**Comment personnaliser les composants de React avec des props**

Introduction

Dans ce tutoriel, vous allez créer des composants personnalisés en passant des *props* à votre composant. Les props sont des arguments que vous fournissez à un [élément JSX](https://www.digitalocean.com/community/tutorials/how-to-create-react-elements-with-jsx). Ils ressemblent à des props HTML standard, mais ils ne sont pas prédéfinis et peuvent avoir de nombreux [types de données JavaScript](https://www.digitalocean.com/community/tutorials/understanding-data-types-in-javascript) différents, notamment des nombres, des chaînes de caractères, des [fonctions](https://www.digitalocean.com/community/tutorials/how-to-define-functions-in-javascript), des [tableaux](https://www.digitalocean.com/community/tutorials/understanding-arrays-in-javascript) et même d'autres [composants React](https://www.digitalocean.com/community/tutorials/how-to-create-custom-components-in-react). Vos composants personnalisés peuvent utiliser des props pour afficher des données ou utiliser les données pour rendre les composants interactifs. Les props sont un élément clé de la création de composants adaptables à différentes situations, et en apprendre davantage à leur sujet vous donnera les outils nécessaires pour développer des composants personnalisés pouvant gérer des situations uniques.

Après avoir ajouté des props à votre composant, vous utiliserez les PropTypes pour définir le type de données qu'un composant doit recevoir. Les PropTypes sont un système de type simple permettant de vérifier que les données correspondent aux types attendus pendant l'exécution. Ils servent à la fois de documentation et de vérificateur d'erreurs qui vous aideront à maintenir la prévisibilité de votre demande au fur et à mesure de son évolution.

À la fin du tutoriel, vous saurez utiliser une variété de props pour construire une petite application qui prendra un ensemble de données sur les animaux et en affichera les informations (nom, nom scientifique, taille, régime alimentaire et informations supplémentaires).

**Note** : La première étape consiste à mettre en place un projet vierge sur lequel vous allez construire l'exercice du tutoriel. Si vous avez déjà un projet de travail et que vous souhaitez directement travailler avec des props, commencez par [**l'étape 2**](https://www.digitalocean.com/community/tutorials/how-to-customize-react-components-with-props#step-2-%E2%80%94-building-dynamic-components-with-props).

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](https://www.digitalocean.com/community/tutorials/how-to-install-node-js-and-create-a-local-development-environment-on-macos) ou la section **Installation à l'aide d'un PPA** du tutoriel [Comment installer Node.js sur Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-18-04).
* En suivant ce tutoriel, vous utiliserez [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). Ce tutoriel suppose également une connaissance des composants de React que vous pouvez découvrir dans notre tutoriel [Comment créer des composants personnalisés dans React](https://www.digitalocean.com/community/tutorials/how-to-create-custom-com).
* Vous devrez également connaître les bases de JavaScript que vous trouverez [dansComment coder en JavaScript](https://www.digitalocean.com/community/tutorial_series/how-to-code-in-javascript), ainsi que des connaissances de base en HTML et 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 - Créer un projet vide

Dans cette étape, vous allez créer un nouveau projet en utilisant [Create React App](https://github.com/facebook/create-react-app). Ensuite, vous supprimerez le projet type et les fichiers connexes qui sont installés lorsque vous démarrez le projet. Enfin, vous créerez une structure de fichiers simple pour organiser vos éléments.

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

* npx create-react-app prop-tutorial

Une fois le projet terminé, passez dans le répertoire :

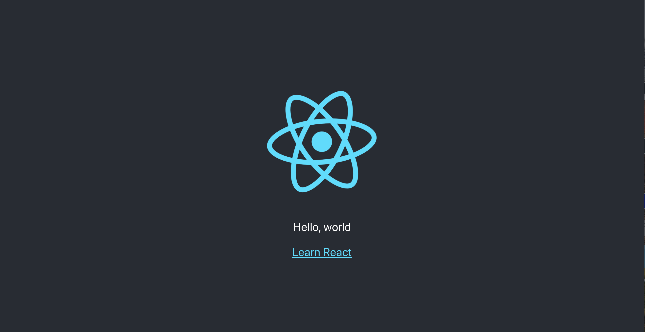
* cd prop-tutorial

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 l'ouvrir en naviguant vers <http://localhost:3000/>. Si vous l'exécutez à partir d'un serveur distant, l'adresse serahttp://your\_domain:3000.

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



Vous allez construire un tout nouvel ensemble de composants personnalisés. Vous commencerez par effacer un code passe-partout afin de pouvoir avoir un projet vide.

Pour commencer, ouvrez src/App.js dans un éditeur de texte. C'est le composant root (racine) qui est injecté dans la page. Tous les composants partiront de là. Vous pouvez trouver plus d'informations sur les App.js à [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#step-5-%E2%80%94-modifying-the-heading-tag-and-styling).

Ouvrez src/App.js :

Vous verrez un fichier comme ceci :

prop-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;

Supprimez le logo d'importation de ligne de '/logo.svg';   Remplacez ensuite tout ce qui figure dans la déclaration de retour pour renvoyer un ensemble de balises vides : <></>. Vous obtiendrez ainsi une page de validation qui ne renvoie rien. Le code final ressemblera à ceci :

prop-tutorial/src/App.js

import React from 'react';

import './App.css';

function App() {

return <></>;

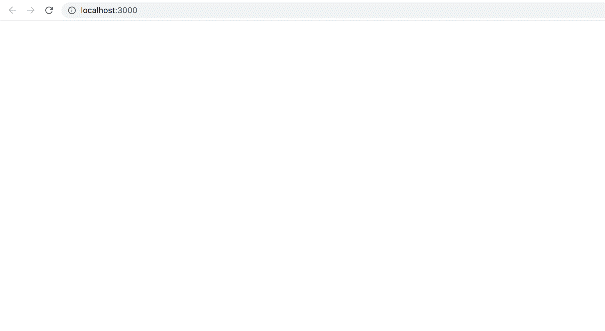
}

export default App;

Sauvegardez et quittez l'éditeur de texte.

Enfin, supprimez le logo. Vous ne l'utiliserez pas dans votre demande et vous devez supprimer les fichiers inutilisés au fur et à mesure de votre travail. Cela vous évitera toute confusion à l'avenir.

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



Maintenant que vous avez éliminé l'exemple de projet Create React App, créez une structure de fichiers simple. Cela vous aidera à maintenir l'isolement et l'indépendance de vos composants.

Créez un répertoire appelé components dans le répertoire src. Celui-ci contiendra tous vos composants personnalisés.

Chaque composant aura son propre répertoire pour stocker le fichier du composant ainsi que les styles, les images s'il y en a, et les tests.

Créez un répertoire pour App :

Déplacez tous les fichiers App dans ce répertoire. Utilisez le joker, \*, pour sélectionner tous les fichiers qui commencent par App. quelle que soit leur extension.Placer les dans le nouveau répertoire.

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

La déclaration d'importation doit pointer vers le fichier App.js dans le répertoire App, donc faites la modification suivante en surbrillance :

prop-tutorial/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 mis en place, vous pouvez créer votre premier composant.

Étape 2 - Construire des composants dynamiques avec des props

Au cours de cette étape, vous créerez un composant qui changera en fonction des informations saisies, appelées *props*. Les props sont les arguments que vous passez à une fonction ou à une classe, mais comme vos composants sont transformés en objets de type HTML avec JSX, vous passerez les props comme s'il s'agissait d'attributs HTML. Contrairement aux éléments HTML, vous pouvez transmettre de nombreux types de données différents, des chaînes de caractères, des tables, des objets et même des fonctions.

Ici, vous allez créer un composant qui affichera des informations sur les animaux. Ce composant prendra le nom et le nom scientifique de l'animal en tant que chaînes, la taille en tant que nombre entier, le régime alimentaire en tant que tableau de chaînes et des informations supplémentaires en tant qu'objet. Vous transmettrez les informations au nouveau composant en tant que props et consommerez ces informations dans votre composant.

À la fin de cette étape, vous aurez un composant personnalisé qui consommera différents props. Vous pourrez également réutiliser le composant pour afficher un ensemble de données à l'aide d'un composant commun.

Ajout de données

Tout d'abord, vous avez besoin de quelques échantillons de données. Créez un fichier dans le répertoire src/App appelé data.

Ouvrez le nouveau fichier dans votre éditeur de texte :

Ensuite, ajoutez un tableau [d'objets](https://www.digitalocean.com/community/tutorials/understanding-objects-in-javascript) que vous utiliserez comme échantillon de données :

prop-tutorial/src/components/App/data.js

export default [

{

name: 'Lion',

scientificName: 'Panthero leo',

size: 140,

diet: ['meat'],

},

{

name: 'Gorilla',

scientificName: 'Gorilla beringei',

size: 205,

diet: ['plants', 'insects'],

additional: {

notes: 'This is the eastern gorilla. There is also a western gorilla that is a different species.'

}

},

{

name: 'Zebra',

scientificName: 'Equus quagga',

size: 322,

diet: ['plants'],

additional: {

notes: 'There are three different species of zebra.',

link: 'https://en.wikipedia.org/wiki/Zebra'

}

}

]

L'ensemble des objets contient une variété de données et vous donnera l'occasion d'essayer divers props. Chaque objet est un animal séparé avec le nom de l'animal, le nom scientifique, la taille, le régime alimentaire et un champ optionnel appelé additional, qui contiendra des liens ou des notes. Dans ce code, vous avez également exporté le tableau par défaut.

Enregistrez et quittez le fichier.

Créer des composants

Ensuite, créez un composant placeholder appelé AnimalCard. Ce composant prendra éventuellement des props et affichera les données.

Tout d'abord, créez un répertoire dans src/components appelé AnimalCard puis touchez un fichier appelé src/components/AnimalCard/AnimalCard.js et un fichier CSS appelé src/components/AnimalCard/AnimalCard.css.

Ouvrez AnimalCard.js dans votre éditeur de texte :

Ajoutez un composant de base qui importe le CSS et renvoie une balise <h2>.

prop-tutorial/src/components/AnimalCard/AnimalCard.js

import React from 'react';

import './AnimalCard.css'

export default function AnimalCard() {

return <h2>Animal</h2>

}

Enregistrez et quittez le fichier. Vous devez maintenant importer les données et le composant dans votre composant App de base.

Ouvrez src/composants/App/App.js :

Importez les données et le composant, puis bouclez les données en retournant le composant pour chaque élément du tableau :

prop-tutorial/src/components/App/App.js

import React from 'react';

import data from './data';

import AnimalCard from '../AnimalCard/AnimalCard';

import './App.css';

function App() {

return (

<div className="wrapper">

<h1>Animals</h1>

{data.map(animal => (

<AnimalCard key={animal.name}/>

))}

</div>

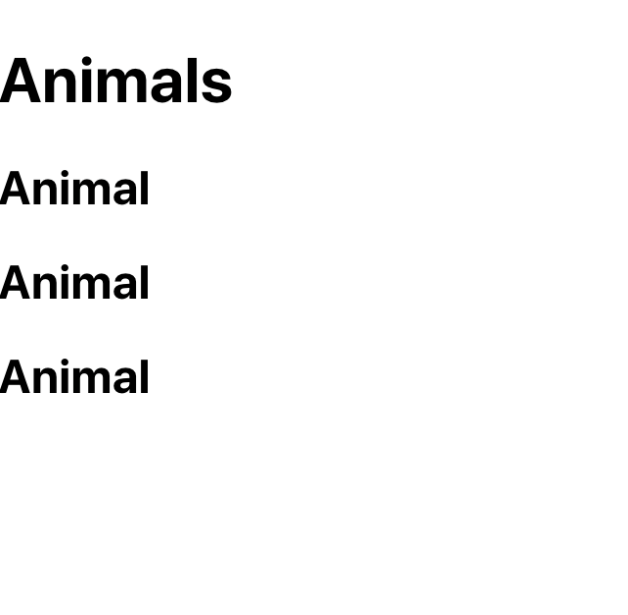
)

}

export default App;

Enregistrez et quittez le fichier. Ici, vous utilisez la [méthode du tableau .map()](https://www.digitalocean.com/community/tutorials/how-to-use-array-methods-in-javascript-iteration-methods#map()) pour itérer sur les données. En plus d'ajouter cette boucle, vous disposez également d'un diviseur d'enveloppe avec une classe que vous utiliserez pour le style et une balise <h1> pour étiqueter votre projet.

Lorsque vous enregistrez, le navigateur se rechargera et vous verrez une étiquette pour chaque carte.



Ensuite, ajoutez un peu de style pour aligner les articles. Ouvrez App.css :

Remplacez le contenu par ce qui suit pour organiser les éléments :

prop-tutorial/src/components/App/App.css

.wrapper {

display: flex;

flex-wrap: wrap;

justify-content: space-between;

padding: 20px;

}

.wrapper h1 {

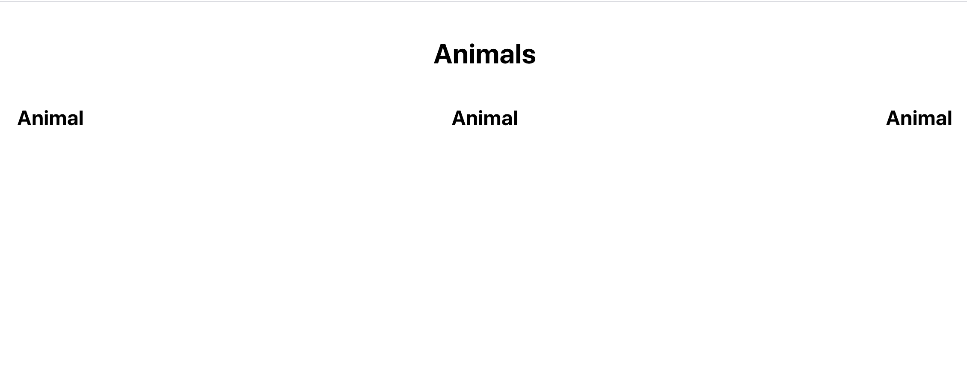
text-align: center;

width: 100%;

}

Ce système utilisera la [flexbox](https://alligator.io/css/flexbox-primer/#display-flex) pour réorganiser les données afin qu'elles s'alignent. Le padding donne de l'espace dans la fenêtre du navigateur. justify-content répartira l'espace supplémentaire entre les éléments, et .wrapper h1 donnera au label Animal toute la largeur.

Enregistrez et quittez le fichier. Lorsque vous le ferez, le navigateur se rafraîchira et vous verrez certaines données espacées.



Ajouter des props

Maintenant que vous avez mis en place vos composants, vous pouvez ajouter votre premier prop. Lorsque vous avez bouclé vos données, vous avez eu accès à chaque objet du tableau de données et aux éléments qu'il contenait. Vous ajouterez chaque élément des données à un prop distinct que vous utiliserez ensuite dans votre composant AnimalCard.

Ouvrez App.js:

Ajoutez un prop name à AnimalCard.

prop-tutorial/src/components/App/App.js

import React from 'react';

...

function App() {

return (

<div className="wrapper">

<h1>Animals</h1>

{data.map(animal => (

<AnimalCard

key={animal.name}

name={animal.name}

/>

))}

</div>

)

}

export default App;

Enregistrez et quittez le fichier. Le prop name ressemble à un attribut HTML standard, mais au lieu d'une chaîne, vous passerez la propriété name de l'objet animal en accolades.

Maintenant que vous avez passé un prop à la nouvelle composante, vous devez l'utiliser. Ouvrez le fichier AnimalCard.js :

Tous les props que vous passez dans le composant sont rassemblés dans un objet qui sera le premier argument de votre fonction. [Déstructurez](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) l'objet pour en retirer des props individuels :

prop-tutorial/src/components/AnimalCard/AnimalCard.js

import React from 'react';

import './AnimalCard.css'

export default function AnimalCard(props) {

const { name } = props;

return (

<h2>{name}</h2>

);

}

Notez que vous n'avez pas besoin de déstructurer un prop pour l'utiliser, mais que c'est une méthode utile pour traiter les données types de ce tutoriel.

Après avoir déstructuré l'objet, vous pouvez utiliser les données individuelles. Dans ce cas, vous utiliserez le titre dans une balise <h2>, entourant la valeur d'accolades pour que React sache l'évaluer en JavaScript.

Vous pouvez également utiliser une propriété sur l'objet prop en utilisant la notation par points. Par exemple, vous pourriez créer un élément <h2> comme celui-ci : <h2>{props.title}</h2>.   L'avantage de la déstructuration est que vous pouvez collecter les props non utilisés et utiliser l’[opérateur de repos d'objet](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax#Spread_in_object_literals).

Enregistrez et quittez le fichier. Lorsque vous le ferez, le navigateur se rechargera et vous verrez le nom spécifique de chaque animal au lieu d'un espace réservé.



La propriété name est une chaîne de caractères, mais les props peuvent être n'importe quel type de données que vous pourriez passer à une fonction JavaScript. Pour voir cela à l'œuvre, ajoutez le reste des données.

Ouvrez le fichier App.js :

Ajoutez un prop pour chacun des éléments suivants : scientificName, size, diet et additional. Ceux-ci comprennent des chaînes de caractères, des entiers, des tableaux et des objets.

prop-tutorial/src/components/App/App.js

import React from 'react';

...

function App() {

return (

<div className="wrapper">

<h1>Animals</h1>

{albums.map(album => (

<AnimalCard

additional={animal.additional}

diet={animal.diet}

key={animal.name}

name={animal.name}

scientificName={animal.scientificName}

size={animal.size}

/>

))}

</div>

)

}

export default App;

Puisque vous créez un objet, vous pouvez les ajouter dans l'ordre que vous voulez. L'ordre alphabétique permet de parcourir plus facilement une liste de props, surtout dans une liste plus longue. Vous pouvez également les ajouter sur la même ligne, mais le fait de les séparer à une par ligne permet de garder les choses lisibles.

Enregistrez et fermez le fichier. Ouvrez AnimalCard.js.

Cette fois, déstructurez les props dans la liste des paramètres de fonction et utilisez les données dans le composant :

prop-tutorial/src/components/AnimalCard/AnimalCard.js

import React from 'react';

import './AnimalCard.css'

export default function AnimalCard({

additional,

diet,

name,

scientificName,

size

}) {

return (

<div>

<h2>{name}</h2>

<h3>{scientificName}</h3>

<h4>{size}kg</h4>

<div>{diet.join(', ')}.</div>

</div>

);

}

Après avoir extrait les données, vous pouvez ajouter scientificName et size dans les balises d'en-tête, mais vous devrez convertir le tableau en une chaîne pour que React puisse l'afficher sur la page. Vous pouvez le faire avec [join(', ')](https://www.digitalocean.com/community/tutorials/how-to-use-array-methods-in-javascript-accessor-methods#join()), ce qui créera une liste séparée par des virgules.

Enregistrez et fermez le fichier. Lorsque vous le ferez, le navigateur se rafraîchira et vous verrez les données structurées.



Vous pourriez créer une liste similaire avec l'objet additional, mais en ajoutant une fonction pour alerter l'utilisateur avec les données. Cela vous donnera la possibilité de passer des fonctions comme props et d'utiliser ensuite les données à l'intérieur d'un composant lorsque vous appelez une fonction.

Ouvrez App.js:

Créez une fonction appelée showAdditionalData qui convertira l'objet en une chaîne de caractères et l'affichera comme une alerte.

prop-tutorial/src/components/App/App.js

import React from 'react';

...

function showAdditional(additional) {

const alertInformation = Object.entries(additional)

.map(information => `${information[0]}: ${information[1]}`)

.join('\n');

alert(alertInformation)

};

function App() {

return (

<div className="wrapper">

<h1>Animals</h1>

{data.map(animal => (

<AnimalCard

additional={animal.additional}

diet={animal.diet}

key={animal.name}

name={animal.name}

scientificName={animal.scientificName}

showAdditional={showAdditional}

size={animal.size}

/>

))}

</div>

)

}

export default App;

La fonction showAdditional convertit l'objet en un tableau de paires où le premier élément est la clé et le second la valeur. Elle établit ensuite une correspondance avec les données en convertissant la paire de clés en une chaîne. Puis elle les joint avec un saut de ligne - \n avant de passer la chaîne complète à la fonction d'alerte.

Comme JavaScript peut accepter des fonctions comme arguments, React peut également accepter des fonctions comme props. Vous pouvez donc passer showAdditional à AnimalCard comme un prop appelé showAdditional.

Enregistrez et fermez le fichier. Ouvrez AnimalCard :

Tirez la fonction showAdditional de l'objet props, puis créez un <bouton> avec un [événement onClick](https://www.digitalocean.com/community/tutorials/how-to-create-react-elements-with-jsx#step-3-%E2%80%94-adding-events-to-elements) qui appelle la fonction avec l'objet additional :

prop-tutorial/src/components/AnimalCard/AnimalCard.js

import React from 'react';

import './AnimalCard.css'

export default function AnimalCard({

additional,

diet,

name,

scientificName,

showAdditional,

size

}) {

return (

<div>

<h2>{name}</h2>

<h3>{scientificName}</h3>

<h4>{size}kg</h4>

<div>{diet.join(', ')}.</div>

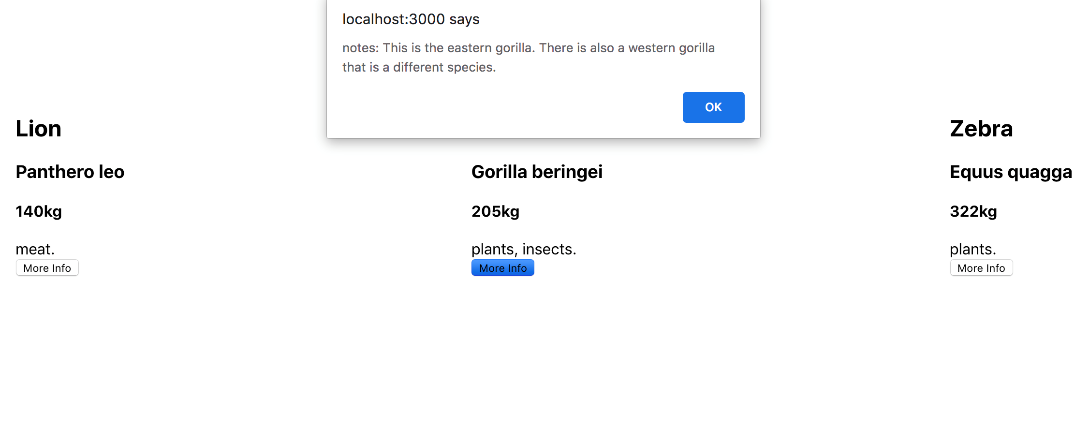
<button onClick={() => showAdditional(additional)}>More Info</button>

</div>

);

}

Sauvegardez le fichier. Lorsque vous le ferez, le navigateur se rafraîchira et vous verrez un bouton après chaque carte. Lorsque vous cliquez sur le bouton, vous obtenez une alerte avec les données supplémentaires.



Si vous essayez de cliquer sur **Plus d'infos** pour le Lion, vous obtiendrez une erreur. C'est parce qu'il n'y a pas de données supplémentaires pour le lion. Vous verrez comment régler ce problème à l'étape 3.

Enfin, ajoutez un peu de style à la carte musicale. Ajoutez une classeNom de l’animal-wrapper à la div dans AnimalCard :

prop-tutorial/src/components/AnimalCard/AnimalCard.js

import React from 'react';

import './AnimalCard.css'

export default function AnimalCard({

...

return (

<div className="animal-wrapper">

...

</div>

)

}

Enregistrez et fermez le fichier. Ouvrez AnimalCard.css :

Ajoutez du CSS pour donner aux cartes et au bouton une petite bordure et du padding :

prop-tutorial/src/components/AnimalCard/AnimalCard.css

.animal-wrapper {

border: solid black 1px;

margin: 10px;

padding: 10px;

width: 200px;

}

.animal-wrapper button {

font-size: 1em;

border: solid black 1px;

padding: 10;

background: none;

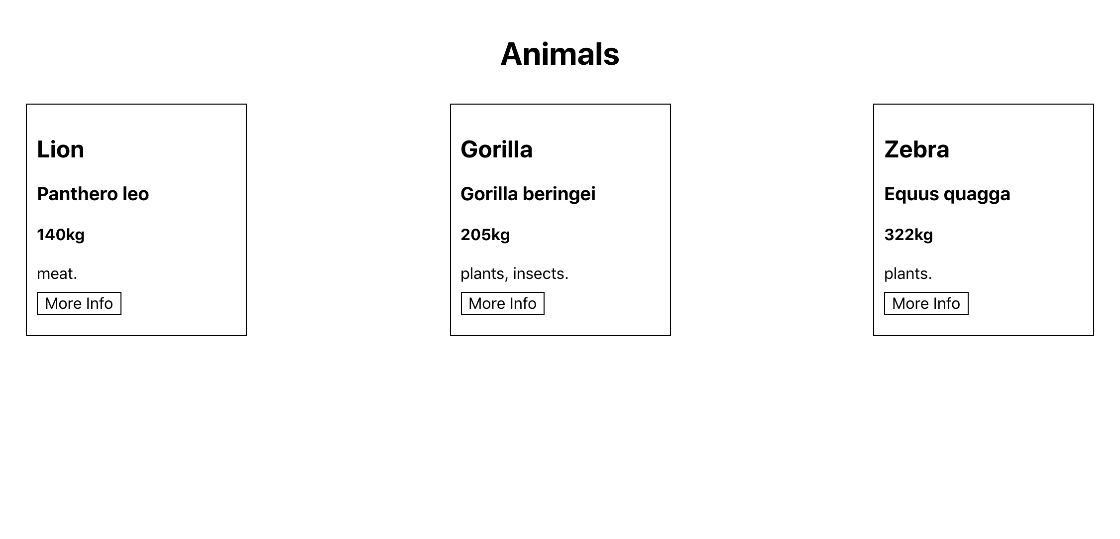
cursor: pointer;

margin: 10px 0;

}

Ce CSS ajoutera une légère bordure à la carte et remplacera le style du bouton par défaut par une bordure et du paddding. cursor: pointer changera le curseur lorsque vous passerez sur le bouton.

Enregistrez et fermez le fichier. Lorsque vous le faites, le navigateur se rafraîchit et vous voyez les données dans les cartes individuelles.



À ce stade, vous avez créé deux composants personnalisés. Vous avez transmis des données au deuxième composant à partir du premier composant en utilisant des props. Les props incluent une variété de données, telles que des chaînes de caractères, des entiers, des tableaux, des objets et des fonctions. Dans votre deuxième composant, vous avez utilisé les props pour créer un composant dynamique en utilisant JSX.

Dans l'étape suivante, vous utiliserez un système de types appelé prop-types pour spécifier la structure que votre composant s'attend à voir, ce qui créera de la prévisibilité dans votre application et empêchera les bugs.

Étape 3 - Créer des props prévisibles avec les PropTypes et les defaultProps

Dans cette étape, vous ajouterez un système de type léger à vos composants avec PropTypes. Les PropTypes agissent comme les autres systèmes de type en définissant explicitement le type de données que vous vous attendez à recevoir pour un certain prop. Ils vous donnent également la possibilité de définir des données par défaut dans les cas où le prop n'est pas toujours nécessaire. Contrairement à la plupart des systèmes de types, PropTypes est une vérification d'exécution, donc si les accessoires ne correspondent pas au type, le code se compilera quand même, mais affichera aussi une erreur de console.

À la fin de cette étape, vous ajouterez de la prévisibilité à votre composant personnalisé en définissant le type de chaque prop. Ainsi, la prochaine personne qui travaillera sur le composant aura une idée claire de la structure des données dont le composant aura besoin.

Le paquet prop-types est inclus dans l'installation de Create React App ; donc pour l'utiliser, il vous suffit de l'importer dans votre composant.

Ouvrez AnimalCard.js :

Ensuite, importez les PropTypes à partir des prop-types :

prop-tutorial/src/components/AnimalCard/AnimalCard.js

import React from 'react';

import PropTypes from 'prop-types';

import './AnimalCard.css'

export default function AnimalCard({

...

}

Ajoutez PropTypes directement à la fonction du composant.   En JavaScript, [les fonctions sont des objets](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions), ce qui signifie que vous pouvez ajouter des propriétés en utilisant la syntaxe des points. Ajoutez les PropTypes suivants à AnimalCard.js :

prop-tutorial/src/components/AnimalCard/AnimalCard.js

import React from 'react';

import PropTypes from 'prop-types';

import './AnimalCard.css'

export default function AnimalCard({

...

}

AnimalCard.propTypes = {

additional: PropTypes.shape({

link: PropTypes.string,

notes: PropTypes.string

}),

diet: PropTypes.arrayOf(PropTypes.string).isRequired,

name: PropTypes.string.isRequired,

scientificName: PropTypes.string.isRequired,

showAdditional: PropTypes.func.isRequired,

size: PropTypes.number.isRequired,

}

Enregistrez et fermez le fichier.

Comme vous pouvez le voir, il existe de nombreux types de PropTypes différents. Ce n'est qu'un petit échantillon ; consultez la [documentation officielle de React](https://reactjs.org/docs/typechecking-with-proptypes.html#proptypes) pour voir les autres que vous pouvez utiliser.

Commençons par le prop name. Ici, vous spécifiez que name doit être une chaîne de caractères. La propriété scientificName est la même. size est un nombre, qui peut inclure à la fois des flotteurs comme 1,5 et des entiers comme 6. showAdditional est une fonction (func).

diet, en revanche, est un peu différent. Dans ce cas, vous spécifiez que diet sera un tableau, mais vous devez également préciser ce que ce tableau contiendra. Dans ce cas, le tableau ne contiendra que des chaînes de caractères. Si vous voulez mélanger les types, vous pouvez utiliser un autre prop appelé oneOfType qui prend un ensemble de PropTypes valides. Vous pouvez utiliser oneOfType n'importe où, donc si vous voulez que size soit soit un nombre ou une chaîne, vous pouvez la changer en ceci :

size: PropTypes.oneOfType([PropTypes.number, PropTypes.string])

Le prop additional est également un peu plus complexe. Dans ce cas, vous spécifiez un objet, mais pour être un peu plus clair, vous indiquez ce que vous voulez que l'objet contienne. Pour ce faire, vous utilisezPropTypes.shape, qui prend un objet avec des champs supplémentaires qui auront besoin de leurs propres PropTypes. Dans ce cas, link et notes sont tous deux PropTypes.string.

Actuellement, toutes les données sont bien formées et correspondent aux props. Pour voir ce qui se passe si les PropTypes ne correspondent pas, ouvrez vos données :

Changez la size en une chaîne sur le premier article :

prop-tutorial/src/components/App/data.js

export default [

{

name: 'Lion',

scientificName: 'Panthero leo',

size: '140',

diet: ['meat'],

},

...

]

Sauvegardez le fichier. Lorsque vous le faites, le navigateur se rafraîchit et vous voyez une erreur dans la console.

Error

index.js:1 Warning: Failed prop type: Invalid prop `size` of type `string` supplied to `AnimalCard`, expected `number`.

in AnimalCard (at App.js:18)

in App (at src/index.js:9)

in StrictMode (at src/index.js:8)



Contrairement à d'autres systèmes de types tels que [TypeScript](https://www.typescriptlang.org/), PropTypes ne vous donnera pas d'avertissement au moment de la construction, et tant qu'il n'y a pas d'erreurs de code, il continuera à se compiler. Cela signifie que vous pourriez accidentellement publier du code avec des erreurs de props.

Remplacez les données par le type correct :

prop-tutorial/src/components/App/data.js

export default [

{

name: 'Lion',

scientificName: 'Panthero leo',

size: 140,

diet: ['meat'],

},

...

]

Enregistrez et fermez le fichier.

Ouvrez AnimalCard.js :

Tous les props, à l'exception d’additional a la propriété isRequired.  Cela signifie qu'ils sont nécessaires. Si vous n'incluez pas un prop requis, le code se compilera quand même, mais vous verrez une erreur d'exécution dans la console.

Si un prop n'est pas nécessaire, vous pouvez ajouter une valeur par défaut. Il est de bonne pratique de toujours ajouter une valeur par défaut pour éviter les erreurs d'exécution si un prop n'est pas nécessaire. Par exemple, dans le composant AnimalCard, vous appelez une fonction avec la donnée additional. Si elle n'est pas là, la fonction va essayer de modifier un objet qui n'existe pas et l'application va planter.

Pour éviter ce problème, ajoutez un defaultProp pour additional :

prop-tutorial/src/components/AnimalCard/AnimalCard.js

import React from 'react';

import PropTypes from 'prop-types';

import './AnimalCard.css'

export default function AnimalCard({

...

}

AnimalCard.propTypes = {

additional: PropTypes.shape({

link: PropTypes.string,

notes: PropTypes.string

}),

...

}

AnimalCard.defaultProps = {

additional: {

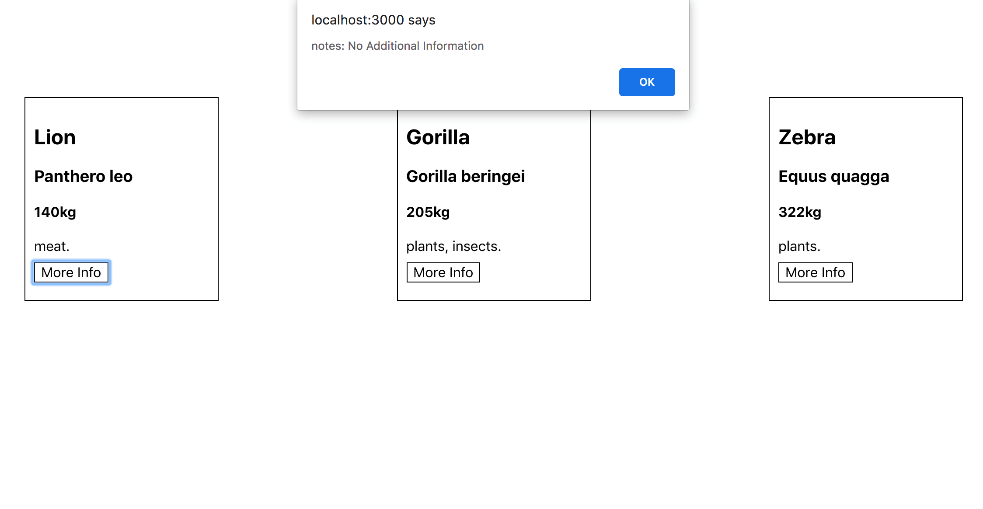
notes: 'No Additional Information'

}

}

Vous ajoutez le defaultProps à la fonction en utilisant la syntaxe des points comme vous l'avez fait avec propTypes, puis vous ajoutez une valeur par défaut que le composant doit utiliser si le prop est undefined. Dans ce cas, vous faites correspondre la forme d'additional, y compris un message indiquant qu'il n'y a pas d'informations supplémentaires.

Enregistrez et fermez le fichier. Lorsque vous le faites, le navigateur se rafraîchit. Après l'avoir actualisé, cliquez sur le bouton **More info** pour **Lion**. Il n'y a pas de champ additional dans les données, donc le prop est undefined.   Mais AnimalCard se substituera au prop par défaut.



Vos props sont désormais bien documentés et sont soit obligatoires, soit dotés d'une valeur par défaut pour garantir un code prévisible. Cela aidera les futurs développeurs (y compris vous-même) à comprendre les props dont un composant a besoin. Cela facilitera l'échange et la réutilisation de vos composants en vous donnant des informations complètes sur la manière dont le composant utilisera les données qu'il reçoit.

Conclusion

Dans ce tutoriel, vous avez créé plusieurs composants qui utilisent des props pour afficher les informations d'un parent. Les props vous donnent la souplesse nécessaire pour commencer à briser les gros composants en morceaux plus petits et plus ciblés. Maintenant que vous n'avez plus vos données étroitement couplées à vos informations d'affichage, vous avez la possibilité de faire des choix sur la manière de segmenter votre application.

Les props sont un outil crucial dans la construction d'applications complexes, donnant la possibilité de créer des composants qui peuvent s'adapter aux données qu'ils reçoivent. Avec les PropTypes, vous créez des composants prévisibles et lisibles qui donneront à une équipe la possibilité de réutiliser le travail de chacun pour créer une base de code souple et stable. Si vous souhaitez consulter d'autres tutoriels de la série React, consultez notre [page thématique React](https://www.digitalocean.com/community/tags/react) ou retournez à [la page de la série Comment coder en React.js](https://www.digitalocean.com/community/tutorial_series/how-to-code-in-react-js).