

5. Composant en React

5.1. Introduction :

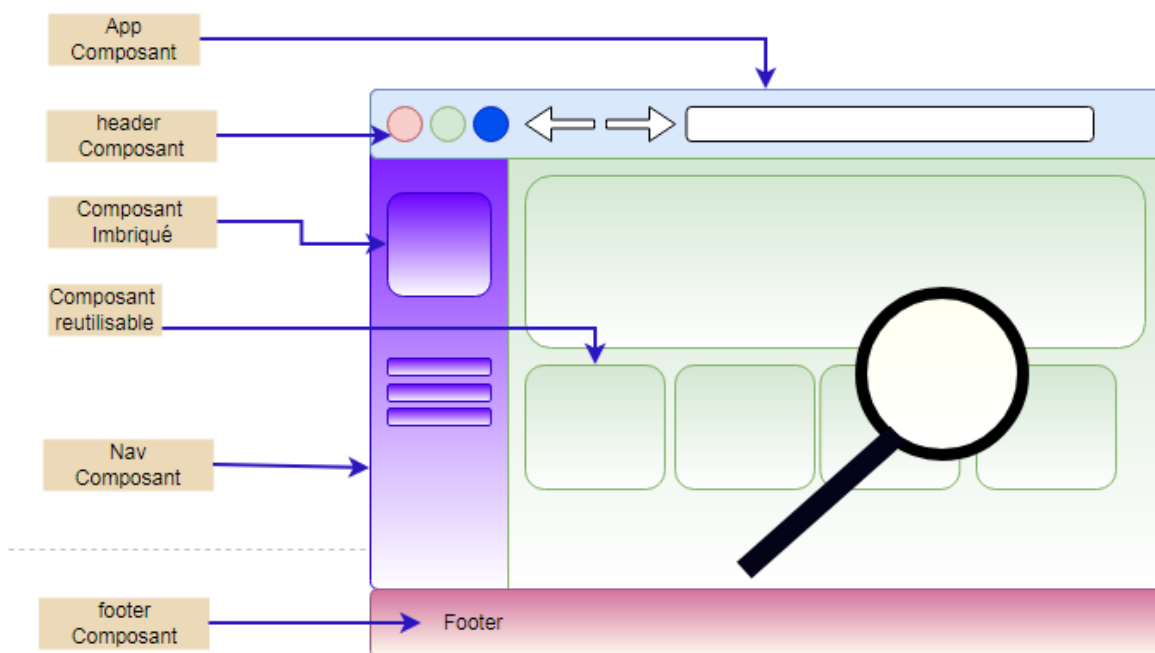
Chaque application React est constitué d'un ensemble de composants.

Avant de construire une Application React on doit identifier les parties qui constituent des composants réutilisables.

Avec les composants vous allez pouvoir avec un même ensemble de code regrouper la structure les styles et les comportements, et vous pouvez utiliser vos composants autant que vous voulez en vous faisant gagner beaucoup du temps.

Dans cette séance vous allez découvrir comment créer des éléments web avec des composants React

On va voir aussi les spécificités du JSX, notamment comment combiner plusieurs composants et comment utiliser des expressions java script directement dans les composants



5.2. Prise en main création d'une application React

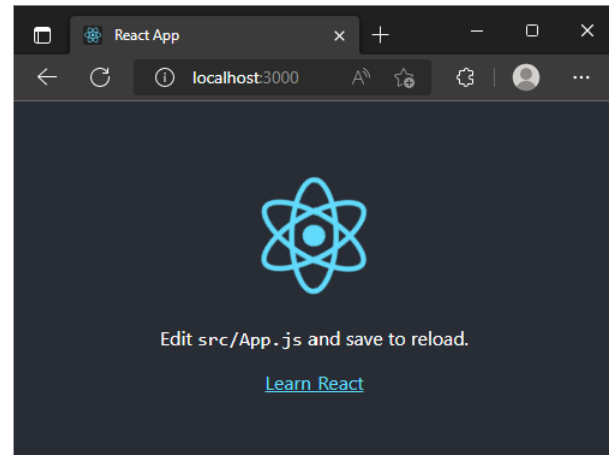
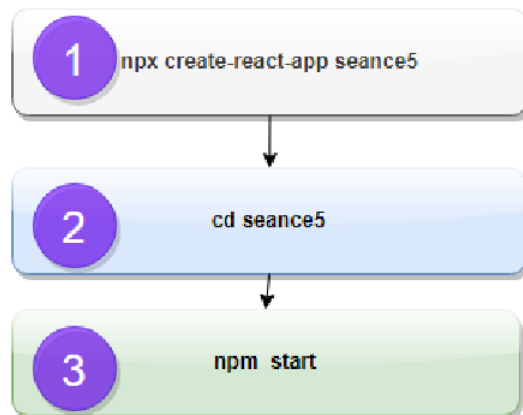
Après avoir créer un projet React voir séance 3

Création de l'application seance5

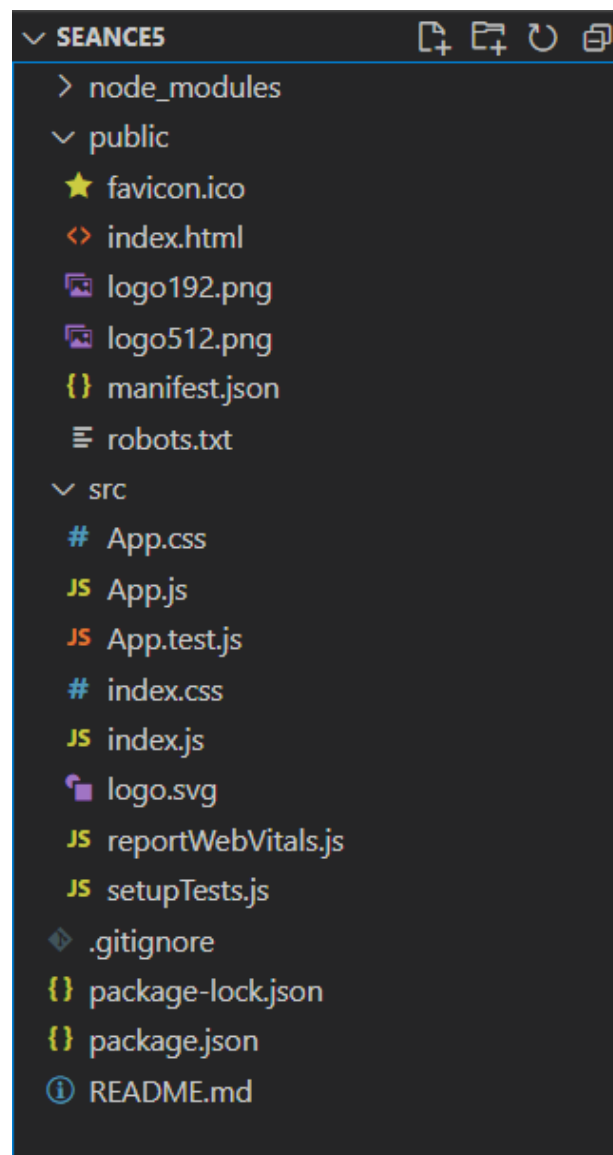
```
npx create-react-app seance5
```

```
Creating a new React app in seance5.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
[ ] | idealTree:webpack-dev-server: sill fetch manifest emojis-list@^3.0.0
```

Etapes à suivre

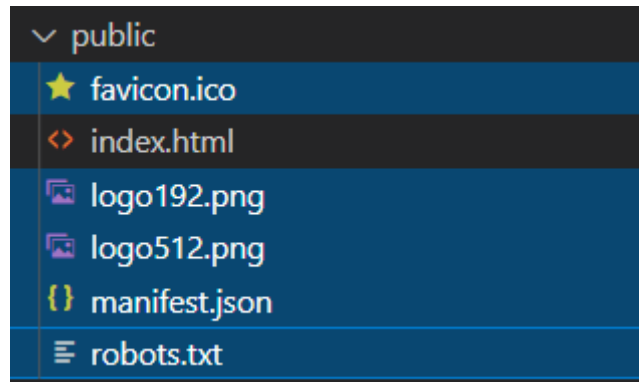


Structure d'une application React :

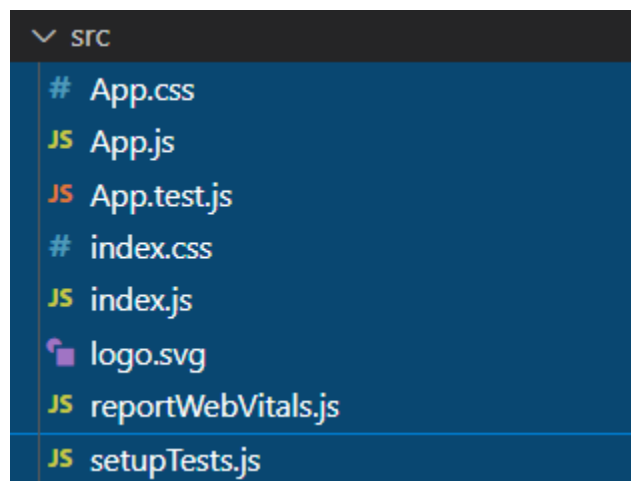


Pour vocation pédagogique, et pour démarrer un projet React avec le minimum de fichiers. dans un premier temps on va :

- Supprimer tous les fichiers qui se trouvent dans le dossier src
- Supprimer tous les fichiers qui se trouvent dans le dossier public sauf le fichier index.html

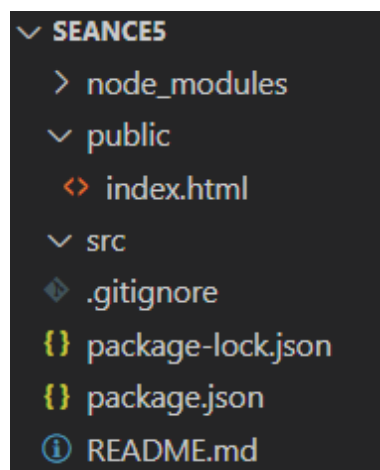


- Supprimer les fichiers sélectionnés de dossier public



- Supprimer tous les fichiers du dossier src

La structure de projet après modification



Dans le dossier src ajouter le fichier index.js

Voir contenu du fichier index.js

```
// 1) importer React et ReactDOM
import React from 'react';
import ReactDOM from 'react-dom/client'

//2) faire reference à div id=root de index.html

const element=document.getElementById("root");
//3) prendre le controle de l'element par React

const root=ReactDOM.createRoot(element)
// 4) Creation d'un composant (component)

//un composant est une fonction qui retourne jsx

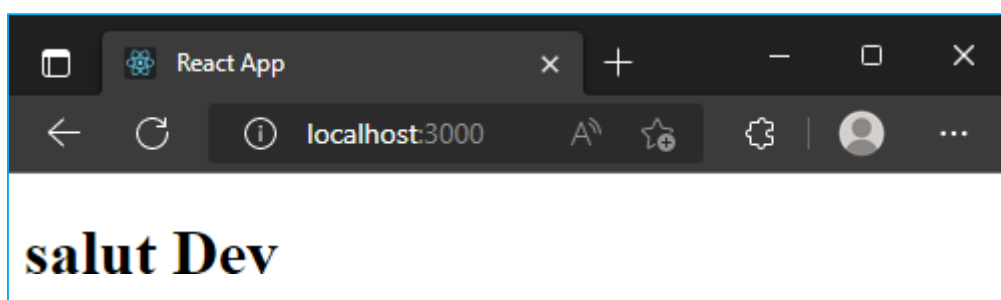
function App(){

  return(
    <h1>salut Dev</h1>
  )
}

// 5) afficher le composant dans le browser

root.render(<App/>)
```

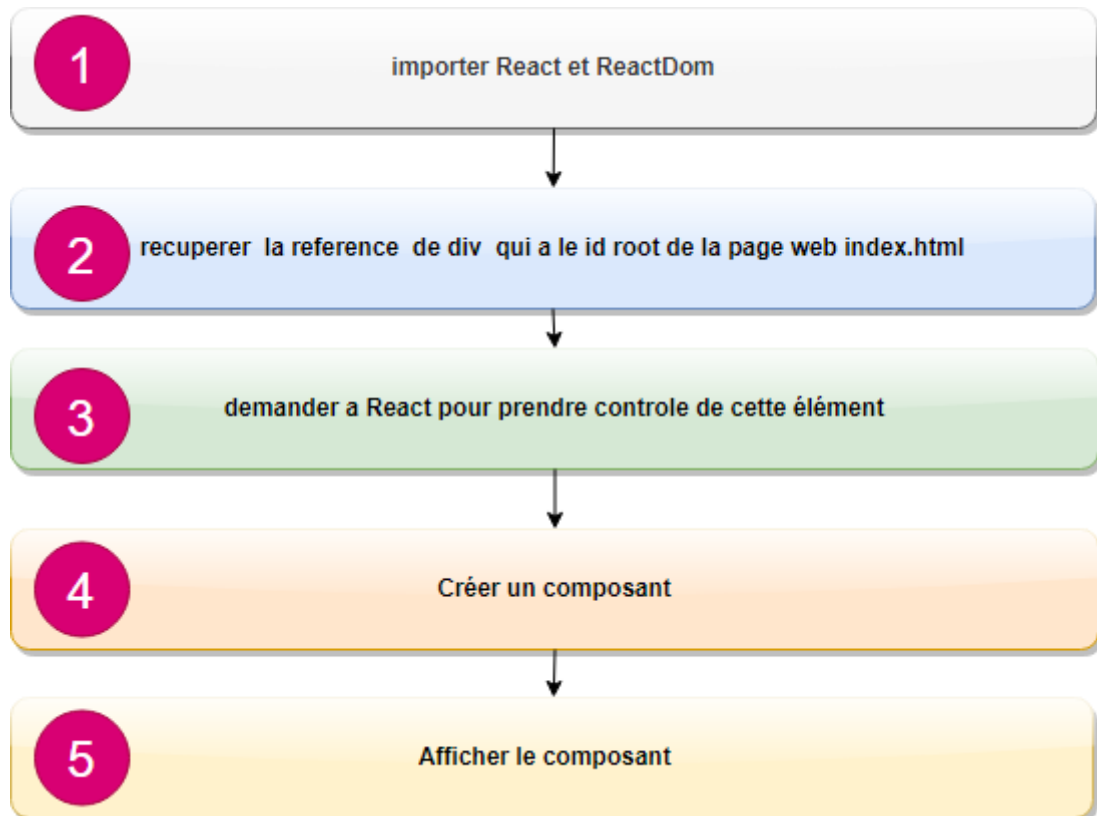
Le rendu sera :



5.3. Explication des étapes pour créer un composant :

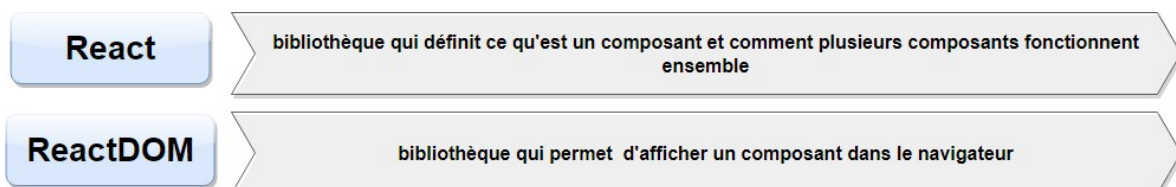
Un composant est **fonction** qui retourne de JSX

Ou une **classe** qui hérite de la classe **React.Component** possédant la fonction render qui retourne du JSX



Etape 1 :

```
// 1) importer React et ReactDOM
import React from 'react';
import ReactDOM from 'react-dom/client'
```



Le package react-dom/client fournit des méthodes spécifiques au client utilisées pour initialiser une application sur le client. La plupart de vos composants ne devraient pas avoir besoin d'utiliser ce module.

Etape 2 :

```
//2) faire reference à div id=root de index.html
```

```
const element=document.getElementById("root");
```

on crée une référence vers l'élément div id='root' de la page public/index.html (la page index.html contient `<div id= 'root'></div>`)

C'est dans cette élément que React va injecter le code HTML

Etape 3 :

```
//3) prendre le controle de l'element par React
```

```
const root=ReactDOM.createRoot(element)
```

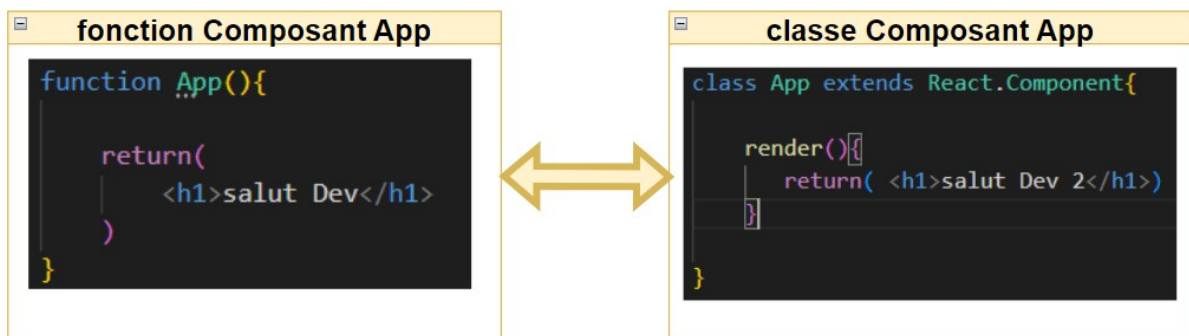
On prend le contrôle de l'élément div id='root' de la page public/index.html par React

Créez une racine React pour le conteneur fourni et renvoyez la racine. La racine peut être utilisée pour restituer un élément React dans le DOM avec render :

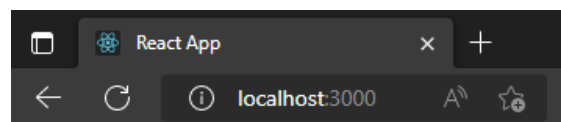
```
root.render(<App/>)
```

Etape 4

On crée un composant qui s'appelle App, comme on avait expliqué avant un composant est une fonction qui retourne de JSX, il peut être aussi une classe qui hérite de la classe React.Component possédant la fonction render qui retourne du JSX,



Remarque : Quand vous remplacez la fonction App par la classe App vous allez avoir le résultat



salut Dev 2

Pour une bonne organisation de l'application chaque composant doit être dans un fichier js portant le même nom de composant.

Etape 5

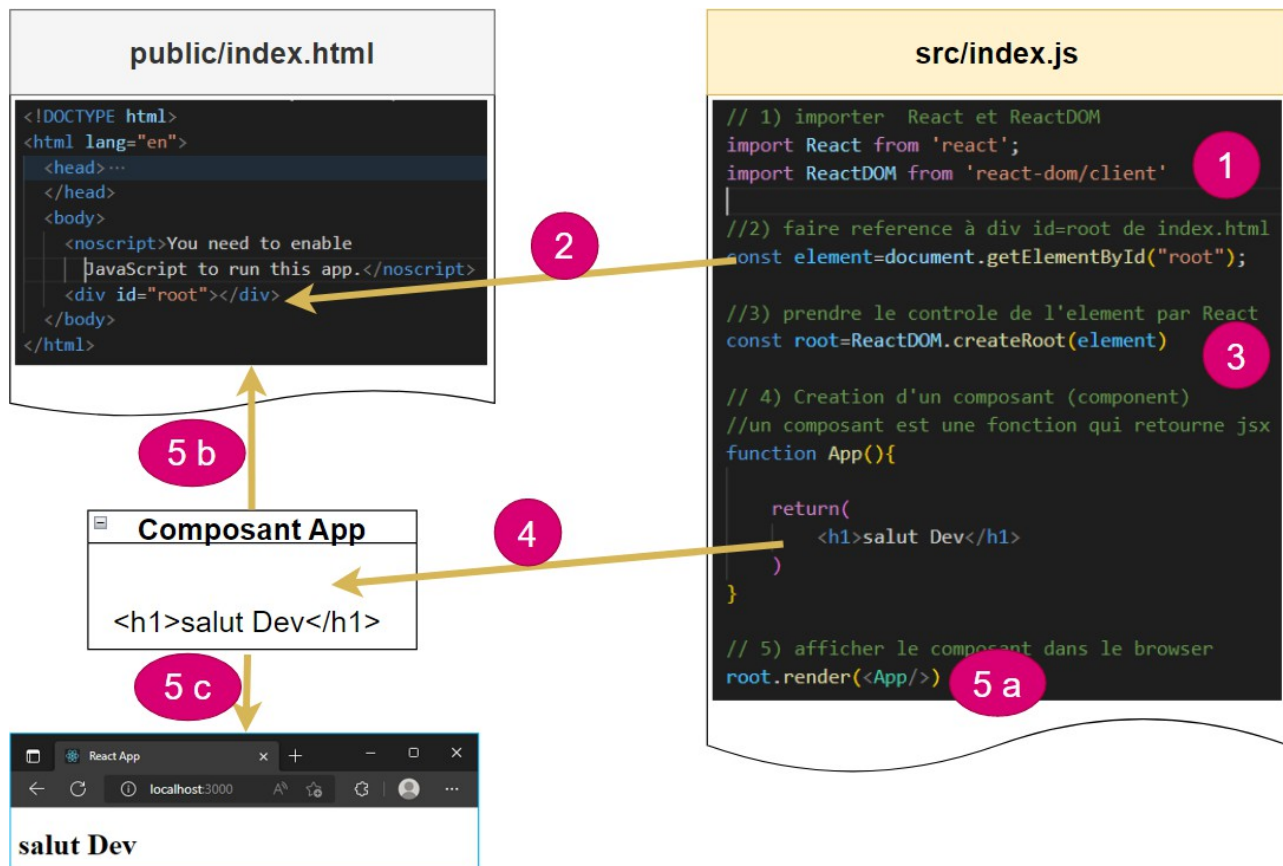
```
// 5) afficher le composant dans le browser
```

```
root.render(<App/>)
```

on passe App composant comme JSX élément a la méthode root.render()

permet d'afficher le contenu du composant App dans le <div id='root'></div> de la page indx.html

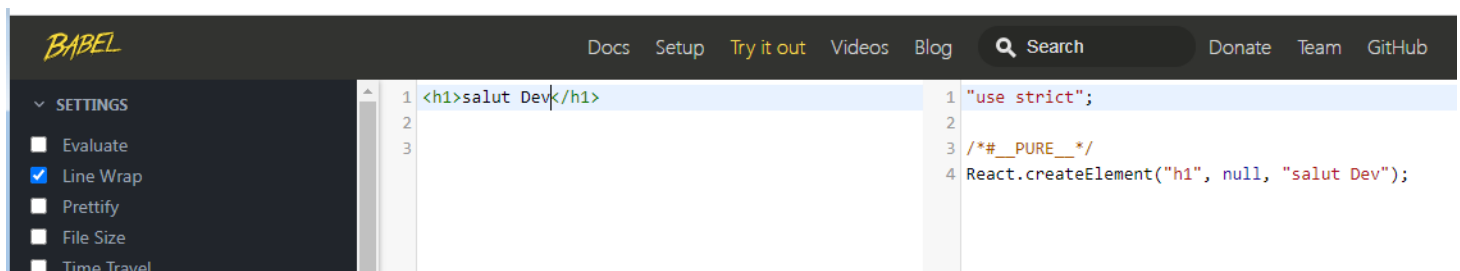
Comment ça marche

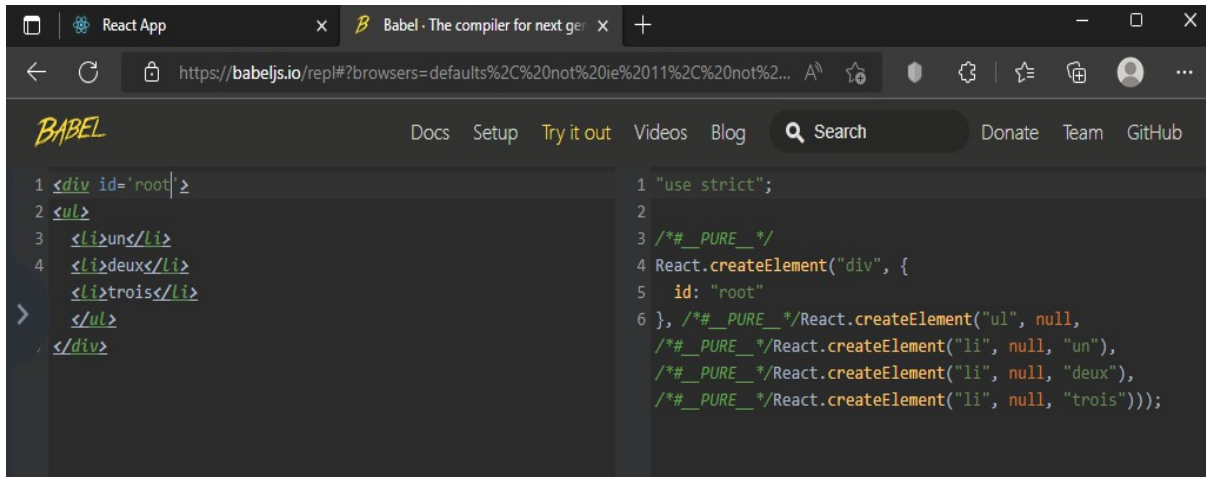


Comment se fait l'interprétation de JSX

Utilisation de l'outils Utilisation de l'outils : babeljs.io/repl

Outil pour vous montrer en quoi votre jsx est transformé :





```

1 <div id='root'>
2   <ul>
3     <li>un</li>
4     <li>deux</li>
5     <li>trois</li>
6   </ul>
7 </div>

```

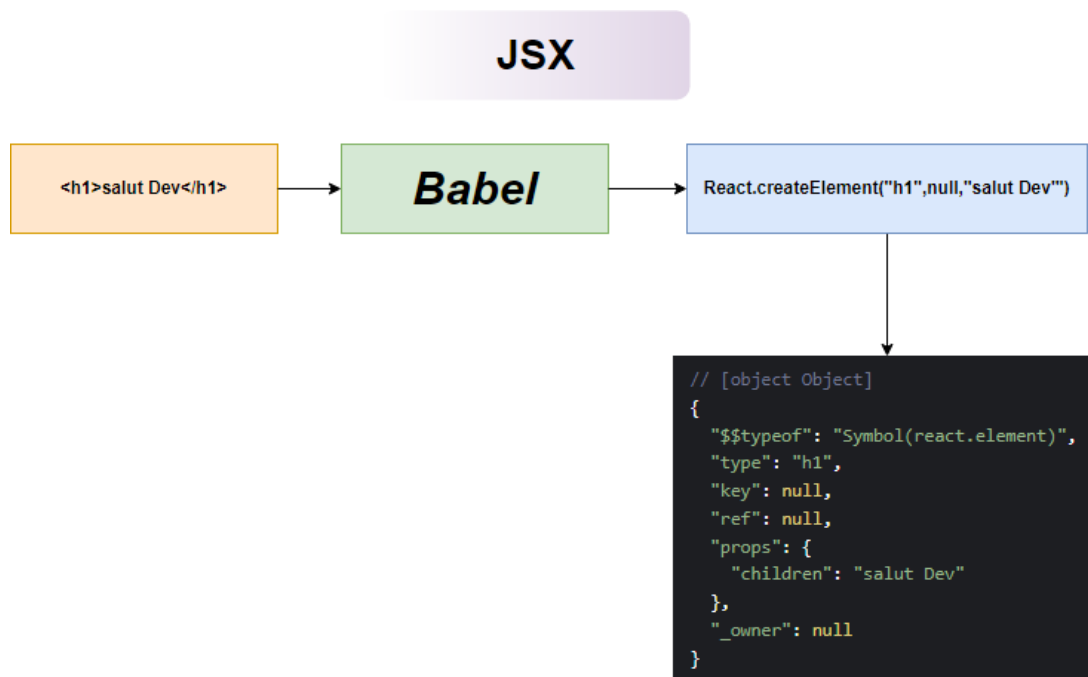
```

1 "use strict";
2
3 /*#__PURE__*/
4 React.createElement("div", {
5   id: "root"
6 }, /*#__PURE__*/React.createElement("ul", null,
  /*#__PURE__*/React.createElement("li", null, "un"),
  /*#__PURE__*/React.createElement("li", null, "deux"),
  /*#__PURE__*/React.createElement("li", null, "trois")));

```

Comme vous remarquer le code JSX est très facile et compréhensible alors que le code java script correspondant est difficile pour le comprendre beaucoup d'imbrications d'appel de createElement.

JSX facilite beaucoup le travail de développeur.



Le code JSX `<h1> salut Dev </h1>` est transformé par la Bibliothèque Babel en code javascript

`React.createElement("h1",null,"salut Dev")` qui retourne un objet java script voir figure

Cet objet contient toutes les informations nécessaires à React pour connaître l'élément à afficher dans le navigateur.

Par exemple 'type' : 'h1' précise le type de l'élément

'props' : { 'children' : 'salut Dev' } précise le texte contenu dans h1


```
<h1>saut Dev</h1>
```

cette ecriture ne fait pas automatiquement d'affichage sur le navigateur

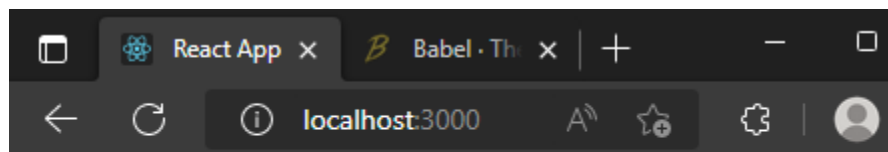
mais cette ecriture a de sens pour React
React crée un element h1

cette ecriture doit etre retourné par un composant pour que React puisse l'utiliser et l'afficher

5.4. Ajouter de variables et expression java script dans JSX

```
function App(){
  const nom='RAMI'
  return(
    <h1>salut {nom}</h1>
  )
}
```

Le rendu



salut RAMI

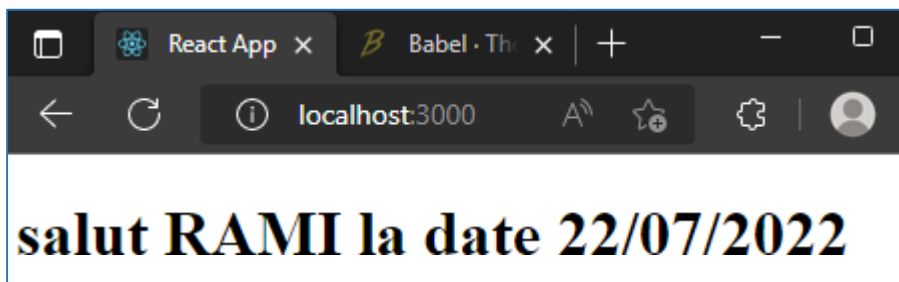
Si on veut utiliser une variable ou expression js dans JSX on utilise les accolades {},ici on passe la variable nom dans JSX

```
<h1>salut {nom}</h1>
```

Exemple 2 :

```
function App(){
  const nom='RAMI'
  const time=new Date().toLocaleDateString();
  return(
    <h1>salut {nom} la date {time}</h1>
  )
}
```

Rendu



Exemple 2 bis expression js dans jsx

```
function App(){
  const nom='RAMI'
  return(
    <h1>salut {nom} la date {new Date().toLocaleDateString()}</h1>
  )
}
```

Exemple 3 :

Afficher la remise

Si le nombre d'article acheté est supérieur a 5 remise 2% si non remise 0%

Solution :

```
function App(){
  const nbArticle=6
  let remise=0
  if(nbArticle>=5){
    remise=2
  }
  return(<div>
    <h2> Remise </h2>
    <p> votre remise est: {remise} %</p>
  </div>
  )
}
```

Rendu



Exercice d'entraînement :

On souhaite afficher le nom, le prenom et l'âge dans l'élément p du composant suivant :

Pour calculer l'âge utiliser la fonction getAge()

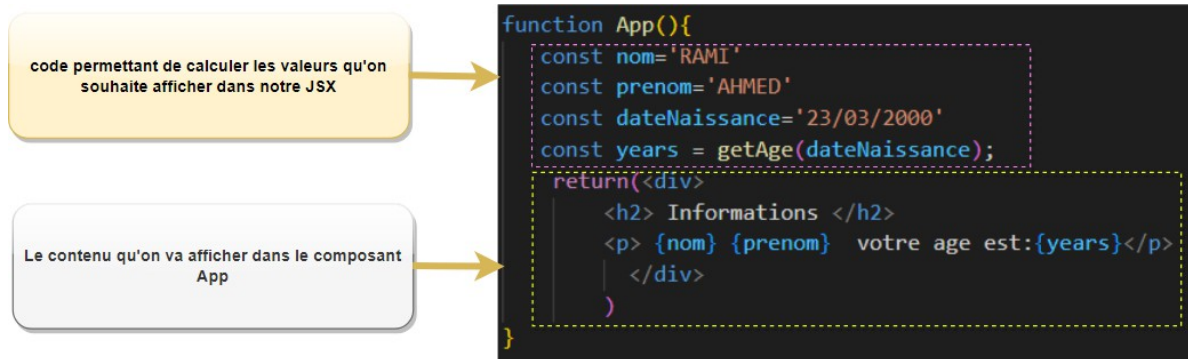
```
import React from 'react';
import ReactDOM from 'react-dom/client';
const element=document.getElementById("root");
const root=ReactDOM.createRoot(element)
//fonction getAge reçoit la date en format DD/MM/YYYY
function getAge(dateNaissance){
  let from = dateNaissance.split("/");
  let birthdateTimeStamp = new Date(from[2], from[1] - 1, from[0]);
  let cur = new Date();
  let diff = cur - birthdateTimeStamp;
  // difference en milliseconds
  let currentAge = Math.floor(diff/31557600000);
  // Division par 1000*60*60*24*365.25
  return currentAge;
}
function App(){
  const nom='RAMI'
  const prenom='AHMED'
  const dateNaissance='23/03/2000'
  return(<div>
    <h2> Informations </h2>
    <p> ----- </p>
  </div>
  )
}
root.render(<App/>)
```

Le rendu

Informations

RAMI AHMED votre age est:22

Solution :



Reprendre l'exercice précédent en utilisant classe composante

Solution :

```
import React from "react";
function getAge(dateNaissance){
  let from = dateNaissance.split("/");
  let birthdateTimeStamp = new Date(from[2], from[1] - 1, from[0]);
  let cur = new Date();
  let diff = cur - birthdateTimeStamp;
  // difference en milliseconds
  let currentAge = Math.floor(diff/31557600000);
  // Division par 1000*60*60*24*365.25
  return currentAge;
}
export default class App extends React.Component{
  constructor(props) {
    super(props)
    this.nom="RAMI";
    this.prenom="AHMED";
    this.dateNaissance="10/03/2000"
  }
  render(){
    return(<div>
      <h2> Informations </h2>
      <p> {this.nom} {this.prenom} votre age est
{getAge(this.dateNaissance)} ans</p>
    </div>
    )
  }
}
```