**Comment gérer l'état sur les composants de classe React**

introduction

Dans [React](https://reactjs.org/) , l' *état* fait référence à une structure qui suit l'évolution des données au fil du temps dans votre application. La gestion de l'état est une compétence cruciale dans React car elle vous permet de créer des composants interactifs et des applications Web dynamiques. L'état est utilisé pour tout, du suivi des entrées de formulaire à la capture de données dynamiques à partir d'une API. Dans ce didacticiel, vous découvrirez un exemple de gestion de l' [état](https://reactjs.org/docs/state-and-lifecycle.html) sur des [composants](https://www.digitalocean.com/community/tutorials/how-to-create-custom-components-in-react) basés sur des classes .

Au moment de la rédaction de ce didacticiel, la [documentation](https://reactjs.org/docs/hooks-faq.html#should-i-use-hooks-classes-or-a-mix-of-both) officielle de [React](https://reactjs.org/docs/hooks-faq.html" \l "should-i-use-hooks-classes-or-a-mix-of-both) encourage les développeurs à adopter [React Hooks](https://reactjs.org/docs/hooks-state.html) pour gérer l'état avec [des composants fonctionnels](https://www.digitalocean.com/community/tutorials/how-to-create-custom-components-in-react#step-4-%E2%80%94-building-a-functional-component) lors de l'écriture de nouveau code, plutôt que d'utiliser [des composants basés](https://www.digitalocean.com/community/tutorials/how-to-create-custom-components-in-react#step-2-%E2%80%94-creating-an-independent-component-with-react-classes) sur des [classes](https://www.digitalocean.com/community/tutorials/how-to-create-custom-components-in-react#step-2-%E2%80%94-creating-an-independent-component-with-react-classes) . Bien que l'utilisation de React Hooks soit considérée comme une pratique plus moderne, il est également important de comprendre comment gérer l'état sur les composants basés sur les classes. L'apprentissage des concepts de la gestion d'état vous aidera à naviguer et à dépanner la gestion d'état basée sur les classes dans les bases de code existantes et vous aidera à décider quand la gestion d'état basée sur les classes est plus appropriée. Il existe également une méthode basée sur les classes appelée [componentDidCatch](https://reactjs.org/docs/error-boundaries.html) qui n'est pas disponible dans Hooks et qui nécessitera de définir l'état à l'aide de méthodes de classe.

Ce didacticiel vous montrera d'abord comment définir l'état à l'aide d'une valeur statique, ce qui est utile dans les cas où l'état suivant ne dépend pas du premier état, comme la définition des données d'une API qui remplace les anciennes valeurs. Ensuite, il expliquera comment définir un état comme état actuel, ce qui est utile lorsque l'état suivant dépend de l'état actuel, comme le basculement d'une valeur. Pour explorer ces différentes façons de définir l'état, vous allez créer un composant de page produit que vous mettrez à jour en ajoutant des achats à partir d'une liste d'options.

Conditions préalables

* Vous aurez besoin d'un environnement de développement exécutant [Node.js](https://nodejs.org/en/about/) ; ce tutoriel a été testé sur Node.js version 10.20.1 et npm version 6.14.4. Pour l'installer sur macOS ou Ubuntu 18.04, suivez les étapes décrites dans [Comment installer Node.js et créer un environnement de développement local sur macOS](https://www.digitalocean.com/community/tutorials/how-to-install-node-js-and-create-a-local-development-environment-on-macos) ou dans la section **Installation à l'aide d'un PPA** de [Comment installer Node.js sur Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-18-04) .
* Dans ce didacticiel, vous allez créer des applications avec [Create React App](https://github.com/facebook/create-react-app) . Vous pouvez trouver des instructions pour installer une application avec Create React App sur [Comment configurer 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 aurez également besoin d'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 que d'une connaissance de base de HTML et CSS. Une bonne ressource pour HTML et CSS est le [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML) .

Étape 1 - Création d'un projet vide

Dans cette étape, vous allez créer un nouveau projet à l'aide de [Create React App](https://github.com/facebook/create-react-app) . Ensuite, vous supprimerez l'exemple de projet et les fichiers associés qui sont installés lorsque vous amorcez le projet. Enfin, vous allez créer une structure de fichiers simple pour organiser vos composants. Cela vous donnera une base solide sur laquelle créer l'exemple d'application de ce didacticiel pour la gestion de l'état sur les composants basés sur les classes.

Pour commencer, créez un nouveau projet. Dans votre terminal, exécutez le script suivant pour installer un nouveau projet en utilisant create-react-app:

* npx create-react-app state-class-tutorial

Copie

Une fois le projet terminé, accédez au répertoire:

* cd state-class-tutorial

Copie

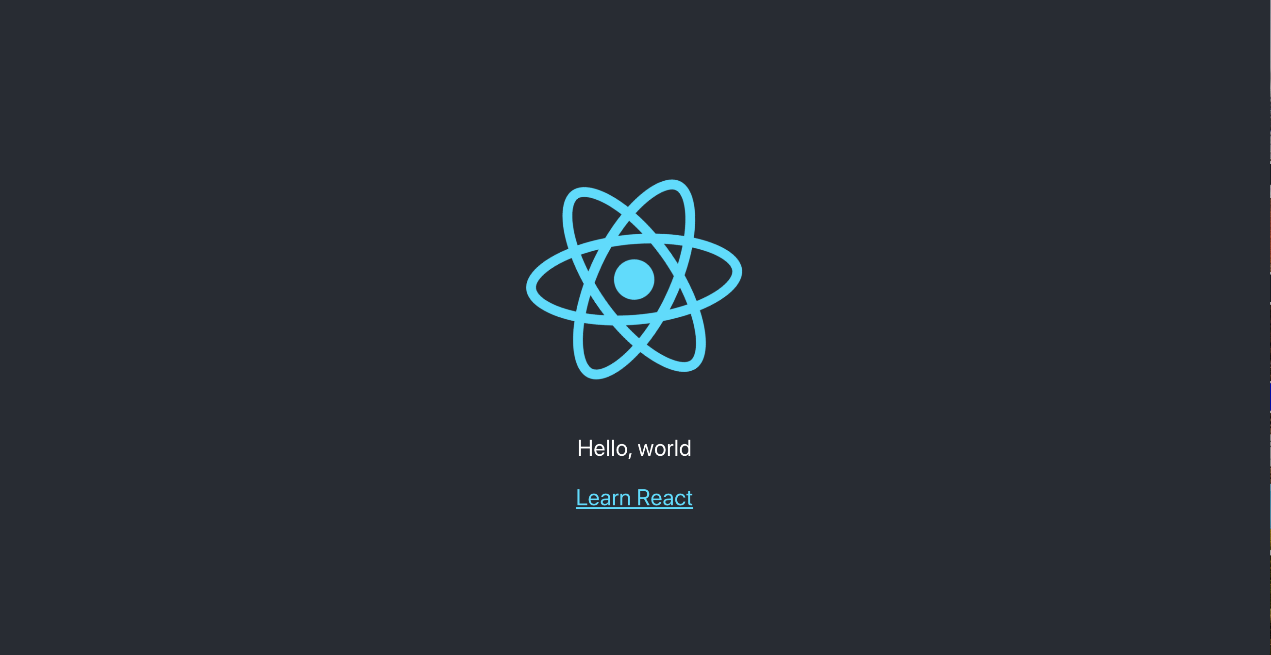
Dans un nouvel onglet ou une nouvelle fenêtre de terminal, démarrez le projet à l'aide du [script de démarrage Create React App](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-react-project-with-create-react-app#step-3-%E2%80%94-starting-the-server) . Le navigateur s'actualisera automatiquement en cas de modification, alors laissez ce script en cours d'exécution pendant que vous travaillez:

* npm start

Copie

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

Votre navigateur se chargera avec une simple 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 standard pour pouvoir avoir un projet vide.

Pour commencer, ouvrez src/App.js dans un éditeur de texte. C'est le composant racine qui est injecté dans la page. Tous les composants commenceront à partir d'ici. Vous pouvez trouver plus d’informations sur App.jsà [Comment mettre en place un projet REACT avec App Créer React](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) .

Ouvrez src/App.jsavec la commande suivante:

* nano src/App.js

Copie

Vous verrez un fichier comme celui-ci:

tutoriel-classe-d'état / 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;

Supprimez la ligne import logo from './logo.svg';. Remplacez ensuite tout dans l' returninstruction pour renvoyer un ensemble de balises vides: <></>. Cela vous donnera une page valide qui ne renvoie rien. Le code final ressemblera à ceci:

tutoriel-classe-d'état / src / App.js

import React from 'react';

import './App.css';

function App() {

return <></>;

}

export default App;

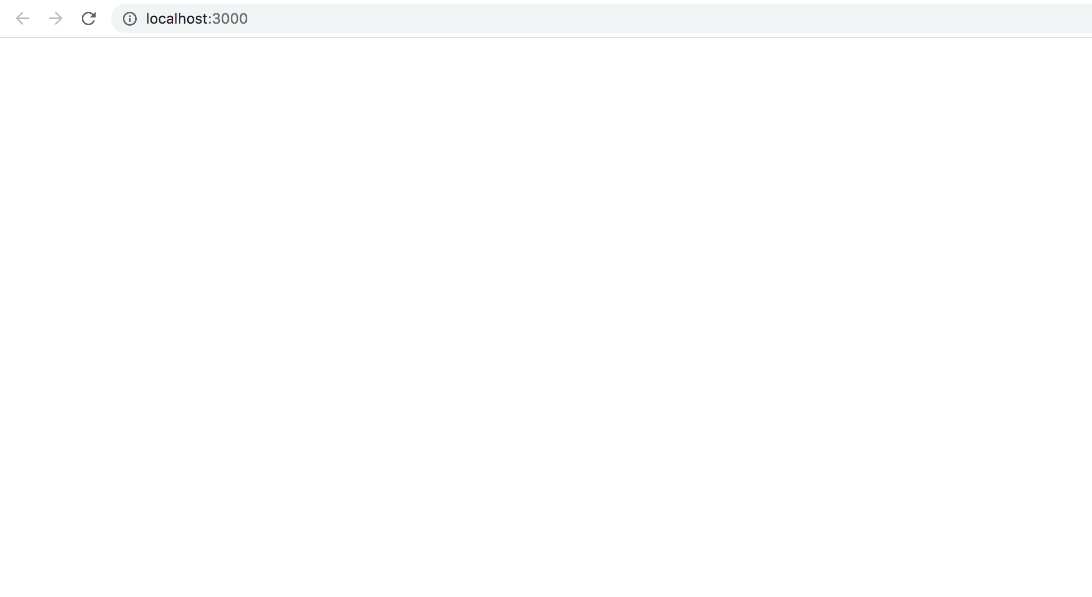
Enregistrez et quittez l'éditeur de texte.

Enfin, supprimez le logo. Vous ne l'utiliserez pas dans votre application et vous devez supprimer les fichiers inutilisés pendant que vous travaillez. Cela vous évitera la confusion à long terme.

Dans la fenêtre du terminal, tapez la commande suivante:

* rm src/logo.svg

Si vous regardez votre navigateur, vous verrez un écran vide.



Maintenant que vous avez effacé l'exemple de projet Create React App, créez une structure de fichiers simple. Cela vous aidera à garder vos composants isolés et indépendants.

Créez un répertoire appelé componentsdans le srcrépertoire. Cela contiendra tous vos composants personnalisés.

* mkdir src/components

Copie

Chaque composant aura son propre répertoire pour stocker le fichier du composant avec les styles, images et tests.

Créez un répertoire pour App:

* mkdir src/components/App

Copie

Déplacez tous les Appfichiers dans ce répertoire. Utilisez le caractère générique,, \*pour sélectionner les fichiers commençant par, App.quelle que soit l'extension de fichier. Ensuite, utilisez la mvcommande pour les mettre dans le nouveau répertoire:

* mv src/App.\* src/components/App

Ensuite, mettez à jour le chemin d'importation relatif dans index.js, qui est le composant racine qui amorce l'ensemble du processus:

* nano src/index.js

L'instruction d'importation doit pointer vers le fichier App.js dans le répertoire App, apportez donc la modification suivante en surbrillance:

tutoriel-classe-d'état / 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';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

Enregistrez et quittez le fichier.

Maintenant que le projet est configuré, vous pouvez créer votre premier composant.

Étape 2 - Utilisation de l'état dans un composant

Dans cette étape, vous allez définir l'état initial d'un composant sur sa classe et référencer l'état pour afficher une valeur. Vous créerez ensuite une page de produit avec un panier qui affiche le total des articles dans le panier en utilisant la valeur de l'état. À la fin de l'étape, vous saurez les différentes façons de conserver une valeur et quand vous devriez utiliser state plutôt qu'un accessoire ou une valeur statique.

Construire les composants

Commencez par créer un répertoire pour Product:

* mkdir src/components/Product

Ensuite, ouvrez Product.jsdans ce répertoire:

* nano src/components/Product/Product.js

Commencez par créer un composant sans état. Le composant comprendra deux parties : le panier, qui contient le nombre d'articles et le prix total, et le produit, qui a un bouton pour ajouter et supprimer un article. Pour l'instant, les boutons n'auront aucune action.

Ajoutez le code suivant à Product.js:

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

export default class Product extends Component {

render() {

return(

<div className="wrapper">

<div>

Shopping Cart: 0 total items.

</div>

<div>Total: 0</div>

<div className="product"><span role="img" aria-label="ice cream">🍦</span></div>

<button>Add</button> <button>Remove</button>

</div>

)

}

}

Vous avez également inclus quelques éléments div qui ont des noms de classe [JSX](https://www.digitalocean.com/community/tutorials/how-to-create-react-elements-with-jsx) afin que vous puissiez ajouter un style de base.

Enregistrez et fermez le fichier, puis ouvrez Product.css:

* nano src/components/Product/Product.css

Copie

Donnez un style léger pour augmenter font-sizele texte et les emoji:

tutoriel-classe-d'état / src / components / Product / Product.css

.product span {

font-size: 100px;

}

.wrapper {

padding: 20px;

font-size: 20px;

}

.wrapper button {

font-size: 20px;

background: none;

}

L'emoji aura besoin d'une taille de police beaucoup plus grande que le texte, car il agit comme l'image du produit dans cet exemple. En outre, vous supprimez l'arrière-plan dégradé par défaut des boutons en définissant la valeur background sur none.

Enregistrez et fermez le fichier.

Maintenant, effectuez le rendu du composant Product dans le composant App afin que vous puissiez voir les résultats dans le navigateur. Ouvrez App.js:

* nano src/components/App/App.js

Importez le composant et effectuez le rendu. Vous pouvez également supprimer l'importation CSS puisque vous ne l'utiliserez pas dans ce didacticiel:

tutoriel-classe-d'état / src / components / App / App.js

import React from 'react';

import Product from '../Product/Product';

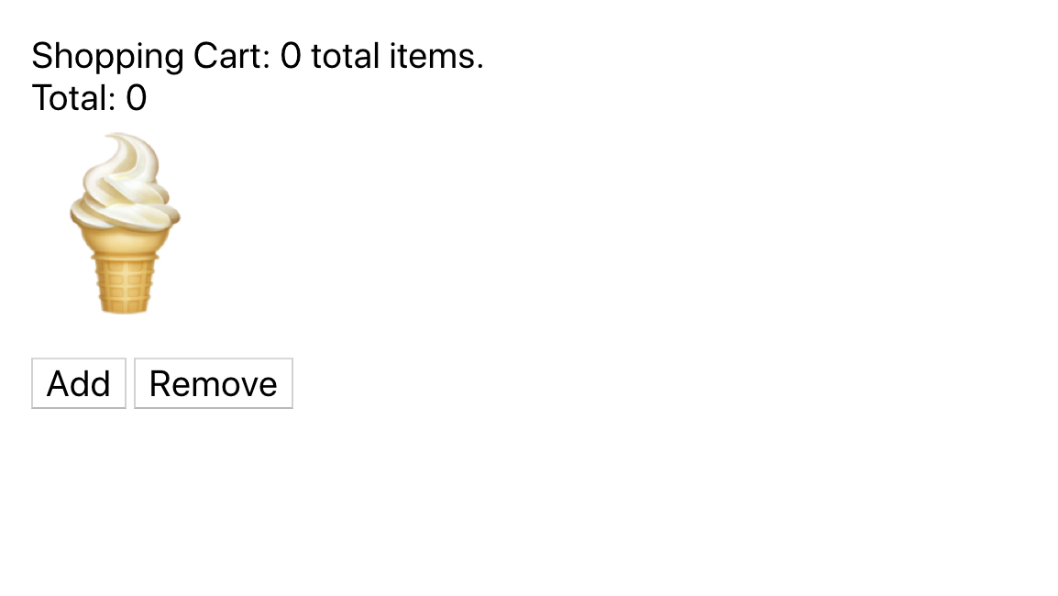
function App() {

return <Product />

}

export default App;

Enregistrez et fermez le fichier. Lorsque vous le faites, le navigateur s'actualisera et vous verrez le composant Product.



Définition de l'état initial sur un composant de classe

Il y a deux valeurs dans les valeurs de vos composants qui vont changer dans votre affichage : le nombre total d'articles et le coût total. Au lieu de les coder en dur, dans cette étape, vous les déplacerez dans un [objet](https://www.digitalocean.com/community/tutorials/understanding-objects-in-javascript) appelé state.

La propriété state d'une classe React est une propriété spéciale qui contrôle le rendu d'une page. Lorsque vous modifiez l'état, React sait que le composant est obsolète et sera automatiquement rendu. Lorsqu'un composant effectue un nouveau rendu, il modifie la sortie rendue pour inclure les informations les plus à jour dans state. Dans cet exemple, le composant sera rendu à nouveau chaque fois que vous ajoutez un produit au panier ou que vous le supprimez du panier. Vous pouvez ajouter d'autres propriétés à une classe React, mais elles n'auront pas la même capacité à déclencher un nouveau rendu.

Ouvrez Product.js:

* nano src/components/Product/Product.js

Ajoutez une propriété appelée state à la classe Product. Ajoutez ensuite deux valeurs à l'objet state : cart et total. Le cart sera un [tableau](https://www.digitalocean.com/community/tutorials/understanding-arrays-in-javascript) , car il peut éventuellement contenir de nombreux éléments. Le total sera un nombre. Après leur attribution, remplacez les références aux valeurs par : this.state.property

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

export default class Product extends Component {

state = {

cart: [],

total: 0

}

render() {

return(

<div className="wrapper">

<div>

Shopping Cart: {this.state.cart.length} total items.

</div>

<div>Total {this.state.total}</div>

<div className="product"><span role="img" aria-label="ice cream">🍦</span></div>

<button>Add</button> <button>Remove</button>

</div>

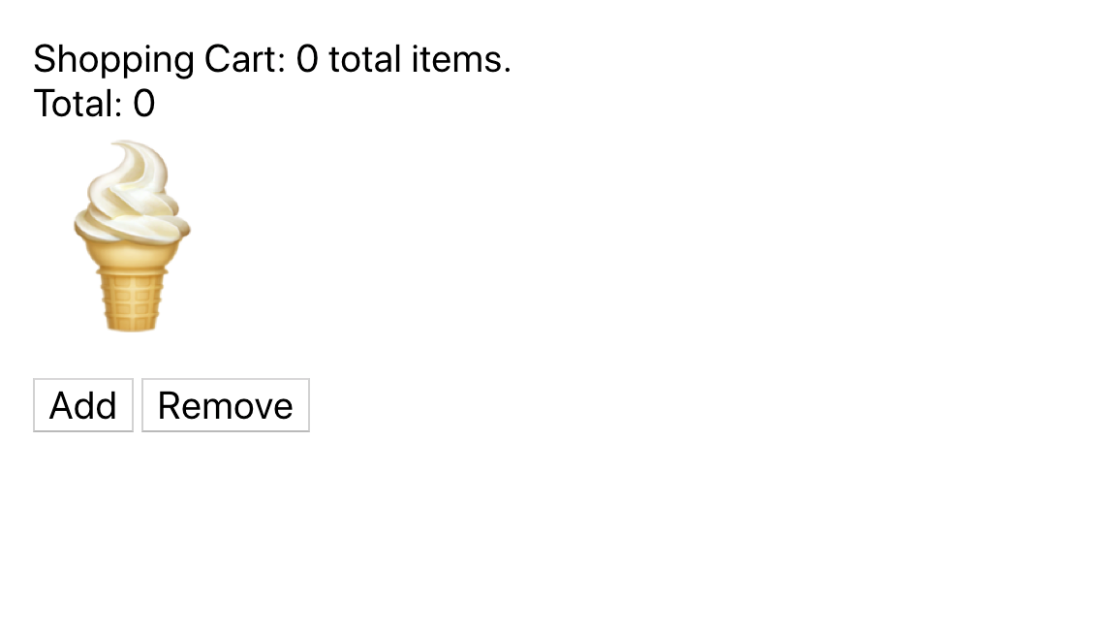
)

}

}

Notez que dans les deux cas, puisque vous référencez JavaScript à l'intérieur de votre JSX, vous devez envelopper le code entre accolades. Vous affichez également le length du tableau cart pour obtenir un décompte du nombre d'éléments dans le tableau.

Enregistrez le fichier. Lorsque vous le faites, le navigateur s'actualisera et vous verrez la même page qu'avant.



La propriété state est une propriété de classe standard, ce qui signifie qu'elle est accessible dans d'autres méthodes, pas seulement dans la méthode render.

Ensuite, au lieu d'afficher le prix sous forme de valeur statique, convertissez-le en une chaîne à l'aide de la méthode [toLocaleString](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toLocaleString), qui convertira le nombre en une chaîne correspondant à la façon dont les nombres sont affichés dans la région du navigateur.

Créez une méthode appelée getTotal() qui prend le state et le convertit en une chaîne localisée à l'aide d'un tableau de currencyOptions. Ensuite, remplacez la référence à state dans le JSX par un appel de méthode :

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

export default class Product extends Component {

state = {

cart: [],

total: 0

}

currencyOptions = {

minimumFractionDigits: 2,

maximumFractionDigits: 2,

}

getTotal = () => {

return this.state.total.toLocaleString(undefined, this.currencyOptions)

}

render() {

return(

<div className="wrapper">

<div>

Shopping Cart: {this.state.cart.length} total items.

</div>

<div>Total {this.getTotal()}</div>

<div className="product"><span role="img" aria-label="ice cream">🍦</span></div>

<button>Add</button> <button>Remove</button>

</div>

)

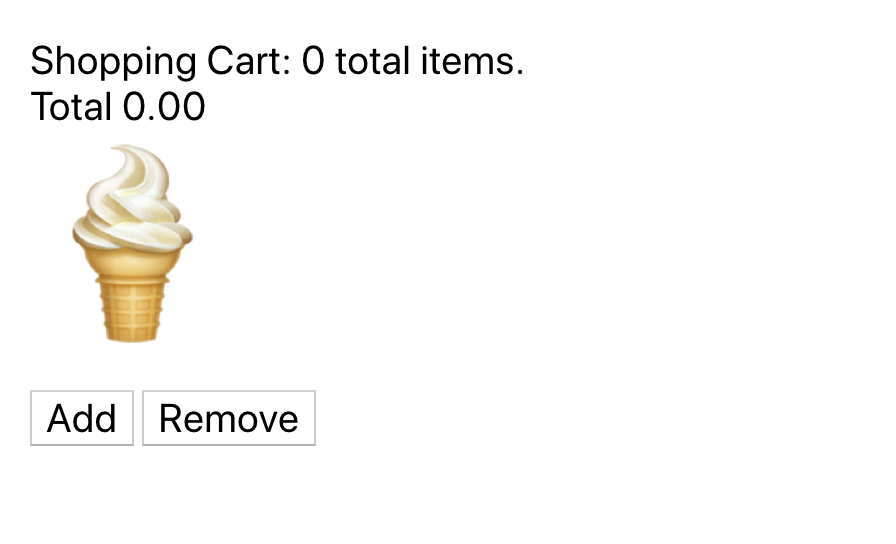
}

}

Puisqu'il total s'agit d'un prix pour les marchandises, vous passez currencyOptions à deux le nombre maximum et minimum de décimales pour votre total. Notez que cela est défini comme une propriété distincte. Souvent, les développeurs React débutants mettront des informations comme celle-ci dans l'objet state, mais il est préférable de n'ajouter que les informations state que vous prévoyez de modifier. De cette façon, les informations contenues state seront plus faciles à conserver au fur et à mesure que votre application évoluera.

Une autre modification importante que vous avez apportée a été de créer la méthode getTotal()en attribuant une [fonction flèchée](https://www.digitalocean.com/community/tutorials/getting-started-with-es6-arrow-functions-in-javascript) à une propriété de classe. Sans utiliser la fonction flèchée, cette méthode créerait une nouvelle liaison [this](https://www.digitalocean.com/community/conceptual_articles/understanding-this-bind-call-and-apply-in-javascript)  , ce qui interférerait avec la liaison this actuelle et introduirait un bogue dans notre code. Vous en verrez plus à ce sujet à l'étape suivante.

Enregistrez le fichier. Lorsque vous le faites, la page s'actualisera et vous verrez la valeur convertie en décimale.



Vous avez maintenant ajouté un état à un composant et l'avez référencé dans votre classe. Vous avez également accédé aux valeurs de la méthode render et d'autres méthodes de classe. Ensuite, vous allez créer des méthodes pour mettre à jour l'état et afficher les valeurs dynamiques.

Étape 3 - Définition de l'état à partir d'une valeur statique

Jusqu'à présent, vous avez créé un état de base pour le composant et vous avez référencé cet état dans vos fonctions et votre code JSX. Dans cette étape, vous allez mettre à jour votre page produit pour modifier les state par les clics sur les boutons. Vous apprendrez à transmettre un nouvel objet contenant des valeurs mises à jour à une méthode spéciale appelée setState, qui définira ensuite le state avec les données mises à jour.

Pour mettre à jour state, les développeurs React utilisent une méthode spéciale appelée setState qui est héritée de la classe Component de base. La méthode setState peut prendre un objet ou une fonction comme premier argument. Si vous avez une valeur statique qui n'a pas besoin de référencer le state, il est préférable de transmettre un objet contenant la nouvelle valeur, car elle est plus facile à lire. Si vous avez besoin de référencer l'état actuel, vous passez une fonction pour éviter toute référence à obsolète state.

Commencez par ajouter un événement aux boutons. Si votre utilisateur clique sur **Ajouter** , le programme ajoutera l'élément au cart et mettra à jour le total. S'ils cliquent sur **Supprimer** , le panier sera réinitialisé à un tableau vide et total à 0. Par exemple, le programme ne permettra pas à un utilisateur d'ajouter un élément plus d'une fois.

Ouvrez Product.js:

* nano src/components/Product/Product.js

À l'intérieur du composant, créez une nouvelle méthode appelée add, puis passez la méthode à l'accessoire onClick pour le bouton **Ajouter** :

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

export default class Product extends Component {

state = {

cart: [],

total: 0

}

add = () => {

this.setState({

cart: ['ice cream'],

total: 5

})

}

currencyOptions = {

minimumFractionDigits: 2,

maximumFractionDigits: 2,

}

getTotal = () => {

return this.state.total.toLocaleString(undefined, this.currencyOptions)

}

render() {

return(

<div className="wrapper">

<div>

Shopping Cart: {this.state.cart.length} total items.

</div>

<div>Total {this.getTotal()}</div>

<div className="product"><span role="img" aria-label="ice cream">🍦</span></div>

<button onClick={this.add}>Add</button>

<button>Remove</button>

</div>

)

}

}

Dans la méthode add, vous appelez la méthode setState et passez un objet contenant la mise à jour de cart avec un seul article ice cream et le prix mis à jour de 5. Notez que vous avez à nouveau utilisé une fonction de flèche pour créer la méthode add. Comme mentionné précédemment, cela garantira que la fonction a le bon contexte this lors de l'exécution de la mise à jour. Si vous ajoutez la fonction en tant que méthode sans utiliser la fonction de flèche, le setState n'existerait pas sans [lier](https://www.digitalocean.com/community/conceptual_articles/understanding-this-bind-call-and-apply-in-javascript) la fonction au contexte actuel.

Par exemple, si vous avez créé la fonction add de cette façon :

export default class Product extends Component {

...

add() {

this.setState({

cart: ['ice cream'],

total: 5

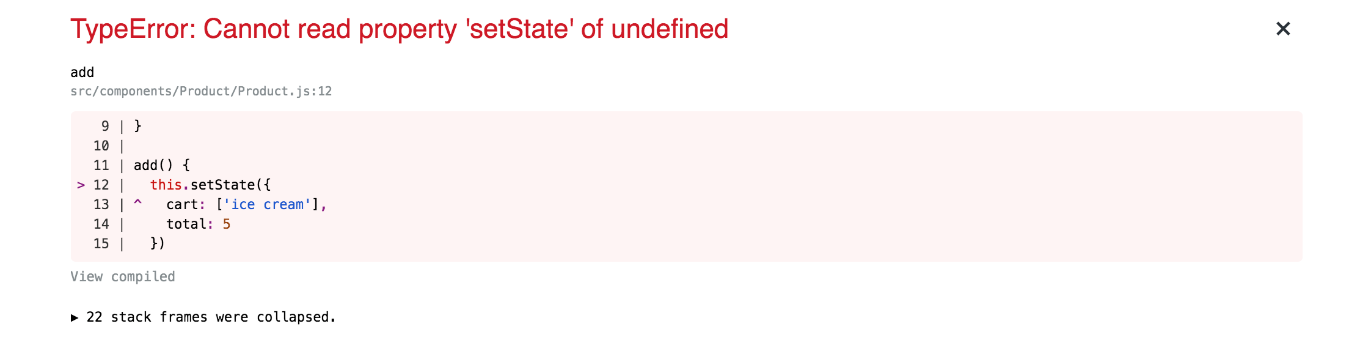
})

}

...

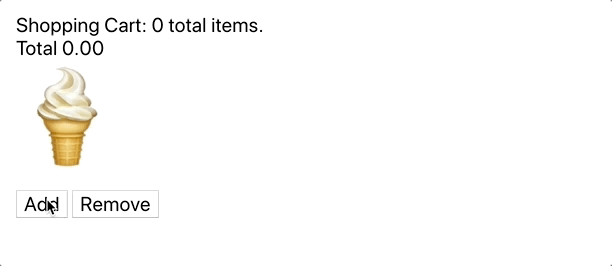
}

L'utilisateur recevrait une erreur lorsqu'il cliquait sur le bouton **Ajouter** .



L'utilisation d'une fonction de flèche garantit que vous aurez le bon contexte pour éviter cette erreur.

Enregistrez le fichier. Lorsque vous le faites, le navigateur se recharge et lorsque vous cliquez sur le bouton **Ajouter,** le panier se met à jour avec le montant actuel.



Avec la méthode add, vous avez transmis les deux propriétés de l'objet state: cart et total. Cependant, vous n'avez pas toujours besoin de passer un objet complet. Il vous suffit de transmettre un objet contenant les propriétés que vous souhaitez mettre à jour, et tout le reste restera le même.

Pour voir comment React peut gérer un objet plus petit, créez une nouvelle fonction appelée remove. Passez un nouvel objet contenant uniquement le cart avec un tableau vide, puis ajoutez la méthode à la propriété onClick du bouton **Supprimer** :

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

export default class Product extends Component {

...

remove = () => {

this.setState({

cart: []

})

}

render() {

return(

<div className="wrapper">

<div>

Shopping Cart: {this.state.cart.length} total items.

</div>

<div>Total {this.getTotal()}</div>

<div className="product"><span role="img" aria-label="ice cream">🍦</span></div>

<button onClick={this.add}>Add</button>

<button onClick={this.remove}>Remove</button>

</div>

)

}

}

Enregistrez le fichier. Lorsque le navigateur s'actualise, cliquez sur les boutons **Ajouter** et **Supprimer** . Vous verrez la mise à jour du panier, mais pas le prix. La valeur total d'état est conservée lors de la mise à jour. Cette valeur n'est conservée qu'à des fins d'exemple ; avec cette application, vous souhaitez mettre à jour les deux propriétés de l'objet state. Mais vous aurez souvent des composants avec des propriétés avec état qui ont des responsabilités différentes, et vous pouvez les faire persister en les laissant en dehors de l'objet mis à jour.

Le changement dans cette étape était statique. Vous saviez exactement quelles seraient les valeurs à l'avance, et elles n'avaient pas besoin d'être recalculées state. Mais si la page produit contenait de nombreux produits et que vous souhaitiez pouvoir les ajouter plusieurs fois, passer un objet statique ne fournirait aucune garantie de référencer les plus à jour state, même si votre objet utilisait une valeur this.state. Dans ce cas, vous pouvez utiliser à la place une fonction.

Dans l'étape suivante, vous effectuerez la mise à jour state à l'aide de fonctions qui référencent l'état actuel.

Étape 4 - Définition de l'état à l'aide de l'état actuel

Vous devrez souvent référencer un état précédent pour mettre à jour un état actuel, comme la mise à jour d'un tableau, l'ajout d'un nombre ou la modification d'un objet. Pour être aussi précis que possible, vous devez référencer l'objet le plus à jour state. Contrairement à la mise à jour  state avec une valeur prédéfinie, dans cette étape, vous passerez une fonction à la méthode setState, qui prendra l'état actuel comme argument. En utilisant cette méthode, vous mettrez à jour l'état d'un composant en utilisant l'état actuel.

Un autre avantage du réglage state avec une fonction est une fiabilité accrue. Pour améliorer les performances, React peut effectuer des appels setState par lots , ce qui signifie que cela peut ne pas être entièrement fiable. Par exemple, si vous mettez à jour rapidement à plusieurs endroits, il est possible qu'une valeur soit obsolète. Cela peut se produire lors des extractions de données, des validations de formulaires ou de toute situation dans laquelle plusieurs actions se produisent en parallèle. Mais utiliser une fonction avec la plus à jour comme argument garantit que ce bogue n'entrera pas dans votre code. this.state.valuestatestate

Pour illustrer cette forme de gestion des états, ajoutez quelques éléments supplémentaires à la page du produit. Tout d'abord, ouvrez le fichier Product.js :

* nano src/components/Product/Product.js

Ensuite, créez un tableau d'objets pour différents produits. Le tableau contiendra l'emoji, le nom et le prix du produit. Ensuite, parcourez la baie pour afficher chaque produit avec un bouton **Ajouter** et **supprimer** :

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

const products = [

{

emoji: '🍦',

name: 'ice cream',

price: 5

},

{

emoji: '🍩',

name: 'donuts',

price: 2.5,

},

{

emoji: '🍉',

name: 'watermelon',

price: 4

}

];

export default class Product extends Component {

...

render() {

return(

<div className="wrapper">

<div>

Shopping Cart: {this.state.cart.length} total items.

</div>

<div>Total {this.getTotal()}</div>

<div>

{products.map(product => (

<div key={product.name}>

<div className="product">

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

</div>

<button onClick={this.add}>Add</button>

<button onClick={this.remove}>Remove</button>

</div>

))}

</div>

</div>

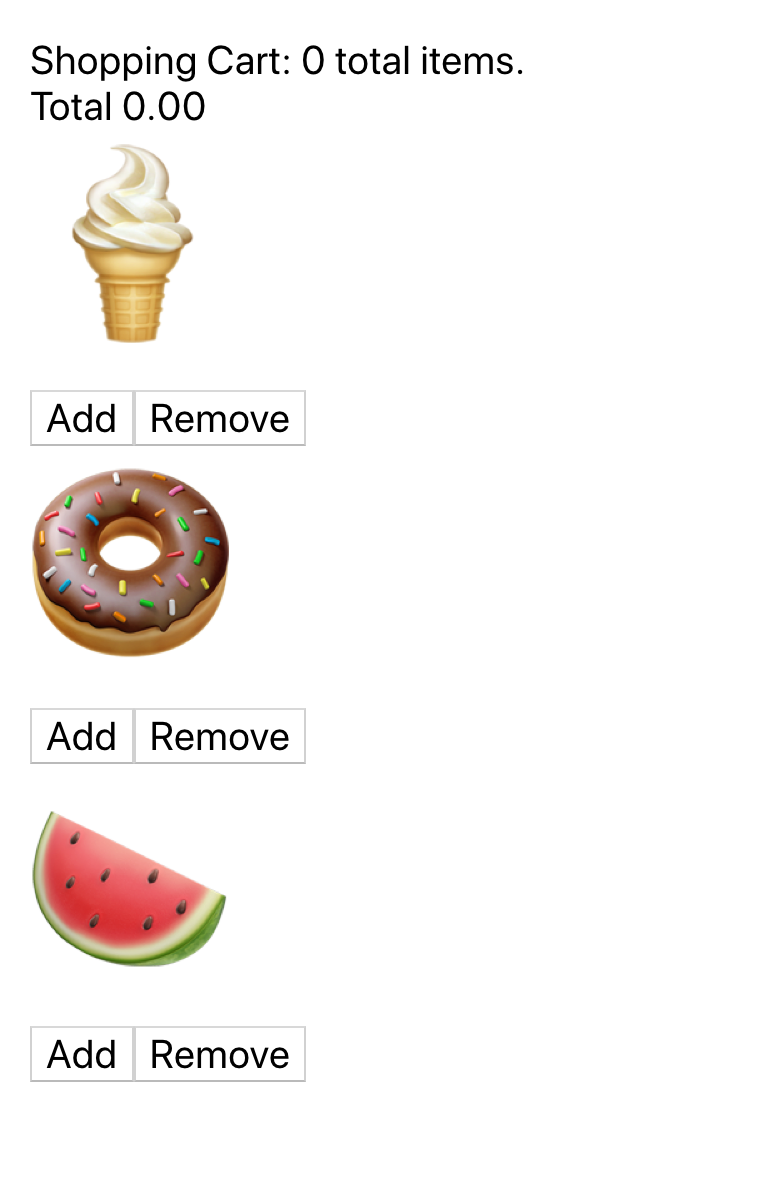
)

}

}

Dans ce code, vous utilisez la méthode array [map()](https://www.digitalocean.com/community/tutorials/how-to-use-array-methods-in-javascript-iteration-methods#map()) pour boucler sur le tableau products et renvoyer le JSX qui affichera chaque élément dans votre navigateur.

Enregistrez le fichier. Lorsque le navigateur se recharge, vous verrez une liste de produits mise à jour :



Vous devez maintenant mettre à jour vos méthodes. Tout d'abord, changez la méthode add() pour prendre le product comme argument. Ensuite, au lieu de passer un objet à setState(), passez une fonction qui prend le state comme argument et renvoie un objet cart qui a été  mis à jour avec le nouveau produit et total mis à jour avec le nouveau prix:

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

...

export default class Product extends Component {

state = {

cart: [],

total: 0

}

add = (product) => {

this.setState(state => ({

cart: [...state.cart, product.name],

total: state.total + product.price

}))

}

currencyOptions = {

minimumFractionDigits: 2,

maximumFractionDigits: 2,

}

getTotal = () => {

return this.state.total.toLocaleString(undefined, this.currencyOptions)

}

remove = () => {

this.setState({

cart: []

})

}

render() {

return(

<div className="wrapper">

<div>

Shopping Cart: {this.state.cart.length} total items.

</div>

<div>Total {this.getTotal()}</div>

<div>

{products.map(product => (

<div key={product.name}>

<div className="product">

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

</div>

<button onClick={() => this.add(product)}>Add</button>

<button onClick={this.remove}>Remove</button>

</div>

))}

</div>

</div>

)

}

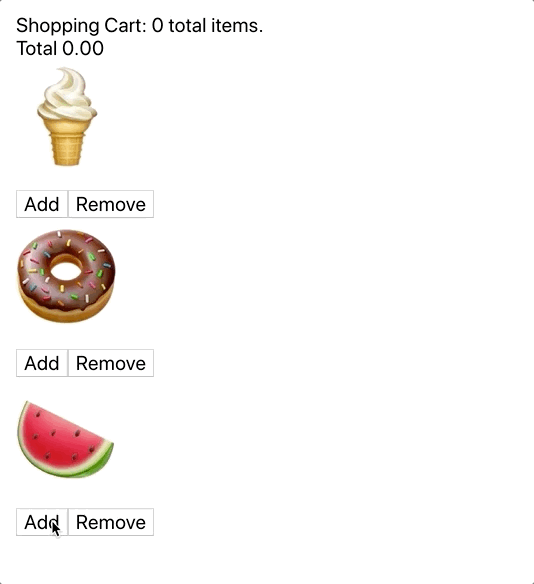
}

Dans la fonction anonyme à laquelle vous passez setState(), assurez-vous de référencer l'argument state— et non l'état du composant— this.state. Sinon, vous courez toujours le risque d'obtenir un objet  state obsolète . Le state dans votre fonction sera par ailleurs identique.

Veillez à ne pas muter directement l'état. Au lieu de cela, lorsque vous ajoutez une nouvelle valeur à la cart, vous pouvez ajouter la nouvelle product à la state en utilisant la [syntaxe de répartition](https://www.digitalocean.com/community/tutorials/understanding-destructuring-rest-parameters-and-spread-syntax-in-javascript#spread) sur la valeur actuelle et en ajoutant la nouvelle valeur à la fin.

Enfin, mettez à jour l'appel à this.add en modifiant l'accessoire onClick()pour prendre une fonction anonyme qui appelle this.add() avec le produit concerné.

Enregistrez le fichier. Lorsque vous le faites, le navigateur se rechargera et vous pourrez ajouter plusieurs produits.



Ensuite, mettez à jour la méthode remove(). Suivez les mêmes étapes : convertir setState pour prendre une fonction, mettre à jour les valeurs sans mutation et mettre à jour le prop onChange() :

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

...

export default class Product extends Component {

...

remove = (product) => {

this.setState(state => {

const cart = [...state.cart];

cart.splice(cart.indexOf(product.name))

return ({

cart,

total: state.total - product.price

})

})

}

render() {

return(

<div className="wrapper">

<div>

Shopping Cart: {this.state.cart.length} total items.

</div>

<div>Total {this.getTotal()}</div>

<div>

{products.map(product => (

<div key={product.name}>

<div className="product">

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

</div>

<button onClick={() => this.add(product)}>Add</button>

<button onClick={() => this.remove(product)}>Remove</button>

</div>

))}

</div>

</div>

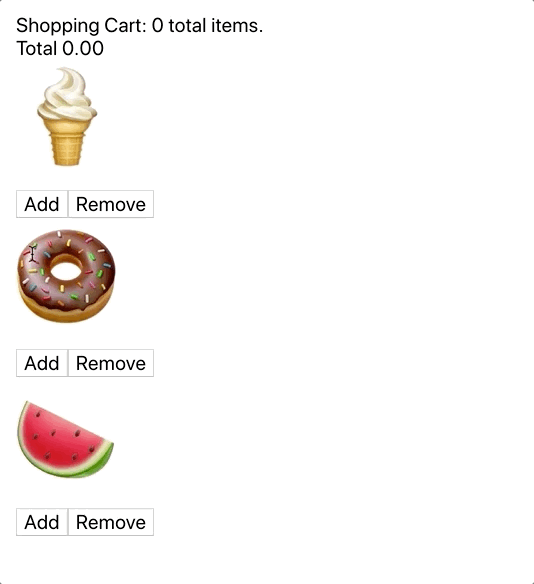
)

}

}

Pour éviter de muter l'objet d'état, vous devez d'abord en faire une copie à l'aide de l'opérateur spread. Ensuite, vous pouvez [séparer](https://www.digitalocean.com/community/tutorials/how-to-use-array-methods-in-javascript-mutator-methods#splice()) l'élément souhaité de la copie et renvoyer la copie dans le nouvel objet. En copiant state dans un premier temps, vous pouvez être sûr de ne pas muter l'objet state.

Enregistrez le fichier. Lorsque vous le faites, le navigateur s'actualisera et vous pourrez ajouter et supprimer des éléments :



Il y a toujours un bogue dans cette application : dans la méthode remove, un utilisateur peut soustraire du total même si l'élément n'est pas dans le cart. Si vous cliquez sur **Supprimer** sur la glace sans l'ajouter à votre panier, votre total sera de **-5,00** .

Vous pouvez corriger le bogue en vérifiant l'existence d'un élément avant de soustraire, mais un moyen plus simple est de garder votre objet d'état petit en ne gardant que les références aux produits et en ne séparant pas les références aux produits et le coût total. Essayez d'éviter les doubles références aux mêmes données. Au lieu de cela, stockez les données brutes dans state- dans ce cas, l'objet  product entier - puis effectuez les calculs en dehors de state.

Refactorisez le composant afin que la méthode add() ajoute l'objet entier, la méthode remove()supprime l'objet entier et la méthode getTotal utilise cart:

tutoriel-classe-d'état / src / components / Product / Product.js

import React, { Component } from 'react';

import './Product.css';

...

export default class Product extends Component {

state = {

cart: [],

}

add = (product) => {

this.setState(state => ({

cart: [...state.cart, product],

}))

}

currencyOptions = {

minimumFractionDigits: 2,

maximumFractionDigits: 2,

}

getTotal = () => {

const total = this.state.cart.reduce((totalCost, item) => totalCost + item.price, 0);

return total.toLocaleString(undefined, this.currencyOptions)

}

remove = (product) => {

this.setState(state => {

const cart = [...state.cart];

const productIndex = cart.findIndex(p => p.name === product.name);

if(productIndex < 0) {

return;

}

cart.splice(productIndex, 1)

return ({

cart

})

})

}

render() {

...

}

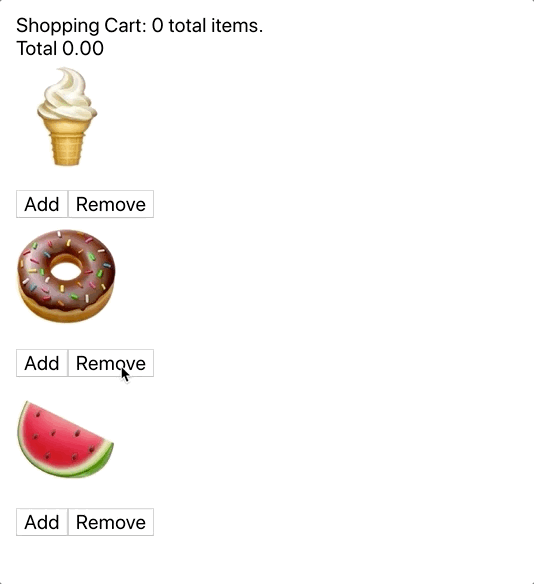
}

La méthode add()est similaire à ce qu'elle était auparavant, sauf que la référence à la propriété total a été supprimée. Dans la méthode remove(), vous trouvez l'index du product avec findByIndex. Si l'index n'existe pas, vous obtiendrez un fichier -1. Dans ce cas, vous utilisez une [instruction conditionnelle pour](https://www.digitalocean.com/community/tutorials/how-to-write-conditional-statements-in-javascript) ne rien renvoyer. En ne renvoyant rien, React saura que le state n'a pas changé et ne déclenchera pas un nouveau rendu. Si vous retournez state ou un objet vide, il déclenchera toujours un nouveau rendu.

Lorsque vous utilisez la méthode splice(), vous passez maintenant 1 comme deuxième argument, ce qui supprimera une valeur et conservera le reste.

Enfin, vous calculez le total à l'aide de la méthode [reduce()](https://www.digitalocean.com/community/tutorials/list-processing-with-map-filter-and-reduce) de tableau.

Enregistrez le fichier. Lorsque vous le faites, le navigateur s'actualisera et vous aurez votre finale cart:



La fonction setState que vous passez peut avoir un argument supplémentaire des accessoires actuels, ce qui peut être utile si vous avez un état qui a besoin de référencer les accessoires actuels. Vous pouvez également passer une fonction de rappel en setState tant que deuxième argument, que vous transmettiez ou non un objet ou une fonction pour le premier argument. Ceci est particulièrement utile lorsque vous définissez state après avoir récupéré des données à partir d'une API et que vous devez effectuer une nouvelle action une fois la  mise à jour state terminée.

Dans cette étape, vous avez appris à mettre à jour un nouvel état en fonction de l'état actuel. Vous avez transmis une fonction à la fonction setState et calculé de nouvelles valeurs sans modifier l'état actuel. Vous avez également appris à quitter une fonction setState s'il n'y a pas de mise à jour de manière à empêcher un nouveau rendu, en ajoutant une légère amélioration des performances.

Conclusion

Dans ce didacticiel, vous avez développé un composant basé sur des classes avec un état dynamique que vous avez mis à jour de manière statique et en utilisant l'état actuel. Vous disposez désormais des outils nécessaires pour réaliser des projets complexes qui répondent aux utilisateurs et aux informations dynamiques.

React dispose d'un moyen de gérer l'état avec Hooks, mais il est utile de comprendre comment utiliser l'état sur les composants si vous devez travailler avec des composants qui doivent être basés sur des classes, tels que ceux qui utilisent la méthode componentDidCatch.

La gestion de l'état est la clé de presque tous les composants et est nécessaire pour créer des applications interactives. Grâce à ces connaissances, vous pouvez recréer de nombreux composants Web courants, tels que des curseurs, des accordéons, des formulaires, etc. Vous utiliserez ensuite les mêmes concepts lorsque vous construisez des applications à l'aide de hooks ou que vous développez des composants qui extraient des données de manière dynamique à partir d'API.

Si vous souhaitez consulter d'autres didacticiels React, consultez notre [page React Topic](https://www.digitalocean.com/community/tags/react) ou revenez à la [page de la série Comment coder dans React.js](https://www.digitalocean.com/community/tutorial_series/how-to-code-in-react-js) .