
ReactJS

— Une introduction —

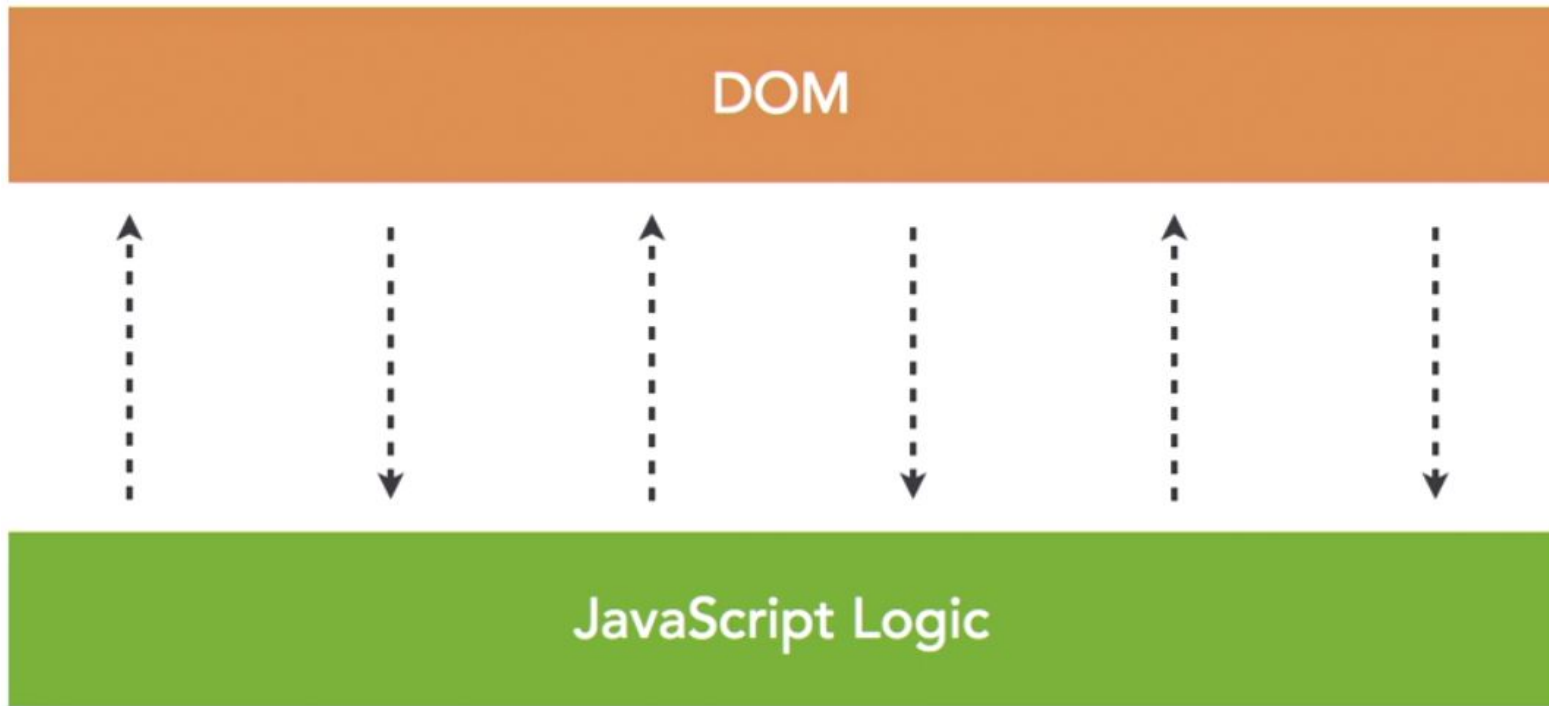
Introduction

- Une librairie pour créer des interfaces utilisateur.
- Créer par Facebook et Instagram (mars 2013)
- Suivi par React native pour les téléphones mobiles

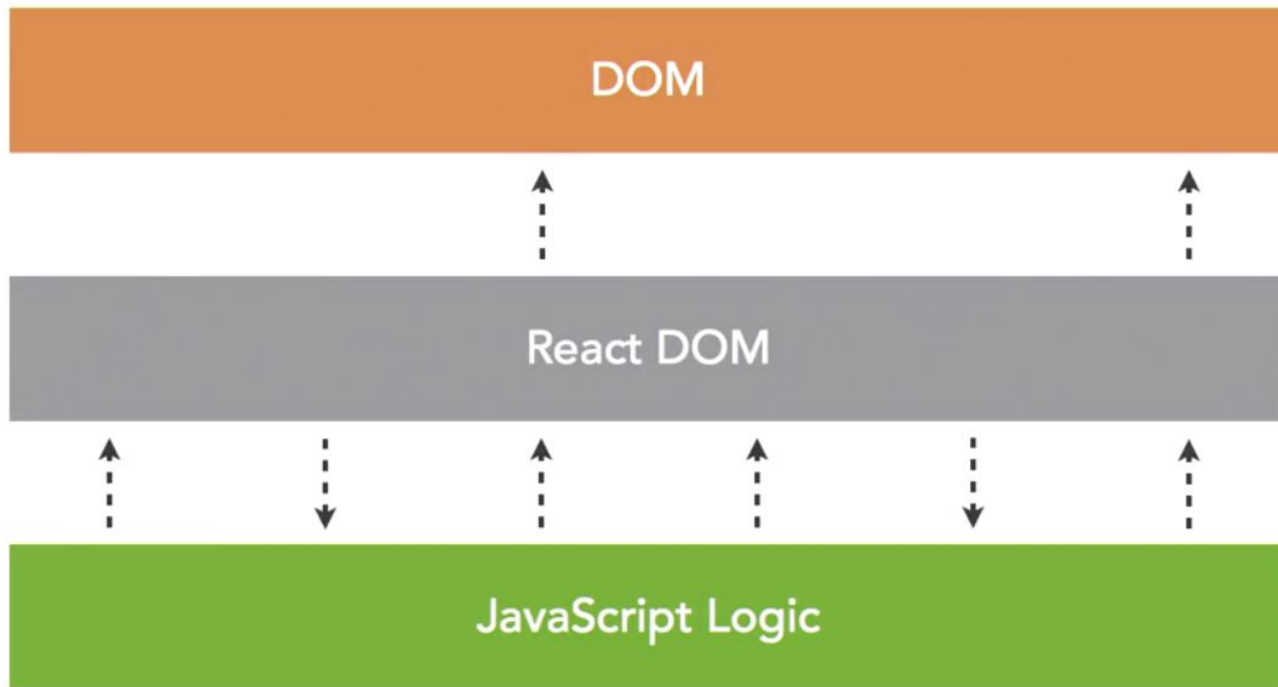
DOM diffing

- Compare la UI actuel avec la nouvelle
- Effectue uniquement les modification nécessaires
- Une comparaison des objets javascript
- Plus rapide que les modification directes sur le DOM

Sans ReactJS



Avec ReactJS (Virtual DOM)



ReactJS: Example

```
1  const app=React.createElement('h1',
2    {id: 'titre', className:'header'},
3    'Bonjour!!');
4
5  ReactDOM.render(app,
6    document.getElementById('root'));
7
```

```
1  <html>
2  <body>
3    <head>
4      <meta charset="utf-8">
5      <title>Ma premiere React app</title>
6      <script type="text/javascript" src='../react.js'></script>
7      <script type="text/javascript" src='../react-dom.js'></script>
8    </head>
9    <body>
10     <div id='root'></div>
11     <script type="text/javascript" src='index.js'></script>
12   </body>
13 </html>
```

ReactJS: Example

```
1  const {createElement} = React;
2  const {render} = ReactDOM;
3
4  const app=createElement('h1',
5    {id: 'titre', className:'header'},
6    'Bonjour!!');
7  render(app, document.getElementById('root'));
8
```

```
3  <head>
4    <meta charset="utf-8">
5    <title>Ma premiere React app</title>
6    <script type="text/javascript" src='../react.js'></script>
7    <script type="text/javascript" src='../react-dom.js'></script>
8  </head>
9  <body>
10   <div id='root'></div>
11   <script type="text/javascript" src='index.js'></script>
12 </body>
13 </html>
```

ReactJS: Example

```
1  const {createElement} = React;
2  const {render} = ReactDOM;
3
4  const stl={
5    backgroundColor: 'orange',
6    color: 'white',
7    fontFamily: 'Verdana'
8  };
9
10 const app=createElement('h1',
11   {id: 'titre', className:'header', style: stl},
12   'Bonjour!!!');
13 render(app, document.getElementById('root'));
```

```
7  <script type= "text/javascript" src='../react-dom.js'></script>
8  </head>
9  <body>
10 <div id='root'></div>
11 <script type="text/javascript" src='index.js'></script>
12 </body>
13 </html>
```

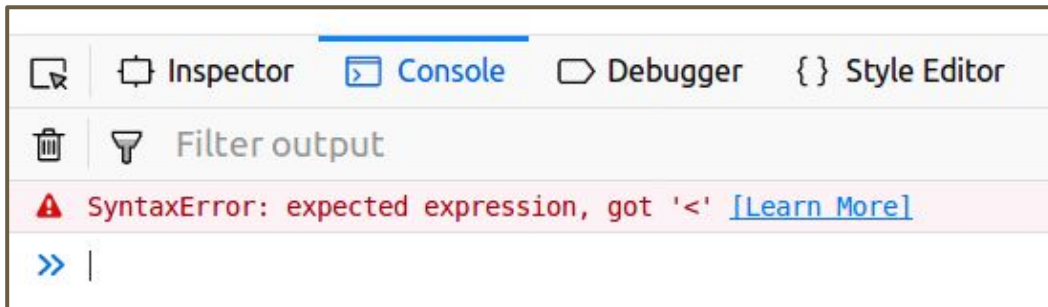
```
title>
src='../react.js'></script>
src='../react-dom.js'></script>
```


JSX

- Du *"HTML"* dans *javascript*

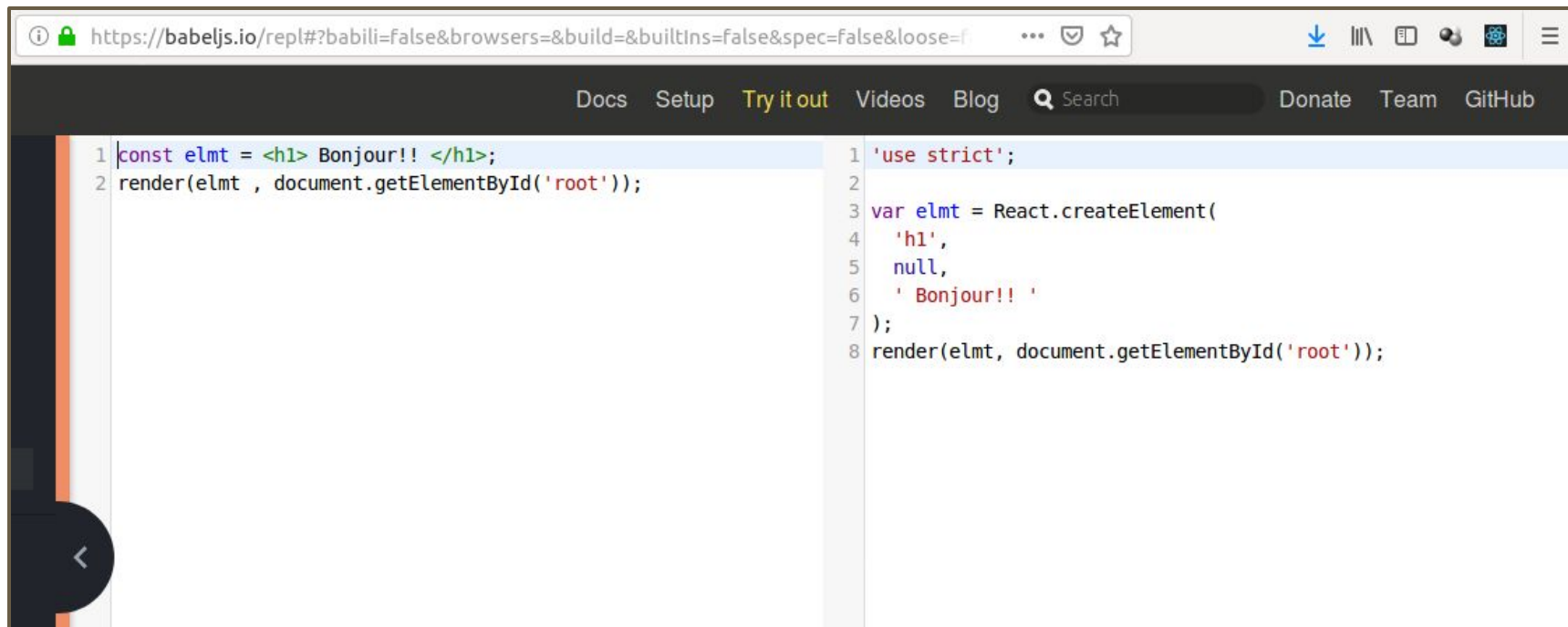
```
1  const {render} = ReactDOM;  
2  
3  const elmt = <h1> Bonjour!! </h1>;  
4  render(elmt , document.getElementById('root'));  
5
```

⇒ *Erreur*



Besoin d'un interpréteur ⇒ *Babel*

JSX: Babel



The screenshot shows the Babel REPL interface in a web browser. The address bar displays the URL: `https://babeljs.io/repl/#?babili=false&browsers=&build=&builtins=false&spec=false&loose=f`. The interface includes a navigation bar with links for Docs, Setup, Try it out, Videos, Blog, a search bar, and links to Donate, Team, and GitHub. The main area is split into two panels. The left panel contains the input code:

```
1 |const elmt = <h1> Bonjour!! </h1>;  
2 |render(elmt , document.getElementById('root'));
```

 The right panel shows the output code after transformation:

```
1 'use strict';  
2  
3 var elmt = React.createElement(  
4   'h1',  
5   null,  
6   ' Bonjour!! '  
7 );  
8 render(elmt, document.getElementById('root'));
```

JSX: Babel

```
1  const {render} = ReactDOM;  
2  
3  const elmt = <h1> Bonjour!! </h1>;  
4  render(elmt , document.getElementById('root'));  
5
```

index.js

index.html

```
3  <head>  
4    <meta charset="utf-8">  
5    <title>First React app</title>  
6    <script type="text/javascript" src='../react.js'></script>  
7    <script type="text/javascript" src='../react-dom.js'></script>  
8    <script type="text/javascript" src='https://cdnjs.cloudflare.com/ajax/libs/  
9      babel-core/5.8.38/browser.js'></script>  
10  </head>  
11  <body>  
12    <div id='root'></div>  
13    <script type="text/babel" src='index.js'></script>  
14  </body>  
15  </html>
```

JSX: Babel

index.js

```
1  const {render} = ReactDOM;
2
3  const stl={
4    backgroundColor: 'orange',
5    color: 'white',
6    fontFamily: 'Verdana'
7  };
8
9  const elmt = <h1 id='titre'
10                 className='header'
11                 style={stl}>
12    Bonjour!!
13  </h1>;
14
15  render(elmt , document.getElementById('root'));
16
```

index.html

```
    et="utf-8">
    : React app</title>
    ="text/javascript" src='../react.js'></script>
    ="text/javascript" src='../react-dom.js'></script>
    ="text/javascript" src='
    s.cloudflare.com/ajax/libs/
    8.38/browser.js'></script>
9  </head>
10 <body>
11 <div id='root'></div>
12 <script type="text/babel" src='index.js'></script>
13 </body>
14 </html>
15
```

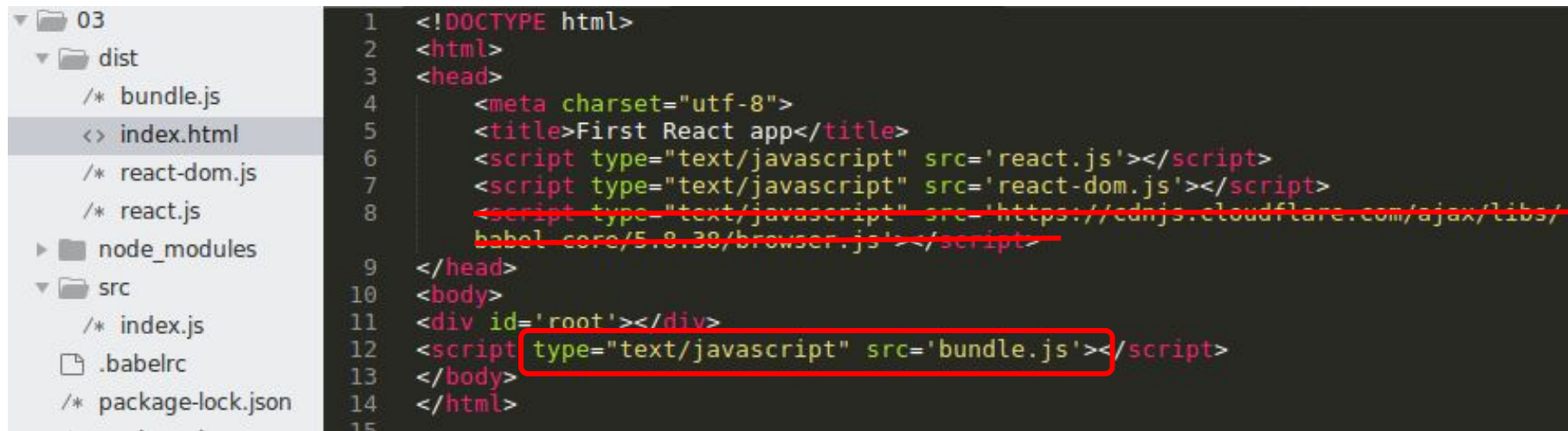
JSX: Babel

- Avec cette méthode, la transformation du jsx en javascript se fait dans le browser.
- Methode valable pour le teste
- Pour la production il faut utiliser le fichier js préalablement transformé

Babel: installation

- \$ *npm init* ⇒ package.json
- \$*npm install --save-dev babel-cli* ou *sudo npm install -g babel-cli*
- \$*npm install --save-dev babel-preset-latest babel-preset-react*
babel-preset-stage-0
- Dans le fichier *.babelrc*: { 'presets': ['latest', 'react', 'stage-0'] }

JSX: Babel

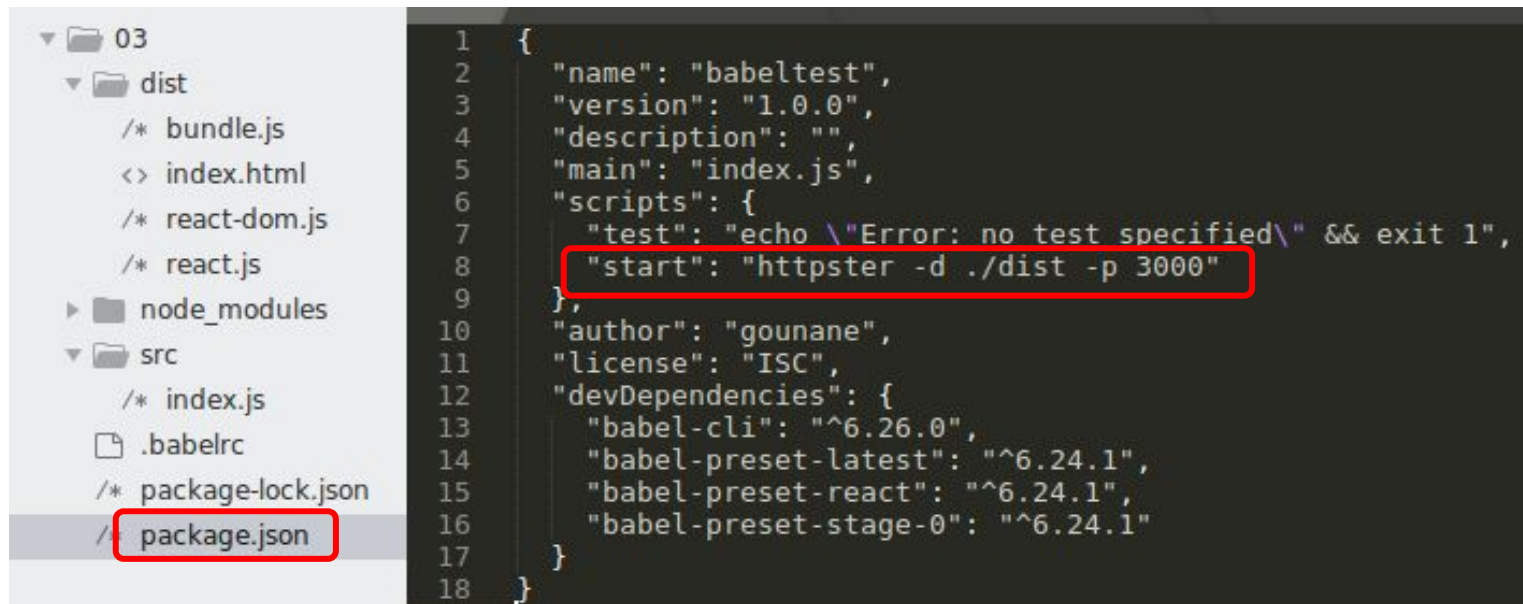


The image shows a file explorer on the left and a code editor on the right. The file explorer displays a directory structure with files like `dist/bundle.js`, `index.html`, `react-dom.js`, `react.js`, `node_modules`, `src/index.js`, `.babelrc`, and `package-lock.json`. The code editor shows an HTML file with JSX syntax. Line 8, which contains a script tag for Babel, is crossed out with a red line. Line 12, which contains a script tag for the generated `bundle.js`, is highlighted with a red box.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>First React app</title>
6   <script type="text/javascript" src='react.js'></script>
7   <script type="text/javascript" src='react-dom.js'></script>
8   <script type="text/javascript" src='https://cdnjs.cloudflare.com/ajax/libs/
  babel-core/5.8.38/browser.js'></script>
9 </head>
10 <body>
11 <div id='root'></div>
12 <script type="text/javascript" src='bundle.js'></script>
13 </body>
14 </html>
15
```

- `./node_modules/.bin/babel ./src/index.js --out-file ./dist/bundle.js` ou
- `$babel ./src/index.js --out-file ./dist/bundle.js`

JSX: Babel



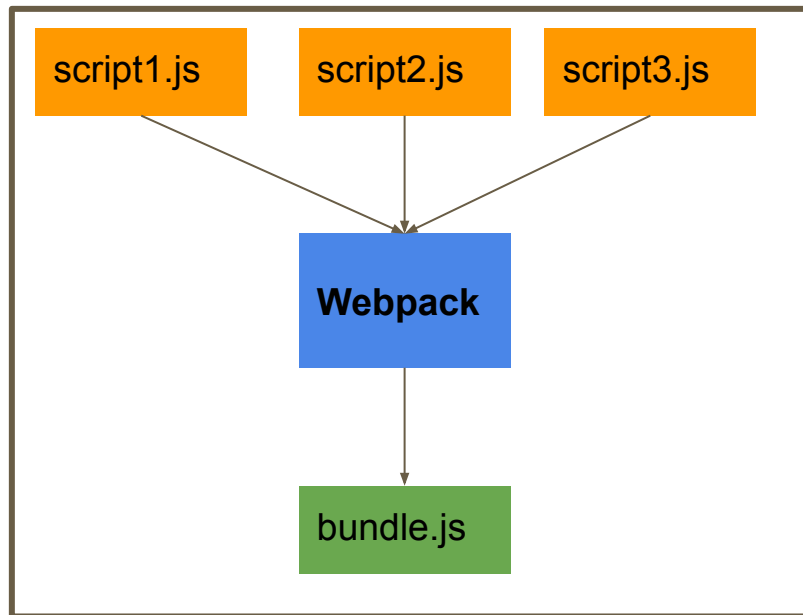
The image shows a file explorer on the left and a code editor on the right. The file explorer displays a project structure with a folder named '03' containing a 'dist' folder and a 'src' folder. The 'dist' folder contains 'bundle.js', 'index.html', 'react-dom.js', and 'react.js'. The 'src' folder contains 'index.js', '.babelrc', 'package-lock.json', and 'package.json'. The 'package.json' file is highlighted with a red box. The code editor on the right shows the contents of 'package.json' with line numbers 1 through 18. The 'scripts' section is highlighted with a red box, showing the 'start' script: `"start": "httpster -d ./dist -p 3000"`.

```
1  {
2    "name": "babeltest",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1",
8      "start": "httpster -d ./dist -p 3000"
9    },
10   "author": "gounane",
11   "license": "ISC",
12   "devDependencies": {
13     "babel-cli": "^6.26.0",
14     "babel-preset-latest": "^6.24.1",
15     "babel-preset-react": "^6.24.1",
16     "babel-preset-stage-0": "^6.24.1"
17   }
18 }
```

- `$npm start`

Webpack

- Est un module bundler
- Création des fichiers statiques
- Automatisation des tâches



Webpack

Créer un fichier webpack.config.js

```
▼ 04
  ▼ dist
    <> index.html
    /* react-dom.js
    /* react.js
  ► node_modules
  ▼ src
    /* index.js
    /* package-lock.json
    /* package.json
    /* webpack.config.js
```

```
1  var webpack = require("webpack");
2
3  module.exports = {
4    entry: "./src/index.js",
5    output: {
6      path: "dist/assets",
7      filename: "bundle.js",
8      publicPath: "assets"
9    },
10   devServer: {
11     inline: true,
12     contentBase: './dist',
13     port: 3000
14   },
15   module: {
16     loaders: [
17       {
18         test: /\.js$/,
19         exclude: /(node_modules)/,
20         loader: ["babel-loader"],
21         query: {
22           presets: ["latest", "stage-0", "react"]
23         }
24       }
25     ]
26   }
27 }
```

Webpack

- `$ npm install webpack babel-loader webpack-dev-server --save-dev`
- `$/node_modules/.bin/webpack`
- Dans le fichier ***package.json*** modifier la ligne "start":
 - `"start": "$./node_modules/.bin/webpack-dev-server"`
- `$ npm start`

```
5     <title>First React app</title>
6     <script type="text/javascript" src='react.js'></script>
7     <script type="text/javascript" src='react-dom.js'></script>
8 </head>
9 <body>
10 <div id='root'></div>
11 <script type="text/javascript" src='assets/bundle.js'></script>
12 </body>
```

Webpack

The image shows a code editor interface with a file explorer on the left and a code editor on the right.

File Explorer (Left):

- FOLDERS
 - 04
 - dist
 - assets
 - /* bundle.js** (highlighted with a red box)
 - <> index.html
 - /* react-dom.js
 - /* react.js
 - node_modules
 - src
 - /* index.js
 - /* package-lock.json
 - /* package.json
 - /* webpack.config.js

Code Editor (Right):

The editor shows the content of `index.html`:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>First React app</title>
6   <script type="text/javascript" src='react.js'></script>
7   <script type="text/javascript" src='react-dom.js'></script>
8 </head>
9 <body>
10  <div id='root'></div>
11  <script type="text/javascript" src='assets/bundle.js'></script>
12 </body>
13 </html>
14
15
```

The `src='assets/bundle.js'` attribute in line 11 is highlighted with a red box.

Webpack

- `$ npm install --save react react-dom`
- `$. /node_modules/.bin/webpack`
- `$ npm start`

```
1 const {render} = ReactDOM;
2 import React from "react";
3 import ReactDOM from "react-dom";
4 const stl={
5   backgroundColor: 'orange',
6   color: 'white',
7   fontFamily: 'Verdana'
8 };
9
10 const elmt = <h1 id='titre'
11               className='header'
12               style={stl}>
13               Bonjour!!
14             </h1>;
15
16 ReactDOM.render(elmt , document.getElementById('root'));
```

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>First React app</title>
6   <script type="text/javascript" src='react.js'></script>
7   <script type="text/javascript" src='react dom.js'></script>
8 </head>
9 <body>
10  <div id='root'></div>
11  <script type="text/javascript" src='assets/bundle.js'></script>
12 </body>
13 </html>
```

Créer une application react sans configuration

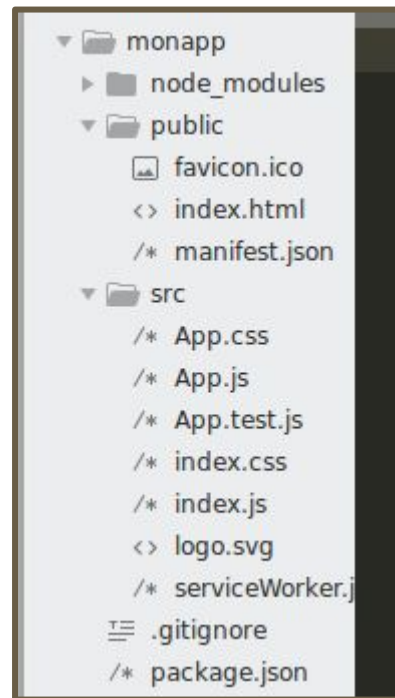
Pour créer une application reactjs sans passer par les étapes précédentes, la commande create-react-app se charge de toutes les configurations nécessaires.

Pour installer create-react-app:

\$sudo npm install -g create-react-app

Pour créer une application "***monapp***":

\$ create-react-app monapp



Lancer une application react

\$ *cd monapp*

\$ *npm start*

Principe

Index.html



```
11 <body>
12   <noscript>
13     You need to enable JavaScript to run this app.
14   </noscript>
15   <div id="root"> </div>
16 </body>
17 </html>
```

index.js



```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5
6 ReactDOM.render(<App />, document.getElementById('root'));
7
```


React Component

1- Étendre la
classe

Component

2- Redéfinir
render()

3- JSX

4- export pour
l'utiliser dans un
autre fichier js

```
1 import React, { Component } from 'react';
2 import logo from './logo.svg';
3 import './App.css';
4
5 class App extends Component {
6   render() {
7     const titre="MonApp React";
8     return (
9       <div className="App">
10         <h1>{titre}</h1>
11         <p>Voici ma premiere application ReactJS</p>
12       </div>
13     );
14   }
15 }
16
17 export default App;
```

Les Props

Pour passer des arguments à un **Component** on utilise les **props**

`<Moncomponent arg1="val1" arg2="val2" arg3={js expression} />`



Pour accéder à ces props on a accès à l'objet props dans la méthode render de moncomponent.js

- **`this.props.arg1`**
- **`this.props.arg2`**
- **`this.props.arg3`**

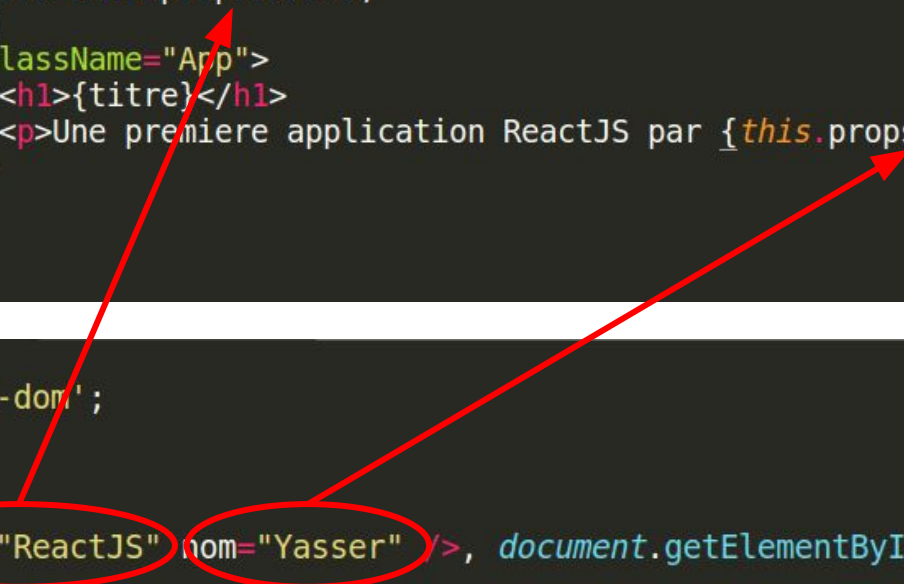
Les props

app.js

```
5 class App extends Component {  
6   render() {  
7     const titre=this.props.titre;  
8     return (  
9       <div className="App">  
10         <h1>{titre}</h1>  
11         <p>Une premiere application ReactJS par {this.props.nom}</p>  
12       </div>  
13     );  
14   }  
15 }
```

index.js

```
1 import React from 'react';  
2 import ReactDOM from 'react-dom';  
3 import './index.css';  
4 import App from './App';  
5  
6 ReactDOM.render(<App titre="ReactJS" nom="Yasser"/>, document.getElementById('root'));  
7
```



Le state

1. L'objet **state** contient des données utilisées par le **Component**
2. Les changements dans cet objet seront répercutés automatiquement sur ce **component**.
3. La modification de l'objet state ne se fait que par la méthode **setStat(newstate)**;

```
1 import React, { Component } from 'react';
2 import './style/bootstrap.min.css';
3 import './style/App.css';
4 import {textVal} from './textVal';
5
6 class App extends Component {
7   state={
8     text: textVal
9   }
10  render() {
11    const titre=this.props.titre;
12    return (
13      <div className="container">
14        <div className="titre">
15          <h1>{titre}</h1>
16          <p>Par: {this.props.nom}</p>
17        </div>
18        <div className="row">
19          <div className="col-sm-6">
20            <textarea rows="20" className="form-control"
21              value={this.state.text}</textarea>
22          </div>
23          <div className="col-sm-6">
24            <h1>Resultats</h1>
25          </div>
26        </div>
27      </div>
28    );
29  }
30 }
```

Le state

1. Lors de l'édition dans le textArea
2. Les modification sont automatiquement apporté au div

```
6 class App extends Component {
7   state={
8     text: ""
9   }
10  handleEdit=(e)=>{
11    const t=e.target.value;
12    const newState={
13      text: t
14    };
15    this.setState(newState);
16  }
17  render() {
18    const titre=this.props.titre;
19    return (
20      <div className="container">
21        <div className="titre">
22          <h1>{titre}</h1>
23          <p>Par: {this.props.nom}</p>
24        </div>
25        <div className="row">
26          <div className="col-sm-6">
27            <textarea rows="20" className="form-control"
28              onChange={(e)=> this.handleEdit(e)}></textarea>
29          </div>
30          <div className="col-sm-6">
31            <h1>{this.state.text}</h1>
```

Le cycle de vie d'un Component

- Mounting
- Updating
- Unmounting

Le cycle de vie d'un Component

- Mounting
 - constructor()
 - componentWillMount()
 - render()
 - componentDidMount()

Le cycle de vie d'un Component

- Updating
 - `getDerivedStateFromProps()`
 - `shouldComponentUpdate()`
 - `render()`
 - `getSnapshotBeforeUpdate()`
 - `componentDidUpdate()`

Le cycle de vie d'un Component

- Unmounting
 - `componentWillUnmount()`

Exercice

Ecrire une application **React** pour :

1. Générer une page contenant une liste d'utilisateurs avec des images de profile issus de (<https://robohash.org/>). Les détails des utilisateur est préalablement stocké dans un Array d'objet **User={firstName, lastName, email, img}**
2. Un bouton pour changer la couleur des cartes
3. Une zone de recherche pour filtrer ces utilisateur par nom.

Robohash (<https://robohash.org/>) est un service Web simple qui permet de fournir facilement des images uniques pour n'importe quel texte.



Les composants fonctionnels

- Les composants fonctionnels sont une partie fondamentale du développement avec React.
- Ils sont des fonctions JavaScript qui peuvent accepter des propriétés (ou "props") en entrée et renvoyer des éléments React.
- Avec l'introduction des Hooks dans React 16.8, les composants fonctionnels peuvent également gérer l'état local et utiliser des effets de manière similaire aux composants de classe.

Les composants fonctionnels

- Les composants fonctionnels sont une partie fondamentale du développement avec React.
- Ils sont des fonctions JavaScript qui peuvent accepter des propriétés (ou "props") en entrée et renvoyer des éléments React.
- Avec l'introduction des Hooks dans React 16.8, les composants fonctionnels peuvent également gérer l'état local et utiliser des effets de manière similaire aux composants de classe.

Les composants fonctionnels

- Les composants fonctionnels offrent une syntaxe plus concise que les composants de classe,
- avec l'introduction des Hooks, ils peuvent gérer l'état et les effets de manière plus expressive.
- Ils sont largement adoptés dans le développement React moderne.

Les composants fonctionnels

- **MaComposante** est une fonction qui prend **props** en paramètre.
- Les **props** contiennent généralement des données ou des configurations passées de composant parent à enfant.
- La fonction renvoie un **élément JSX** qui définit la structure de la composante.

```
import React from 'react';

const MaComposante = (props) => {
  return (
    <div>
      <h1>{props.titre}</h1>
      <p>{props.texte}</p>
    </div>
  );
};

export default MaComposante;
```

Les Props

- Les props, ou propriétés, sont un mécanisme essentiel dans React pour ***transmettre*** des données d'un composant ***parent*** à un composant ***enfant***.
- Les props sont utilisées pour transférer des informations à partir du composant parent, et elles ***sont immuables*** dans le composant enfant (c'est-à-dire qu'elles ne peuvent pas être modifiées par le composant enfant).

Les props

```
// Composant parent
import React from 'react';
import MonAutreComposantEnfant from
  './Enfant';

const Parent = () => {
  const texte = "Bonjour";
  const nombre = 42;
  const tableau = [1, 2, 3];

  return (
    <MonAutreComposantEnfant texte={texte}
    nombre={nombre} tableau={tableau} />
  );
};
```

```
// Composant enfant
const Enfant = (props) => {
  return (
    <div>
      <p>{props.texte}</p>
      <p>{props.nombre}</p>
      <ul>
        {props.tableau.map(item => (
          <li key={item}>{item}</li>
        ))}
      </ul>
    </div>
  );
};
```


Les Props

Il est important de noter quelques points clés à propos des props :

- ***Unidirectionnelle (One-Way Data Flow)*** : Les données descendant du composant parent vers les composants enfants.
- ***Immuabilité*** : Un composant enfant ne peut pas modifier directement les valeurs des props.
- ***Types de Données*** : Les props peuvent être de n'importe quel type de données, y compris des chaînes de caractères, des nombres, des objets, des tableaux, des fonctions, etc.

Les Hooks

- Les hooks sont une fonctionnalité introduite dans React 16.8
- Elle permet aux composants fonctionnels de gérer l'état local, les effets de bord et d'accéder au cycle de vie du composant sans avoir besoin d'utiliser de classes.
- Ils sont une alternative aux classes pour écrire des composants plus simples, plus lisibles et réutilisables.
- Parmi les hooks les plus couramment utilisés dans React :
 - `useState`, `useEffect`, `useReducer`, `useContext`, `useCallback`, `useMemo` ...
 -

useState

- **useState** est l'un des hooks fondamentaux de React.
- Il permet aux composants fonctionnels de gérer un état local.
- **Avant** l'introduction des **hooks**, les composants fonctionnels étaient "**stateless**", c'est-à-dire qu'ils ne pouvaient pas conserver d'état entre les rendus.
- **useState** a changé cela en permettant aux composants fonctionnels de gérer leur propre état.

useState

- `useState(0)` initialise l'état `count` à 0.
- ***useState*** retourne un tableau avec deux éléments :
 - la valeur de l'état
 - une fonction pour le mettre à jour.
- ***count*** est la valeur actuelle de l'***état*** (state).
- ***setCount*** est la fonction qui permet de mettre à jour la valeur de `count`.
- Lorsque cette fonction est appelée avec une nouvelle valeur, le composant se réaffiche avec la nouvelle valeur de `count`.

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

const MonComposant = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Incrementer
      </button>
    </div>
  );
};
```

useState

- Il est important de noter que la fonction de mise à jour de l'état (setCount dans cet exemple) peut être appelée de manière asynchrone, et React fusionnera les mises à jour d'état pour optimiser les rendus.
- Cela signifie que vous ne devriez pas compter sur la valeur actuelle de l'état lors de la mise à jour.
- Pour éviter ce problème et garantir la cohérence il faut utiliser la forme de fonction:

```
// Exemple de mise à jour basée sur la valeur précédente de l'état  
setCount((prevCount) => prevCount + 1);
```

useState

Il est également possible de gérer plusieurs états dans un même composant en utilisant useState plusieurs fois.

```
const MonComposant = () => {  
  const [count1, setCount1] = useState(0);  
  const [count2, setCount2] = useState(0);  
  return (  
    <div>  
      <p>Count 1: {count1}</p>  
      <button onClick={() => setCount1(count1+1)}>  
        Incrementer Count 1  
      </button>  
    </div>  
  );  
};
```

```
      <p>Count 2: {count2}</p>  
      <button onClick={() =>  
setCount2(count2+1)}>  
        Incrementer Count 2  
      </button>  
    </div>  
  );  
};
```

useEffect

- **useEffect** est un hook de gestion des effets de bord dans React.
- Il permet **d'effectuer** des opérations **après le rendu** du composant, ce qui est utile pour des tâches telles que
 - les appels réseau,
 - la manipulation du DOM,
 - la gestion d'abonnements,
 - etc.
- De plus, il offre une solution élégante pour gérer le cycle de vie des composants fonctionnels.

useEffect

```
import React, { useState, useEffect } from
'react';
const MonComposant = () => {
  const [data, setData] = useState(null);
  useEffect(() => {
    // Code à exécuter après chaque rendu
    fetchData();
    // Fonction de nettoyage à exécuter avant
    le démontage du composant
    return () => {
      // Code de nettoyage
    };
  }, []); // Le tableau vide signifie que
  l'effet ne dépend d'aucune variable
```

```
const fetchData = async () => {
  // Effectuer un appel réseau, par exemple
  const response = await
  fetch('https://jsonplaceholder.typicode.com/to
  dos/1');
  const result = await response.json();
  setData(result);
};
return (
  <div>
    <p>Data: {data}</p>
  </div>
);
};
```


useEffect

Dans l'exemple précédent:

- La ***fonction*** passée à **useEffect** est exécutée après ***chaque rendu*** du composant.
- La fonction `fetchData` effectue un appel réseau pour obtenir des données et met à jour l'état local (`data`) du composant.
- Le ***tableau vide []*** en deuxième argument de **useEffect** indique que cet effet ***ne dépend d'aucune variable***, et il ne sera donc exécuté qu'**une** fois après le ***premier rendu***.

useEffect

- Si on fournit un tableau de dépendances useEffect, l'effet sera exécuté à chaque fois que l'une des dépendances change.

```
useEffect(() => {  
  // Code à exécuter lorsque myVariable change  
  fetchData();  
}, [myVariable]);
```

useEffect

- L'utilisation de `useEffect` avec les dépendances permet de gérer de manière efficace les effets de bord et d'éviter les problèmes liés aux mises à jour de l'état asynchrones.
- De plus, la fonction de nettoyage retournée par `useEffect` est utile pour effectuer des tâches de nettoyage avant le démontage du composant, comme l'annulation d'abonnements ou la libération de ressources.

useReducer

- useReducer est un hook de React qui est utilisé pour gérer l'état de manière plus complexe que useState.
- useReducer est utile lorsque l'état dépend de l'action actuelle et des états précédents. Alors que, useState est souvent suffisant pour gérer des états simples,

useReducer

```
import React, { useReducer } from 'react';
// Reducer est une fonction qui spécifie comment l'état
// doit évoluer en réponse à une action
const reducer = (state, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
};
```

```
const MonComposantAvecReducer = () => {
  // useReducer renvoie l'état actuel et une fonction de
  // dispatch
  const [state, dispatch] = useReducer(reducer, { count: 0
});
  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={
        () => dispatch({ type: 'INCREMENT'
      })>Incrementer</button>
      <button onClick={
        () => dispatch({ type: 'DECREMENT'
      })>Décrementer</button>
    </div>
  );
};
```

useReducer

Dans l'exemple précédent

- ***reducer*** est une fonction qui spécifie comment l'état doit évoluer en réponse à une action.
- ***useReducer***
 - prend deux arguments : la fonction du reducer et l'état initial.
 - renvoie un tableau avec l'état actuel et une fonction de dispatch. La fonction de dispatch est utilisée pour envoyer des actions au reducer.
- Lorsqu'un bouton est cliqué, une action est dispatchée, et le reducer est appelé pour calculer le nouvel état en fonction de l'action.

useReducer

- L'utilisation de ***useReducer*** devient particulièrement ***avantageuse*** lorsque la logique de ***gestion de l'état devient complexe***.
- Elle peut également rendre le ***code plus lisible*** en regroupant la gestion de l'état dans une seule fonction.
- ***useReducer*** est souvent préféré à ***useState*** lorsque le nouvel état dépend de l'état précédent, car il offre une manière plus sûre de mettre à jour l'état.

useContext

- useContext est un hook de React qui permet à un composant d'accéder au contexte de React.
- Le contexte est une fonctionnalité qui permet de partager des valeurs telles que des thèmes, des paramètres d'authentification, ou toute autre information que vous souhaitez partager entre plusieurs composants, sans avoir à passer explicitement ces valeurs via les props à chaque niveau.

useContext

- useContext est un hook de React qui permet à un composant d'accéder au contexte de React.
- Le contexte est une fonctionnalité qui permet de partager des valeurs telles que des thèmes, des paramètres d'authentification, ou toute autre information que vous souhaitez partager entre plusieurs composants, sans avoir à passer explicitement ces valeurs via les props à chaque niveau.
- Création du contexte :

useContext

```
// 2- MonContexteProvider.js
import React from 'react';
import MonContexte from './MonContexte';
const MonContexteProvider = ({ children }) => {
  const valeurDuContexte = 'Valeur partagée';
  return (
    <MonContexte.Provider value={valeurDuContexte}>
      {children}
    </MonContexte.Provider>
  );
};
export default MonContexteProvider;
```

```
// 1- MonContexte.js
import { createContext } from 'react';
const MonContexte = createContext();
export default MonContexte;
```

useContext

```
// 3- MonComposantEnfant.js
import React, { useContext } from 'react';
import MonContexte from './MonContexte';
const MonComposantEnfant = () => {
  const valeurDuContexte = useContext(MonContexte);
  return (
    <div>
      <p>Valeur du contexte : {valeurDuContexte}</p>
    </div>
  );
};
export default MonComposantEnfant;
```

useContext

```
// 4- App.js
import React from 'react';
import MonContexteProvider from './MonContexteProvider';
import MonComposantEnfant from './MonComposantEnfant';
const App = () => {
  return (
    <MonContexteProvider>
      <MonComposantEnfant />
    </MonContexteProvider>
  );
};
export default App;
```

useContext

Dans l'exemple précédent

- MonContexte est créé à l'aide de createContext.
- MonContexteProvider utilise MonContexte.Provider pour envelopper ses composants enfants avec le contexte. La valeur du contexte (valeurDuContexte) est fournie via la prop value.
- MonComposantEnfant utilise useContext(MonContexte) pour accéder à la valeur du contexte.
- Cela permet de partager la valeur du contexte ('Valeur partagée') avec tous les composants descendants de MonContexteProvider sans avoir à passer explicitement la valeur en tant que prop à chaque niveau.

useContext

- useContext est particulièrement utile dans les situations où plusieurs composants enfants ont besoin d'accéder à la même valeur partagée.
- Cela améliore la maintenabilité et la lisibilité du code, en particulier pour des valeurs partagées à travers une grande partie de l'application.

ReactRouter

React Router

- React Router est une bibliothèque très populaire dans l'écosystème React qui facilite la gestion de la navigation dans les applications à page unique (SPA).
- Elle permet de synchroniser l'état de l'URL avec l'état de l'application, facilitant ainsi la création d'interfaces utilisateur dynamiques et interactives.
- Pour installer react router
 - \$ ***npm install react-router-dom***

React Router

Parmi les principales composantes de react Router en trouve

- `<BrowserRouter>` et `<HashRouter>`
- `<Routes>` (`<Switch>` v6-)
- `<Route>`
- `<Link>` et `<NavLink>`
- `<Redirect>`

<BrowserRouter> et <HashRouter>

Ces composants fournissent le contexte de routage pour les composants enfants. On utilise l'un ou l'autre.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```


React Router

Parmi les principaux composants de react Router en trouve

- `<BrowserRouter>` et `<HashRouter>` : Ces composants fournissent le contexte de routage pour les composants enfants. Vous utilisez l'un ou l'autre en fonction de votre environnement d'hébergement.
- `<Route>` : Ce composant est utilisé pour définir des itinéraires et rendre des composants associés à ces itinéraires.
- `<Switch>` : Ce composant rend le premier `<Route>` ou `<Redirect>` qui correspond à l'emplacement actuel. Utile pour rendre une seule route à la fois.
- `<Link>` et `<NavLink>` : Ces composants sont utilisés pour créer des liens de navigation.

- `<Link>` : Crée un lien de navigation simple