

React

TypeScript

React : Présentation

React est une librairie javascript qui permet de créer des application web

Elle utilise **Javascript**

React : Présentation

Pour utiliser React, nous devons installer les prérequis : **nodejs**

Cette installation installe 3 outils :

- **NodeJS** : Le compilateur javascript
- **NPM** : Le gestionnaire de dépendance javascript
- **NPX** : L'exécuteur de commande javascript

React : Présentation

Pour s'assurer que tout est bien installé vous pouvez exécuter les commandes suivante dans votre terminal :

```
● ● ●

# Affiche la version de nodejs
node -v

# Affiche la version de NPM
npm -v

# Affiche la version de NPX
npx -v
```

Quelques Liens Utiles :

- [Le site officiel de React](#) : Vous y retrouverez la documentation de react, ainsi qu'un tutoriel complet
- [Le site de la MDN](#) : C'est LA référence lorsque l'on parle de javascript. Vous y retrouverez l'intégralité de la documentation du langage de programmation JavaScript.

Installation

React : Installation

Pour mettre en place un nouveaux projet react il faut tout d'abord créez un répertoire vide (ATTENTION : Pas de majuscule, pas d'espace ni d'accent ou autre character spéciaux !)

Exemple : **mon-application-react**

React : Installation

Maintenant, ouvrez le répertoire vide avec
VSCode

Toujours dans vscode, ouvrez un terminal
(Terminal > Nouveau Terminal) et rentrez la
commande suivante :



```
1 # Crée une application react dans le répertoire en cours
2 npx create-react-app . --template typescript
```

React : Installation

Et maintenant, vous pouvez lancer l'application react en exécutant la commande suivante :

```
● ● ●  
npm start
```

Cotre navigateur web devrait s'ouvrir et l'application react s'afficher :)

Configuration de VSCode

React : VSCode

Il est recommandé avant de commencer un projet react de configurer correctement VSCode.

Pour cela nous avons besoin de l'extension **prettier** permettant de formater le code javascript automatiquement !

React : VSCode

1. Téléchargez l'**extension** et installer la sur
votre éditeur

React : VSCode

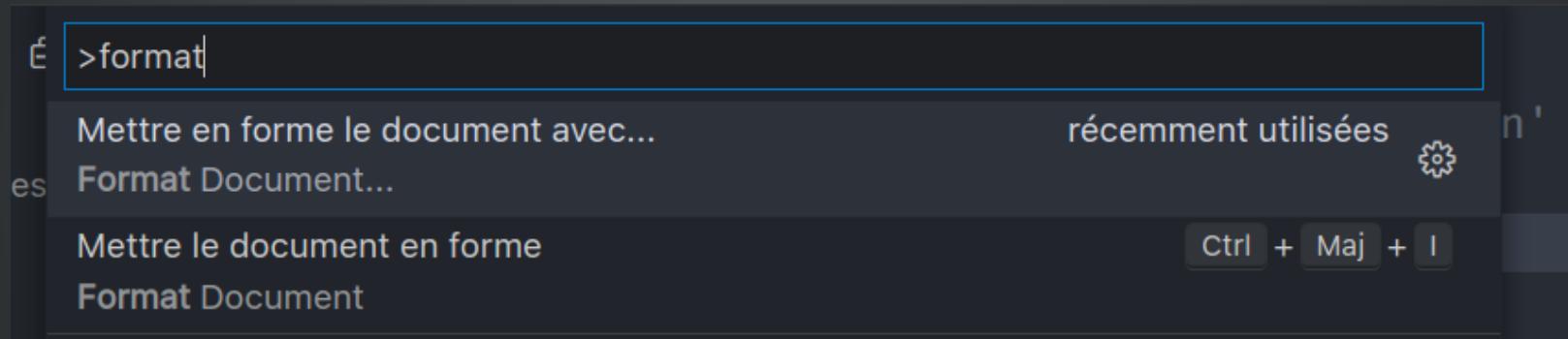
2. Activez le formatage automatique du code lors de la sauvegarde (Fichier > Préférences > Paramètres) :

The screenshot shows the VSCode settings interface. At the top, there is a search bar with the text "format on save". Below the search bar, a message says "3 paramètres trouvés". There are two tabs: "Utilisateur" and "Espace de travail", with "Utilisateur" selected. A blue button labeled "Activer la synchronisation des paramètres" is visible. The main area displays two settings:

- Editor: Format On Save**: This setting has a checked checkbox and a descriptive text: "Met en forme un fichier à l'enregistrement. Un formateur doit être disponible, le fichier ne doit pas être enregistré après un délai et l'éditeur ne doit pas être en cours d'arrêt."
- Editor: Format On Save Mode**: This setting has a descriptive text: "Permet de contrôler si la mise en forme au moment de l'enregistrement met en forme la totalité du fichier ou seulement les modifications apportées. S'applique uniquement quand [Editor: Format On Save](#) est activé."

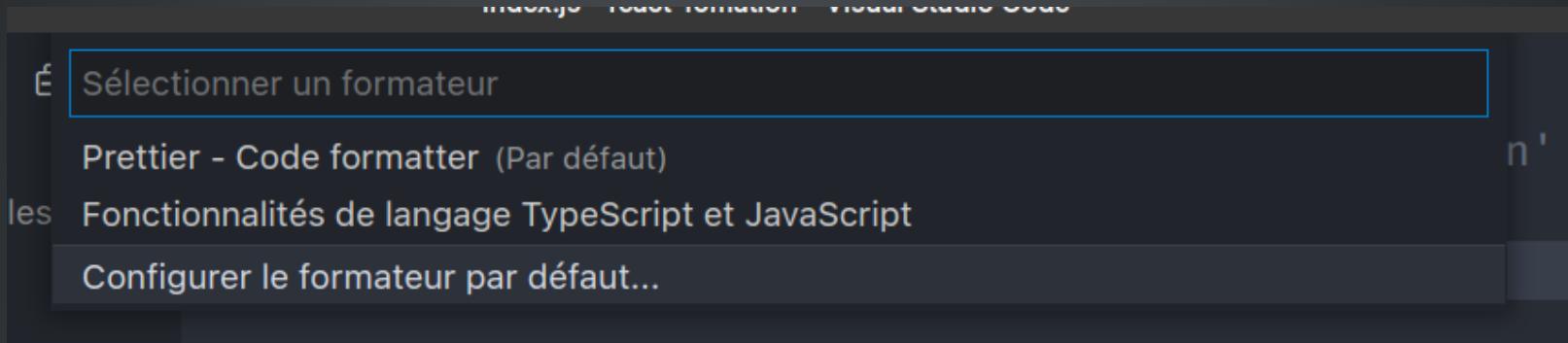
A dropdown menu at the bottom left shows the word "file".

3. Ouvrez un fichier .tsx et ouvrez le panneau de commande VSCode (**Ctrl + Maj + P** ou **Cmd + Maj + P** sur mac) et recherchez "format" :

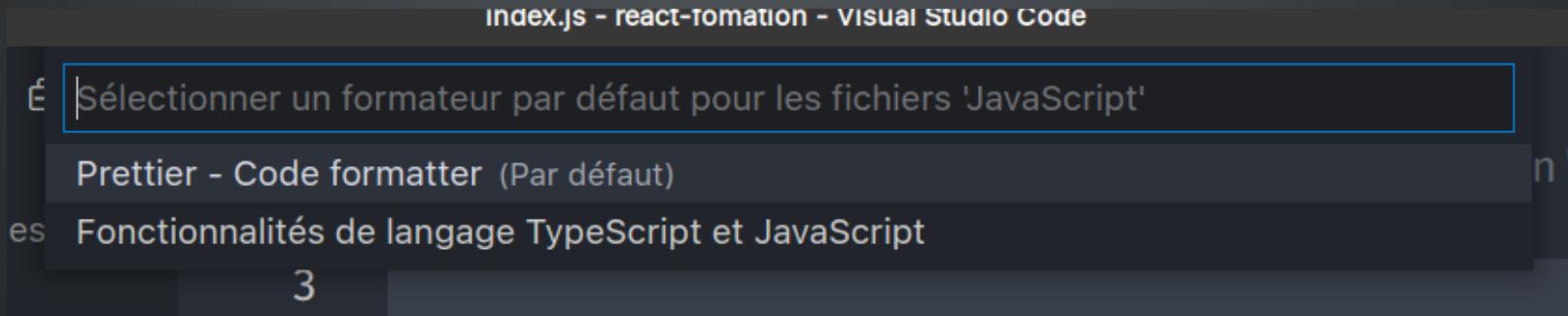


3. Cliquez sur "Mettre en forme le document avec ..."

4. Choisissez "Configurez le formateur par défaut"



5. Cliquez sur "Prettier"



6. Ajoutez à la racine du projet un fichier `.

```
1  {
2    "arrowParens": "avoid",
3    "bracketSameLine": false,
4    "bracketSpacing": true,
5    "embeddedLanguageFormatting": "auto",
6    "htmlWhitespaceSensitivity": "css",
7    "insertPragma": false,
8    "jsxSingleQuote": false,
9    "printWidth": 80,
10   "proseWrap": "preserve",
11   "quoteProps": "as-needed",
12   "requirePragma": false,
13   "semi": false,
14   "singleQuote": true,
15   "tabWidth": 2,
16   "trailingComma": "all",
17   "useTabs": false,
18   "vueIndentScriptAndStyle": false
19 }
```

Structure d'un projet

React : Structure d'un projet

```
> build  
> node_modules  
> public  
> src  
⚙ .editorconfig  
≡ .gitignore  
{} .prettierrc  
{} package.json  
ⓘ README.md  
👤 yarn.lock
```

Contient les dépendances
javascript ainsi que la configuration
de **yarn** et **npm**

C'est ici que sont noté toutes les
librairies javascript nécessaire pour
démarrer une application **React**

React : Structure d'un projet

```
> build  
> node_modules  
> public  
> src  
⚙ .editorconfig  
≡ .gitignore  
{} .prettierrc  
{} package.json  
ⓘ README.md  
👤 yarn.lock
```

Contient la configuration de **prettier**, l'outil nous permettant de formater le code javascript automatiquement

React : Structure d'un projet

```
> build  
> node_modules  
> public  
> src  
⚙ .editorconfig  
Ξ .gitignore  
{ } .prettierrc  
{ } package.json  
ⓘ README.md  
👤 yarn.lock
```

Contient le fichier de documentation principal. C'est aussi ce fichier qui s'affichera sur github

React : Structure d'un projet

```
> build  
> node_modules  
> public  
> src  
⚙ .editorconfig  
≡ .gitignore  
{} .prettierrc  
{} package.json  
ⓘ README.md  
👤 yarn.lock
```

Contient les fichiers ignorés par **git** et **github**

React : Structure d'un projet

```
> build  
> node_modules  
> public  
> src  
⚙ .editorconfig  
≡ .gitignore  
{ } .prettierrc  
{ } package.json  
ⓘ README.md  
👤 yarn.lock
```

Contient les fichiers publics accessible sur notre application web.

Généralement le HTML, les fonts, des images etc ...

React : Structure d'un projet

```
> build  
> node_modules  
> public  
> src  
⚙ .editorconfig  
≡ .gitignore  
{ } .prettierrc  
{ } package.json  
ⓘ README.md  
👤 yarn.lock
```

Contient le code source de notre application **React**.



C'est ici que nous passerons la majorité de notre temps

React : Structure d'un projet

```
> build  
> node_modules  
> public  
> src  
⚙ .editorconfig  
≡ .gitignore  
{ } .prettierrc  
{ } package.json  
ⓘ README.md  
👤 yarn.lock
```

Contient les librairies javascript nécessaire au bon fonctionnement de **React**

React : Structure d'un projet

```
> build  
> node_modules  
> public  
> src  
⚙ .editorconfig  
≡ .gitignore  
{} .prettierrc  
{} package.json  
ⓘ README.md  
👤 yarn.lock
```

Contient l'application distribuée et compilée.

Vous pouvez créer ce répertoire avec la commande :



```
npm run build
```

React

JSX

React : JSX

Le **JSX** est une syntaxe très proche de **HTML** utilisé pour afficher des balises à l'écran :

```
● ● ●

1 import React from 'react'
2 import { createRoot } from 'react-dom'
3
4 // Création d'un élément à afficher
5 // à l'écran
6 const element = (
7   <div>
8     <h1>Bonjour !</h1>
9     <p>Content de te voir ici.</p>
10  </div>
11 )
12
13 // Affichage de l'élément dans la balise avec
14 // l'id root
15 createRoot(document.querySelector('#root')).render(element)
```

React : JSX

Nous commençons par importer **React**, ceci est **obligatoire** dès que nous **faisons du JSX**



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 // Création d'un élément à afficher
5 // à l'écran
6 const element = (
7   <div>
8     <h1>Bonjour !</h1>
9     <p>Content de te voir ici.</p>
10  </div>
11 )
12
13 // Affichage de l'élément dans la balise avec
14 // l'id root
15 render(element, document.querySelector('#root'))
```

React : JSX

Ensuite nous importons "createRoot" qui permet d'afficher du JSX à l'écran



```
1 import React from 'react'
2 import { createRoot } from 'react-dom'
3
4 // Création d'un élément à afficher
5 // à l'écran
6 const element = (
7   <div>
8     <h1>Bonjour !</h1>
9     <p>Content de te voir ici.</p>
10  </div>
11 )
12
13 // Affichage de l'élément dans la balise avec
14 // l'id root
15 createRoot(document.querySelector('#root')).render(element)
```

React : JSX

Nous créons ensuite une variable élément avec du
JSX (Noté que la syntax est très similaire à HTML
!)

```
● ● ●

1 import React from 'react'
2 import { createRoot } from 'react-dom'
3
4 // Création d'un élément à afficher
5 // à l'écran
6 const element = (
7   <div>
8     <h1>Bonjour !</h1>
9     <p>Content de te voir ici.</p>
10   </div>
11 )
12
13 // Affichage de l'élément dans la balise avec
14 // l'id root
15 createRoot(document.querySelector('#root')).render(element)
```

React : JSX

Il est important de placer des parenthèses autour de nos balises HTML :

```
● ● ●

1 import React from 'react'
2 import { createRoot } from 'react-dom'
3
4 // Création d'un élément à afficher
5 // à l'écran
6 const element = (
7   <div>
8     <h1>Bonjour !</h1>
9     <p>Content de te voir ici.</p>
10  </div>
11 )
12
13 // Affichage de l'élément dans la balise avec
14 // l'id root
15 createRoot(document.querySelector('#root')).render(element)
```

React : JSX

Ensuite, nous demandons à notre JSX de s'afficher dans la balise avec l'id #root de notre HTML :

```
● ● ●

1 import React from 'react'
2 import { createRoot } from 'react-dom'
3
4 // Création d'un élément à afficher
5 // à l'écran
6 const element = (
7   <div>
8     <h1>Bonjour !</h1>
9     <p>Content de te voir ici.</p>
10  </div>
11 )
12
13 // Affichage de l'élément dans la balise avec
14 // l'id root
15 createRoot(document.querySelector('#root')).render(element)
```

React : JSX

Il est possible d'afficher des variables en JSX en utilisant des { }

```
1 import React from 'react'  
2  
3 const name = 'John Doe'  
4  
5 const element = (  
6   <div>  
7     <h1>Bonjour {name}</h1>  
8     <p>Content de te voir ici.</p>  
9   </div>  
10 )
```

React : JSX

Il est possible aussi de définir des attributs de balise :

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 const element = (  
5   <div className="ma-class">  
6     <h1>Bonjour !</h1>  
7     <p>Content de te voir ici.</p>  
8   </div>  
9 )  
10  
11 // Affichage de l'élément dans la balise avec  
12 // l'id root  
13 render(element, document.querySelector('#root'))
```

“ “**class**” est un mot réservé en javascript, en
JSX nous utilisons : “**className**” !

Les variables peuvent aussi être utilisées dans les attributs



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 const id = 'super-div'
5
6 const element = (
7   <div id={id}>
8     <h1>Bonjour !</h1>
9     <p>Content de te voir ici.</p>
10  </div>
11 )
12
13 // Affichage de l'élément dans la balise avec
14 // l'id root
15 render(element, document.querySelector('#root'))
```

React : JSX

Il est aussi possible de composer des attributs complexe grâce à l'interpolation :

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 const name = 'john'  
5 const age = 24  
6  
7 const element = (  
8   <div dataPersona={` ${name}- ${age} `}>  
9     <h1>Bonjour !</h1>  
10    <p>Content de te voir ici.</p>  
11  </div>  
12 )  
13  
14 // Affichage de l'élément dans la balise avec  
15 // l'id root  
16 render(element, document.querySelector('#root'))
```

React : JSX

On peut aussi boucler et afficher tout les éléments d'un tableaux en utilisant "map" !



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 const notes = [12, 15, 8, 9]
5
6 const element = (
7   <div>
8     <h1>Vos notes :</h1>
9     <ul>
10       {notes.map(note => (
11         <li>{note} / 20</li>
12       ))}
13     </ul>
14   </div>
15 )
16
17 // Affichage de l'élément dans la balise avec
18 // l'id root
19 render(element, document.querySelector('#root'))
```

React : JSX

Attention pour des raisons de performance il est obligatoire de générer un attribut "key" pour chaque élément JSX d'un tableaux !

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 const notes = [12, 15, 8, 9]  
5  
6 const element = (  
7   <div>  
8     <h1>Vos notes :</h1>  
9     <ul>  
10       {notes.map((note, index) => (  
11         <li key={`note-${index}`}>{note} / 20</li>  
12       ))}  
13     </ul>  
14   </div>  
15 )  
16  
17 // Affichage de l'élément dans la balise avec  
18 // l'id root  
19 render(element, document.querySelector('#root'))
```

React : JSX

Pour bien comprendre JSX, voici comment React traduis le JSX :

```
● ● ●  
1 // L'élément suivant :  
2 const element = (  
3   <div>  
4     <h1>Bonjour</h1>  
5   </div>  
6 )  
7  
8 // Est compilé comme suivis :  
9 const element = React.createElement('div', {}, [  
10   React.createElement('h1', {}, 'Bonjour'),  
11 ])
```

Les composant

React : Les composant

Les composant en **React** c'est un peu comme des balises HTML personnel.

En effet, React donne la possibilité de créer nos propres balises HTML : **Les Composant**

Un composant est une **simple fonction** qui **commence par une majuscule** et qui **retourne du JSX**

React : Les composant

Voici un exemple de composant :

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function HelloWorld(): JSX.Element {  
5   return <h1>Bonjour tout le monde !</h1>  
6 }  
7  
8 const element = (  
9   <div>  
10     <HelloWorld />  
11   </div>  
12 )  
13  
14 render(element, document.querySelector('#root'))
```

React : Les composant

Nous créons un composant "HelloWorld"



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function HelloWorld() {
5   return <h1>Bonjour tout le monde !</h1>
6 }
7
8 const element = (
9   <div>
10     <HelloWorld />
11   </div>
12 )
13
14 render(element, document.querySelector('#root'))
```

React : Les composant

Nous l'utilisons comme une balise HTML :

```
● ● ●

1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function HelloWorld() {
5   return <h1>Bonjour tout le monde !</h1>
6 }
7
8 const element = (
9   <div>
10     <HelloWorld />
11   </div>
12 )
13
14 render(element, document.querySelector('#root'))
```

React : Les composant

ATTENTION

Un composant est une **fonction** qui
commence toujours par une majuscule

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function HelloWorld() {  
5   return <h1>Bonjour tout le monde !</h1>  
6 }  
7  
8 const element = (  
9   <div>  
10     <HelloWorld />  
11   </div>  
12 )  
13  
14 render(element, document.querySelector('#root'))
```

React : Les composant

Il est aussi possible de spécifier des "attributs" html à nos composants

C'est attributs sont nommé les **props**

C'est un objet passé en premier paramètre de notre composant. Cette objet contient tout les attributs ou **props** de notre composant

React : Les composant

Voici un exemple de props :

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function Hello(props) {  
5   return (  
6     <div>  
7       <h1>Bonjour {props.name}</h1>  
8       <p>Vous avez {props.age} ans</p>  
9     </div>  
10   )  
11 }  
12  
13 const element = (  
14   <div>  
15     <Hello name="John" age={34} />  
16   </div>  
17 )  
18  
19 render(element, document.querySelector('#root'))
```

React : Les composant

Notre composant reçoit un objet "props"
contenant les attributs de la balise (ici
"name" et "age")

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function Hello(props) {  
5   return (  
6     <div>  
7       <h1>Bonjour {props.name}</h1>  
8       <p>Vous avez {props.age} ans</p>  
9     </div>  
10   )  
11 }  
12  
13 const element = (  
14   <div>  
15     <Hello name="John" age={34} />  
16   </div>  
17 )  
18  
19 render(element, document.querySelector('#root'))
```

React : Les composant

Nous affichons le contenu des props "name"
et "age"

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function Hello(props) {  
5   return (  
6     <div>  
7       <h1>Bonjour {props.name}</h1>  
8       <p>Vous avez {props.age} ans</p>  
9     </div>  
10   )  
11 }  
12  
13 const element = (  
14   <div>  
15     <Hello name="John" age={34} />  
16   </div>  
17 )  
18  
19 render(element, document.querySelector('#root'))
```

React : Les composant

Lors de l'utilisation du composant, nous passons le "name" et l'age en attribut de notre balise :

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function Hello(props) {  
5   return (  
6     <div>  
7       <h1>Bonjour {props.name}</h1>  
8       <p>Vous avez {props.age} ans</p>  
9     </div>  
10   )  
11 }  
12  
13 const element = (  
14   <div>  
15     <Hello name="John" age={34} />  
16   </div>  
17 )  
18  
19 render(element, document.querySelector('#root'))
```

React : Les composant

Il est aussi possible de « déstructurer » nos props pour améliorer la lisibilité de nos composants :

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function Hello({ name, age }) {  
5   return (  
6     <div>  
7       <h1>Bonjour {name}</h1>  
8       <p>Vous avez {age} ans</p>  
9     </div>  
10   )  
11 }  
12  
13 const element = (  
14   <div>  
15     <Hello name="John" age={34} />  
16   </div>  
17 )  
18  
19 render(element, document.querySelector('#root'))
```

React : Les composant

Les composants deviennent ainsi plus simple à lire et à utiliser :

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function Hello({ name, age }) {  
5   return (  
6     <div>  
7       <h1>Bonjour {name}</h1>  
8       <p>Vous avez {age} ans</p>  
9     </div>  
10   )  
11 }  
12  
13 const element = (  
14   <div>  
15     <Hello name="John" age={34} />  
16   </div>  
17 )  
18  
19 render(element, document.querySelector('#root'))
```

React : Les composant

Il existe une props « spéciale » que l'on appelle les "**children**"

Cette props correspond aux enfants de notre balise :

```
● ● ●  
1 <div>  
2   <p>Child 1</p>  
3   <p>Child 2</p>  
4 </div>
```

Ici, la balise "div" possède 2 enfants (les balise "p")

React : Les composant

De la même manière que en HTML, **React** donne la possibilité de créer ce système :

```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function BigText({ children }) {
5   return (
6     <div className="super-big">
7       <h1>{children}</h1>
8     </div>
9   )
10 }
11
12 const element = (
13   <div>
14     <BigText>Bonjour la compagnie</BigText>
15   </div>
16 )
17
18 render(element, document.querySelector('#root'))
```

React : Les composant

Tout d'abord, nous déclarons que notre composant accepte des enfants (children) :

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function BigText({ children }) {  
5   return (  
6     <div className="super-big">  
7       <h1>{children}</h1>  
8     </div>  
9   )  
10 }  
11  
12 const element = (  
13   <div>  
14     <BigText>Bonjour la compagnie</BigText>  
15   </div>  
16 )  
17  
18 render(element, document.querySelector('#root'))
```

React : Les composant

Nous affichons les enfants de notre composant "BigText" dans une balise h1



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function BigText({ children }) {
5   return (
6     <div className="super-big">
7       <h1>{children}</h1>
8     </div>
9   )
10 }
11
12 const element = (
13   <div>
14     <BigText>Bonjour la compagnie</BigText>
15   </div>
16 )
17
18 render(element, document.querySelector('#root'))
```

React : Les composant

Lors de l'utilisation de notre composant,
nous pouvons lui spécifier n'importe quels
enfants :



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function BigText({ children }) {
5   return (
6     <div className="super-big">
7       <h1>{children}</h1>
8     </div>
9   )
10 }
11
12 const element = (
13   <div>
14     <BigText>Bonjour la compagnie</BigText>
15   </div>
16 )
17
18 render(element, document.querySelector('#root'))
```

React : Les composant

Ou bien :

```
● ● ●

1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function BigText({ children }) {
5   return (
6     <div className="super-big">
7       <h1>{children}</h1>
8     </div>
9   )
10 }
11
12 const element = (
13   <div>
14     <BigText>
15       Bonjour la compagnie
16       <br />
17       <span>Comment allez-vous ?!</span>
18     </BigText>
19   </div>
20 )
21
22 render(element, document.querySelector('#root'))
```

React : Les composant

Il est aussi possible d'utiliser une solution très puissantes pour rendre nos composants éditables comme des balise HTML

Ce sont les "restProps" qui permettent à un composant d'accepter une infinité de props et de les spécifié à un enfant

Cette technique est utilisé lors du "**wrap**" de composant. C'est à dire lorsque l'on veut personnaliser une balise HTML déjà existante

React : Les composant

Un exemple de **wrap** et de **restProps** avec une **checkbox custom** :



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function MyCheckBox({ size = 'normal', ...restProps }) {
5   return (
6     <label className={`my-checkbox size-${size}`}>
7       <input type="checkbox" {...restProps} />
8
9       {restProps.checked ? <div className="checker"></div> : null}
10    </label>
11  )
12 }
13
14 const element = (
15   <div>
16     <MyCheckBox checked={true} id="newsletter"></MyCheckBox>
17   </div>
18 )
19
20 render(element, document.querySelector('#root'))
```

React : Les composant

Nous ajoutons la props size et récupérons toutes les autres props dans un objet "restProps"



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function MyCheckBox({ size = 'normal', ...restProps }) {
5   return (
6     <label className={`my-checkbox size-${size}`}>
7       <input type="checkbox" {...restProps} />
8
9       {restProps.checked ? <div className="checker"></div> : null}
10    </label>
11  )
12}
13
14 const element = (
15   <div>
16     <MyCheckBox checked={true} id="newsletter"></MyCheckBox>
17   </div>
18 )
19
20 render(element, document.querySelector('#root'))
```

React : Les composant

Nous appliquons toutes les autres props à
l'input



```
1 import React from 'react'
2 import { render } from 'react-dom'
3
4 function MyCheckBox({ size = 'normal', ...restProps }) {
5   return (
6     <label className={`my-checkbox size-${size}`}>
7       <input type="checkbox" {...restProps} />
8
9       {restProps.checked ? <div className="checker"></div> : null}
10    </label>
11  )
12}
13
14 const element = (
15   <div>
16     <MyCheckBox checked={true} id="newsletter"></MyCheckBox>
17   </div>
18 )
19
20 render(element, document.querySelector('#root'))
```

React : Les composant

Ici la props checked et id seront passées à l'input grâce au restProps

```
● ● ●  
1 import React from 'react'  
2 import { render } from 'react-dom'  
3  
4 function MyCheckBox({ size = 'normal', ...restProps }) {  
5   return (  
6     <label className={`my-checkbox size-${size}`}>  
7       <input type="checkbox" {...restProps} />  
8  
9       {restProps.checked ? <div className="checker"></div> : null}  
10    </label>  
11  )  
12}  
13  
14 const element = (  
15   <div>  
16     <MyCheckBox checked={true} id="newsletter"></MyCheckBox>  
17   </div>  
18 )  
19  
20 render(element, document.querySelector('#root'))
```

React : Les composant

En React chaque composant possède son propre **module javascript**

Ce module doit être **nommé à l'identique** que le **composant qu'il contient**

Le composant à l'intérieur de ce module lui doit être **exporté par défaut**

React : Les composant

Exemple avec le composant Hello

```
● ● ●  
1 // src>Hello.js  
2 import React from 'react'  
3  
4 export default function Hello({ name }) {  
5   return (  
6     <div className="greetings">  
7       <h1>Hello {name}</h1>  
8     </div>  
9   )  
10 }  
11 }
```

React : Les composant

Et maintenant dans l'index.js



```
1 // src/index.js
2 import React from 'react'
3 import { render } from 'react-dom'
4 import Hello from './Hello'
5
6 render(<Hello />, document.querySelector('#root'))
```

React : Les composant

Il existe un composant spécial contenant le cœur de notre application :

Le composant App

```
● ● ●  
1 // src/App.js  
2 import React from 'react'  
3  
4 export default function App() {  
5   return (  
6     <div className="app">  
7       /* Ici vient toute l'application */  
8     </div>  
9   )  
10 }  
11 }
```

React

Le Style

React : Le Style

En React il existe plusieurs façons de faire du style (css) pour nos composants.

La plus simple est l'import d'une feuille de style par composant

Il est aussi possible d'utiliser des modules css

Ou bien d'installer une librairie comme **styled component** !

React : Le Style - Import CSS

En **react** il est tout à fait possible d'importer une feuille de style directement depuis un fichier javascript :

```
● ● ●  
1 /* src/App.css */  
2 .App {  
3   background-color: antiquewhite;  
4   color: darkslategray;  
5 }
```

```
● ● ●  
1 import React from 'react'  
2 import './App.css'  
3  
4 export default function App() {  
5   return (  
6     <div className="App">  
7       /* Ici vient toute l'application */  
8     </div>  
9   )  
10 }
```

React : Le Style - Import CSS

Création de la feuille de style et import de cette dernière dans la fichier javascript :)

```
● ● ●  
1 /* src/App.css */  
2 .App {  
3   background-color: antiquewhite;  
4   color: darkslategray;  
5 }
```

```
● ● ●  
1 import React from 'react'  
2 import './App.css'  
3  
4 export default function App() {  
5   return (  
6     <div className="App">  
7       /* Ici vient toute l'application */  
8     </div>  
9   )  
10 }
```

React : Le Style - Import CSS

Cette technique CSS suffit pour de petit projet, mais souffre d'un grand défaut

Les classes CSS peuvent entrer en « collision »

Rien ne garantit que la class css « App » ne soit pas déjà définie dans une autre feuille de style ainsi provoquant un conflit de class !

L'import CSS généralement ne suffit pas à la mise en place d'un design complexe

React : Le Style - Module CSS

React ajoute la possibilité d'importer une feuille de style de la même manière qu'un module javascript :



```
1 /* src/App.module.css */
2 .app {
3     background-color: antiquewhite;
4     color: darkslategray;
5 }
```



```
1 // src/App.js
2 import React from 'react'
3 import style from './App.module.css'
4
5 export default function App() {
6     return (
7         <div className={style.app}>
8             /* Ici vient toute l'application */
9         </div>
10    )
11 }
```

React : Le Style - Module CSS

On commence par nommé notre feuille de style `"*.module.css"`

```
1 /* src/App.module.css */
2 .app {
3   background-color: antiquewhite;
4   color: darkslategray;
5 }
```

```
1 // src/App.js
2 import React from 'react'
3 import style from './App.module.css'
4
5 export default function App() {
6   return (
7     <div className={style.app}>
8       /* Ici vient toute l'application */
9     </div>
10   )
11 }
```

React : Le Style - Module CSS

Nous ajoutons le css sous forme de class



```
1 /* src/App.module.css */
2 .app {
3   background-color: antiquewhite;
4   color: darkslategray;
5 }
```



```
1 // src/App.js
2 import React from 'react'
3 import style from './App.module.css'
4
5 export default function App() {
6   return (
7     <div className={style.app}>
8       /* Ici vient toute l'application */
9     </div>
10   )
11 }
```

React : Le Style - Module CSS

Nous importons le module css comme un module javascript !



```
1 /* src/App.module.css */
2 .app {
3     background-color: antiquewhite;
4     color: darkslategray;
5 }
```



```
1 // src/App.js
2 import React from 'react'
3 import style from './App.module.css'
4
5 export default function App() {
6     return (
7         <div className={style.app}>
8             /* Ici vient toute l'application */
9         </div>
10    )
11 }
```

React : Le Style - Module CSS

Nous appliquons la class contenu dans le module css



```
1 /* src/App.module.css */
2 .app {
3   background-color: antiquewhite;
4   color: darkslategray;
5 }
```



```
1 // src/App.js
2 import React from 'react'
3 import style from './App.module.css'
4
5 export default function App() {
6   return (
7     <div className={style.app}>
8       /* Ici vient toute l'application */
9     </div>
10   )
11 }
```

React : Le Style - Module CSS

Cette dernière sera générée avec un nom unique évitant ainsi d'éventuel conflit de nommage



```
1 /* src/App.module.css */
2 .app {
3   background-color: antiquewhite;
4   color: darkslategray;
5 }
```



```
1 // src/App.js
2 import React from 'react'
3 import style from './App.module.css'
4
5 export default function App() {
6   return (
7     <div className={style.app}>
8       /* Ici vient toute l'application */
9     </div>
10   )
11 }
```

React : Le Style - Module CSS

Il est aussi possible de designer une application en utilisant une librairie plus puissantes comme [styled-components](#) ou encore [tailwind](#)

Les Events

React : Les Events

En react, tout comme en html, il est possible d'attacher des événements à nos éléments

HTML comme :

- Lors du clique
- Lors du survole de la souris
- Lorsque l'on rentre du text
- etc ...

Il existe un très **grand nombre** d'événements supporté par React nous permettant d'exécuter du code lors d'une action spécifique

React : Les Events

Les events en React s'applique directement sur les attributs de la balise concerné, voici un exemple avec un simple bouton :

```
● ● ●  
1 import React from 'react'  
2  
3 export default function MyButton() {  
4   const myClickEvent = () => {  
5     console.log('Click !')  
6   }  
7  
8   return <button onClick={myClickEvent}></button>  
9 }
```

React : Les Events

Nous appliquons un événement lors du clique sur notre bouton grâce à l'attribut "onClick" :

```
● ● ●  
1 import React from 'react'  
2  
3 export default function MyButton() {  
4   const myClickEvent = () => {  
5     console.log('Click !')  
6   }  
7  
8   return <button onClick={myClickEvent}></button>  
9 }
```

React : Les Events

Nous créons une fonction qui se lancera à chaque clique sur notre bouton



```
1 import React from 'react'  
2  
3 export default function MyButton() {  
4   const myClickEvent = () => {  
5     console.log('Click !')  
6   }  
7  
8   return <button onClick={myClickEvent}></button>  
9 }
```

React : Les Events

On donne la fonction à l'attribut "onClick" de notre bouton (ainsi à chaque clique sur le bouton notre fonction sera lancé) :

```
● ● ●  
1 import React from 'react'  
2  
3 export default function MyButton() {  
4   const myClickEvent = () => {  
5     console.log('Click !')  
6   }  
7  
8   return <button onClick={myClickEvent}></button>  
9 }
```

React : Les Events

Il existe une très grande variété d'événements tel que :

- onClick
- onPress
- onMouseEnter
- onMouseLeave
- etc ...

Vous pouvez retrouver une liste de tout ces événements juste [ici](#)

React : Les Events

Il est aussi possible d'obtenir des informations sur l'événement en utilisant l'objet "**event**" passé en **premier paramètre** de notre **fonction event**.

Cet objet contient des informations divers sur ce qui vient de se produire, nous pouvons y retrouver des informations comme :

- L'élément HTML ayant produit l'événement
 - La position de la souris
- Le touche du clavier ayant été pressée
 - etc ...

React : Les Events

Éxemple : Récupérer le contenu d'un input type text lorsque ce dernier change



```
1 import React from 'react'  
2  
3 export default function MyInput() {  
4   const onChange = ev => {  
5     const inputValue = ev.target.value  
6  
7     console.log(`L'input contient la valeur : ${inputValue}`)  
8   }  
9  
10  return <input type="text" onChange={onChange} />  
11 }
```

Éxemple : Récupérer la position de la souris lorsque cette dernière se déplace dans un élément



```
1 import React from 'react'  
2  
3 export default function MyGame() {  
4   const mousePosition = ev => {  
5     const x = ev.screenX  
6     const y = ev.screenY  
7  
8     console.log(`Position de la souris : x(${x}) y(${y})`)  
9   }  
10  
11   return <div onMouseMove={mousePosition} />  
12 }
```

React : Les Events

Éxemple : Traiter et valider les données d'un formulaire

```
1 import React from 'react'  
2  
3 export default function MyForm() {  
4   const onSubmit = ev => {  
5     console.log('Traitement et validation du formulaire ...')  
6   }  
7  
8   return (  
9     <form onSubmit={onSubmit}>  
10       <input type="text" name="email" />  
11       <input type="password" name="password" />  
12       <button type="submit">Envoyer</button>  
13     </form>  
14   )  
15 }
```

React : Les Events

Example : Il est aussi possible, tout comme en html, de prévenir le comportement par défaut du navigateur



```
1 import React from 'react'
2
3 export default function MyForm() {
4   const onSubmit = ev => {
5     ev.preventDefault()
6     console.log('Traitement et validation du formulaire ...')
7   }
8
9   return (
10     <form onSubmit={onSubmit}>
11       <input type="text" name="email" />
12       <input type="password" name="password" />
13       <button type="submit">Envoyer</button>
14     </form>
15   )
16 }
```

Example : Ainsi que de stopper la propagation de l'événement



```
1 import React from 'react'
2
3 export default function MyForm() {
4   const handleClick = ev => {
5     console.log('Arret de la propagation')
6     ev.stopPropagation()
7   }
8
9   return (
10     <div className="one">
11       <div className="two">
12         <div className="three" onClick={handleClick}></div>
13       </div>
14     </div>
15   )
16 }
```

React : Les Events

Grâce aux événements, une application web devient véritablement dynamique !

Cependant, notre interface, notre visuel se doit de réagir lors d'événement, par exemple :

Si je clique sur envoyé, alors une barre de chargement doit apparaître

etc ...

Le State

React : Le State

Aujourd'hui, toute application moderne se doit d'être dynamique

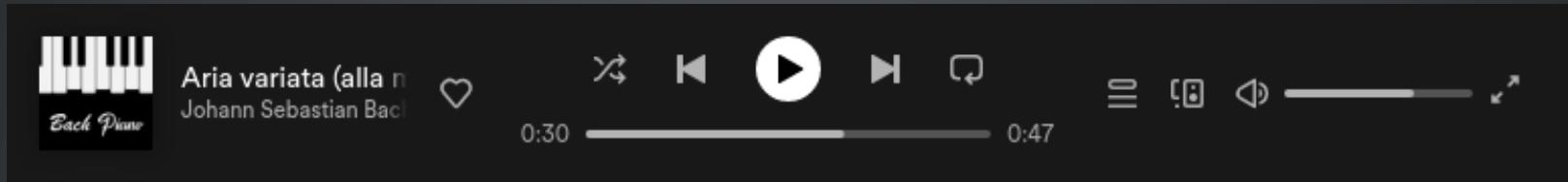
Lorsque l'on fais certaines actions sur l'interface l'on s'attend à ce que l'interface évolue, change, s'adapte à notre action

React met à disposition une notion centrale pour gérer ce « dynamisme »

Le State

React : Le State

Afin de bien comprendre de quoi nous parlons, prenons l'exemple d'un lecteur de musique :

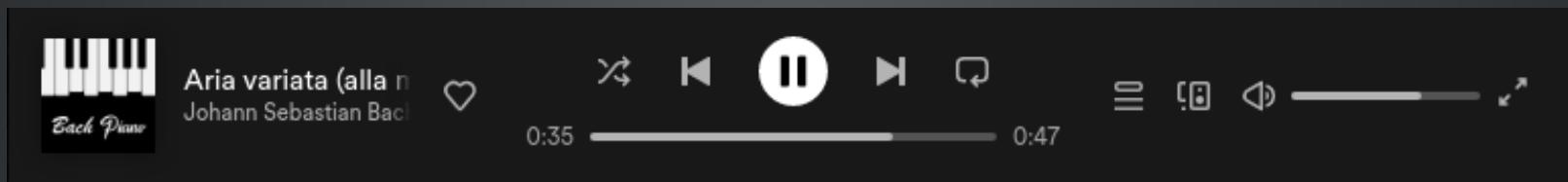


Ce lecteur possède des états (state) :

- Il est en pause
- Son volume est d'environ 75%
- Nous avons joué 30 seconds du titre
 - etc

React : Le State

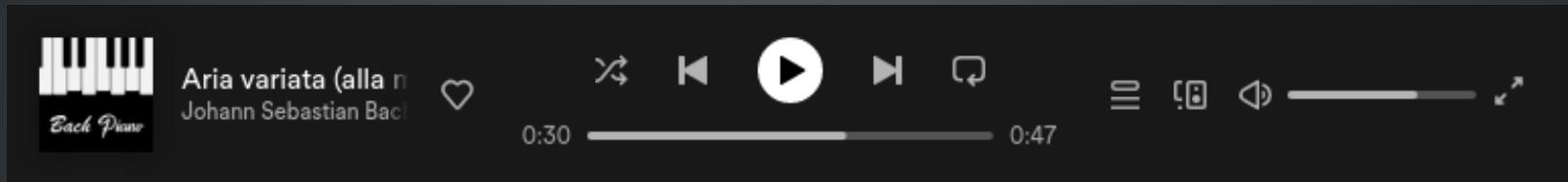
De la même manière si je clique sur "play",
le lecteur change d'état :



- Il est en lecture
- Nous avons joué 35 secondes du titre
 - etc

React : Le State

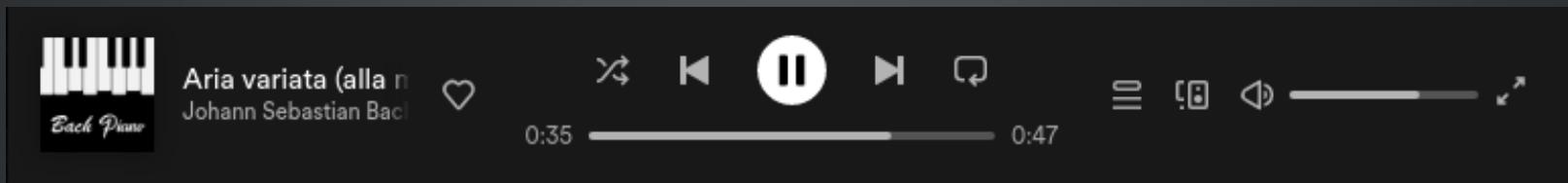
Ainsi nous pouvons créer des variables contenant les informations de notre état :



```
1 const estEnPause = true
2 const tempsEcoule = 30
3 const tempsTotal = 47
4 const volume = 75
```

React : Le State

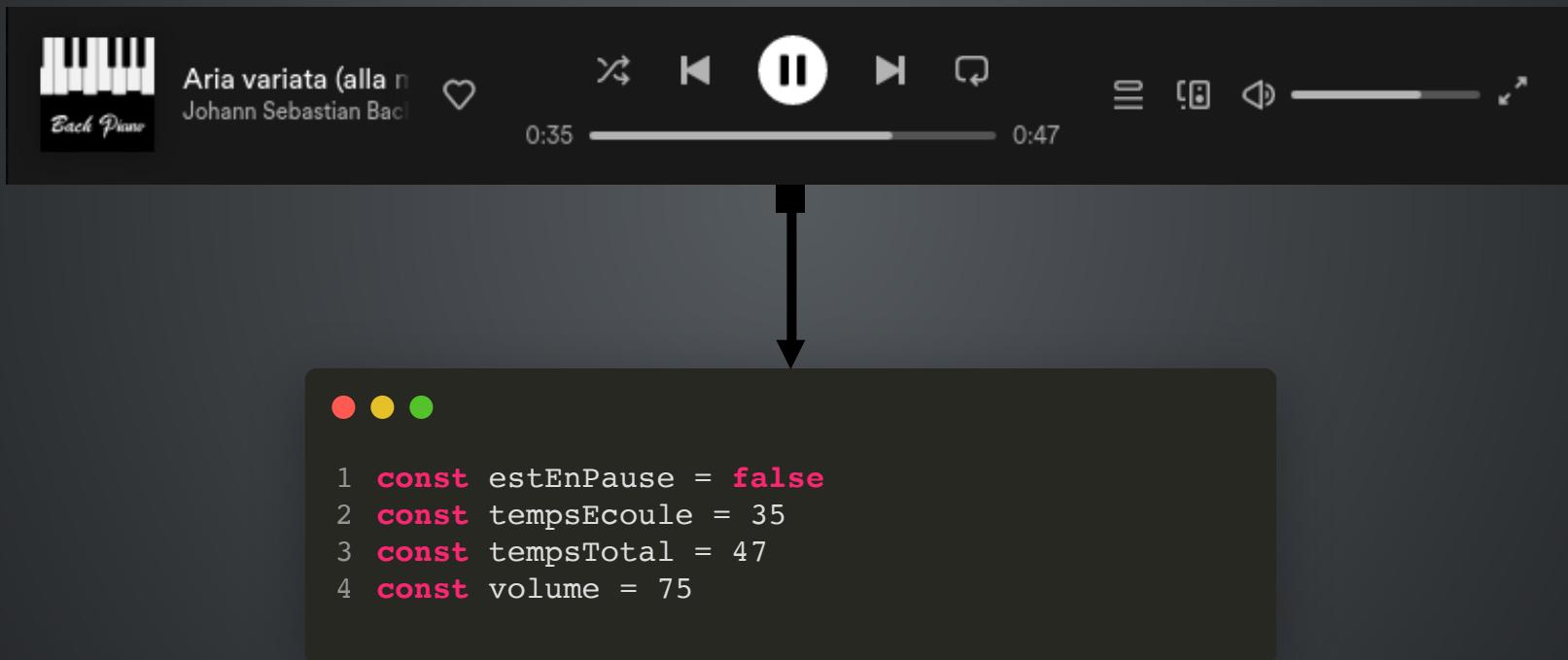
Et lors du clique sur play, nous changeons notre état :



```
1 const estEnPause = false
2 const tempsEcoule = 35
3 const tempsTotal = 47
4 const volume = 75
```

React : Le State

Et bien c'est exactement ce que permet de faire **react** grâce à l'état !



```
● ● ●  
1 const estEnPause = false  
2 const tempsEcoule = 35  
3 const tempsTotal = 47  
4 const volume = 75
```

React : Le State

Voici un exemple d'état simple avec un conteur :

```
● ● ●

1 import React, { useState } from 'react'
2
3 export default function AppEvent() {
4   const [counter, setCounter] = useState(0)
5
6   const onPlus = () => {
7     setCounter(counter + 1)
8   }
9
10  const onMinus = () => {
11    setCounter(counter - 1)
12  }
13
14  return (
15    <div>
16      <button onClick={onMinus}>-</button>
17      <p>Conteur : {counter}</p>
18      <button onClick={onPlus}>+</button>
19    </div>
20  )
21 }
```

React : Le State

Ici nous importons "useState" permettant de créer ces variables d'états



```
1 import React, { useState } from 'react'
2
3 export default function AppEvent() {
4   const [counter, setCounter] = useState(0)
5
6   const onPlus = () => {
7     setCounter(counter + 1)
8   }
9
10  const onMinus = () => {
11    setCounter(counter - 1)
12  }
13
14  return (
15    <div>
16      <button onClick={onMinus}>-</button>
17      <p>Conteur : {counter}</p>
18      <button onClick={onPlus}>+</button>
19    </div>
20  )
21 }
```

React : Le State

Création d'une variable "counter" et de la fonction "setCounter" qui permet de modifier la variable :



```
1 import React, { useState } from 'react'
2
3 export default function AppEvent() {
4   const [counter, setCounter] = useState(0)
5
6   const onPlus = () => {
7     setCounter(counter + 1)
8   }
9
10  const onMinus = () => {
11    setCounter(counter - 1)
12  }
13
14  return (
15    <div>
16      <button onClick={onMinus}>-</button>
17      <p>Conteur : {counter}</p>
18      <button onClick={onPlus}>+</button>
19    </div>
20  )
21 }
```

Gestion des événements avec changement de l'état



```
1 import React, { useState } from 'react'
2
3 export default function AppEvent() {
4     const [counter, setCounter] = useState(0)
5
6     const onPlus = () => {
7         setCounter(counter + 1)
8     }
9
10    const onMinus = () => {
11        setCounter(counter - 1)
12    }
13
14    return (
15        <div>
16            <button onClick={onMinus}>-</button>
17            <p>Conteur : {counter}</p>
18            <button onClick={onPlus}>+</button>
19        </div>
20    )
21 }
```

React : Le State

Affichage du conteur



```
1 import React, { useState } from 'react'
2
3 export default function AppEvent() {
4     const [counter, setCounter] = useState(0)
5
6     const onPlus = () => {
7         setCounter(counter + 1)
8     }
9
10    const onMinus = () => {
11        setCounter(counter - 1)
12    }
13
14    return (
15        <div>
16            <button onClick={onMinus}>-</button>
17            <p>Conteur : {counter}</p>
18            <button onClick={onPlus}>+</button>
19        </div>
20    )
21 }
```

Les Effets

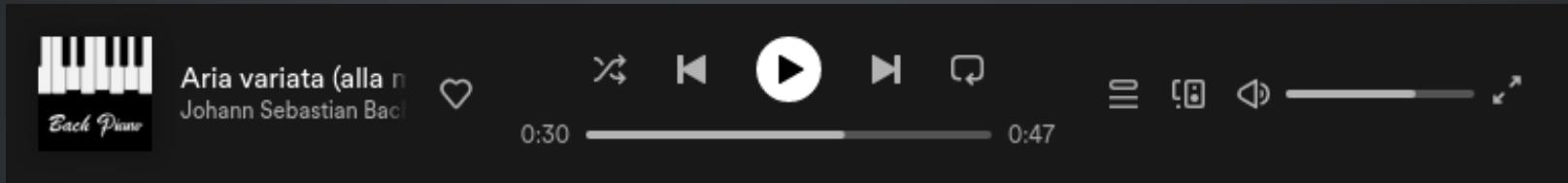
React : Les Effets

Les effets en react sont des **fonctions** s'exécutant à la suite d'un **chagement d'état**

Elles permettent de mettre en place tout un panel d'opérations qui ne sont pas visible pour l'utilisateur mais essentiel pour le bon fonctionnement de l'application !

React : Les Effets

Exemple avec un « lecteur de musique »



Lorsque l'on clique sur « play », la musique
doit **joué** et pause elle doit **s'arrêter**

Nous avons donc besoin d'une fonction qui
démarre et stop la musique lorsque l'état «
play » change !

React : Les Effets

Les effets peuvent être utilisés dans tout un panel d'action :

- Allez chercher des données sur une api
- Attendre que l'envoie d'un formulaire soit fais
- Recevoir un message ou une notification
- Envoyer les données d'un formulaire
 - etc ...

Ils sont absolument central à toute application web !

React : Les Effets

Prenons un cas pratique : un chronomètre



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   return (
5     <>
6       <button>Demarrer</button>
7       <p>
8         Temps écoulé : 0 minute(s) et 0 seconde(s)
9       </p>
10      <button>Arreter</button>
11    </>
12  )
13 }
```

React : Les Effets

Affiche le chrono



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   return (
5     <>
6       <button>Demarrer</button>
7       <p>
8         Temps écoulé : 0 minute(s) et 0 seconde(s)
9       </p>
10      <button>Arreter</button>
11    </>
12  )
13 }
```

React : Les Effets

Démarre le chronomètre



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   return (
5     <>
6       <button>Demarrer</button>
7       <p>
8         Temps écoulé : 0 minute(s) et 0 seconde(s)
9       </p>
10      <button>Arreter</button>
11    </>
12  )
13 }
```

React : Les Effets

Stop le chronomètre



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   return (
5     <>
6       <button>Demarrer</button>
7       <p>
8         Temps écoulé : 0 minute(s) et 0 seconde(s)
9       </p>
10      <button>Arreter</button>
11    </>
12  )
13 }
```

React : Les Effets

Création des états



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   return (
9     <>
10       <button>Demarrer</button>
11       <p>
12         Temps écoulé : {minutes} minute(s) et {seconds} seconde(s)
13       </p>
14       <button>Arreter</button>
15     </>
16   )
17 }
```

React : Les Effets

Arrête et démarre le chronomètre



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   return (
9     <>
10       <button>Demarrer</button>
11       <p>
12         Temps écoulé : {minutes} minute(s) et {seconds} seconde(s)
13       </p>
14       <button>Arreter</button>
15     </>
16   )
17 }
```

React : Les Effets

Nombre de secondes du chronomètre



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   return (
9     <>
10       <button>Demarrer</button>
11       <p>
12         Temps écoulé : {minutes} minute(s) et {seconds} seconde(s)
13       </p>
14       <button>Arreter</button>
15     </>
16   )
17 }
```

React : Les Effets

Nombre de minutes du chronomètre



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   return (
9     <>
10       <button>Demarrer</button>
11       <p>
12         Temps écoulé : {minutes} minute(s) et {seconds} seconde(s)
13       </p>
14       <button>Arreter</button>
15     </>
16   )
17 }
```

React : Les Effets

Affichage du temps écoulé



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   return (
9     <>
10       <button>Demarrer</button>
11       <p>
12         Temps écoulé : {minutes} minute(s) et {seconds} seconde(s)
13       </p>
14       <button>Arreter</button>
15     </>
16   )
17 }
```

React : Les Effets

On attache les « events » lors du clique sur start et stop



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   const start = () => setStarted(true)
9   const stop = () => setStarted(false)
10
11  return (
12    <>
13      <button onClick={start}>Demarrer</button>
14      <p>
15        Temps écoulé : {minutes} minute(s) et {seconds} seconde(s)
16      </p>
17      <button onClick={stop}>Arreter</button>
18    </>
19  )
20 }
```

React : Les Effets

Création d'un effet !



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   const start = () => setStarted(true)
9   const stop = () => setStarted(false)
10
11  useEffect(() => {
12    console.log('Je me déclenche dès que started change !')
13  }, [started])
14
15  return (...)
16 }
```

React : Les Effets

On importe useEffect de React



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   const start = () => setStarted(true)
9   const stop = () => setStarted(false)
10
11  useEffect(() => {
12    console.log('Je me déclenche dès que started change !')
13  }, [started])
14
15  return (...)
16 }
```

React : Les Effets

On crée une fonction qui se déclenchera à chaque changement de l'état "started"



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   const start = () => setStarted(true)
9   const stop = () => setStarted(false)
10
11  useEffect(() => {
12    console.log('Je me déclenche dès que started change !')
13  }, [started])
14
15  return (...)
16 }
```

React : Les Effets

Si started est « faux » on ne fais rien



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         if (!started) {
8             return
9         }
10    }, [started])
11
12    return (...)
```

React : Les Effets

Création d'une variable qui contiendra le nombre de secondes écoulé



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         if (!started) {
8             return
9         }
10
11         let elapsedSeconds = 0
12     }, [started])
13
14     return (...)
```

React : Les Effets

Création d'un « interval »; une fonction qui se déclenche toutes les secondes !



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         if (!started) {
8             return
9         }
10
11         let elapsedSeconds = 0
12
13         let timer = window.setInterval(() => {
14             console.log('Je me lance toute les secondes !')
15         }, 1000)
16     }, [started])
17
18     return (...)
```

React : Les Effets

On augmente le nombre de secondes écoulées



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         if (!started) {
8             return
9         }
10
11         let elapsedSeconds = 0
12
13         let timer = window.setInterval(() => {
14             elapsedSeconds += 1
15             }, 1000)
16         }, [started])
17
18         return (...)
```

React : Les Effets

On calcule les minutes et les seconds



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         // ...
8
9         let elapsedSeconds = 0
10
11         let timer = window.setInterval(() => {
12             elapsedSeconds += 1
13
14             let newMinutes = Math.floor(elapsedSeconds / 60)
15             let newSeconds = elapsedSeconds - newMinutes * 60
16         }, 1000)
17     }, [started])
18
19     return (...)
```

React : Les Effets

On change nos variables d'états



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         // ...
8
9         let elapsedSeconds = 0
10
11         let timer = window.setInterval(() => {
12             elapsedSeconds += 1
13
14             let newMinutes = Math.floor(elapsedSeconds / 60)
15             let newSeconds = elapsedSeconds - newMinutes * 60
16
17             setMinutes(newMinutes)
18             setSeconds(newSeconds)
19         }, 1000)
20     }, [started])
21
22     return (...)
23 }
```

React : Les Effets

On retourne une fonction de « nettoyage » (« clean up »)



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         // ...
8
9         let elapsedSeconds = 0
10
11         let timer = // ...
12
13         return () => {
14             console.log(`
15                 Je me déclenche à la fin de l'effet, lorsque
16                 l'état « started » passe de « true » à « false » !
17             `)
18         }
19     }, [started])
20
21     return (...)
```

React : Les Effets

On arrête « l'interval », la fonction qui se déclenche toutes les secondes



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         // ...
8
9         let elapsedSeconds = 0
10
11         let timer = // ...
12
13         return () => {
14             window.clearInterval(timer)
15         }
16     }, [started])
17
18     return (...)
```

React : Les Effets

On met les secondes et les minutes à 0 !



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4     // ...
5
6     useEffect(() => {
7         // ...
8
9         let elapsedSeconds = 0
10
11         let timer = // ...
12
13         return () => {
14             window.clearInterval(timer)
15             setSeconds(0)
16             setMinutes(0)
17         }
18     }, [started])
19
20     return (...)
```

React : Les Effets

Le code final !



```
1 import React, { useEffect, useState } from 'react'
2
3 export default function App() {
4   const [started, setStarted] = useState(false)
5   const [seconds, setSeconds] = useState(0)
6   const [minutes, setMinutes] = useState(0)
7
8   const start = () => setStarted(true)
9   const stop = () => setStarted(false)
10
11  useEffect(() => {
12    if (!started) {
13      return
14    }
15
16    let elapsedSeconds = 0
17    let timer = window.setInterval(() => {
18      elapsedSeconds += 1
19      let newMinutes = Math.floor(elapsedSeconds / 60)
20      let newSeconds = elapsedSeconds - newMinutes * 60
21
22      setMinutes(newMinutes)
23      setSeconds(newSeconds)
24    }, 1000)
25
26    return () => {
27      window.clearInterval(timer)
28    }
29  })
30}
```