



CONTENTS

Introduction

Conditions préalables

Étape 1 — Création de l'app React et modification du composant de l'app

Étape 2 — Création d'un serveur Express et rendu du composant de l'app

Étape 3 — Configuration des scripts Webpack, Babel et npm

Conclusion

RELATED

Comment créer des éléments React avec JSX

Tutorial

Comment configurer un projet React avec Create React App

Tutorial

// Tutorial //

Comment activer le rendu côté serveur d'une app React

Published on November 23, 2020

React

By [Alligator.io](#)

Developer and author at DigitalOcean.

Français



Introduction

Le *rendu côté serveur* (SSR) est une technique couramment utilisée pour obtenir un rendu d'une *application simple page* (SPA) côté client sur le serveur et que cette page soit renvoyée au client avec un rendu complet. Cette technique permet de présenter les composants dynamiques sous forme de balise HTML.

Cette approche peut vous être utile dans le cadre de l'optimisation des moteurs de recherche (SEO) lorsque l'indexation ne traite pas correctement JavaScript. Elle peut également s'avérer être très pratique si la lenteur du réseau vient altérer le téléchargement d'un grand paquet JavaScript.

Au cours de ce tutoriel, vous allez initialiser une app React en utilisant [Create React App](#). Ensuite, vous modifierez le projet pour activer le rendu côté serveur.

À la fin de ce tutoriel, vous disposerez d'un projet actif avec une application React côté client et une application Express côté serveur.

Remarque : vous pouvez également envisager d'utiliser l'approche moderne proposée par [Next.js](#) qui vous permet de créer des applications statiques et serveurs développées avec React.

Conditions préalables

Pour suivre ce tutoriel, vous aurez besoin de :

- Node.js installé localement, ce que vous pouvez faire en suivant [Comment installer Node.js et créer un environnement de développement local](#).

Les vérifications effectuées sur ce tutoriel ont été réalisées avec Node v14.4.0 et `npm` v6.14.5.

Étape 1 – Création de l'app React et modification du composant de l'app

Dans un premier lieu, nous allons utiliser `npx` pour lancer une nouvelle app React en utilisant la dernière version de Create React App.

Appelons notre app `my-ssr-app` :

```
$ npx create-react-app@3.4.1 my-ssr-app
```

Copy

Ensuite, nous la copions (`cd`) dans le nouveau répertoire :

```
cd my-ssr-app
```

Pour finir, nous pouvons lancer notre nouvelle app côté client afin d'en vérifier l'installation :

```
$ npm start
```

Copy

Vous devriez voir apparaître un exemple d'affichage de l'app React dans la fenêtre de votre navigateur.

Maintenant, créons un composant `<Home>` :

```
$ nano src/Home.js
```

Copy

Ensuite, ajoutons le code suivant au fichier `Home.js` :

```
src/Home.js  
import React from 'react';  
  
export default props => {  
  return <h1>Hello {props.name}!</h1>;  
};
```

Copy

Cela créera un titre `<h1>` accompagné d'un message « Bonjour » adressé à un nom.

Ensuite, nous allons permettre le rendu de `<Home>` dans le composant `<App>`. Ouvrez le fichier `App.js` :

```
$ nano src/App.js
```

Copy

Maintenant, remplacez les lignes de code existantes par les nouvelles lignes de code suivantes :

```
src/App.js  
import React from 'react';  
import Home from './Home';  
  
export default () => {  
  return <Home name="Sammy" />;  
};
```

Copy

Cela transmet un `name` au composant `<Home>` de manière à ce que le message suivant s'affiche : « Bonjour Sammy ! ».

Dans le fichier `index.js` de notre app, nous allons utiliser la [méthode d'hydratation](#) de ReactDOM au lieu du `rendu`. Nous indiquons ainsi au moteur de rendu DOM que nous allons réhydrater l'app une fois qu'un rendu sera obtenu côté serveur.

Ouvrez le fichier `index.js` :

```
$ nano index.js
```

Copy

Ensuite, remplacez le contenu du fichier `index.js` par le code suivant :

```
index.js  
import React from 'react';  
import ReactDOM from 'react-dom';  
import App from './App';  
  
ReactDOM.hydrate(<App />, document.getElementById('root'));
```

Copy

Vous venez de terminer la configuration côté client. Nous pouvons maintenant passer à celle du côté serveur.

Étape 2 – Création d'un serveur Express et rendu du composant de l'app

Maintenant que notre app est installée, créons un serveur qui enverra une version rendue. Nous utiliserons [Express for our server](#). Ajoutez-le au projet en saisissant la commande suivante dans la fenêtre de votre terminal :

```
$ npm install express@4.17.1
```

Copy

Ou, en utilisant yarn :

```
$ yarn add express@4.17.1
```

Copy

Maintenant, créez un répertoire `server` à côté du répertoire `src` de l'app :

```
$ mkdir server
```

Copy

Créez un nouveau fichier `index.js` qui contiendra le code du serveur Express :

```
$ nano server/index.js
```

Copy

Ajoutez les importations nécessaires et définissez les quelques constantes suivantes :

```
server/index.js

import path from 'path';
import fs from 'fs';

import React from 'react';
import express from 'express';
import ReactDOMServer from 'react-dom/server';

import App from '../src/App';

const PORT = process.env.PORT || 3006;
const app = express();
```

Copy

Ensuite, ajoutez le code du serveur avec un traitement des erreurs :

```
server/index.js

// ...

app.get('/', (req, res) => {
  const app = ReactDOMServer.renderToString(<App />);

  const indexPath = path.resolve('./build/index.html');
  fs.readFile(indexPath, 'utf8', (err, data) => {
    if (err) {
      console.error('Something went wrong:', err);
      return res.status(500).send('Oops, better luck next time!');
    }

    return res.send(
      data.replace('<div id="root"></div>', `<div id="root">${app}</div>`)
    );
  });
});

app.use(express.static('./build'));

app.listen(PORT, () => {
  console.log(`Server is listening on port ${PORT}`);
});
```

Copy

Comme vous le voyez, nous pouvons importer notre composant `<App>` de l'app client directement depuis le serveur.

Il se passe trois choses importantes à ce stade :

- Nous indiquons à Express de desservir le contenu du répertoire `build` sous forme de fichiers statiques.
- Nous utilisons une méthode de `ReactDOMServer`, `renderToString` pour obtenir un rendu de notre app sur une chaîne HTML statique.
- Ensuite, nous lisons le fichier statique `index.html` de l'app client créée, nous injectons le contenu statique de notre app dans le `<div>` avec un `id` « `root` » et envoyons cela sous forme de réponse à la requête.

Étape 3 – Configuration des scripts Webpack, Babel et npm

Pour que notre code serveur fonctionne, nous devons le grouper et le transposer, en utilisant [Webpack](#) et [Babel](#). Pour ce faire, ajoutez les dépendances de dév. au projet en saisissant la commande suivante dans la fenêtre de votre terminal :

```
$ npm install webpack@4.42.0 webpack-cli@3.3.12 webpack-node-externals@1.7.2 @babel/
```

Copy

Ou, en utilisant yarn :

```
$ yarn add webpack@4.42.0 webpack-cli@3.3.12 webpack-node-externals@1.7.2 @babel/core
```

Remarque : une version antérieure de ce tutoriel était dédiée à l'installation de `babel-core`, `babel-preset-env` et `babel-preset-react-app`. Depuis, ces paquets ont été archivés et remplacés par des versions mono repo.

Ensuite, créez un fichier de configuration Babel :

```
$ nano .babelrc.json
```

Copy

Puis, ajoutez les prérèglages de `env` et `react-app` :

```
.babelrc.json
```

Copy

```
{
  "presets": [
    "@babel/preset-env",
    "@babel/preset-react"
  ]
}
```

Remarque : dans une version antérieure de ce tutoriel, on utilisait un fichier `.babelrc` (aucune extension de fichier `.json`). Il s'agissait d'un fichier de configuration pour Babel 6, mais ce n'est plus le cas pour Babel 7.

Maintenant, nous allons créer un Webpack de configuration pour le serveur qui utilise Babel Loader et pourvoir ainsi transposer le code. Commencez par créer le fichier :

```
$ nano webpack.server.js
```

Copy

Maintenant, ajoutez les paramètres suivants au fichier `webpack.server.js` :

```
webpack.server.js
```

Copy

```
const path = require('path');
const nodeExternals = require('webpack-node-externals');

module.exports = {
  entry: './server/index.js',
  target: 'node',
  externals: [nodeExternals()],
  output: {
    path: path.resolve('server-build'),
    filename: 'index.js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        use: 'babel-loader'
      }
    ]
  }
};
```

Une fois cette configuration faite, notre bundle serveur transposé sera généré dans le dossier `server-build` situé dans le fichier `index.js`.

Notez que l'on utilise `target: 'node'` et `externals: [nodeExternals()]` de `webpack-node-externals`. Cela permettra d'omettre les fichiers `node_modules` qui se trouvent dans le paquet et au serveur d'accéder directement à ces fichiers.

Ceci achève l'installation de la dépendance et la configuration du Webpack et de Babel.

Maintenant, retournons à `paquet.json` pour ajouter les scripts `npm` d'aide :

```
$ nano package.json
```

Copy

Nous allons ajouter les scripts `dev:build-server`, `dev:start` et `dev` au fichier `package.json` pour développer et desservir facilement notre application SSR :

```
package.json
```

Copy

```
"scripts": {
  "dev:build-server": "NODE_ENV=development webpack --config webpack.server.js --mode=development"
```

```
"dev": "nodemon ./server-build/index.js",
"dev": "npm-run-all --parallel build dev:*",
...
},
```

Lorsque nous apportons des modifications au serveur, nous utilisons `nodemon` pour le redémarrer. De son côté, `npm-run-all` permet d'exécuter plusieurs commandes en parallèle.

Vous pouvez maintenant procéder à l'installation de ces paquets en saisissant les commandes suivantes dans la fenêtre de votre terminal :

```
$ npm install nodemon@2.0.4 npm-run-all@4.1.5 --save-dev
```

Copy

Ou, en utilisant yarn :

```
$ yarn add nodemon@2.0.4 npm-run-all@4.1.5 --dev
```

Copy

Une fois que tout cela est en place, vous pouvez exécuter les éléments suivants pour développer l'app côté client, regrouper et transposer le code du serveur et lancer le serveur en utilisant `:3006` :

```
$ npm run dev
```

Copy

Ou, en utilisant yarn :

```
$ yarn run dev
```

Copy

En configurant ainsi le Webpack de notre serveur, les changements seront surveillés et le serveur redémarré à chaque fois que des modifications seront apportées. Cependant, du côté de l'app client, il faudra en créer une à chaque changement. Il s'agit ici d'un [problème qui n'a pas encore été résolu](#).

Maintenant, ouvrez `http://localhost:3006/` dans votre navigateur web. Vous verrez que votre app est rendue côté serveur.

Auparavant, le code source révélait ce qui suit :

```
Output
<div id="root"></div>
```

Copy

Mais maintenant, avec les changements que vous avez apportés, le code source révèle ceci :

```
Output
<div id="root"><h1 data-reactroot="">Hello ←!— →Sammy←!— →!</h1></div>
```

Copy

Le rendu côté serveur a réussi à convertir le composant `<App>` en HTML.

Conclusion

Au cours de ce tutoriel, vous avez initialisé une application React et activé le rendu côté serveur.

Cependant, cet article n'aborde qu'une infime partie de ce qu'il est possible de faire. Les choses ont tendance à devenir un peu plus compliquées lorsque le rendu d'une app côté serveur implique un routage, la récupération de données ou Redux.

L'un des principaux avantages de l'utilisation de SSR est que le contenu d'une app peut être analysé, même par les robots d'exploration qui n'exécutent pas de code JavaScript. Cela peut s'avérer être très pratique pour optimiser les moteurs de recherche (SEO) et alimenter les canaux des médias sociaux en métadonnées.

Souvent, SSR pourra également vous aider à optimiser votre performance car l'app lancée sera entièrement chargée à partir du serveur à la première requête. Pour les apps non triviales, le parcours peut être quelque peu différent. En effet, la configuration de SSR peut s'avérer un peu compliquée et créer une plus grande charge sur le serveur. La décision d'utiliser un rendu côté serveur de votre app React dépend de vos besoins spécifiques et des compromis les plus pertinents pour votre utilisation.

Si vous souhaitez en savoir plus sur React, consultez notre série [Comment coder dans React.js](#) ou consultez [notre page thématique React](#) pour des exercices et des projets de programmation.



Want to learn more? Join the DigitalOcean Community!

Join our DigitalOcean community of over a million developers for free! Get help and share knowledge in our Questions & Answers section, find tutorials and tools that will help you grow as a developer and scale your project or business, and subscribe to topics of interest.

[Sign up →](#)



About the authors



[Alligator.io](#) Author

Developer and author at DigitalOcean.



[Bradley Kouchi](#) Editor

Developer and author at DigitalOcean.

Still looking for an answer?

[Ask a question](#)

[Search for more help](#)

Was this helpful?

[Yes](#)

[No](#)



Comments

Leave a comment

B I U N C ⌂ ↲ H1 H2 H3 ≡ 1. “ ” ⓘ ☰ <>



Leave a comment...

[Login to Comment](#)



This work is licensed under a Creative Commons Attribution-NonCommercial- ShareAlike 4.0 International License.



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a
Newsletter.

HOLLIE'S HUB FOR GOOD

Working on improving health and
education, reducing inequality,
and spurring economic growth?
We'd like to help.

BECOME A CONTRIBUTOR

You get paid; we donate to tech
nonprofits.

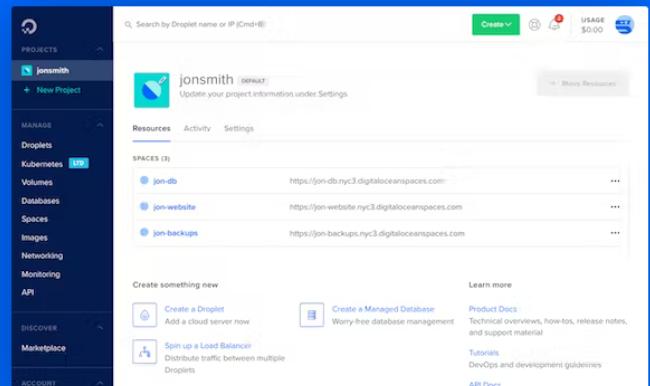
Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage Marketplace VPC Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

[Learn More](#)



© 2022 DigitalOcean, LLC. All rights reserved.

Company

About
Leadership
Blog
Careers
Partners
Referral Program
Press
Legal
Security & Trust Center

Products

Pricing
Products Overview
Droplets
Kubernetes
Managed Databases
Spaces
Marketplace
Load Balancers
Block Storage
API Documentation
Documentation
Release Notes

Community

Tutorials
Q&A
Tools and Integrations
Tags
Write for DigitalOcean
Presentation Grants
Hatch Startup Program
Shop Swag
Research Program
Open Source
Code of Conduct

Contact

Get Support
Trouble Signing In?
Sales
Report Abuse
System Status
Share your ideas