**Comment créer des composants personnalisés dans React**

Introduction

Dans ce tutoriel, vous apprendrez à créer des *composants* personnalisés dans [React](https://reactjs.org/). Les composants sont des éléments de fonctionnalité indépendants que vous pouvez réutiliser dans votre application, et sont les éléments de base de toutes les applications React. Souvent, ils peuvent être de simples [fonctions](https://www.digitalocean.com/community/tutorials/how-to-define-functions-in-javascript) et [classes JavaScript](https://www.digitalocean.com/community/tutorials/understanding-classes-in-javascript), mais vous les utilisez comme s'il s'agissait d'éléments HTML personnalisés. Les boutons, les menus et tout autre contenu de la page d'accueil peuvent tous être créés comme composants. Les composants peuvent également contenir des informations sur l'état et afficher une démarque.

Après avoir appris à créer des composants dans React, vous serez capable de diviser des applications complexes en petits morceaux plus faciles à construire et à entretenir.

Dans ce tutoriel, vous allez créer une liste d'emojis qui afficheront leur nom lors du clic. Les emojis seront construits à l'aide d'un composant personnalisé et seront appelés de l'intérieur d'un autre composant personnalisé. À la fin de ce tutoriel, vous aurez fabriqué des composants personnalisés en utilisant à la fois des classes et des fonctions JavaScript, et vous comprendrez comment séparer le code existant en morceaux réutilisables et comment stocker les composants dans une structure de fichier lisible.

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 utiliserez la syntaxe JSX, que vous pouvez découvrir dans notre tutoriel [Comment créer des éléments avec JSX](https://www.digitalocean.com/community/tutorials/how-to-create-react-elements-with-jsx).
* 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 - Mettre en place le projet React

Dans cette étape, vous allez créer une base pour votre projet en utilisant l'application Create React App. Vous modifierez également le projet par défaut pour créer votre projet de base en mappant une liste d'emojis et en ajoutant une petite quantité de style.

D'abord, créez un nouveau projet. Ouvrez un terminal, puis exécutez la commande suivante :

* npx create-react-app tutorial-03-component

Une fois cette opération terminée, passez dans le répertoire du projet :

* cd tutorial-03-component

Ouvrez le code App.js dans votre éditeur de texte :

Ensuite, retirez le code modèle créé par Create React App, puis remplacez le contenu par le nouveau code React qui affiche une liste d'emojis :

tutorial-03-component/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";

const displayAction = false;

return(

<div className="container">

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

{displayAction && <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;

Ce code utilise la syntaxe JSX pour [mapper()](https://www.digitalocean.com/community/tutorials/how-to-use-array-methods-in-javascript-iteration-methods#map()) sur le [tableau](https://www.digitalocean.com/community/tutorials/understanding-arrays-in-javascript) des emojis et les répertorier en tant qu'éléments de liste <li>. Il joint également les [événements](https://www.digitalocean.com/community/tutorials/understanding-events-in-javascript) onClick pour afficher les données emoji dans le navigateur. Pour explorer le code plus en détail, consultez la page [Comment créer des éléments React avec JSX](https://www.digitalocean.com/community/tutorials/how-to-create-react-elements-with-jsx) qui contient une explication détaillée du JSX.

Enregistrez et fermez le fichier. Vous pouvez maintenant supprimer le logo.svg puisqu'il faisait partie du modèle et que vous n'y faites plus référence :

Maintenant, actualisez le style. Ouvrez src/App.css :

Remplacez le contenu par le CSS suivant pour centrer les éléments et ajuster la police :

tutorial-03-component/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;

}

Cela utilise flex pour centrer les principaux <h1> et éléments de la liste. Cela supprime également les styles de boutons par défaut et les styles <li> pour que les emojis se disposent en rang. Plus de détails sont disponibles dans [Comment créer des éléments React avec JSX](https://www.digitalocean.com/community/tutorials/how-to-create-react-elements-with-jsx).

Enregistrez et quittez le fichier.

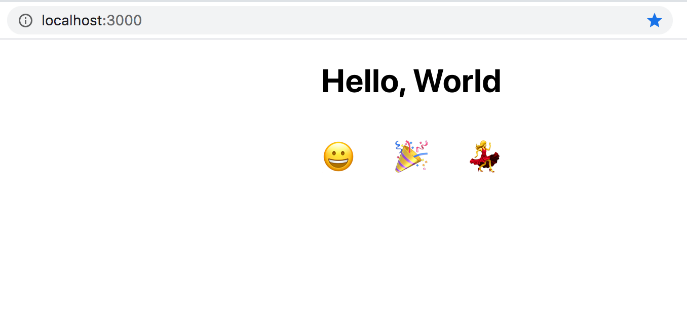
Ouvrez une autre fenêtre de terminal dans la racine de votre projet. Démarrez le projet avec la commande suivante :

* npm start

Après l'exécution de la commande, vous verrez le projet s'exécuter dans votre navigateur web à [http://localhost:3000](http://localhost:3000/).

Laissez-le fonctionner pendant toute la durée de votre projet. Chaque fois que vous enregistrez le projet, le navigateur se rafraîchit automatiquement et affiche le code le plus récent.

Vous verrez votre page de projet avec **Hello, World** et les trois emojis dont vous avez fait la liste dans votre fichier App.js :



Maintenant que vous avez configuré votre code, vous pouvez commencer à rassembler les composants dans React.

Étape 2 - Créer un composant indépendant avec des classes React

Maintenant que votre projet est lancé, vous pouvez commencer à fabriquer votre composant personnalisé. Dans cette étape, vous allez créer un composant React indépendant en étendant la [classe](https://www.digitalocean.com/community/tutorials/understanding-classes-in-javascript) de base de composants React. Vous allez créer une nouvelle classe, ajouter des méthodes et utiliser la fonction de rendu pour afficher les données.

Les composants React sont des éléments autonomes que vous pouvez réutiliser tout au long d'une page. En fabriquant de petits morceaux de code ciblés, vous pouvez déplacer et réutiliser des morceaux à mesure que votre application se développe. La clé ici est qu'ils sont autonomes et ciblés, vous permettant ainsi de séparer le code en morceaux logiques. En fait, vous avez déjà travaillé avec des composants logiquement séparés : le fichier App.js est un composant fonctionnel que vous verrez plus en détail à l'étape 3.

Il existe deux types de composants personnalisés : *par classe* et *fonctionnels*. Le premier élément que vous allez réaliser est un élément basé sur la classe. Vous allez créer un nouveau composant appelé Instructions, qui explique les instructions pour la visionneuse emoji.

**Note :** Les composants basés sur les classes étaient autrefois la façon la plus populaire de créer des composants React. Mais avec l'introduction de [React Hooks](https://reactjs.org/docs/hooks-intro.html), de nombreux développeurs et bibliothèques se tournent vers l'utilisation de composants fonctionnels.

Bien que les composants fonctionnels soient maintenant la norme, vous trouverez souvent des composants de classe dans le code existant. Vous n'avez pas besoin de les utiliser, mais vous devez savoir comment les reconnaître. Ils donnent également une introduction claire à de nombreux concepts futurs, tels que la gestion de l'état. Dans ce tutoriel, vous apprendrez à fabriquer des composants de classe et des composants fonctionnels.

Pour commencer, créez un nouveau fichier. Par convention, les fichiers de composants sont en majuscules : src/Instructions.js

Ensuite, ouvrez le fichier dans votre éditeur de texte :

Tout d'abord, importez React et la classe de composant, puis exportez Instructions avec les lignes suivantes :

tutorial-03-component/src/Instructions.js

import React, { Component } from 'react';

export default class Instructions extends Component {}

Importer React permettra de convertir le JSX. Le composant est une classe de base que vous étendez pour créer votre composant. Pour étendre cela, vous avez créé une classe qui porte le nom de votre composant (Instructions) et vous avez étendu le composant de base avec la ligne d’exportation. Vous exportez également cette classe par défaut avec des mots-clés par défaut d'exportation au début de la déclaration de classe.

Le nom de la classe doit être en majuscule et doit correspondre au nom du fichier. Ceci est important lors de l'utilisation d'outils de débogage qui afficheront le nom du composant. Si le nom correspond à la structure du fichier, il sera plus facile de localiser l'élément concerné.

La classe de composants de base a [plusieurs méthodes](https://reactjs.org/docs/react-component.html#the-component-lifecycle) que vous pouvez utiliser dans votre classe personnalisée. La méthode la plus importante et la seule que vous utiliserez dans ce tutoriel, est la méthode render(). La méthode render() renvoie le code JSX que vous voulez afficher dans le navigateur.

Pour commencer, ajoutez une petite explication de l'application au moyen d'une balise <p> :

tutorial-03-component/src/Instructions.js

import React, { Component } from 'react';

export class Instructions extends Component {

render() {

return(

<p>Click on an emoji to view the emoji short name.</p>

)

}

}

Enregistrez et fermez le fichier. À ce stade, votre navigateur n'a pas encore été modifié. C'est parce que vous n'avez pas encore utilisé le nouveau composant. Pour utiliser ce composant, vous devrez l'ajouter à un autre composant qui se connecte au composant racine. Dans ce projet, <App> est le composant root dans index.js. Pour le faire apparaître dans votre application, vous devez l'ajouter au composant <App>.

Ouvrez src/App.js dans un éditeur de texte :

Tout d'abord, vous devez importer le composant :

tutorial-03-component/src/App.js

import React from 'react';

import Instructions from './Instructions';

import './App.css';

...

export default App;

Comme il s'agit de l'importation par défaut, vous pouvez importer vers n'importe quel nom. Il est préférable de garder les noms cohérents pour des raisons de lisibilité—l'importation doit correspondre au nom du composant qui doit correspondre au nom du fichier—mais la seule règle impérative est que le composant doit commencer par une majuscule. C'est ainsi que React sait que c'est [un composant React](https://reactjs.org/docs/jsx-in-depth.html#specifying-the-react-element-type).

Maintenant que vous avez importé le composant, ajoutez-le au reste de votre code comme s'il s'agissait d'un élément HTML personnalisé :

tutorial-03-component/src/App.js

import React from 'react';

import Instructions from './Instructions.js'

...

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>}

<Instructions />

<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>

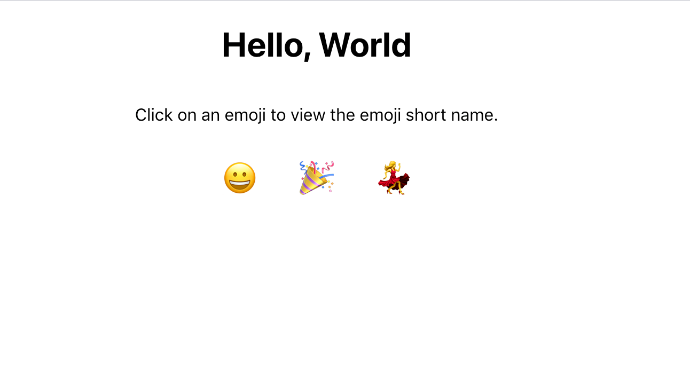
)

}

export default App;

Dans ce code, vous avez mis le composant entre crochets. Comme ce composant n'a pas d'enfants, il peut se fermer de lui-même en se terminant par />.

Sauvegardez le fichier. Lorsque vous le ferez, la page se rafraîchira et vous verrez le nouveau composant.



Maintenant que vous avez du texte, vous pouvez ajouter une image. Téléchargez une image d'emoji sur [wikimedia](https://commons.wikimedia.org/wiki/File:Twemoji_1f602.svg) et sauvegardez-la dans le répertoire src comme emoji.svg

Ensuite, ouvrez votre fichier de composants : src/Instructions.js

Importez l'emoji et ajoutez-le à votre composant personnalisé avec un lien dynamique :

tutorial-03-component/src/Instructions.js

import React, { Component } from 'react';

import emoji from './emoji.svg'

export default class Instructions extends Component {

render() {

return(

<>

<img alt="laughing crying emoji" src={emoji} />

<p>Click on an emoji to view the emoji short name.</p>

</>

)

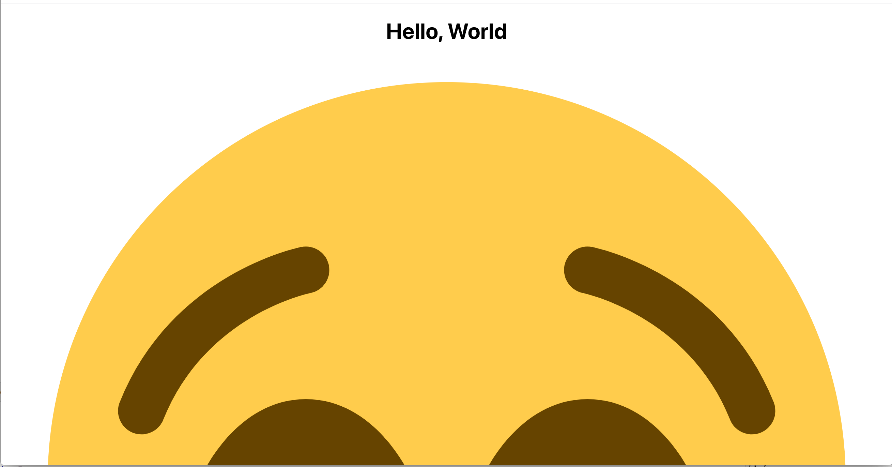
}

}

Notez que vous devez inclure l'extension de fichier .svg lors de l'importation. Lorsque vous effectuez une importation, vous importez un chemin dynamique qui est créé par [webpack](https://webpack.js.org/) lorsque le code se compile. Pour plus d'informations, voir [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 envelopper le <img> et <p> avec des balises vides pour s'assurer que vous ne renvoyez qu'un seul élément.

Sauvegardez le fichier. Lorsque vous relancez, l'image sera très grande par rapport au reste du contenu :



Pour rendre l'image plus petite, vous devrez ajouter du CSS et un nom de classe à votre composant personnalisé.

Tout d'abord, dans Instructions.js, changez les balises vides en div et donnez-lui un className d’instructions :

tutorial-03-component/src/Instructions.js

import React, { Component } from 'react';

import emoji from './emoji.svg'

export default class Instructions extends Component {

render() {

return(

<div className="instructions">

<img alt="laughing crying emoji" src={emoji} />

<p>Click on an emoji to view the emoji short name.</p>

</div>

)

}

}

Enregistrez et fermez le fichier. Ensuite, ouvrez App.css :

Créer des règles pour le [sélecteur de classe](https://developer.mozilla.org/en-US/docs/Web/CSS/Class_selectors) .instructions :

tutorial-03-component/src/App.css

.container {

display: flex;

flex-direction: column;

align-items: center;

}

...

.instructions {

display: flex;

flex-direction: column;

align-items: center;

}

Lorsque vous ajoutez un affichage de style flex, vous centrez la img et le p avec [flexbox](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox). Vous avez changé de direction pour que tout s'aligne verticalement avec flex-direction: column;. La ligne align-items: center; va centrer les éléments à l'écran.

Maintenant que vos éléments sont alignés, vous devez modifier la taille de l'image. Donnez à l’img à l'intérieur de la div une largeur et hauteur de 100px.

tutorial-03-component/src/App.css

.container {

display: flex;

flex-direction: column;

align-items: center;

}

...

.instructions {

display: flex;

flex-direction: column;

align-items: center;

}

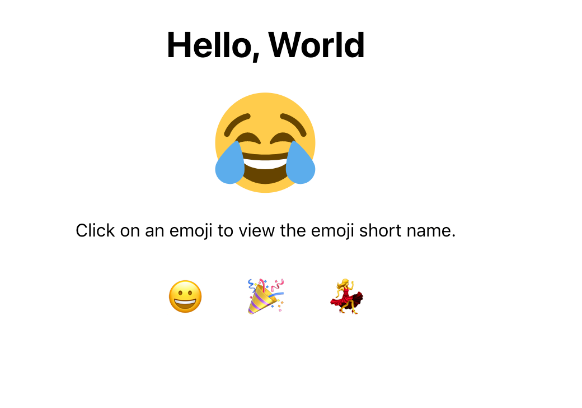
.instructions img {

width: 100px;

height: 100px;

}

Enregistrez et fermez le fichier. Le navigateur va se recharger et vous verrez que l'image est beaucoup plus petite :



À ce stade, vous avez créé un composant personnalisé indépendant et réutilisable. Pour voir comment il est réutilisable, ajoutez une deuxième instance à App.js.

Ouvrez App.js:

Dans App.js, ajoutez une deuxième instance du composant :

tutorial-03-component/src/App.js

import React from 'react';

import Instructions from './Instructions.js'

...

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>}

<Instructions />

<Instructions />

<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>

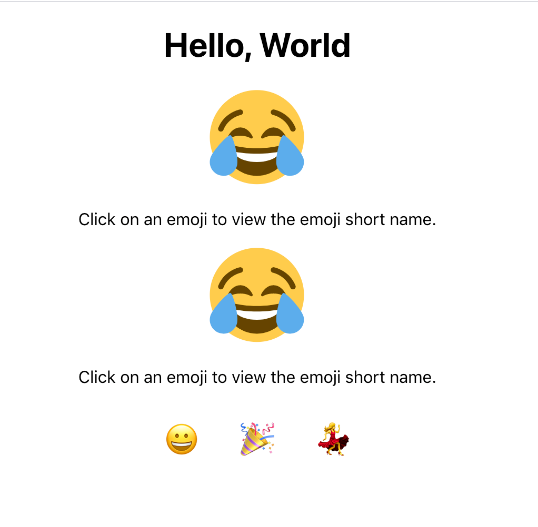
</div>

)

}

export default App;

Sauvegardez le fichier. Lorsque le navigateur se recharge, vous verrez le composant deux fois.



Dans ce cas, vous ne voudriez pas avoir deux cas d’Instructions mais vous pouvez voir que le composant peut être efficacement réutilisé. Lorsque vous créez des boutons ou des tables personnalisés, vous les utiliserez probablement plusieurs fois sur une même page, ce qui les rendra parfaits pour les composants personnalisés.

Pour l'instant, vous pouvez supprimer la balise d'image supplémentaire. Dans votre éditeur de texte, supprimez le deuxième <Instructions /> et sauvegardez le fichier :

tutorial-03-component/src/App.js

import React from 'react';

import Instructions from './Instructions.js'

...

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>}

<Instructions />

<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;

Vous disposez désormais d'un composant indépendant et réutilisable que vous pouvez ajouter plusieurs fois à un composant parent. La structure que vous avez maintenant fonctionne pour un petit nombre de composants, mais il y a un léger problème. Tous les dossiers sont mélangés. L'image relative à <Instructions> se trouve dans le même répertoire que les actifs de <App>. Vous mélangez également le code CSS pour <App> avec le CSS pour <Instructions>.

Dans l'étape suivante, vous créerez une structure de fichiers qui donnera à chaque composant son indépendance en regroupant ses fonctionnalités, ses styles et ses dépendances, vous donnant ainsi la possibilité de les déplacer selon vos besoins.

Étape 3 - Créer une structure de fichier lisible

Au cours de cette étape, vous allez créer une structure de fichiers pour organiser vos composants et leurs actifs, tels que les images, les CSS et autres fichiers JavaScript. Vous regrouperez les codes par composant, et non par type d'actif. En d'autres termes, vous n'aurez pas de répertoire séparé pour le CSS, les images et le JavaScript. Au lieu de cela, vous aurez un répertoire séparé pour chaque composant qui contiendra les CSS, JavaScript et images pertinents. Dans les deux cas, vous [séparez les préoccupations](https://github.com/alexnm/react-training/blob/master/03-introduction-to-react.md).

Comme vous avez une composante indépendante, vous avez besoin d'une structure de fichiers qui regroupe le code correspondant. Actuellement, tout est dans le même répertoire. Faites la liste des éléments de votre répertoire src :

Le résultat montrera que les choses deviennent assez encombrées :

Output

App.css Instructions.js index.js

App.js emoji.svg serviceWorker.js

App.test.js index.css setupTests.js

Vous avez un code pour le composant <App> (App.css, App.js, et App.test.js) qui se trouve à côté de votre composant root (index.css et index.js) et votre composant personnalisé Instructions.js.

React est [intentionnellement agnostique](https://reactjs.org/docs/faq-structure.html) au sujet de la structure des fichiers. Il ne recommande pas une structure particulière, et le projet peut fonctionner avec une variété de hiérarchies de fichiers différentes. Mais nous recommandons d'ajouter un peu d'ordre pour éviter de surcharger votre répertoire root avec des composants, des fichiers CSS et des images qui seront difficiles à naviguer. En outre, une désignation explicite peut permettre de voir plus facilement quels éléments de votre projet sont liés. Par exemple, un fichier image nommé Logo.svg peut ne pas faire clairement partie d'un composant appelé Header.js.

L'une des structures les plus simples consiste à créer un répertoire de composants avec un répertoire séparé pour chaque composant. Cela vous permettra de regrouper vos composants séparément de votre code de configuration, par exemple serviceWorker, tout en regroupant les actifs avec les composants.

Créer un répertoire de composants

Pour commencer, créez un répertoire appelé composants:

Ensuite, déplacez les composants et le code suivants dans le répertoire : App.css, App.js, App.test.js, Instructions.jset emoji.svg :

Ici, vous utilisez un [joker (\*)](https://tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm) pour sélectionner tous les fichiers qui commencent par App..

Après avoir déplacé le code, vous verrez une erreur dans le fonctionnement de votre terminal npm start

Failed to compile.

./src/App.js

Error: ENOENT: no such file or directory, open 'your\_file\_path/tutorial-03-component/src/App.js'

N'oubliez pas que tout le code importe en utilisant des chemins relatifs. Si vous changez le chemin d'accès de certains fichiers, vous devrez mettre à jour le code.

Pour ce faire, ouvrez index.js.

Ensuite, changez le chemin de la fonction d'importation de l’App à l'importation à partir du répertoire components/.

tutorial-03-component/src/index.js

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './components/App';

import \* as serviceWorker from './serviceWorker';

...

serviceWorker.unregister();

Enregistrez et fermez le fichier. Votre script détectera les changements et l'erreur disparaîtra.

Vous avez maintenant les composants dans un répertoire séparé. À mesure que vos applications deviennent plus complexes, vous pouvez disposer de répertoires pour les services API, les magasins de données et les fonctions utilitaires. Séparer le code du composant est la première étape, mais vous avez toujours le code CSS pour Instructions mélangé dans le fichierApp.css de l'entreprise. Pour créer cette séparation logique, vous devez d'abord déplacer les composants dans des répertoires séparés.

Déplacer des composants vers des répertoires individuels

Tout d'abord, créez un répertoire spécifique pour le composant <App> :

Ensuite, déplacez les fichiers correspondants dans le nouveau répertoire :

Lorsque vous le ferez, vous obtiendrez une erreur similaire à la dernière section :

Failed to compile.

./src/components/App.js

Error: ENOENT: no such file or directory, open 'your\_file\_path/tutorial-03-component/src/components/App.js'

Dans ce cas, vous devrez mettre à jour deux choses. Tout d'abord, vous devez mettre à jour le chemin d'accès dans index.js.

Ouvrez le fichier index.js :

Ensuite, mettez à jour le chemin d'importation pour App pour qu'il pointe vers le répertoire App.

tutorial-03-component/src/index.js

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './components/App/App';

import \* as serviceWorker from './serviceWorker';

...

serviceWorker.unregister();

Enregistrez et fermez le fichier. L'application ne fonctionne toujours pas. Vous verrez une erreur de ce type :

Failed to compile.

./src/components/App/App.js

Module not found: Can't resolve './Instructions.js' in 'your\_file\_path/tutorial-03-component/src/components/App'

Puisque <Instructions> n'est pas au même niveau de répertoire que le composant<App>, vous devez modifier le chemin d'importation. Avant cela, créez un répertoire pour Instructions. Créer un répertoire appelé Instructions dans le répertoire src/components :

Puis déplacez Instructions.js et emoji.svg dans ce répertoire :

Maintenant que le composant Instructions a été créé, vous pouvez terminer la mise à jour des chemins de fichiers pour connecter votre composant à votre application.

Mettre à jour les chemins d’importation

Maintenant que les composants sont dans des répertoires individuels, vous pouvez ajuster le chemin d'importation dans App.js.

Ouvrez App.js:

Comme le chemin est relatif, vous devrez remonter d'un répertoire -src/components- puis dans le répertoire Instructions pour Instructions.js, mais comme il s'agit d'un fichier JavaScript, vous n'avez pas besoin de l'importation finale.

tutorial-03-component/src/components/App/App.js

import React from 'react';

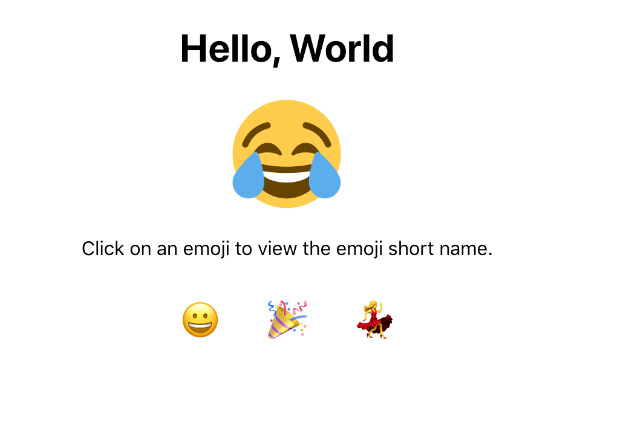
import Instructions from '../Instructions/Instructions.js';

import './App.css';

...

export default App;

Enregistrez et fermez le fichier. Maintenant que vos importations utilisent toutes le bon chemin, votre navigateur se met à jour et affiche l'application.



**Note :** Vous pouvez également appeler le fichier root dans chaque répertoire index.js. Par exemple, au lieu de src/components/App/App.js vous pourriez créer src/components/App/index.js. L'avantage est que vos importations sont légèrement plus petites. Si le chemin mène à un répertoire, l'importation cherchera un fichier index.js. L'importation pour src/components/App/index.js dans le fichier src/index.js serait import ./components/App. L'inconvénient de cette approche est que de nombreux fichiers portent le même nom, ce qui peut rendre la lecture difficile dans certains éditeurs de texte. En fin de compte, c'est une décision personnelle et collective, mais il est préférable d'être cohérent.

Séparer le code dans les fichiers partagés

Aujourd'hui, chaque composant a son propre répertoire, mais tout n'est pas totalement indépendant. La dernière étape consiste à extraire le CSS pour Instructions dans un fichier séparé.

Créez d'abord un fichier CSS dans src/components/instructions:

Ensuite, ouvrez le fichier CSS dans votre éditeur de texte :

Ajoutez les instructions CSS que vous avez créées dans une section précédente :

tutorial-03-component/src/components/Instructions/Instructions.css

.instructions {

display: flex;

flex-direction: column;

align-items: center;

}

.instructions img {

width: 100px;

height: 100px;

}

Enregistrez et fermez le fichier. Ensuite, supprimez les instructions CSS de src/components/App/App.css.

Supprimez les lignes concernant .instructions. Le fichier final ressemblera à ceci :

tutorial-03-component/src/components/App/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;

}

Enregistrez et fermez le fichier. Enfin, importez le CSS dans Instructions.js :

Importez le CSS en utilisant le chemin relatif :

tutorial-03-component/src/components/Instructions/Instructions.js

import React, { Component } from 'react';

import './Instructions.css';

import emoji from './emoji.svg'

export default class Instructions extends Component {

render() {

return(

<div className="instructions">

<img alt="laughing crying emoji" src={emoji} />

<p>Click on an emoji to view the emoji short name.</p>

</div>

)

}

}

Enregistrez et fermez le fichier. La fenêtre de votre navigateur aura le même aspect qu'auparavant, sauf que maintenant tous les fichiers sont regroupés dans le même répertoire.



Maintenant, jetez un dernier regard sur la structure. Tout d'abord, le répertoire src/ :

Vous avez le composant root index.js et le CSS index.css correspondant à côté du répertoire components/ et des fichiers utilitaires tels que serviceWorker.js et setupTests.js :

components serviceWorker.js

index.css setupTests.js

index.js

Ensuite, regardez à l'intérieur des composants:

Vous verrez un répertoire pour chaque composant :

App Instructions

Si vous regardez à l'intérieur de chaque composant, vous verrez le code du composant, le CSS, le test et les fichiers images s'ils existent.

App.css App.js App.test.js

Instructions.css Instructions.js emoji.svg

À ce stade, vous avez créé une structure solide pour votre projet. Vous avez déplacé beaucoup de code, mais maintenant que vous avez une structure, il sera plus facile de l'adapter.

Ce n'est pas la seule façon de composer votre structure. Certaines structures de fichiers peuvent tirer profit du [fractionnement du code](https://reactjs.org/docs/code-splitting.html) en spécifiant un répertoire qui sera divisé en différents paquets. D'autres structures de [fichiers divisent par itinérair](https://survivejs.com/react/advanced-techniques/structuring-react-projects/#directory-per-view)e et utilisent un répertoire commun pour les composants utilisés sur les différents itinéraires.

Pour l'instant, tenez-vous en à une approche moins complexe. Lorsque le besoin d'une autre structure se fait sentir, il est toujours plus facile de passer du simple au complexe. Commencer avec une structure complexe avant d'en avoir besoin rendra le remaniement difficile.

Maintenant que vous avez créé et organisé un composant basé sur la classe, l'étape suivante consiste à créer un composant fonctionnel.

Étape 4 - Construire un composant fonctionnel

Dans cette étape, vous allez créer un composant fonctionnel. Les composants fonctionnels sont les plus courants dans le code React contemporain. Ces composants ont tendance à être plus courts et, contrairement aux composants basés sur la classe, ils peuvent utiliser [des crochets React](https://reactjs.org/docs/hooks-intro.html), une nouvelle forme de gestion de l'état et des événements.

Un composant fonctionnel est une fonction JavaScript qui renvoie du JSX. Il n'est pas nécessaire d'étendre quoi que ce soit et il n'y a pas de méthodes spéciales à mémoriser.

Pour remanier <Instructions> en tant que composant fonctionnel, vous devez transformer la classe en fonction et supprimer la méthode de rendu afin de ne conserver que la déclaration de retour.

Pour ce faire, il faut d'abord ouvrir Instructions.js dans un éditeur de texte.

Changez la déclaration de classe en une déclaration de fonction :

tutorial-03-component/src/components/Instructions/Instructions.js

import React, { Component } from 'react';

import './Instructions.css';

import emoji from './emoji.svg'

export default function Instructions() {

render() {

return(

<div className="instructions">

<img alt="laughing crying emoji" src={emoji} />

<p>Click on an emoji to view the emoji short name.</p>

</div>

)

}

}

Ensuite, supprimez l'importation de { Component } :

tutorial-03-component/src/components/Instructions/Instructions.js

import React from 'react';

import './Instructions.css';

import emoji from './emoji.svg'

export default function Instructions() {

render() {

return(

<div className="instructions">

<img alt="laughing crying emoji" src={emoji} />

<p>Click on an emoji to view the emoji short name.</p>

</div>

)

}

}

Enfin, il faut supprimer la méthode render(). À ce stade, vous ne renvoyez que JSX.

tutorial-03-component/src/components/Instructions/Instructions.js

import React from 'react';

import './Instructions.css';

import emoji from './emoji.svg'

export default function Instructions() {

return(

<div className="instructions">

<img alt="laughing crying emoji" src={emoji} />

<p>Click on an emoji to view the emoji short name.</p>

</div>

)

}

Sauvegardez le fichier. Le navigateur va se remettre à jour et vous verrez votre page telle qu'elle était auparavant.



Vous pouvez également réécrire la fonction comme une [fonction de flèche](https://www.digitalocean.com/community/tutorials/how-to-define-functions-in-javascript#arrow-functions) en utilisant le retour implicite. La principale différence étant que vous perdez le corps de la fonction. Vous devrez également commencer par attribuer la fonction à une variable, puis exporter la variable :

tutorial-03-component/src/components/Instructions/Instructions.js

import React from 'react';

import './Instructions.css';

import emoji from './emoji.svg'

const Instructions = () => (

<div className="instructions">

<img alt="laughing crying emoji" src={emoji} />

<p>Click on an emoji to view the emoji short name.</p>

</div>

)

export default Instructions;

Les composants fonctionnels simples et les composants basés sur la classe sont très similaires. Lorsque vous disposez d'un composant simple qui ne stocke pas l'état, il est préférable d'utiliser un composant fonctionnel. La véritable différence entre les deux est la façon dont vous stockez l'état d'un composant et utilisez ses propriétés. Les composants basés sur les classes utilisent des méthodes et des propriétés pour fixer l'état et ont tendance à être un peu plus longs. Les composants fonctionnels utilisent des hooks pour stocker l'état ou gérer les changements et ont tendance à être un peu plus courts.

Conclusion

Vous avez maintenant une petite application avec des pièces indépendantes. Vous avez créé deux grands types de composants : fonctionnels et de classe. Vous avez séparé des parties des composants dans des répertoires afin de pouvoir conserver des morceaux de code similaires regroupés. Vous avez également importé et réutilisé les composants.

En comprenant les composants, vous pouvez commencer à considérer vos applications comme des pièces que vous pouvez démonter et remonter. Les projets deviennent modulaires et interchangeables. La capacité de voir des applications entières comme une série de composants est une étape importante dans pour penser dans un environnement React. Si vous souhaitez consulter d'autres tutoriels de la série React, consultez notre [page thématique React](https://www.digitalocean.com/community/tutorial_series/how-to-code-in-react-js) ou retournez à la [page de la série Comment coder en React.js](https://www.digitalocean.com/community/tags/react).