18 FÉVRIER 2020 / #REACT

Un guide complet du débutant pour réagir au routeur (y compris les crochets de routeur)

Ibrahima Ndaw

Passionné de JavaScript, développeur et blogueur Full-stack

React est une bibliothèque JavaScript pour créer des interfaces utilisateur. Nous pouvons également l'étendre pour créer des applications multi-pages à l'aide de React Router. Il s'agit d'une bibliothèque tierce qui permet le routage dans nos applications React.

Dans ce tutoriel, nous allons couvrir tout ce que vous devez savoir pour commencer avec React Router.

Mise en place du projet

Pour pouvoir suivre, vous devrez créer une nouvelle application React en exécutant la commande suivante dans votre terminal:

```
npx create-react-app react-router-guide
```

Ensuite, ajoutez ces lignes de code au `App.js` fichier:

```
import React from "react";
import "./index.css"

export default function App() {
  return (
    <main>
      <nav>
        <ul>
          <li><a href="/">Home</a></li>
          <li><a href="/about">About</a></li>
          <li><a href="/contact">Contact</a></li>
        </ul>
        </nav>
     </main>
  );
}
// Home Page
const Home = () => (
  <Fragment>
    <h1>Home</h1>
    <FakeText />
  </Fragment>
  );
// About Page
const About = () => (
  <Fragment>
    <h1>About</h1>
    <FakeText />
  </Fragment>
  );
// Contact Page
const Contact = () => (
  <Fragment>
    <h1>Contact</h1>
    <FakeText />
  </Fragment>
```

```
const FakeText = () => (
  <p>
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore
  </p>
  )
```

Ensuite, si vous êtes prêt à partir, commençons par répondre à une question importante: qu'est-ce que le routage?

Qu'est-ce que le routage?

Le routage est la capacité de montrer différentes pages à l'utilisateur. Cela signifie que l'utilisateur peut se déplacer entre différentes parties d'une application en entrant une URL ou en cliquant sur un élément.

Comme vous le savez peut-être déjà, par défaut, React est livré sans routage. Et pour l'activer dans notre projet, nous devons ajouter une bibliothèque nommée react-router .

Pour l'installer, vous devrez exécuter la commande suivante dans votre terminal:

```
yarn add react-router-dom
```

Ou

```
npm install react-router-dom
```

Maintenant que nous avons installé notre routeur avec succès, commençons à l'utiliser dans la section suivante.

Configuration du routeur

Pour activer le routage dans notre application React, nous devons d'abord importer à `BrowserRou`

Dans le `App.js` fichier, entrez les informations suivantes:

```
import React, { Fragment } from "react";
import "./index.css"

import { BrowserRouter as Router } from "react-router-dom";

export default function App() {
  return (
  <Router>
    <main>
      <nav>
        <ul>
          <li><a href="/">Home</a></li>
          <li><a href="/about">About</a></li>
          <li><a href="/contact">Contact</a></li>
        </ul>
      </nav>
    </main>
</Router>
  );
}
```

Cela devrait contenir tout dans notre application où le routage est nécessaire. Cela signifie que si nous avons besoin de routage dans l'ensemble de notre application, nous devons envelopper notre composant supérieur avec `BrowserRouter` .

Soit dit en passant, vous n'avez pas à renommer `BrowserRouter as Router` comme je le fais ici, je veux juste que les choses soient lisibles.

Un routeur seul ne fait pas grand-chose. Ajoutons donc un itinéraire dans la section suivante.

Itinéraires de rendu

Pour rendre les routes, nous devons importer le `Route` composant à partir du package du routeur.

Dans votre `App.js` fichier, ajoutez le code suivant:

```
import React, { Fragment } from "react";
import "./index.css"

import { BrowserRouter as Router, Route } from "react-router-dom";

export default function App() {
  return (
  <Router>
    <main>
      <nav>
        <ul>
          <li><a href="/">Home</a></li>
          <li><a href="/about">About</a></li>
          <li><a href="/contact">Contact</a></li>
```

```
  <Route path="/" render={() => <h1>Welcome!</h1>} />
    </main>
</Router>
  );
}
```

Ensuite, ajoutez-le là où nous voulons rendre le contenu. Le `Route` composant a plusieurs propriétés. Mais ici, nous avons juste besoin de `path` et `render` .

`path` : le chemin de l'itinéraire. Ici, nous utilisons `/` pour définir le chemin de la page d'accueil.

`render` : affichera le contenu chaque fois que l'itinéraire sera atteint. Ici, nous rendrons un message de bienvenue à l'utilisateur.

Dans certains cas, desservir des itinéraires comme celui-ci est parfaitement bien. Mais imaginez un cas où nous devons traiter avec un composant réel - l'utilisation `render` peut ne pas être la bonne solution.

Alors, comment afficher un vrai composant? Eh bien, le `Route` composant a une autre propriété nommée `component` .

Mettons à jour notre exemple un peu pour le voir en action.

Dans votre `App.js` fichier, ajoutez le code suivant:

```
import React, { Fragment } from "react";
import "./index.css"

import { BrowserRouter as Router, Route } from "react-router-dom";

export default function App() {
  return (
   <Router>
    <main>
      <nav>
        <ul>
          <li><a href="/">Home</a></li>
          <li><a href="/about">About</a></li>
          <li><a href="/contact">Contact</a></li>
        </ul>
      </nav>

    <Route path="/" component={Home} />
    </main>
</Router>
  );
}

const Home = () => (
  <Fragment>
    <h1>Home</h1>
    <FakeText />
  </Fragment>
  );
```

Maintenant, au lieu de rendre un message, notre route va charger le `Home` composant.

Pour obtenir toute la puissance de React Router, nous devons avoir plusieurs pages et liens avec lesquels jouer. Nous avons déjà des pages (des composants si vous le souhaitez aussi), alors maintenant, ajoutons quelques liens afin de pouvoir basculer entre les pages.

Utilisation de liens pour changer de page

Pour ajouter des liens à notre projet, nous utiliserons à nouveau le React Router.

Dans votre `App.js` fichier, ajoutez le code suivant:

```
import React, { Fragment } from "react";
import "./index.css"

import { BrowserRouter as Router, Route, Link } from "react-router-dom";

export default function App() {
  return (
   <Router>
    <main>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/about">About</Link></li>
          <li><Link to="/contact">Contact</Link></li>
        </ul>
      </nav>

    <Route path="/" exact component={Home} />
    <Route path="/about"  component={About} />
    <Route path="/contact"  component={Contact} />

    </main>
</Router>
  );
}

const Home = () => (
  <Fragment>
    <h1>Home</h1>
    <FakeText />
  </Fragment>
  );

const About = () => (
  <Fragment>
    <h1>About</h1>
    <FakeText />
  </Fragment>
  );

const Contact = () => (
  <Fragment>
    <h1>Contact</h1>
    <FakeText />
  </Fragment>
  );
```

Maintenant, au lieu d'utiliser `a` tag et `href` , React Router utilise `Link` et `to` pour, bien, pouvoir basculer entre les pages sans le recharger.

Ensuite, nous devons ajouter deux nouvelles routes `About` et `Contact` , pour pouvoir basculer entre les pages ou les composants.

Maintenant, nous pouvons accéder à différentes parties de notre application via des liens. Mais il y a un problème avec notre routeur: le `Home` composant est toujours affiché même si nous passons à d'autres pages.

Cela est dû au fait que React Router vérifiera si la `path` définition commence par `/` . Si tel est le cas, il rendra le composant. Et ici, notre premier itinéraire commence par `/` , donc le `Home` composant sera rendu à chaque fois.

Cependant, nous pouvons toujours modifier le comportement par défaut en ajoutant la `exact` propriété à `Route` .

Dans `App.js` , ajoutez:

```
<Route path="/" exact component={Home} />
```

En mettant à jour l' `Home` itinéraire avec `exact` , maintenant il ne sera rendu que s'il correspond au chemin complet.

Nous pouvons encore l'améliorer en encapsulant nos routes avec `Switch` pour indiquer à React Router de ne charger qu'une seule route à la fois.

Dans `App.js` , ajoutez:

```
import { BrowserRouter as Router, Route, Link, Switch } from "react-router-dom";

  <Switch>
    <Route path="/" exact component={Home} />
    <Route path="/about"  component={About} />
    <Route path="/contact"  component={Contact} />
  </Switch>
```

Maintenant que nous avons de nouveaux liens, utilisons-les pour passer des paramètres.

Passer les paramètres de l'itinéraire

Pour transmettre des données entre les pages, nous devons mettre à jour notre exemple.

Dans votre `App.js` fichier, ajoutez le code suivant:

```
import "./index.css"

import { BrowserRouter as Router, Route, Link, Switch } from "react-router-dom";

export default function App() {
  const name = 'John Doe'
  return (
   <Router>
    <main>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to={`/about/${name}`}>About</Link></li>
          <li><Link to="/contact">Contact</Link></li>
        </ul>
      </nav>
    <Switch>
      <Route path="/" exact component={Home} />
      <Route path="/about/:name"  component={About} />
      <Route path="/contact"  component={Contact} />
    </Switch>
    </main>
</Router>
  );
}

const Home = () => (
  <Fragment>
    <h1>Home</h1>
    <FakeText />
  </Fragment>
  );

const About = ({match:{params:{name}}}) => (
  // props.match.params.name
  <Fragment>
    <h1>About {name}</h1>
    <FakeText />
  </Fragment>
);

const Contact = () => (
  <Fragment>
    <h1>Contact</h1>
    <FakeText />
  </Fragment>
  );
```

Comme vous pouvez le voir ici, nous commençons par déclarer une nouvelle constante `name` qui sera passée en paramètre à la `About` page. Et nous ajoutons `name` au lien correspondant.

Avec cela, nous devons maintenant mettre à jour l' `About` itinéraire en ajustant son chemin pour recevoir `name` comme paramètre `path="/about/:name"` .

Maintenant, le paramètre sera reçu comme accessoire du `About` composant. La seule chose que nous devons faire maintenant est de détruire les accessoires et de récupérer la `name` propriété. Par ailleurs, `{match:{params:{name}}}` est le même que `props.match.params.name` .

Nous avons fait beaucoup jusqu'ici. Mais dans certains cas, nous ne voulons pas utiliser de liens pour naviguer entre les pages.

Parfois, nous devons attendre la fin d'une opération avant de passer à la page suivante.

Alors, traitons ce cas dans la section suivante.

Navigation par programmation

Les accessoires que nous recevons ont quelques méthodes pratiques que nous pouvons utiliser pour naviguer entre les pages.

Dans `App.js` , ajoutez:

```
const Contact = ({history}) => (
  <Fragment>
    <h1>Contact</h1>
    <button onClick={() => history.push('/') } >Go to home</button>
    <FakeText />
  </Fragment>
  );
```

Ici, nous tirons l' `history` objet des accessoires que nous recevons. Il a des méthodes pratiques comme `goBack` , `goForward` et ainsi de suite. Mais ici, nous allons utiliser la `push` méthode pour pouvoir accéder à la page d'accueil.

Maintenant, traitons le cas lorsque nous voulons rediriger notre utilisateur après une action.

Redirection vers une autre page

Le React Router a un autre composant nommé `Redirect` . Comme vous l'avez deviné, cela nous

Dans `App.js`, ajoutez:

```
import { BrowserRouter as Router, Route, Link, Switch, Redirect } from "react-router-dom";

const About = ({match:{params:{name}}}) => (
  // props.match.params.name
  <Fragment>
    { name !== 'John Doe' ? <Redirect to="/" /> : null }
    <h1>About {name}</h1>
    <FakeText />
  </Fragment>
);
```

Maintenant, si le `name` passé en tant que paramètre n'est pas égal à `John Doe`, l'utilisateur sera redirigé vers la page d'accueil.

Vous pourriez faire valoir que vous devez rediriger l'utilisateur avec `props.history.push('/)`. Eh bien, le `Redirect` composant remplace la page et donc l'utilisateur ne peut pas revenir à la page précédente. Mais, avec la méthode push, ils le peuvent. Cependant, vous pouvez utiliser `props.history.replace('/)` pour imiter le `Redirect` comportement.

Passons maintenant à autre chose et traitons le cas lorsque l'utilisateur atteint une route qui n'existe pas.

Redirection vers une page 404

Pour rediriger l'utilisateur vers une page 404, vous pouvez créer un composant pour l'afficher. Mais ici, pour garder les choses simples, je vais simplement afficher un message avec `render`.

```
import React, { Fragment } from "react";
import "./index.css"

import { BrowserRouter as Router, Route, Link, Switch } from "react-router-dom";

export default function App() {
  const name = 'John Doe'

  return (
   <Router>
    <main>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to={`/about/${name}`}>About</Link></li>
          <li><Link to="/contact">Contact</Link></li>
        </ul>
      </nav>
    <Switch>
      <Route path="/" exact component={Home} />
      <Route path="/about/:name"  component={About} />
      <Route path="/contact"  component={Contact} />
      <Route render={() => <h1>404: page not found</h1>} />

    </Switch>
```

```
  );
}
```

La nouvelle route que nous avons ajoutée capturera tous les chemins qui n'existent pas et redirigera l'utilisateur vers la page 404.

Maintenant, passons à autre chose et apprenons comment protéger nos itinéraires dans la section suivante.

Itinéraires de garde

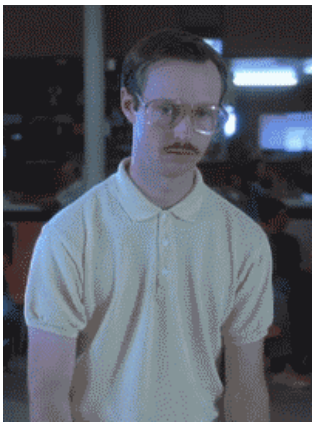
Il existe de nombreuses façons de protéger les itinéraires vers React. Mais ici, je vais juste vérifier si l'utilisateur est authentifié et le rediriger vers la page appropriée.

```
import React, { Fragment } from "react";
import "./index.css"

import { BrowserRouter as Router, Route, Link, Switch } from "react-router-dom";

export default function App() {
  const name = 'John Doe'
  const isAuthenticated = false
  return (
   <Router>
    <main>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to={`/about/${name}`}>About</Link></li>
          <li><Link to="/contact">Contact</Link></li>
        </ul>
      </nav>
    <Switch>
      <Route path="/" exact component={Home} />
      {
      isAuthenticated ?
      <>
      <Route path="/about/:name"  component={About} />
      <Route path="/contact"  component={Contact} />
      </> : <Redirect to="/" />
      }

    </Switch>
    </main>
</Router>
  );
}
```

Comme vous pouvez le voir ici, j'ai déclaré une variable pour imiter l'authentification. Ensuite, vérifiez si l'utilisateur est authentifié ou non. Si tel est le cas, restituez les pages protégées. Sinon, redirigez-les vers la page d'accueil.

Nous avons couvert beaucoup de choses jusqu'à présent, mais une partie intéressante reste: les

Passons à la dernière section et introduisons les crochets.



Crochets de routeur

Les crochets de routeur facilitent les choses. Vous pouvez désormais accéder à l'historique, à l'emplacement ou aux paramètres de manière simple et élégante.

useHistory

Le `useHistory` crochet nous donne accès à l'instance d'historique sans la retirer des accessoires.

```
import { useHistory } from "react-router-dom";

const Contact = () => {
const { history } = useHistory();
return (
  <Fragment>
    <h1>Contact</h1>
    <button onClick={() => history.push('/') } >Go to home</button>
  </Fragment>
  )
  };
```

useParams

Ce crochet nous aide à obtenir le paramètre passé sur l'URL sans utiliser l'objet props.

```
import { BrowserRouter as Router, Route, Link, Switch, useParams } from "react-router-dom";

export default function App() {
  const name = 'John Doe'
  return (
   <Router>
    <main>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to={`/about/${name}`}>About</Link></li>
```

```
    <Switch>
      <Route path="/" exact component={Home} />
      <Route path="/about/:name"  component={About} />
    </Switch>
    </main>
</Router>
  );
}

const About = () => {
  const { name } = useParams()
  return (
  // props.match.params.name
  <Fragment>
    { name !== 'John Doe' ? <Redirect to="/" /> : null }
    <h1>About {name}</h1>
    <Route component={Contact} />
  </Fragment>
)
};
```

useLocation

Ce crochet renvoie l'objet d'emplacement qui représente l'URL actuelle.

```
import { useLocation } from "react-router-dom";

const Contact = () => {
const { pathname } = useLocation();

return (
  <Fragment>
    <h1>Contact</h1>
    <p>Current URL: {pathname}</p>
  </Fragment>
  )
  };
```

Dernières pensées

React Router est une bibliothèque incroyable qui nous aide à passer d'une seule page à une application multi-pages avec une grande convivialité. (Gardez juste à l'esprit - à la fin de la journée, c'est toujours une application d'une seule page).

Et maintenant, avec les crochets de toupie, vous pouvez voir à quel point ils sont faciles et élégants. Ils sont définitivement quelque chose à considérer dans votre prochain projet.

Vous pouvez lire plus de mes articles sur mon blog .

Prochaines étapes

Documentation de React Router

Si vous lisez jusqu'ici, envoyez un tweet à l'auteur pour lui montrer que vous vous souciez de lui.

Tweetez un remerciement

Continuez à lire sur

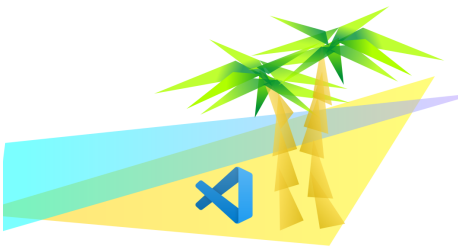
Réagir

5 projets React dont vous avez besoin dans votre portefeuille

The React Cheatsheet pour 2020 (+ exemples réels)

Comment partager des variables entre HTML, CSS et JavaScript à l'aide de Webpack

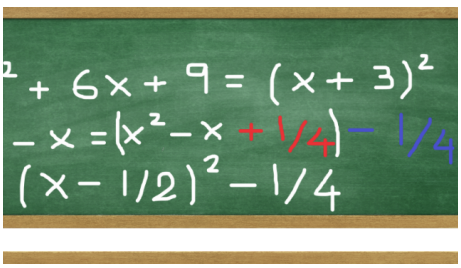
Voir les 491 articles →



#BULLE

Comment je n'ai PAS codé une application d'écriture collaborative

ERIC BUREL IL Y A 7 JOURS



#MATH

Comment compléter le carré: une méthode pour compléter le carré

ALEXANDER AROBELIDZE 8 DAYS AGO

You can make a tax-deductible donation here.

Our Nonprofit

About

Alumni Network

Open Source

Shop

Support

Sponsors

Academic Honesty

Code of Conduct

Privacy Policy

Terms of Service

Copyright Policy

Trending Guides

2019 Web Developer Roadmap

Python Tutorial

CSS Flexbox Guide

JavaScript Tutorial

Python Example

HTML Tutorial

Linux Command Line Guide

JavaScript Example

Git Tutorial

React Tutorial

Java Tutorial

Tutoriel Linux Tutoriel

CSS

Exemple jQuery

Tutoriel

SQL

Exemple

CSS Exemple

React

Tutoriel angulaire

Bootstrap Exemple

Comment configurer les clés SSH

Tutoriel WordPress

Exemple PHP