esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

# Routage

V6

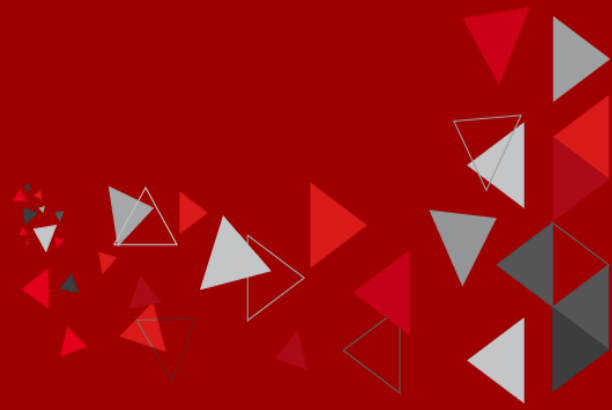
REACT ROUTER

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia

EUR-ACE® Délivrée par la Commission des Titres d'Ingénieur

CONFÉRENCE DES GRANDES ÉCOLES

cdio CONCEIVE DESIGN IMPLEMENT OPERATE



- Routage - Définition
- Configuration des routes
- Définition des routes
- Navigation
- Lazy Loading

# Routage - Définition

Le routage est la gestion des différentes URL de votre application et de la façon dont elles sont liées à des composants React.C'est est le fait de naviguer d'une vue à une autre selon la demande de l'utilisateur.

Un utilisateur peut:

- Taper l'url d'une vue dans le navigateur
- Cliquer sur un lien, bouton, etc..
- Cliquer sur les boutons de retour du navigateur

En utilisant un système de routage, vous pouvez décomposer votre application de différents composants et les organiser de manière cohérente en fonction de leur URL correspondante. Cela permet une meilleure organisation du code et une expérience utilisateur plus fluide.

# Configuration des routes

Pour gérer le routing dans un projet react, nous utilisons la librairie **« react-router-dom »**, une variante de la librairie **« react-router »** spécialement conçue pour le routage des applications web qui fournissent des fonctionnalités avancées telles que la gestion des paramètres de l'URL, la protection des routes, le rendu de composants en fonction de l'URL et bien plus encore.

Afin d'installer la bibliothèque **« react-router-dom »** vous devez utiliser cette commande :

- **npm install react-router-dom ou yarn add react-router-dom**

# Configuration des routes

Afin de configurer les routes , nous aurons besoin d'importer votre routeur **« BrowserRouter »** dans la page **index.js** de votre application et il enveloppera votre composant **App**.

```
import { BrowserRouter } from 'react-router-dom';

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

- **BrowserRouter** fonctionne exactement comme un contexte dans Reactet fournit toutes les informations nécessaires à votre application pour que vous puissiez faire du routage.

# Définition des routes

Définir des routes est aussi simple que de définir un seul composant **Route** pour chaque route de votre application, puis de placer tous ces composants Route dans un seul composant **Routes**.

Lorsque votre URL change, **React Router** consulte les routes définies dans votre composant **Routes** et rend le contenu dans le prop **element** de la route dont le chemin correspond à l'URL.

```
<Routes>
  <Route path="/home" element={<Home />} />
</Routes>
```

# Définition des routes

En plus de prop **path** et **element** , il y a également d'autres **props** qui sont importants pour le composant **Route** comme :

- **« loader » :** Chaque route peut définir une fonction "loader" pour fournir des données à l'élément de la route avant qu'il ne soit rendu.
- **« errorElement » :** Lorsque des exceptions sont levées dans les chargeurs, les actions ou le rendu des composants, au lieu du chemin de rendu normal pour vos Routes (<Route element>), le chemin d'erreur sera rendu (<Route errorElement>) et l'erreur rendue disponible avec hook **useRouteError**.

# Définition des routes

## Route Non Paramétrée

Dans l'exemple ci-dessus, nous allons avoir des routes  non paramétrées:

- Si notre URL était ***/home***, le composant **Home** serait rendu.
- Si notre URL était ***/contact***, le composant **Contact** serait rendu.
- Si notre URL était ***/about***, le composant **About** serait rendu.

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

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

# Définition des routes

## 1.Route Paramétrée

Une « route dynamique » ou « route paramétrée » est une route qui peut prendre en compte des paramètres dans son URL pour afficher des données spécifiques à l'utilisateur.

- Dans l'exemple ci-dessus, nous allons envoyé un paramètre au composant Home:

```
function App() {
  return (
    <Routes>
      <Route path="/home/:username" element={<Home />} />
  </Routes>
  );
}
```

# Définition des routes

## 1.Route Paramétrée

- Afin d'avoir un accès au paramètres envoyés dans l'url , nous aurons besoin du hook **useParams().**

```
import { useParams } from "react-router-dom"

export default function Home() {
    const { username } = useParams()

return <h1> welcome {username} to Home Page</h1>

}
```

Résultat :

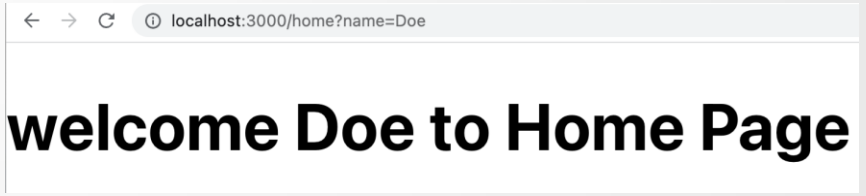
# Définition des routes

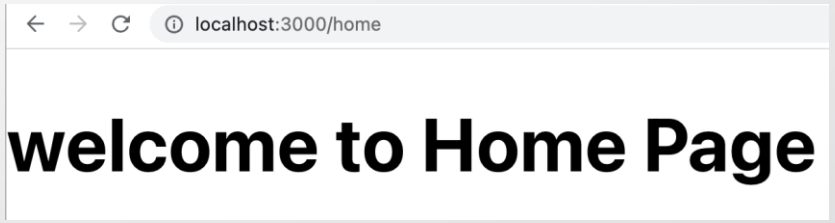
## 2.Route Paramétrée

- On peut avoir aussi des routes avec des paramètres de recherche qui sont tous les paramètres qui viennent après le ? dans une URL (?name=Joe&age=27).
- Pour utiliser les paramètres de recherche, vous devez utiliser le hook **useSearchParams** qui fonctionne de manière très similaire au hook **useState**.

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

function Home() {

    const [searchParams, setSearchParams] = useSearchParams({ name: "" })

  return (
    <>
      <h1>welcome {searchParams.get('name')} to Home Page</h1>

    </>
  );
}

export default Home;
```

Résultat :



# Définition des routes

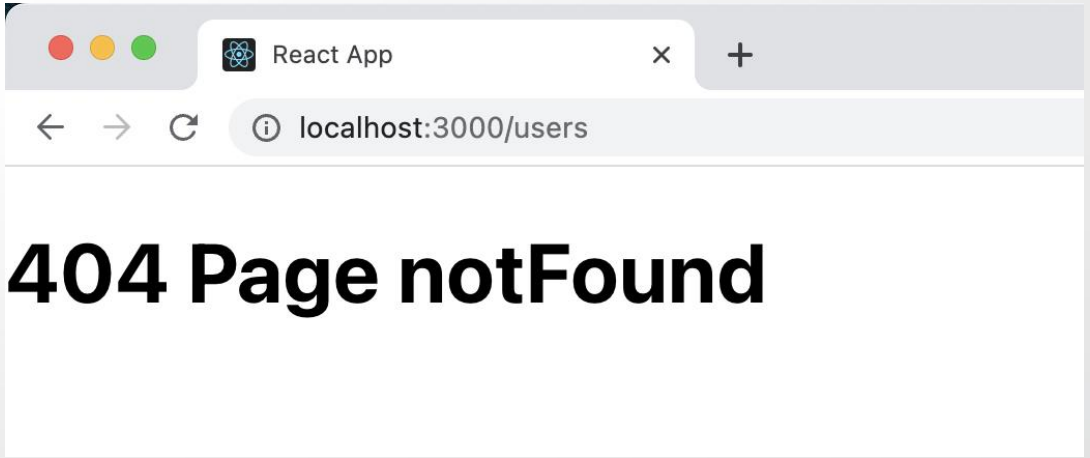
## NotFound

Une route avec * correspondra à n'importe quoi, ce qui le rend parfait pour des choses comme une page 404.

L'exemple çi-dessous montre que la route / users n'existe pas dans la redirection sera pour le composant NotFound:

```
function App() {
  return (
    <Routes>
        <Route path="/home/:username" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="*" element={<NotFound />} />
  </Routes>
  );
}
```

Résultat :



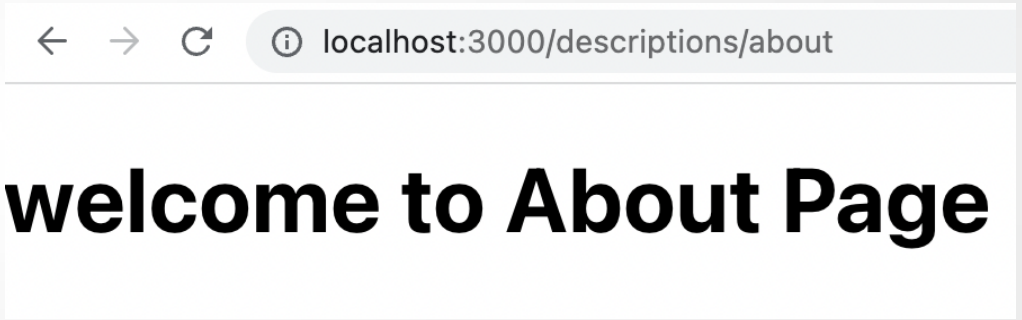
# Définition des routes

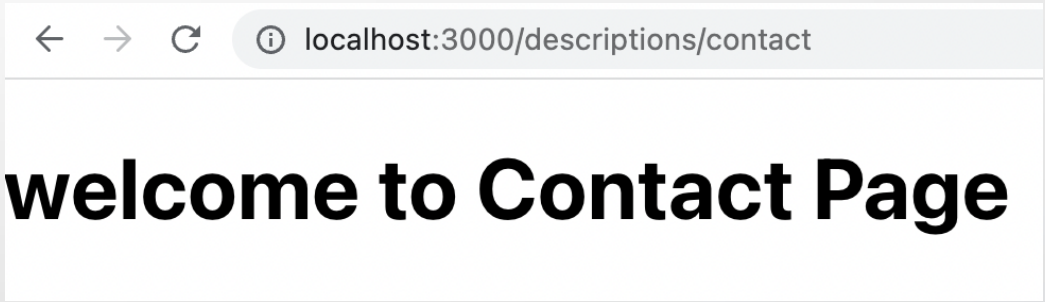
## 1.Route Imbriquée

Une route imbriquée est une route qui se trouve à l'intérieur d'une autre route. Cela permet de créer une arborescence de routes où une route parent peut inclure plusieurs sous-routes.

```
function App() {
  return (
    <Routes>
      <Route path="/home/:username" element={<Home />} />
      <Route path="/descriptions">
        <Route path="about" element={<About />} />
        <Route path="contact" element={<Contact />} />
      </Route>
      <Route path="*" element={<NotFound />} />
  </Routes>
  );
}
```

Résultat :



# Définition des routes

## 2.Route Imbriquée-Templates partagés

```jsx
<Routes>
  <Route path="/home/:username" element={<Home />} />
  <Route path="/products" element={<ProductsLayout />}>
    <Route index element={<ProductsList />} />
    <Route path=":id" element={<Product/>} />
    <Route path="add" element={<NewProduct />} />
  </Route>
  <Route path="*" element={<NotFound />} />
</Routes>
```

```jsx
import {Outlet } from "react-router-dom";

export function ProductsLayout() {
  return (
    <>
      <Outlet />
    </>
  );
}
```

Le composant **<Outlet>** doit être utilisé en tant qu'espace réservé où les routes enfants imbriquées seront rendues.

Le prop **index** signifie simplement que le **path** de la route est le même que celui de la route parente.

# Définition des routes

## 3.Route Imbriquée-Templates partagés

Pour passer les props d'une route parent à une route enfant, React Router 6 a une propriété de Outlet appelée **context**. **context** accepte un tableau d'états.

Pour qu'une route enfant accepte le contexte, le composant enfant doit utiliser le hook fourni par React Router, **useOutletContext()**.

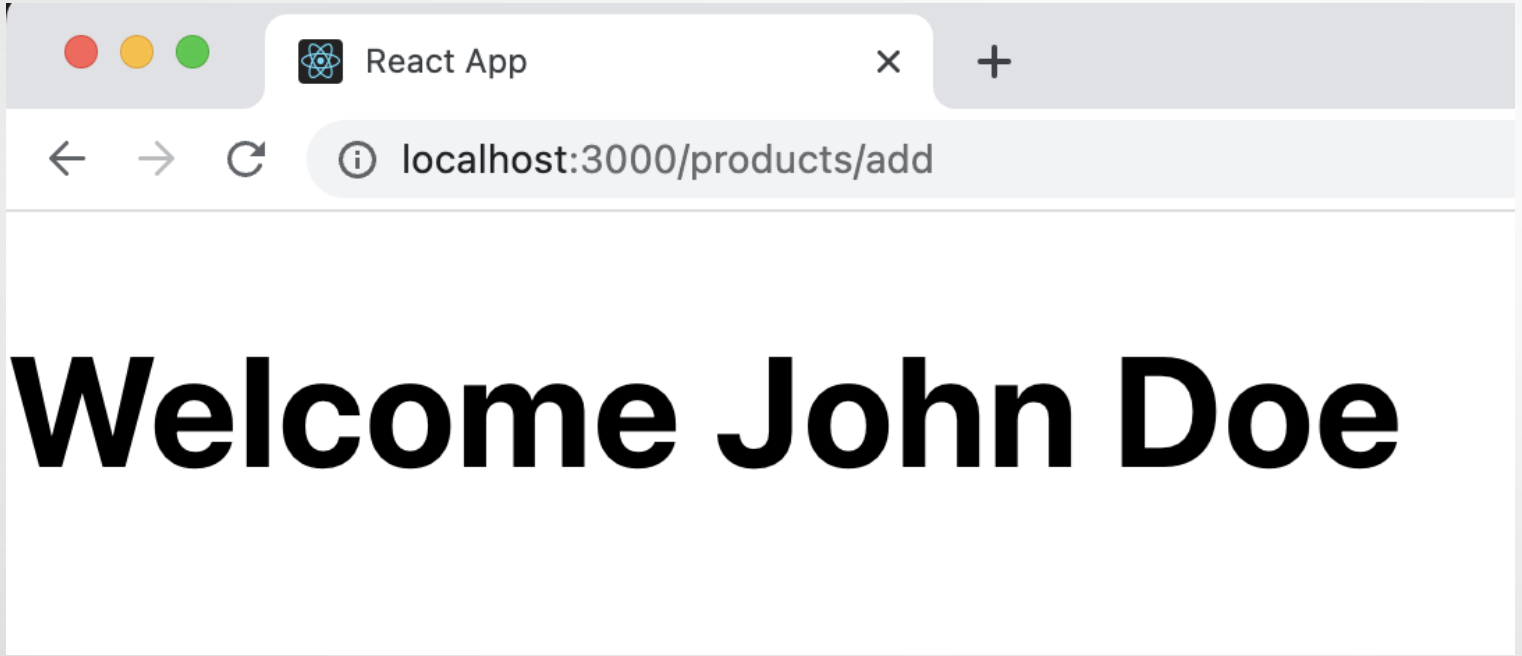
```
import { useState } from "react"
import { Outlet } from "react-router-dom"

export function ProductsLayout() {
const [currentUser, setCurrentUser] = useState([ "John Doe"])
  return (
    <>
      <Outlet context={[currentUser]}/>
    </>
  )
}
```

```
import { useOutletContext } from "react-router-dom"

export default function NewProduct() {
  const [currentUser] = useOutletContext()

    return (
        <div>
            <h1>Welcome {currentUser}</h1>
      <form/>
        </div>
    )
}
```

# Définition des routes

## 3.Route Imbriquée-Templates partagés

Résultat :



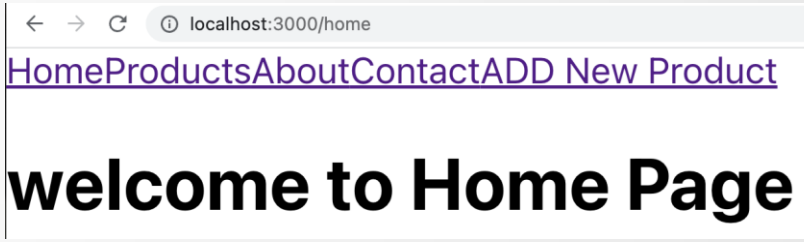
# Navigation

## Link

Le composant Link est utilisé comme une alternative à l'utilisation d'une balise **<a …></a>** classique, car il offre un moyen plus optimisé et plus riche en fonctionnalités de naviguer entre les routes dans votre application React.

```
<Link to="home">Home</Link>
<Link to="products">Products</Link>
<Link to="descriptions/about">About</Link>
<Link to="descriptions/contact">Contact</Link>
<Link to="products/add">ADD New Product </Link>
```

Résultat :




En plus de prop **to**, il y a également d'autres **props** qui sont importants pour le composant **Link** comme :

- **« reloadDocument »**
- **« replace »**
- **« state »**

# Navigation

## Link - reloadDocument

Cette prop **« reloadDocument »** prend une valeur booléenne.Si elle est défini à **true**, votre composant **Link** agira comme une balise **<a …></a>** normale et effectuera un rafraîchissement complet de la page lors de la navigation au lieu de simplement réafficher le contenu dans votre composant **Routes**.

```
<Link to="descriptions/contact" reloadDocument={true}>Contact</Link>
```

# Navigation

## Link - replace

L'option **replace** prend une valeur booléenne qui, lorsqu'il est défini **true**, permet à ce lien de remplacer la page actuelle dans l'historique du navigateur.

- Imaginons que vous ayez l'historique de navigation suivant :

```
/home
/descriptions/about
/home
```

```
<Link to="home">Home</Link>
<Link to="descriptions/about">About</Link>
<Link to="descriptions/contact" replace={true}>Contact</Link>
```

- Si vous cliquez sur un lien qui mène à la page **/descriptions/contact** et que la propriété **replace** est définie sur true, votre nouvel historique ressemblera à ceci :

```
/home
/descriptions/about
/descriptions/contact
```

# Navigation

## Link - state

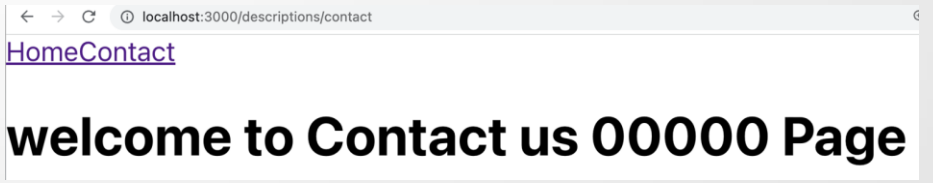
La propriété **state** peut être utilisée pour définir une valeur d'état pour le nouvel emplacement qui est stockée dans l'état historique.

Cette valeur est ensuite accessible via **useLocation()**.

```
<Link to="home" >Home</Link>
<Link to="descriptions/contact" state={{ number: "00000" }}>Contact</Link>
```

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

function Contact() {
  const { state } = useLocation();

  return <h1>welcome to Contact us {state.number} Page </h1>;
}

export default Contact;
```

Résultat :



# Navigation

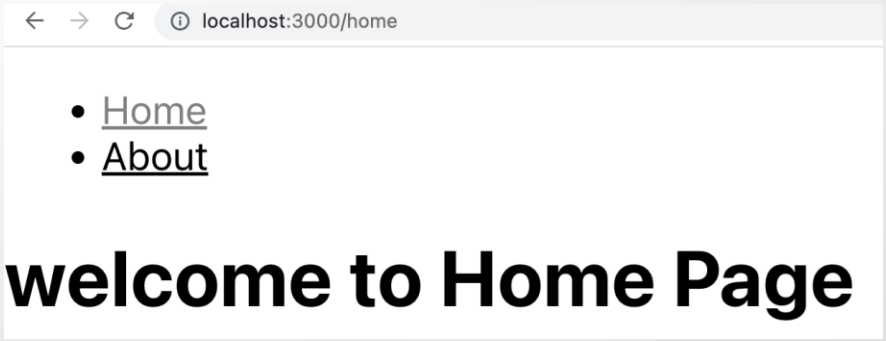
## NavLink

Un **<NavLink>** est un type spécial de **<Link>** qui sait s'il est actif ou non. C'est utile lors de la construction d'un menu de navigation.

- Imaginons que vous avez ce menu de navigation suivant :

```
<nav>
<ul>
  <li>
    <NavLink
      to="home"
      style={({ isActive }) => ({ color: isActive ? "gray" : "black" })}
    >
      Home
    </NavLink>
  </li>
  <li>
    <NavLink
      to="descriptions/about"
      style={({ isActive }) => ({ color: isActive ? "gray" : "black" })}
    >
      About
    </NavLink>
  </li>
</ul>
</nav>
```

Résultat :



# Navigation

## Navigation manuelle - Navigate

Afin de naviguer manuellement un utilisateur en fonction d'éléments tels que la soumission d'un formulaire ou l'impossibilité d'accéder à une page spécifique. Pour ces cas d'utilisation, vous devrez utiliser le composant **Navigate** ou le hook **useNavigate**.

- Dans cet exemple, lorsque le composant **Navigate** est rendu ,il redirige automatiquement l'utilisateur vers la valeur du prop **to** du composant **Na**

```
<Navigate to="................" />
```

- Dans cet exemple , lorsque l'utilisateur clique sur le bouton Go To Contact , il sera redirigé vers la valeur du **navigate**.

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

function Home() {
  const navigate = useNavigate();

  return (
    <>
      <h1>welcome to Home Page</h1>
      <button onClick={() => navigate("/descriptions/contact")}>
        Go to Contact
      </button>
    </>
  );
}
```

# Navigation

## Navigation manuelle - Navigate

Une autre façon d'utiliser la fonction de **navigate** est de lui transmettre un nombre. Cela vous permettra de simuler la clique du bouton avant/arrière.

```
navigate(-1) // Revenir une page en arrière dans l'history
navigate(1) // Avancez d'une page dans l'history
```

# Lazy Loading

## React.lazy

React.lazy est une fonctionnalité de React qui permet de charger de manière asynchrone des composants spécifiques uniquement lorsqu'ils sont nécessaires. Cela signifie que vous pouvez retarder le chargement de composants volumineux jusqu'à ce qu'ils soient vraiment nécessaires, ce qui peut améliorer les performances de votre application et donner une expérience utilisateur plus rapide.

- Dans cet exemple, les composants Home,About et Contact ne seront chargés que lorsqu'il sera nécessaire et affichés à l'écran, ce qui peut améliorer les performances de votre application.

```
const Contact = React.lazy(() => import('./Contact'));
const About = React.lazy(() => import('./About'));
const Home = React.lazy(() => import('./Home'));
```

# Lazy Loading

## React.lazy

Le composant importé dynamiquement devrait être exploité dans un composant **Suspense**, qui nous permet d'afficher un contenu de repli (ex. un indicateur de chargement) en attendant que ce module soit chargé.

```
<Suspense fallback={<p>Chargement...</p>}>
  <Routes>
    <Route path="/home" element={<Home />} />
    <Route path="/descriptions">
      <Route path="about" element={<About />} />
      <Route path="contact" element={<Contact />} />
    </Route>
  </Routes>
</Suspense>
```

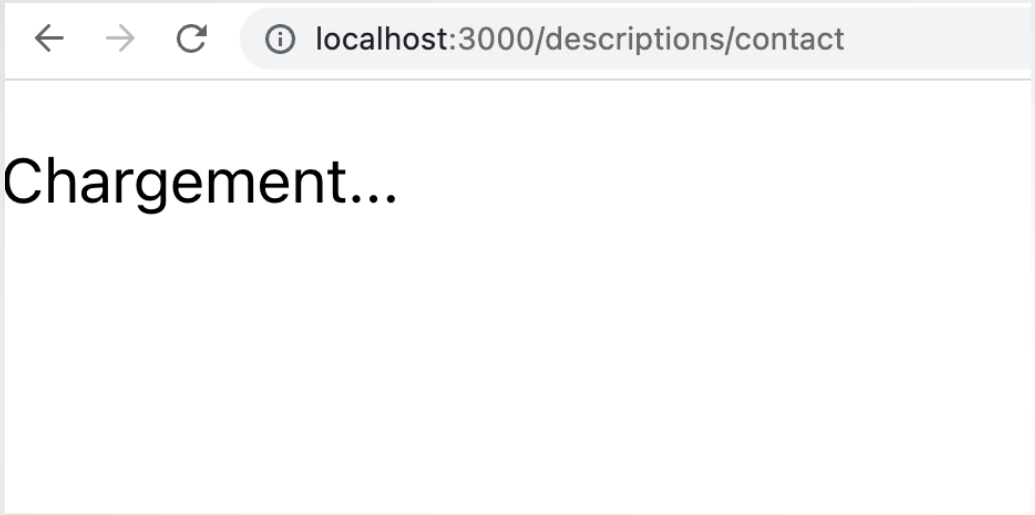
- Remarque : Pour le moment, React.lazy ne prend en charge que les exports par défaut.

Si le module que vous souhaitez importer utilise des exports nommés, vous pouvez créer un module intermédiaire qui ré-exportera le composant voulu en tant qu'export par défaut.

# Lazy Loading

## React.lazy

- Résultat de l'indicateur de chargement:



# Référence

https://reactrouter.com/en/main/upgrading/reach

https://fr.reactjs.org/docs/code-splitting.html#route-based-code-splitting