# Comment créer des éléments React avec JSX

### Introduction

Dans ce tutoriel, vous apprendrez à décrire des éléments avec JSX. JSX est une abstraction qui vous permet d'écrire une syntaxe de type HTML dans votre code [JavaScript](https://www.digitalocean.com/community/tutorial_series/how-to-code-in-javascript) et vous permettra de construire des composants React qui ressemblent au balisage HTML standard.  JSX est le langage de référence de [React](https://reactjs.org/), et constitue donc le fondement de tout balisage que React apportera à votre demande.

Comme JSX vous permet également d'écrire du JavaScript dans votre balisage, vous pourrez tirer parti des fonctions et des méthodes JavaScript, notamment le mappage des [tables](https://www.digitalocean.com/community/tutorials/understanding-arrays-in-javascript) et l'évaluation des courts-circuits pour les [conditionnels](https://www.digitalocean.com/community/tutorials/how-to-write-conditional-statements-in-javascript)

Dans le cadre du tutoriel, vous capturerez les événements de clic sur les boutons directement dans le balisage et repérerez les cas où la syntaxe ne correspond pas exactement au HTML standard, par exemple avec les classes CSS. À la fin de ce tutoriel, vous disposerez d'une application de travail qui utilise diverses fonctionnalités JSX pour afficher une liste d'éléments ayant un auditeur de clics intégré. Il s'agit d'un modèle courant dans les applications React que vous utiliserez souvent durant l'apprentissage du cadre. Vous pourrez également mélanger des éléments HTML standard avec du JavaScript pour voir comment React vous donne la possibilité de créer de petits morceaux de code réutilisables.

## Conditions préalables

* Vous aurez besoin d'un environnement de développement qui exécute [Node.js](https://nodejs.org/en/about/) ; ce tutoriel a été testé sur Node.js version 10.19.0 et npm version 6.13.4. Pour l'installer sur macOS ou Ubuntu 18.04, suivez les étapes du tutoriel **Comment installer Node.js et créer un environnement de développement local sur macOS** ou la section [Installation à l'aide d'un PPA](https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-18-04) du tutoriel [Comment installer Node.js sur Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-node-js-and-create-a-local-development-environment-on-macos).
* Vous devrez être capable de créer des applications avec [Create React App](https://github.com/facebook/create-react-app). Vous pouvez trouver les instructions pour installer une application avec Create React App à l'adresse [Comment mettre en place un projet React avec Create React App](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-react-project-with-create-react-app).
* Vous devez également avoir une connaissance de base de JavaScript, que vous pouvez trouver dans [Comment coder en JavaScript](https://www.digitalocean.com/community/tutorial_series/how-to-code-in-javascript), ainsi qu'une connaissance de base du HTML et du CSS. Le [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML) est une bonne ressource pour le HTML et le CSS.

## Étape 1 - Ajouter des balises à un élément de React

Comme mentionné précédemment, React dispose d'un langage de balisage spécial appelé JSX. Il s'agit d'un mélange de syntaxe HTML et JavaScript qui ressemble à quelque chose comme ça :

<div>

{inventory.filter(item => item.available).map(item => (

<Card>

<div className="title"}>{item.name}</div>

<div className="price">{item.price}</div>

</Card>

))

}

</div>

Vous reconnaîtrez certaines fonctionnalités de JavaScript telles que [.filter](https://www.digitalocean.com/community/tutorials/how-to-use-array-methods-in-javascript-iteration-methods#filter()) et [.map](https://www.digitalocean.com/community/tutorials/how-to-use-array-methods-in-javascript-iteration-methods#map()) ainsi que du HTML standard comme <div>. Mais il y a d'autres parties qui ressemblent à la fois au HTML et au JavaScript, comme <Card> et className.

Il s'agit de JSX, le langage de balisage spécial qui donne aux composants de React la sensation du HTML avec la puissance du JavaScript.

Dans cette étape, vous apprendrez à ajouter une syntaxe de base de type HTML à un élément React existant. Pour commencer, vous ajouterez des éléments HTML standard dans une fonction JavaScript, puis vous verrez le code compilé dans un navigateur. Vous regrouperez également les éléments afin que React puisse les compiler avec un minimum de balisage, laissant une sortie HTML propre.

Pour commencer, faites un nouveau projet. Sur votre ligne de commande, exécutez le script suivant pour installer un nouveau projet en utilisant create-react-app :

* npx create-react-app jsx-tutorial

Une fois le projet terminé, passez dans le répertoire :

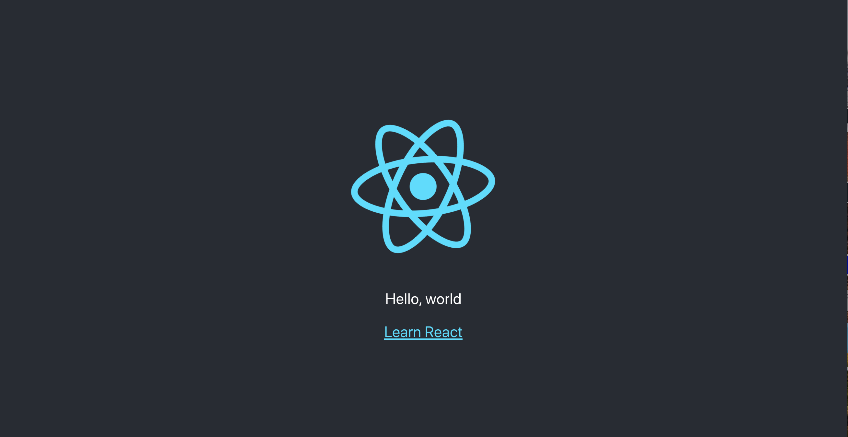
* cd jsx-tutorial

Dans un nouvel onglet ou une nouvelle fenêtre du terminal, démarrez le projet en utilisant le [script de démarrage de l'application Create React](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-react-project-with-create-react-app#step-3-%E2%80%94-starting-the-server) . Le navigateur se réactualise automatiquement en fonction des changements, alors laissez ce script fonctionner pendant toute la durée de votre travail :

* npm start

Vous obtiendrez un serveur local en fonctionnement. Si le projet ne s'est pas ouvert dans une fenêtre de navigateur, vous pouvez le trouver sur <http://localhost:3000/>. Si vous l'exécutez à partir d'un serveur distant, l'adresse sera http://your\_IP\_address:3000.

Votre navigateur se chargera avec une application React incluse dans le cadre de Create React App.



Vous allez créer un tout nouvel ensemble de composants personnalisés, vous devrez donc commencer par effacer du code passe-partout afin d'avoir un projet vide. Pour commencer, ouvrez [App.js](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-react-project-with-create-react-app#step-5-%E2%80%94-modifying-the-heading-tag-and-styling) dans un éditeur de texte. C'est le composant root (racine) qui est injecté dans la page. Tous les composants partiront de là.

Vous verrez un fichier comme ceci :

jsx-tutorial/src/App.js

import React from 'react';

import logo from './logo.svg';

import './App.css';

function App() {

return (

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<p>

Edit <code>src/App.js</code> and save to reload.

</p>

<a

className="App-link"

href="https://reactjs.org"

target="\_blank"

rel="noopener noreferrer"

>

Learn React

</a>

</header>

</div>

);

}

export default App;

Maintenant, supprimez la ligne import logo from './logo.svg et tout ce qui suit la déclaration de retour dans la fonction. Modifiez-le pour retourner null. Le code final ressemblera à ceci :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

return null;

}

export default App;

Sauvegardez et quittez l'éditeur de texte.

Enfin, supprimez le logo. Dans la fenêtre du terminal, tapez la commande suivante :

Vous n'utiliserez pas ce fichier SVG dans votre application, et vous devez supprimer les fichiers inutilisés pendant que vous travaillez. Cela permettra de mieux organiser votre code sur le long terme.

Maintenant que ces parties de votre projet ont été supprimées, vous pouvez passer à l'exploration des facettes de JSX. Ce langage de balisage est compilé par React et devient finalement le HTML que vous voyez sur une page web. Sans aller trop loin [dans les internes](https://reactjs.org/docs/introducing-jsx.html#jsx-represents-objects), React prend le JSX et crée un modèle de ce à quoi ressemblera votre page, puis crée les éléments nécessaires et les ajoute à la page.

Cela signifie que vous pouvez écrire ce qui ressemble à du HTML et vous attendre à ce que le HTML rendu soit similaire. Cependant, il y a quelques pièges.

Tout d'abord, si vous regardez l'onglet ou la fenêtre de votre serveur, vous verrez ceci :

...

./src/App.js

Line 1:8: 'React' is defined but never used no-unused-vars

...

C'est le [linter](https://create-react-app.dev/docs/setting-up-your-editor/#displaying-lint-output-in-the-editor) qui vous dit que vous n'utilisez pas le code React importé. Lorsque vous ajoutez la ligne import React à partir de « react » à votre code, vous importez du code JavaScript qui convertit le code JSX en code React. S'il n'y a pas de JSX, l'importation n'est pas nécessaire.

Changeons cela en ajoutant une petite quantité de JSX. Commencez par remplacer null avec l'exemple Hello, World :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

return <h1>Hello, World</h1>;

}

export default App;

Sauvegardez le fichier. Si vous regardez le terminal avec le serveur en marche, vous verrez que le message d'avertissement a disparu. Si vous consultez votre navigateur, vous verrez le message comme un h1 element.



Ensuite, sous la balise <h1>, ajoutez une balise de paragraphe qui contient la chaîne « I am writing JSX ». Le code ressemblera à ceci :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

return(

<h1>Hello, World</h1>

<p> I am writing JSX</p>

)

}

export default App;

Comme le JSX s'étend sur plusieurs lignes, vous devrez mettre l'expression entre parenthèses.

Sauvegardez le fichier. Lorsque vous le ferez, vous verrez une erreur dans le terminal qui fait fonctionner votre serveur :

./src/App.js

Line 7:5: Parsing error: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>?

5 | return(

6 | <h1>Hello, World</h1>

> 7 | <p>I am writing JSX</p>

| ^

8 | )

9 | }

10 |

Lorsque vous renvoyez JSX à partir d'une fonction ou d'une déclaration, vous devez renvoyer un seul élément. Cet élément peut avoir des enfants nichés, mais il doit y avoir un seul élément de haut niveau. Dans ce cas, vous rendez deux éléments.

Le correctif est un petit changement de code. Entourez le code d'une [balise vide](https://reactjs.org/docs/fragments.html#short-syntax). Une balise vide est un élément HTML sans aucun mot. Il ressemble à ceci : <></>.

Retournez à la page ./src/App.js dans votre éditeur et ajoutez la balise vide :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

return(

<>

<h1>Hello, World</h1>

<p>I am writing JSX</p>

</>

)

}

export default App;

La balise vide crée un élément unique, mais lorsque le code est compilé, il n'est pas ajouté à la balise finale. Cela vous permettra de garder votre code propre tout en donnant à React un seul élément.

**Note :** Vous auriez également pu envelopper le code avec un div au lieu de balises vides à condition que le code renvoie un élément. Dans cet exemple, une balise vide a l'avantage de ne pas ajouter de balisage supplémentaire à la sortie analysée.

Enregistrez le code et quittez le fichier. Votre navigateur rafraîchit et affiche la page mise à jour avec l'élément de paragraphe. En outre, lorsque le code est converti, les balises vides sont supprimées :



Vous avez maintenant ajouté quelques JSX de base à votre composant et appris comment tous les JSX doivent être imbriqués dans un seul composant. Dans l'étape suivante, vous ajouterez un peu de style à votre composant.

## Étape 2 - Ajout de style à un élément avec attributs

Dans cette étape, vous allez styliser les éléments de votre composant pour apprendre comment les attributs HTML fonctionnent avec JSX. Il existe de nombreuses [options de style](https://reactjs.org/docs/faq-styling.html) dans React. Certaines d'entre elles impliquent l'écriture de CSS en Javascript, d'autres utilisent des préprocesseurs. Dans ce tutoriel, vous travaillerez avec des classes CSS et CSS importées.

Maintenant que vous avez votre code, il est temps d'ajouter un peu de style. Ouvrez App.css dans votre éditeur de texte :

Comme vous commencez avec le nouveau JSX, le CSS actuel fait référence à des éléments qui n'existent plus. Comme vous n'avez pas besoin du CSS, vous pouvez le supprimer.

Après avoir effacé le code, vous aurez un fichier vide.

Ensuite, vous ajouterez un style pour centrer le texte. Dans src/App.css, ajoutez le code suivant :

jsx-tutorial/src/App.css

.container {

display: flex;

flex-direction: column;

align-items: center;

}

Dans ce bloc de code, vous avez créé un [sélecteur de classe CSS](https://developer.mozilla.org/en-US/docs/Web/CSS/Class_selectors) appelé .container et utilisé ça pour centrer le contenu en utilisant [display : flex](https://developer.mozilla.org/en-US/docs/Web/CSS/flex).

Sauvegardez le fichier et quittez. Le navigateur sera mis à jour, mais rien ne changera. Pour pouvoir voir le changement, vous devez ajouter la classe CSS à votre composant React. Ouvrez le code JavaScript du composant ./src/App.js :

Le code CSS est déjà importé avec la ligne import './App.css'. Cela signifie que le [webpack](https://webpack.js.org/) va tirer le code pour faire une feuille de style finale, mais pour appliquer le CSS à vos éléments, vous devez ajouter les classes.

Tout d'abord, dans votre éditeur de texte, changez les balises vides <> en <div>.

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

return(

<div>

<h1>Hello, World</h1>

<p>I am writing JSX</p>

</div>

)

}

export default App;

Dans ce code, vous avez remplacé les balises vides -<>-avec les balises div. Les balises vides sont utiles pour regrouper votre code sans ajouter de balises supplémentaires, mais ici vous devez utiliser un div car les balises vides n'en acceptent aucuns [attributs HTML](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes).

Ensuite, vous devez ajouter le nom de la classe. C'est là que JSX va commencer à s'écarter du HTML. Si vous vouliez ajouter une classe à un élément HTML habituel, vous le feriez comme ceci :

<div class="container">

Mais comme JSX est du JavaScript, il a quelques limites. L'une des limites est que JavaScript a [réservé des mots-clés](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords). Cela signifie que vous ne pouvez pas utiliser certains mots dans un code JavaScript. Par exemple, vous ne pouvez pas faire de variable appelée null parce que ce mot est déjà réservé.

L'un des mots réservés est classe. Réagissez en contournant ce mot réservé en le modifiant légèrement. Au lieu d'ajouter l'attribut class, vous ajouterez l'attribut className. En règle générale, si un attribut ne fonctionne pas comme prévu, essayez d'ajouter la version en camel case. Un autre attribut qui est légèrement différent est l'attribut for que vous utiliseriez pour les étiquettes. Il y a [quelques autres cas](https://reactjs.org/docs/dom-elements.html) mais heureusement, la liste est assez courte.

**Remarque :** Dans React, les attributs sont souvent appelés des accessoires. Les accessoires sont des données que vous pouvez transmettre à d'autres composants personnalisés. Ils ont le même aspect que les attributs, sauf qu'ils ne correspondent à aucune spécification HTML. Dans ce tutoriel, nous les appellerons attributs, car ils sont principalement utilisés comme des attributs HTML standard. Cela permettra de les distinguer des accessoires qui ne se comportent pas comme des attributs HTML qui seront abordés plus loin dans cette série.

Maintenant que vous savez comment l'attribut class est utilisé dans React, vous pouvez mettre à jour votre code pour inclure les styles. Dans votre éditeur de texte, ajoutez className="container" à votre étiquette de div d'ouverture :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

return(

<div className="container">

<h1>Hello, World</h1>

<p>I am writing JSX</p>

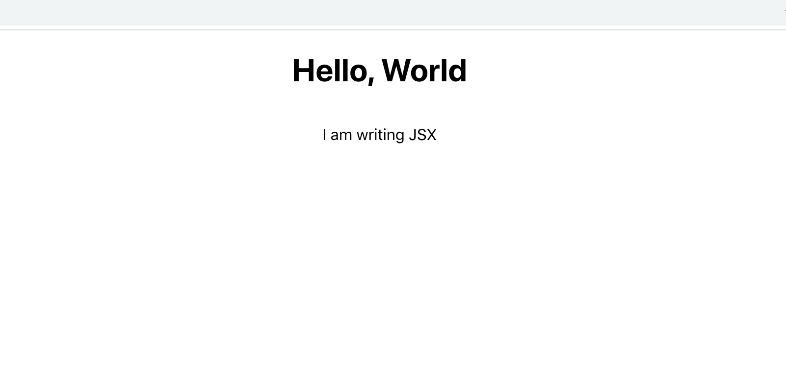
</div>

)

}

export default App;

Sauvegardez le fichier. Lorsque vous le ferez, la page se rechargera et le contenu sera centré.

.

L'attribut className est unique dans React. Vous pouvez ajouter la plupart des attributs HTML à JSX sans aucune modification. Par exemple, retournez à votre éditeur de texte et ajoutez un id de salutation à votre élément <h1>. Il ressemblera à du HTML standard :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

return(

<div className="container">

<h1 id="greeting">Hello, World</h1>

<p>I am writing JSX</p>

</div>

)

}

export default App;

Sauvegardez la page et rechargez le navigateur. Ce sera le même.

Jusqu'à présent, JSX ressemble à un balisage standard, mais l'avantage de JSX est que même s'il ressemble au HTML, il a la puissance du JavaScript. Cela signifie que vous pouvez attribuer des variables et les référencer dans vos attributs. Pour référencer un attribut, enveloppez-le avec des accolades -{}- au lieu de guillemets.

Dans votre éditeur de texte, ajoutez les lignes surlignées suivantes pour faire référence à un attribut :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

const greeting = "greeting";

return(

<div className="container">

<h1 id={greeting}>Hello, World</h1>

<p>I am writing JSX</p>

</div>

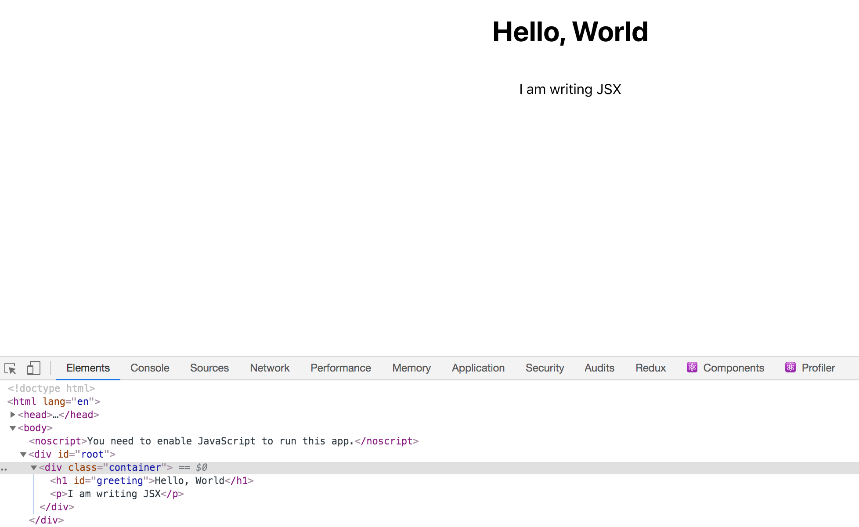
)

}

export default App;

Dans ce code, vous avez créé une variable au-dessus de l'instruction de return appelée greeting avec la valeur de « greeting », puis référencé la variable dans l'attribut id de votre balise <h1>.

Enregistrez et quittez le fichier. La page sera la même, mais avec une balise id.



Jusqu'à présent, vous avez travaillé avec quelques éléments seuls, mais vous pouvez aussi utiliser JSX pour ajouter de nombreux éléments HTML et les imbriquer pour créer des pages complexes.

Pour le démontrer, vous ferez une page avec une liste d'emojis. Ces emoji seront enveloppés d'un élément <button>. Lorsque vous cliquez sur les émojis, vous obtenez leur [nom abrégé CLDR](http://cldr.unicode.org/).

Pour commencer, vous devrez ajouter quelques éléments supplémentaires à la page. Ouvrez src/App.js dans votre éditeur de texte. Gardez-le ouvert tout au long de cette étape.

Tout d'abord, ajoutez une liste d'emojis en ajoutant les lignes surlignées suivantes :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

const greeting = "greeting";

return(

<div className="container">

<h1 id={greeting}>Hello, World</h1>

<p>I am writing JSX</p>

<ul>

</div>

)

}

export default App;

Ici vous avez créé une balise <ul> pour contenir une liste d'emojis. Chaque emoji se trouve dans un élément séparé <li> et est entouré d'un élément <button>. À l'étape suivante, vous ajouterez un [événement](https://www.digitalocean.com/community/tutorials/understanding-events-in-javascript) à ce bouton.

Vous avez également entouré l'emoji avec une balise <span> qui possède quelques autres attributs. Chaque span possède l'attribut de [rôle](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles/Role_Img) fixé au rôle img. Cela signalera au logiciel d'accessibilité que l'élément [agit comme une image](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Roles/Role_Img). En outre, chaque <span> a également une étiquette [aria-label](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Techniques/Using_the_aria-label_attribute) et un id avec le nom de l'emoji. L'étiquette aria-label indiquera aux visiteurs munis de lecteurs d'écran ce qui est affiché. Vous utiliserez l’id lors de la rédaction des événements de l'étape suivante.

Lorsque vous écrivez un code de cette manière, vous utilisez des [éléments sémantiques](https://developer.mozilla.org/en-US/docs/Glossary/Semantics) ce qui permettra de garder la page accessible et facile à analyser pour les lecteurs d'écran.

Enregistrez et quittez le fichier. Votre navigateur va se rafraîchir et vous verrez ceci :



Maintenant, ajoutez un peu de style. Ouvrez le code CSS dans votre éditeur de texte :

Ajoutez le code surligné suivant pour supprimer l'arrière-plan et la bordure par défaut des boutons tout en augmentant la taille de la police :

jsx-tutorial/src/App.css

.container {

display: flex;

flex-direction: column;

align-items: center;

}

button {

font-size: 2em;

border: 0;

padding: 0;

background: none;

cursor: pointer;

}

ul {

display: flex;

padding: 0;

}

li {

margin: 0 20px;

list-style: none;

padding: 0;

}

Dans ce code, vous avez utilisé la taille de la police, la bordure et d'autres paramètres permettant d'ajuster l'apparence de vos boutons et de modifier la police. Vous avez également supprimé les styles de liste et ajouté display : flex à l'élément <ul> pour le rendre horizontal.

Enregistrez et fermez le fichier CSS. Votre navigateur va se rafraîchir et vous verrez ceci :



Vous avez maintenant travaillé avec plusieurs éléments JSX qui ressemblent à du HTML normal. Vous avez ajouté des classes, des ids et des balises aria, et vous avez travaillé avec des données sous forme de chaînes et de variables. Mais React utilise également des attributs pour définir comment vos éléments doivent répondre aux événements des utilisateurs. Dans l'étape suivante, vous commencerez à rendre la page interactive en ajoutant des événements au bouton.

## Étape 3 - Ajouter des événements aux éléments

Dans cette étape, vous ajouterez des événements aux éléments en utilisant des attributs spéciaux et capturerez un événement de clic sur un élément de bouton. Vous apprendrez comment saisir les informations de l'événement pour envoyer une autre action ou utiliser d'autres informations dans le cadre du dossier.

Maintenant que vous disposez d'une page d'information de base, il est temps d'y ajouter quelques événements. Il existe de nombreux [gestionnaires d'événements](https://www.digitalocean.com/community/tutorials/understanding-events-in-javascript#event-handlers-and-event-listeners) que vous pouvez ajouter aux éléments HTML. React vous donne accès à tous ces éléments. Comme votre code JavaScript est couplé à votre balisage, vous pouvez rapidement ajouter les événements tout en gardant votre code bien organisé.

Pour commencer, ajoutez le [gestionnaire d'événements onclick](https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onclick). Cela vous permet d'ajouter un peu de code JavaScript directement à votre élément plutôt que d'attacher un auditeur de l'événement :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

const greeting = "greeting";

return(

<div className="container">

<h1 id={greeting}>Hello, World</h1>

<p>I am writing JSX</p>

<ul>

<li>

<button onClick={event => alert(event.target.id)}>

<span role="img" aria-label="grinning face" id="grinning face">😀</span>

</button>

</li>

<li>

<button onClick={event => alert(event.target.id)}>

<span role="img" aria-label="party popper" id="party popper">🎉</span>

</button>

</li>

<li>

<button onClick={event => alert(event.target.id)}>

<span role="img" aria-label="woman dancing" id="woman dancing">💃</span>

</button>

</li>

</ul>

</div>

)

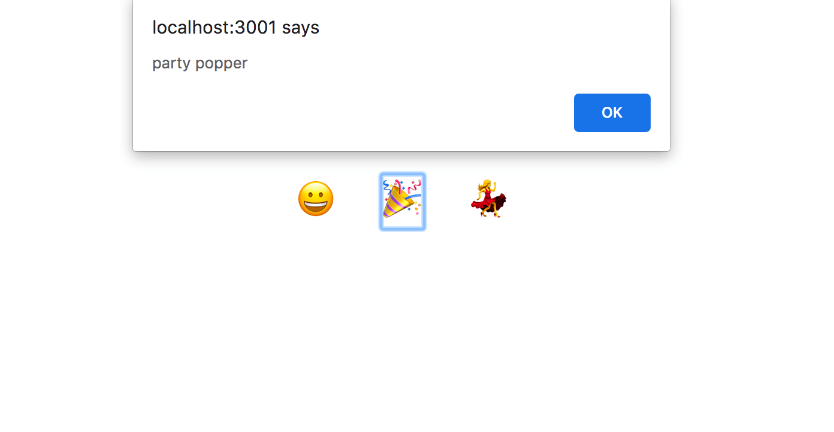
}

export default App;

Comme il s'agit de JSX, vous avez mis onclick en camel case, ce qui signifie que vous l'avez ajouté en tant que onClick. Cet attribut onClick utilise une [fonction d'anonymat](https://www.digitalocean.com/community/tutorials/how-to-define-functions-in-javascript#function-expressions) pour récupérer des informations sur l'élément cliqué.

Vous avez ajouté une [fonction de flèche](https://www.digitalocean.com/community/tutorials/how-to-define-functions-in-javascript#arrow-functions) anonyme qui obtiendra l'événement à partir du bouton cliqué, et l'événement aura une cible qui est l'élément <span>. Les informations dont vous avez besoin se trouvent dans l'attribut id auquel vous pouvez accéder avec event.target.id. Vous pouvez déclencher l'alerte avec la fonction [alert()](https://developer.mozilla.org/en-US/docs/Web/API/Window/alert).

Sauvegardez le fichier. Dans votre navigateur, cliquez sur l'un des emojis et vous obtiendrez une alerte avec le nom.



Vous pouvez réduire une duplication en déclarant la fonction une fois et en la passant à chaque action onClick. Comme la fonction ne repose sur rien d'autre que des entrées et des sorties, vous pouvez la déclarer en dehors de la fonction du composant principal. En d'autres termes, la fonction n'a pas besoin d'accéder au champ d'application du composant. L'avantage de les garder séparés est que la fonction de votre composant est légèrement plus courte et que vous pouvez déplacer la fonction dans un fichier séparé plus tard si vous le souhaitez.

Dans votre éditeur de texte, créez une fonction appelée displayEmojiName qui prend l'événement et appelle la fonction alert() avec un identifiant. Ensuite, passez la fonction à chaque attribut onClick :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

const displayEmojiName = event => alert(event.target.id);

function App() {

const greeting = "greeting";

return(

<div className="container">

<h1 id={greeting}>Hello, World</h1>

<p>I am writing JSX</p>

<ul>

<li>

<button onClick={displayEmojiName}>

<span role="img" aria-label="grinning face" id="grinning face">😀</span>

</button>

</li>

<li>

<button onClick={displayEmojiName}>

<span role="img" aria-label="party popper" id="party popper">🎉</span>

</button>

</li>

<li>

<button onClick={displayEmojiName}>

<span role="img" aria-label="woman dancing" id="woman dancing">💃</span>

</button>

</li>

</ul>

</div>

)

}

export default App;

Sauvegardez le fichier. Dans votre navigateur, cliquez sur un émoji et vous verrez la même alerte.

Dans cette étape, vous avez ajouté des événements à chaque élément. Vous avez également vu comment JSX utilise des noms légèrement différents pour les événements des éléments, et vous avez commencé à écrire du code réutilisable en prenant la fonction et en la réutilisant sur plusieurs éléments. Dans l'étape suivante, vous allez écrire une fonction réutilisable qui renvoie des éléments JSX plutôt que d'écrire chaque élément à la main. Cela permettra de réduire encore la duplication.

## Étape 4 - Recouper les données pour créer des éléments

Dans cette étape, vous irez au-delà de l'utilisation de JSX comme simple balisage. Vous apprendrez à le combiner avec JavaScript pour créer un balisage dynamique qui réduit le code et améliore la lisibilité. Vous refaites votre code dans un tableau que vous passez en boucle pour créer des éléments HTML.

JSX ne vous limite pas à une syntaxe de type HTML. Il vous donne également la possibilité d'utiliser JavaScript directement dans votre balisage. Vous avez déjà un peu essayé en passant des fonctions aux attributs. Vous avez également utilisé des variables pour réutiliser les données. Il est maintenant temps de créer JSX directement à partir des données, en utilisant le code JavaScript standard.

Dans votre éditeur de texte, vous devrez créer un tableau des données emoji dans le fichier src/App.js. Réouvrez le dossier si vous l'avez fermé :

Ajouter une table qui contiendra les [objets](https://www.digitalocean.com/community/tutorials/understanding-objects-in-javascript) qui ont l'émoji et portent le nom de l'émoji. Notez que les emojis doivent être entourés de guillemets. Créez cette table au-dessus de la fonction App :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

const displayEmojiName = event => alert(event.target.id);

const emojis = [

{

emoji: "😀",

name: "grinning face"

},

{

emoji: "🎉",

name: "party popper"

},

{

emoji: "💃",

name: "woman dancing"

}

];

function App() {

...

}

export default App;

Maintenant que vous avez les données, vous pouvez les passer en boucle. Pour utiliser JavaScript dans JSX, vous devez l'entourer d'accolades : {}. C'est la même chose que lorsque vous avez ajouté des fonctions aux attributs.

Pour créer les composants React, vous devez convertir les données en éléments JSX. Pour ce faire, vous allez cartographier les données et renvoyer un élément JSX. Il y a quelques éléments que vous devez garder à l'esprit lorsque vous écrivez le code.

Tout d'abord, un groupe d'articles doit être entouré d'un conteneur <div>. Deuxièmement, chaque objet a besoin d'un bien spécial appelé key. La key, ou clé, doit être une donnée unique que React peut utiliser pour garder une trace des éléments, afin de savoir quand [mettre à jour le composant](https://reactjs.org/docs/lists-and-keys.html#keys). La clé sera retirée du HTML compilé, car elle est réservée à un usage interne. Chaque fois que vous travaillez avec des boucles, vous devez ajouter une simple chaîne comme clé.

Voici un exemple simplifié qui fait correspondre une liste de noms à une liste contenant <div>:

...

const names = [

"Atul Gawande",

"Stan Sakai",

"Barry Lopez"

];

return(

<div>

{names.map(name => <div key={name}>{name}</div>)}

</div>

)

...

Le HTML qui en résulte ressemblerait à ceci :

...

<div>

<div>Atul Gawande</div>

<div>Stan Sakai</div>

<div>Barry Lopez</div>

</div>

...

La conversion de la liste des emojis sera similaire. Le <ul> sera le conteneur. Vous allez cartographier les données et retourner un <li> avec une clé du nom court de l'emoji. Vous remplacerez les données codées en dur dans les champs <button> et <span> avec des informations provenant de la boucle.

Dans votre éditeur de texte, ajoutez ce qui suit :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

const displayEmojiName = event => alert(event.target.id);

const emojis = [

{

emoji: '😀',

name: "test grinning face"

},

{

emoji: '🎉',

name: "party popper"

},

{

emoji: '💃',

name: "woman dancing"

}

];

function App() {

const greeting = "greeting";

return(

<div className="container">

<h1 id={greeting}>Hello, World</h1>

<p>I am writing JSX</p>

<ul>

{

emojis.map(emoji => (

<li key={emoji.name}>

<button

onClick={displayEmojiName}

>

<span role="img" aria-label={emoji.name}

id={emoji.name}>{emoji.emoji}

</span>

</button>

</li>

))

}

</ul>

</div>

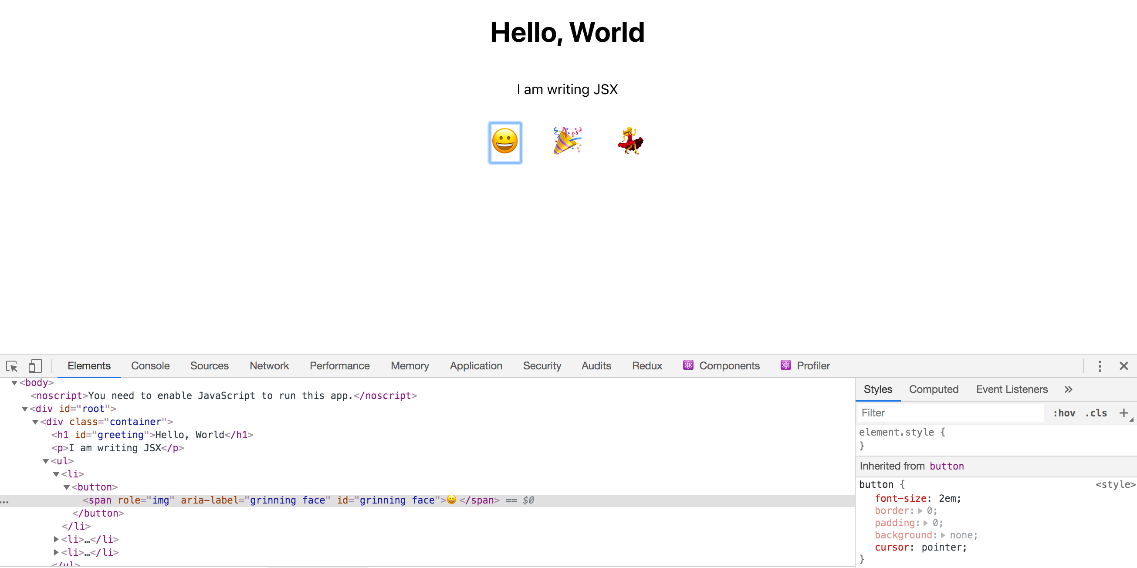
)

}

export default App;

Dans le code, vous avez [mis en correspondance](https://www.digitalocean.com/community/tutorials/how-to-use-array-methods-in-javascript-iteration-methods#map()) sur la table d’emojis dans la balise <ul> et renvoyé un <li>. Dans chaque <li> vous avez utilisé le nom de l'emoji comme accessoire clé. Le bouton aura la même fonction que normalement. Dans l'élément <span>, remplacez l'étiquette aria-label et id par l'élément name. Le contenu de la balise <span> devrait être l'emoji.

Sauvegardez le fichier. Votre fenêtre se rafraîchira et vous verrez les données. Remarquez que la clé n'est pas présente dans le HTML généré.



La combinaison de JSX et de JavaScript standard vous donne de nombreux outils pour créer du contenu de manière dynamique, et vous pouvez utiliser n'importe quel JavaScript standard que vous souhaitez. Dans cette étape, vous avez remplacé le JSX codé en dur par une table et une boucle pour créer du HTML de façon dynamique. Dans l'étape suivante, vous afficherez les informations de manière conditionnelle en utilisant un court-circuit.

## Étape 5 - Présenter sous condition des éléments avec un court-circuit

Dans cette étape, vous utiliserez le court-circuit pour afficher conditionnellement certains éléments HTML. Cela vous permettra de créer des composants qui peuvent masquer ou afficher du HTML en fonction d'informations supplémentaires, ce qui donnera à vos composants la souplesse nécessaire pour gérer plusieurs situations.

Il y a des moments où vous aurez besoin d'un élément pour montrer des informations dans certains cas et pas dans d'autres. Par exemple, vous pouvez vouloir afficher un message d'alerte pour l'utilisateur seulement si certains cas sont vrais, ou vous pouvez vouloir afficher certaines informations de compte pour un administrateur que vous ne voudriez pas qu'un utilisateur normal voit.

Pour ce faire, vous utiliserez un [court-circuit](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators#Short-circuit_evaluation). Cela signifie que vous utiliserez un conditionnel, et si la première partie est vraie, il renverra les informations dans la deuxième partie.

Voici un exemple. Si vous vouliez afficher un bouton uniquement dans le cas où l'utilisateur serait connecté, vous entoureriez l'élément avec des accolades et ajouteriez la condition précédente.

{isLoggedIn && <button>Log Out</button>}

Dans cet exemple, vous utilisez l'opérateur &&, qui renvoie la dernière valeur si tout est vrai. Sinon, il revient comme faux, ce qui indiquera à React de ne pas renvoyer de balisage supplémentaire. Si isLoggedIn est vrai, React affichera le bouton. Si isLoggedIn est faux, il n'affichera pas le bouton.

Pour essayer, ajoutez les lignes surlignées suivantes :

jsx-tutorial/src/App.js

import React from 'react';

import './App.css';

...

function App() {

const greeting = "greeting";

const displayAction = false;

return(

<div className="container">

<h1 id={greeting}>Hello, World</h1>

{displayAction && <p>I am writing JSX</p>}

<ul>

...

</ul>

</div>

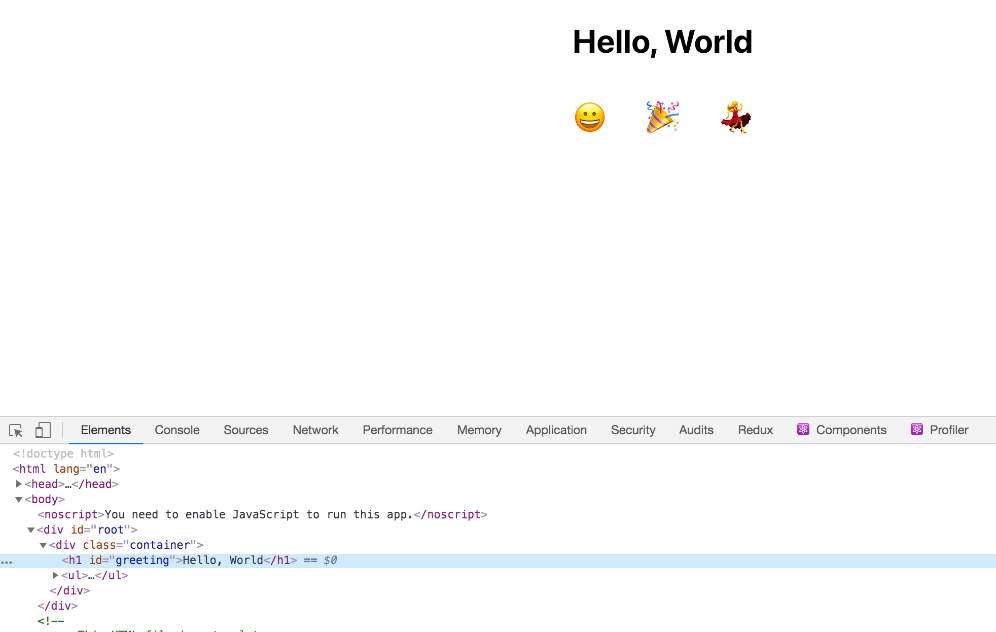
)

}

export default App;

Dans votre éditeur de texte, vous avez créé une variable appelée displayAction avec une valeur assignée comme false (fausse). Vous avez ensuite entouré la balise <p> avec des accolades.  Au début des bretelles bouclées, vous avez ajouté displayAction && pour créer le conditionnel.

Enregistrez le fichier et vous verrez l'élément disparaître dans votre navigateur. Et surtout, il n'apparaîtra pas non plus dans le HTML généré. Ce n'est pas la même chose que de masquer un élément avec le CSS. Il n'existera pas du tout dans le balisage final.



Actuellement, la valeur de displayAction est codée en dur, mais vous pouvez également stocker cette valeur en tant [qu'état](https://reactjs.org/docs/state-and-lifecycle.html) ou la transmettre comme accessoire d'un [composant parent](https://reactjs.org/docs/components-and-props.html).

Dans cette étape, vous avez appris à montrer des éléments de manière conditionnelle. Cela vous donne la possibilité de créer des composants qui sont personnalisables en fonction d'autres informations.

## Conclusion

À ce stade, vous avez créé une application personnalisée avec JSX. Vous avez appris comment ajouter des éléments de type HTML à votre composant, ajouter du style à ces éléments, passer des attributs pour créer un balisage sémantique et accessible, et ajouter des événements aux composants. Vous avez ensuite mélangé JavaScript dans votre JSX pour réduire le code en double et afficher ou masquer les éléments de manière conditionnelle.

C'est la base dont vous avez besoin pour fabriquer les futurs composants. En utilisant une combinaison de JavaScript et de HTML, vous pouvez construire des composants dynamiques qui sont flexibles et permettent à votre application de se développer et de changer.

Si vous souhaitez en savoir plus sur React, consultez notre [page thématique React](https://www.digitalocean.com/community/tags/react).