici l'exemple où l'expression conditionnelle est intégrée dans JSX :

```
const element = <h1>{ (i == 1) ? 'Hello World!' : 'False!' } < /h1>;
ReactDOM.render(
    element,
    document.getElementById("root")
);
```

# Spécifier des attributs en JSX

- JSX nous permet d'utiliser des attributs avec les éléments HTML comme nous le faisons avec du HTML normal.
- On peut utiliser des guillemets pour spécifier des littérales chaînes de caractères dans les attributs.

```
const element = <div tabIndex="0"></div>;
```

- Mais on peut aussi utiliser des accolades pour utiliser une expression JavaScript dans un attribut :

```
const element = <img src={user.avatarUrl}></img>;
```

- Il ne faut pas mettre de guillemets autour des accolades quand on utilise une expression JavaScript dans un attribut.
- On peut utiliser soit des guillemets (pour des valeurs textuelles) soit des accolades (pour des expressions), mais pas les deux à la fois pour un même attribut.

# className 1/3

- Dans la mesure où JSX est plus proche de JavaScript que de HTML, React DOM utilise la casse camelCase comme convention de nommage des propriétés, au lieu des noms d'attributs HTML.
- Par exemple, class devient className en JSX, et tabindex devient tabIndex.
- La principale raison derrière cela est que certains des noms d'attribut en HTML comme «classe» sont des mots-clés réservés dans JavaScripts.

# className 2/3

**Exemple** :

- *mystyle.module.css*

```
.bigblue { color: DodgerBlue;
           padding: 40px;
           font-family: Arial; }
```

- *App.js*

```
import React from 'react';
import ReactDOM from 'react-dom';
import styles from './mystyle.module.css';
class Personne extends React.Component {
  render() {
    return <h1 className={styles.bigblue}>Hello Mohamed !</h1>;
  }
}
export default Personne;
```

# className 3/3

- *index.js*

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

- *index.html*

```
<!DOCTYPE html>
<html>
  <head> <title>React App</title> </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

*Affiche dans localhost:3000*

Hello Mohamed !

# Style en ligne

- React recommande aussi d'utiliser des styles en ligne. Nous devons utiliser la syntaxe camelCase.
- React ajoutera également automatiquement px après la valeur numérique sur des éléments spécifiques.
- L'exemple suivant montre comment ajouter myStyle en ligne à l'élément h1.

```
import React from 'react';
class App extends React.Component {
render() { var myStyle = {
          fontSize: 100,
          color: '#FF0000' }
          return ( <div> <h1 style = {myStyle}>Hello</h1> </div> );
          }
}
export default App;
```

*Affiche dans localhost:3000* Hello