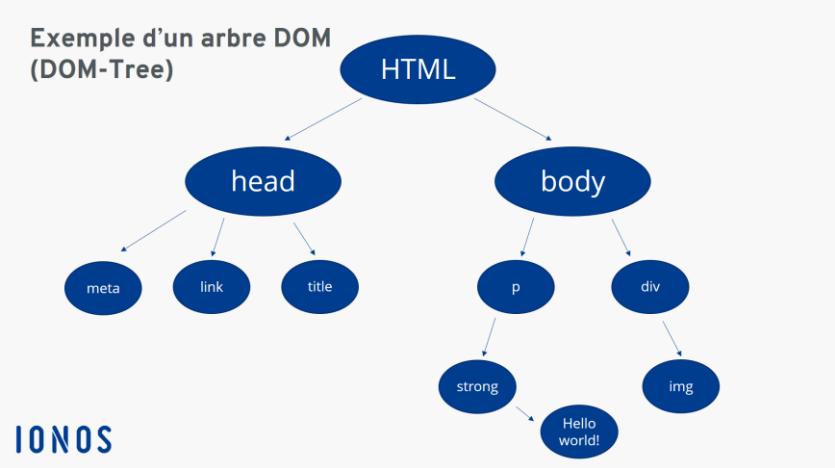
Chapitre 3 : JSX

Table des matières

I. DOM virtual

a. Définition du DOM

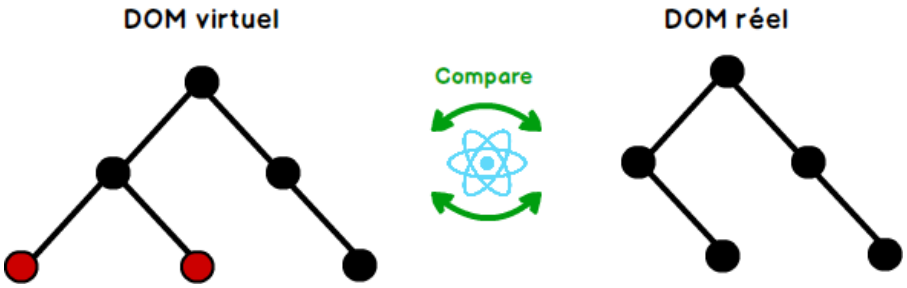
L'objectif du Document Object Model (DOM) est d'offrir aux programmeurs un **accès aux différents composants d'une page Web**, pour ainsi faciliter au mieux l'insertion, la suppression ou l'édition des contenus, d'attributs et de styles.



b. DOM virtuel

Le DOM virtuel est une représentation en mémoire des éléments DOM réels générés par les composants React avant toute modification de la page.

React crée une arborescence d'objets personnalisés représentant une partie du DOM. Par exemple, au lieu de créer un élément DIV contenant un élément UL, il crée un objet React.div contenant un objet React.ul. Il peut manipuler ces objets très rapidement sans toucher au vrai DOM ni passer par l'API du DOM. Ensuite, lors du rendu d'un composant, il utilise ce DOM virtuel pour déterminer ce qu'il doit faire avec le DOM réel pour que les deux arbres correspondent.



c. Le module ReactDOM

ReactDOM est un paquet qui fournit des méthodes spécifiques à DOM qui peuvent être utilisées au niveau supérieur d'une application web pour permettre une gestion efficace des éléments DOM de la page web. Virtual DOM améliore à la fois l'expérience des utilisateurs et le travail du développeur

La méthode render du module ReactDOM est l'une des méthodes les plus importantes de ReactDOM : Cette méthode est utilisée pour rendre un seul composant React ou plusieurs composants enveloppés ensemble dans un composant ou un élément div. Cette fonction utilise les méthodes efficaces de React pour la mise à jour du DOM.

II. JSX

a. Créer des éléments avec React

Pour créer des éléments HTML avec React, nous utilisons la methode React.createElement. Cette méthode accepte 3 paramètres : **type** , **les attributs** , **les enfants**.

- **Type** : Type de l'élément ou du composant html, par exemple : h1, h2, p, div …
- **Les attributs** : Les propriétés de l'élément par exemple style ou le nom de la classe.
- **Les enfants** : Tout ce que vous devez faire passer entre les éléments du dom.

Exemple 1 : Exemple de création de l'élément h1 dans le composants app.js :

```
import React from 'react';

function App() {
  return React.createElement('h1',{},'Premier Exemple');
}

export default App;
```

Exemple 2 : Exemple de création de l'élément h1 en spécifiant le style :

```
import React from "react";

function App() {
  let welcome = React.createElement(
```

```
    "h1",
    { style: { color: "red" } },
    'Welcome to react world'
  );
  return welcome;
}

export default App;
```

Exemple 3 : Exemple de création de l'élément h1 dans l'élément div :

```
import React from "react";

function App() {
  const title = React.createElement("h1", {}, "My First React Code");
  const container = React.createElement("div", {}, title);

  return container;
}

export default App;
```

Exemple 4 : Exemple de création des éléments h1 et p dans un élément div

```
import React from "react";

function App() {
  const title = React.createElement('h1', {}, 'My First React Code');
  const paragraph = React.createElement('p', {}, 'Writing some more
HTML. Cool stuff!');
  const container = React.createElement('div', {}, title, paragraph);

  return container;
}

export default App;
```

Exemple 5 : Exemple de création des éléments h1 et ul dans un élément div

```
import React from "react";

function App() {
  const container = React.createElement(
    "div",
    {},
    React.createElement("h1", {}, "My favorite ice cream flavors"),
```

```
    React.createElement("ul", {}, [
      React.createElement("li", { className: "brown" }, "Chocolate"),
      React.createElement("li", { className: "white" }, "Vanilla"),
      React.createElement("li", { className: "yellow" }, "Banana"),
    ])
  );

  return container;
}

export default App;
```

b. Définition

JSX est l'abréviation de JavaScript XML.

JSX est un format d'écriture qui va nous permettre d'écrire un semblant de HTML dans nos composants React.

Exemple :

```
// Sans JSX
React.createElement(
  "div",
  null,
  React.createElement("h2", null, "Title List"),
  React.createElement("p", null, "Contenu")
);

// Avec JSX
const element = () => (
  <div>
      <h2>Title List</h2>
      <p>Contenu</p>
  </div>
);
```

Le code JSX est facile à lire et à écrire. Il s'écrit comme du code HTML, mais il est saisi dans la partie réservée au code JavaScript. Une même instruction peut s'écrire sur plusieurs lignes et doit obligatoirement commencer par une balise ouvrante et se terminer par balise fermante.

c. JSX expressions

Jusqu'à présent, nous n'avons utilisé que des balises HTML dans le cadre de JSX. Mais JSX devient plus utile lorsque nous y ajoutons réellement du code JavaScript. Pour ajouter du code JavaScript à l'intérieur de JSX, nous devons l'écrire entre crochets comme ceci :

```
import React from "react";

function App() {
  const number = 10;
  return (
    <div>
      <p>Number: {number}</p>
    </div>
  );
}

export default App;
```

Ainsi, cette syntaxe d'utilisation des accolades est souvent connue sous le nom de **d'expression JSX**.

Voici les éléments valides que vous pouvez avoir dans une expression JSX :

- Une chaine de caractères comme "hello"
- Un nombre comme 10
- Un tableau comme [1, 2, 4, 5]
- Une propriété d'objet qui sera évaluée à une certaine valeur
- Appel d'une fonction

Voici un exemple d'affichage des propriétés d'un objet :

```
import React from "react";

function App() {
  const p = {code:1,pname:"produit1",prix:23};
  return (
    <div>
      <h1>Code produit = {p.code}</h1>
      <h1>Nom produit = {p.pname}</h1>
      <h1>Prix produit = {p.prix}</h1>
    </div>
  );
```

```
}

export default App;
```

Utiliser des parenthèses en début et en fin du code JSX : Lorsqu'on retourne un code JSX sur plusieurs lignes (par exemple un élément suivi de plusieurs éléments), l'instruction return doit comporter à la suite, sur la même ligne, le premier élément JSX retourné, sinon une erreur se produit. Cela oblige à décaler vers la droite le code JSX du premier élément retourné.

- Afficher une liste d'éléments sans utiliser des parenthèses

```
function App() {
  return <ul>
    <li>Element 1</li>
    <li>Element 2</li>
    <li>Element 3</li>
    <li>Element 4</li>
  </ul>
}
```

- Afficher une liste d'éléments avec des parenthèses

```
function App() {
  return (
  <ul>
    <li>Element 1</li>
    <li>Element 2</li>
    <li>Element 3</li>
    <li>Element 4</li>
  </ul>
  )
}
```

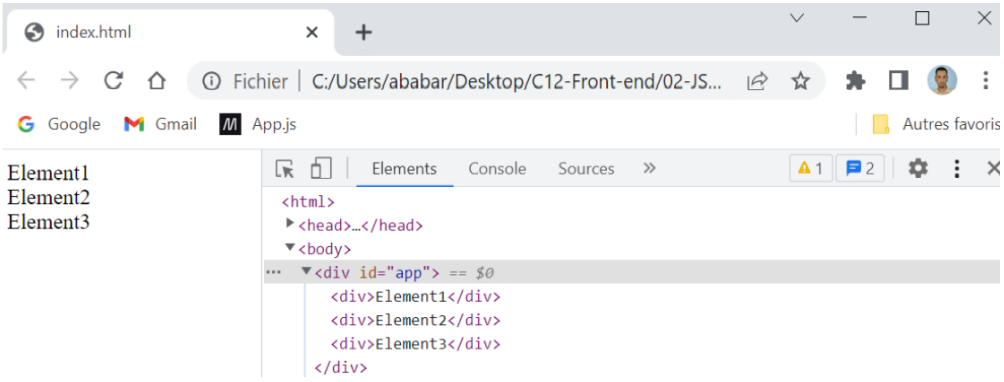
Utiliser un fragment avec le composant : L'ajout d'un parent, tel qu'un élément, fonctionne lorsqu'on souhaite encapsuler plusieurs éléments retournés dans un seul. L'inconvénient de cette solution est que cela ajoute un élément supplémentaire dans le code JSX, sans que cela soit vraiment nécessaire pour l'application React.

Pour cela, React propose un composant spécifique appelé que l'on peut utiliser pour ces cas-là.

Utilisons le composant pour englober un ensemble de trois éléments sans parents. L'élément va devenir le parent des trois éléments, sans apparaître pour autant dans l'arborescence des éléments React.

```
function ListeElements(props) {
    return <React.Fragment>
        <div>Element1</div>
        <div>Element2</div>
        <div>Element3</div>
    </React.Fragment>
}
```

L'élément permet de retourner un seul parent, en évitant l'ajout d'un nouvel élément parent non nécessaire. Remarquez que React ne visualise pas l'élément dans l'arborescence des éléments React.



Appel d'une fonction JavaScript : Voici un exemple d'appel d'une fonction JavaScript dans une expression JSX :

```
import React from "react";

function somme(a,b){
  return a+b;
}

function App() {
  const x = 3, y = 5;
  return (
    <div>
      <h1>La somme = {somme(x,y)}</h1>
    </div>
  );
}

export default App;
```

Opérateurs conditionnels dans les expressions JSX : Nous ne pouvons pas écrire de conditions if dans les expressions JSX, ce qui peut vous sembler un problème. Mais React nous permet d'écrire des opérateurs conditionnels, comme les opérateurs ternaires ainsi que l'opérateur logique de court-circuit && comme ceci :

```
import React from "react";

function App() {
  const number = 10;
  return (
    <div>
      {number > 0 ? (
        <p>Number {number} is positive</p>
      ) : (
        <p>Number {number} is Negative</p>
      )}
    </div>
  );
}

export default App;
```

Voici un autre exemple de l'utilisation de l'expression conditionnelle dans le code JSX :

```
function ListeElements(props) {
    return (
        <ul>
            { props.hideFirstItem ? null : <li>Element1</li> }
            <li>Element2</li>
            <li>Element3</li>
            <li>Element4</li>
            <li>Element5</li>
        </ul>
    )
}
```

Spécifier les classes CSS en JSX : Nous pouvons ajouter des attributs aux éléments JSX, par exemple id et class, de la même manière qu'en HTML.

```
import React from "react";

function App() {
  const id = "some-id";
  return (
    <div>
      <h1 id={id}>This is a heading</h1>
```

```
      <h2 className="active">This is another heading</h2>
    </div>
  );
}

export default App;
```

Notez que dans React, nous devons utiliser className au lieu de class. L'utilisation de l'attribut class génère une avertissement (Warning).

Les commentaires en JSX : Si vous avez une ligne de code comme celle-ci : `<p>This is some text</p>`, et que vous voulez ajouter un commentaire pour ce code, vous devez alors envelopper ce code dans la syntaxe d'expression JSX à l'intérieur des symboles de commentaire /* et */ comme ceci : `{/* <p>This is some text</p> */}`

Les styles en JSX : Pour donner du style à un élément avec l'attribut style, la valeur doit être un objet JavaScript :

```
<h1 style={{color: "red"}}>Hello Style!</h1>
```

Voici un autre exemple d'ajout des styles a un élément a l'aide d'expression JSX :

```
var color = "red";
var styleListe = { listStyleType:"none", color:color };
var liste = <ul id="list1" style={styleListe}>
                <li> Element1 </li>
                <li> Element2 </li>
                <li> Element3 </li>
                <li> Element4 </li>
                <li> Element5 </li>
            </ul>;
```

Retourner un type complexe : Considérant le code suivant qui représente le composant App qui retourne 2 paragraphes.

```
import React from "react";

function App() {
  return (
    <p>This is first JSX Element!</p>
    <p>This is another JSX Element</p>
);
}

export default App;
```

L'exécution de ce code génère une erreur qui indique qu'un composant ne peut pas retourner plusieurs éléments.

La correction du code précèdent est de rassembler les deux paragraphes dans une div, comme le montre le code suivant :

```
import React from "react";

function App() {
  return (
      <div>
          <p>This is first JSX Element!</p>
          <p>This is another JSX Element</p>
      </div>
);
}
export default App;
```