Chapitre 11 : React Route

Table des matières

I. Présentation 2

II. Installation 2

III. BrowserRoute 3

IV. Définir des routes 3

V. Link 4

VI. useNavigate 5

VII. URL parameter 5

a. Définition des routes 5

b. Définir le lien et Navigation 6

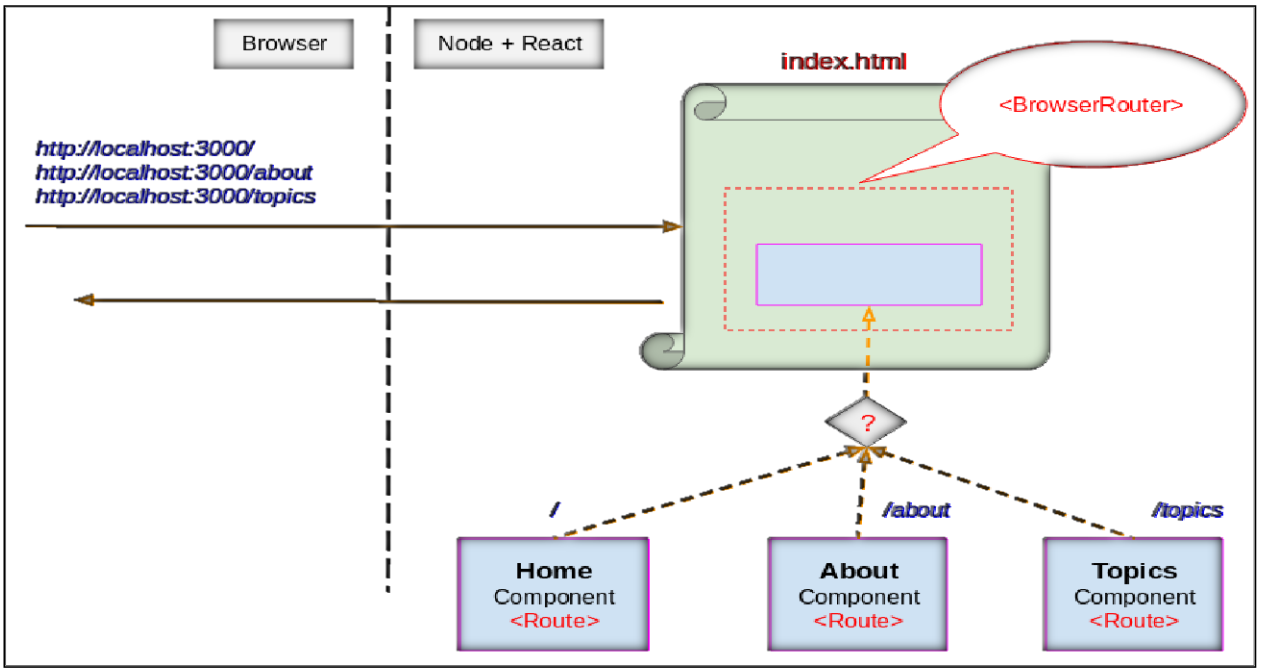
c. Récupérer les données 7

VIII. NavLink 7

IX. NestedRoute 7

I. Présentation

React Router est une bibliothèque standard pour le routage dans React. Il permet la navigation entre les vues de divers composants dans une application React, permet de modifier l'URL du navigateur et maintient l'interface utilisateur synchronisée avec l'URL.



Les principaux composants de React Router sont :

- **BrowserRouter**
- **Routes**
- **Route**
- **Link**
- **NavLink**

II. Installation

La dépendance **'react-router-dom'** doit être installée à la racine du répertoire de votre projet. Voici qu'il faut installer pour ajouter react-router :

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

III. BrowserRoute

La première chose à faire une fois l'installation terminée est de rendre React Router disponible partout dans votre application. Pour ce faire, ouvrez le fichier index.js dans le dossier src et importez BrowserRouter de react-router-dom, puis enveloppez-y le composant racine (le composant App).

BrowserRouter est le type de routage recommandé pour tous les projets web React Router. Il utilise l'API DOM History pour mettre à jour l'URL et gérer la pile d'historique.

```
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

IV. Définir des routes

Puisque le composant App est le composant racine à partir duquel notre code React est rendu initialement, nous allons créer toutes nos routes dans ce composant :

```
import { Route, Routes } from 'react-router-dom';
import './App.css';
import About from './components/About';
import Contact from './components/Contact';
import Home from './components/Home';

function App() {
  return (
    <div>
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='about' element={<About />} />
        <Route path='contact' element={<Contact />} />
      </Routes>
    </div>
  );
}

export default App;
```

Le composant <Routes> agit comme un conteneur/parent pour toutes les routes individuelles qui seront créées dans notre application.

Le composant <Route> définit une relation (mapping) entre une URL et un Component. Cela signifie que lorsque l'utilisateur visite une URL sur le navigateur, un Component correspondant doit être rendu sur l'interface. Route est utilisé pour créer une seule route. Elle prend en compte deux attributs :

- **Path :** qui spécifie le chemin URL du composant souhaité. Vous pouvez appeler ce nom de chemin comme vous le souhaitez.
- **Element :** qui spécifie le composant que la route doit rendre.

V. Link

Le composant Link est utilisé pour créer des liens vers différents chemins et mettre en œuvre la navigation dans l'application. Cela fonctionne comme une balise d'ancrage HTML <a>. Voici un exemple d'utilisation de ce composant :

```
import React from 'react'
import { Link } from 'react-router-dom'

export default function Home() {
  return (
    <div>
        <h1>This is the home page</h1>
        <Link to="about">Vers page About</Link> <br/>
        <Link to="contact">Vers page Contact</Link>
    </div>
  )
}
```

Rappelez-vous que nous avons créé les noms de chemin répertoriés dans le composant App, de sorte que lorsque vous cliquez sur le lien, il va chercher dans vos routes et rendre le composant avec le nom de chemin correspondant.

VI. useNavigate

UseNavigate hook renvoie une fonction qui vous permet de naviguer de manière programmatique.

Exemple 1 :

```
import React from "react";
import { useNavigate } from "react-router-dom";

const About = () => {
  let navigate = useNavigate();

  const goHome = () => {
    navigate("/");
  };

  return (
    <div>
      <h1>About page here!</h1>
      <p>
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Fugit, modi!
      </p>
      <button onClick={goHome}>Go to home page</button>
    </div>
  );
};

export default About;
```

VII. URL parameter

Les params sont comme des routes dynamiques. Ici, au lieu des noms de routes, ils seront remplis avec les valeurs.

a. Définition des routes

En définissant les routes elles-mêmes, vous devez mentionner les variables qui seront utilisées à l'aide du **symbole " : "** Cette méthode n'est efficace que si vous connaissez la variable dans la phase initiale elle-même.

Exemple :

```jsx
import { Route, Routes } from "react-router-dom";
import "./App.css";
import Home from "./components/Home";
import NotFound from "./components/NotFound";
import Posts from "./components/Posts";
import Todos from "./components/Todos";

function App() {
  return (
    <div>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/posts/:Userid" element={<Todos />} />
        <Route path="/todos/:Userid" element={<Posts />} />
        <Routes path="*" element={<NotFound />} />
      </Routes>
    </div>
  );
}

export default App;
```

b. Définir le lien et Navigation

En définissant le lien, remplacez simplement les paramètres par les valeurs.

Exemple :

```jsx
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";

export default function Auth() {
  const navigate = useNavigate();


  const connexion = () => {
    navigate('home/1');
  };

  return (
    <div>
        <button onClick={connexion}>Connexion</button>
    </div>
  );
}
```

c. Récupérer les données

Les données sont récupérées à l'aide de useParams Hook.

Exemple :

```
import * as React from 'react';
import { Routes, Route, useParams } from 'react-router-dom';

function ProfilePage() {
  // Get the userId param from the URL.
  let { userId } = useParams();
  // ...
}

function App() {
  return (
    <Routes>
      <Route path="users">
        <Route path=":userId" element={<ProfilePage />} />
        <Route path="me" element={...} />
      </Route>
    </Routes>
  );
}
```

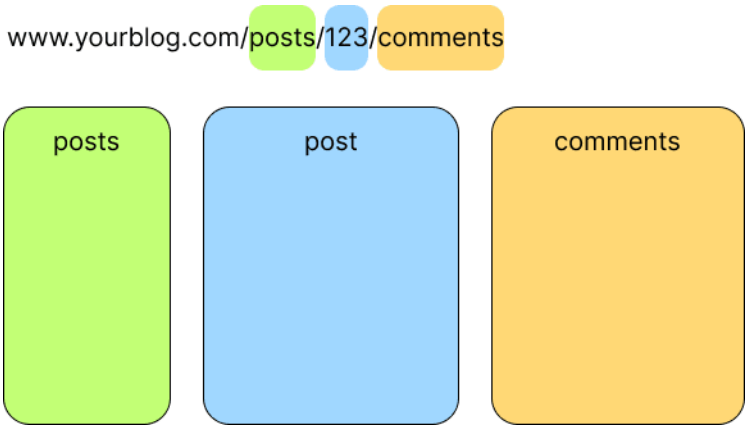
VIII. NavLink

Un <NavLink> est un type spécial de <Link> qui sait s'il est "actif" ou non.

```
        <NavLink
            to="tasks"
            className={({ isActive }) =>
              isActive ? activeClassName : undefined
            }
          >
            Tasks
          </NavLink>
```

IX. NestedRoute

(source https://dev.to/tywenk/how-to-use-nested-routes-in-react-router-6-4jhd)



La version 6 de React Router facilite l'imbrication des routes. Les routes imbriquées vous permettent d'avoir plusieurs composants rendus sur la même page avec une parité de route.

```
import { Link, Outlet, Route, Routes } from "react-router-dom";
import "./App.css";

function Home() {
  return (
    <div>
      <h1>Component Home</h1>

      <nav>
        <Link to="user">User</Link>
      </nav>
    </div>
  );
}

function User() {
  return (
    <div>
      <h1>Component USER</h1>


      <nav>
        <Link to="profile">Profile</Link> <br/>
        <Link to="account">Account</Link>
      </nav>

      <Outlet />
     </div>
   );
  }
```

```
function NoMatch()
  { return (
    <div>
      <h1>Component No match</h1>

    </div>
  );
}

function Profile()
  { return (
    <div>
      <h1>Component Profile</h1>
    </div>
  );
}

function Account()
  { return (
    <div>
      <h1>Component Account</h1>
    </div>
  );
}
function App() {
  return (
    <div>
      <Routes>
        <Route index element={<Home />} />
        <Route path="home" element={<Home />} />
        <Route path="user" element={<User />}>
          <Route index element={<Profile />} />
          <Route path="profile" element={<Profile />} />
          <Route path="account" element={<Account />} />
        </Route>
        <Route path="*" element={<NoMatch />} />
      </Routes>
    </div>
  );
}
export default App;
```

Un <Outlet> doit être utilisé dans les éléments de route parents pour rendre leurs éléments de route enfants. Cela permet à l'interface utilisateur imbriquée de s'afficher lorsque les routes enfant sont rendues. Si la route parent correspond exactement, elle rendra une route index enfant ou rien s'il n'y a pas de route index.