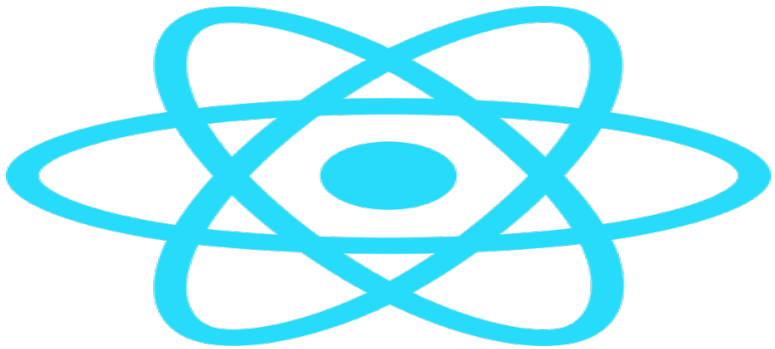
React : routage

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

Plan

React.js

Récapitulatif

- À la création d'un nouveau composant, on ajoute sa balise dans le template de `App.js` ou un de ses composants enfants
- Souvent, dans les application Web, on n'affiche que le·s composant·s demandé·s
- Une route demandée $\Rightarrow$ un composant affiché

React.js

Gestion de route

- Mapping : route/composant (élément)
- Une nouvelle balise pour les liens : pour ne pas recharger la page (aspect **SPA** de **React.js**)
- Une nouvelle balise pour spécifier l'emplacement dédié à l'affichage du composant demandé

React.js

Pour installer le module de routage

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

React.js

Pour installer le module de routage

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


Ensuite, on va

- créer un fichier `routes.js` dans un dossier `router` : fichier contenant le mapping route/composant
- modifier `index.js` pour utiliser le router
- créer un dossier `views` avec deux composants : `About.js` et `Home.js`
- modifier `App.js` : remplacement du code précédent avec un menu avec deux éléments pointant vers `About.js` et `Home.js`

React.js

Question

Quelle est la différence entre `views` et `components` ?

React.js

Question

Quelle est la différence entre `views` et `components` ?

Réponse

On recommande d'utiliser

- `views` (ou `pages`) pour les composants associés à au moins une route
- `components` pour les composants sans route ou aux composants enfants des composants définis dans `views`

React.js

Commençons par créer le composant `Home.js` dans `views` avec le contenu suivant

```
const Home = () => {
    return <h2>Home Page</h2>;
};

export default Home;
```

React.js

Commençons par créer le composant `Home.js` dans `views` avec le contenu suivant

```
const Home = () => {
    return <h2>Home Page</h2>;
};

export default Home;
```


Remarque

Déplaçons tout le contenu d'`App.js` dans `About.js`.

React.js

Contenu de `src/views/About.js`

```
import Hi from '../components/Hi/Hi';
import Hello from '../components/Hello/Hello';
import { useState } from 'react';

function About() {
    let [country, setCountry] = useState(null)
    function updateValue(value) {
        setCountry(value)
    }

    return (
        <div className="About">
            <Hi nom="Wick">Marseille</Hi>
            <Hello nom="Wick" onValeurChange={updateValue}>
              Paris</Hello>

        </div>
    );
}
```

React.js

Contenu de `src/router/routes.js`

```
import { Routes, Route } from "react-router-dom";
import Home from "../views/Home";
import Primeur from "../views/Primeur";
import Compteur from "../views/Compteur";
import About from "../views/About";

const AppRoutes = () => {
    return (
        <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
            <Route path="/primeur" element={<Primeur />} />
            <Route path="/compteur" element={<Compteur />} />
        </Routes>
    );
};

export default AppRoutes;
```

React.js

Explications

- `path` : chemin du composant.
- `element` : composant à afficher pour le chemin spécifié.

Nouveau contenu de `App.js`

```
import './App.css';
import AppRoutes from "./router/routes";
import { Link } from "react-router-dom";

function App() {

  return (
    <div className="App">

      {/* Le menu */}
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/compteur">Compteur</Link>
          </li>
          <li>
            <Link to="/primeur">Primeur</Link>
          </li>
        </ul>
      </nav>
      {/* Les routes */}
      <AppRoutes />
    </div>

  );
}

export default App;
```

React.js

Explications

- `<Link>` : similaire à la balise `a` en **HTML** mais ne recharge pas la page.
- `to`
  - équivalent de l'attribut `href` de la balise `a`,
  - prenant comme valeur une route définie dans `routes.js`.

React.js

Contenu de `index.js`

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { Provider } from './context/GlobalContext';
import { BrowserRouter as Router } from "react-router-dom";

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

reportWebVitals();
```

React.js

Objectif

- Déplacer le menu dans un nouveau composant `Menu.js` (à créer dans `components`)
- Déclarer `Menu` comme composant enfant de `App`

React.js

Créons le composant `Menu.js` dans `components` avec le contenu suivant

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

const Menu = () => {
    return (
        <nav>
            <ul>
                <li>
                    <Link to="/">Home</Link>
                </li>
                <li>
                    <Link to="/about">About</Link>
                </li>
                <li>
                    <Link to="/compteur">Compteur</Link>
                </li>
                <li>
                    <Link to="/primeur">Primeur</Link>
                </li>
            </ul>
        </nav>
    );
};

export default Menu;
```

React.js

Nouveau contenu d'`App.js`

```
import './App.css';
import AppRoutes from "./router/routes";
import Menu from "./components/Menu";

function App() {

  return (
    <div className="App">
      <Menu />
      <AppRoutes />
    </div>

  );
}

export default App;
```

React.js

Deux formes de paramètres de requête

- `/chemin/param1/param2`
- `/chemin?var1=value1&var2=value2`

React.js

Deux formes de paramètres de requête

- `/chemin/param1/param2`
- `/chemin?var1=value1&var2=value2`

Deux hooks pour la récupération de ces paramètres

- `useParams()` pour `/chemin/param1/param2`
- `useSearchParams()` pour `/chemin?var1=value1&var2=value2`

React.js

Pour la suite

Créons les trois composants suivants dans `views`

- `Personne.js`
- `PersonneDetails.js`
- `Adresse.js`

React.js

Commençons par définir une route pour le composant `PersonneDetail.js` dans le tableau `routes` de `index.js`

```
const AppRoutes = () => {
    return (
        <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
            <Route path="/primeur" element={<Primeur />} />
            <Route path="/compteur" element={<Compteur />} />
            <Route path="/personne" element={<Personne />} />
            <Route path="/personne/:id" element={<PersonneDetails />} />
        </Routes>
    );
};

export default AppRoutes;
```

React.js

Commençons par définir une route pour le composant `PersonneDetail.js` dans le tableau `routes` de `index.js`

```
const AppRoutes = () => {
    return (
        <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
            <Route path="/primeur" element={<Primeur />} />
            <Route path="/compteur" element={<Compteur />} />
            <Route path="/personne" element={<Personne />} />
            <Route path="/personne/:id" element={<PersonneDetails />} />
        </Routes>
    );
};

export default AppRoutes;
```


Explication

On utilise le préfixe `:` pour `id` pour le définir comme paramètre (et non pas comme un path)

React.js

Commençons par déclarer le tableau `personnes` dans `PersonneDetails.js`

```
export default function PersonneDetails() {
    const personnes = [
        { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
        { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
        { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
    ];
    return (
        <div>
            <h2>Détails de la personne</h2>

        </div>
    );
}
```

React.js

Commençons par déclarer le tableau `personnes` dans `PersonneDetails.js`

```
export default function PersonneDetails() {
    const personnes = [
        { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
        { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
        { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
    ];
    return (
        <div>
            <h2>Détails de la personne</h2>

        </div>
    );
}
```

Objectif

Récupérer l'`id` de la barre d'adresse et afficher la personne correspondante dans `template`.

React.js

Pour récupérer les paramètres d'une route de la forme `personne/:id/`

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


export default function PersonneDetails() {
    const personnes = [
        { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
        { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
        { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
    ];
    const { id } = useParams();

    return (
        <div>
            <h2>Détails de la personne : {id}</h2>

        </div>
    );
}
```

React.js

Pour afficher les détails de la personne dont l'`id` est passé en paramètre

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

export default function PersonneDetails() {
    const personnes = [
        { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
        { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
        { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
    ];

    const { id } = useParams();

    const personne = personnes.find(p => p.id === parseInt(id));


    return (
        <div>
            <h2>Détails de la personne : {id}</h2>
            {personne ? (
                <div>
                    <p>Nom: {personne.nom}</p>
                    <p>Prénom: {personne.prenom}</p>
                    <p>Âge: {personne.age}</p>
                </div>
            ) : (
                <p>Aucune personne trouvée pour l'dentifiant {id}</p>
            )}
        </div>
    );
}
```

React.js

Commençons par définir une route pour le composant `Adresse.js` dans le tableau `routes` de `routes.js`

```
const AppRoutes = () => {
    return (
        <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
            <Route path="/adresse" element={<Adresse />} />
            <Route path="/primeur" element={<Primeur />} />
            <Route path="/compteur" element={<Compteur />} />
            <Route path="/personne" element={<Personne />} />
            <Route path="/personne/:id" element={<PersonneDetails />} />
        </Routes>
    );
};

export default AppRoutes;
```

React.js

Commençons par définir une route pour le composant `Adresse.js` dans le tableau `routes` de `routes.js`

```
const AppRoutes = () => {
    return (
        <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
            <Route path="/adresse" element={<Adresse />} />
            <Route path="/primeur" element={<Primeur />} />
            <Route path="/compteur" element={<Compteur />} />
            <Route path="/personne" element={<Personne />} />
            <Route path="/personne/:id" element={<PersonneDetails />} />
        </Routes>
    );
};

export default AppRoutes;
```


Remarque

Pas besoin d'inclure les paramètres dans la définition de la route

React.js

Contenu d'`Adresse.js`
(URL de test : `http://localhost:3000/adresse?ville=Paris&codePostal=75000`)

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

export default function Adresse() {
    const location = useLocation();
    const queryParams = new URLSearchParams(location.search);

    const ville = queryParams.get("ville");
    const codePostal = queryParams.get("codePostal");
    return (
        <div>
            <h2>Adresse</h2>
            <ul>
                <li>Ville : {ville}</li>
                <li>Code postal : {codePostal}</li>
            </ul>
        </div>

    );
}
```

Pour récupérer et modifier les paramètres, on utilise le hook `useSearchParams`

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

export default function Adresse() {
    const [searchParams, setSearchParams] = useSearchParams();

    const ville = searchParams.get("ville");
    const codePostal = searchParams.get("codePostal");
    return (
        <div>
            <h2>Adresse</h2>
            <ul>
                <li>Ville : {ville}</li>
                <li>Code postal : {codePostal}</li>
            </ul>

            <button onClick={() => setSearchParams({ ville: "Gap" })}>
                Aller à Gap
            </button>
        </div>
    );
}
```

React.js

Exercice

- Créer un composant `Calcul` accessible via le chemin `calcul/:op`
- Les valeurs possibles de `op` sont `plus`, `moins`, `fois` et `div`
- Si l'adresse saisie dans la barre d'adresse contient `/calcul/plus?value1=2&value2=5`, la réponse attendue dans le template est `2 + 5 = 7`

React.js

Solution possible

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

export default function Calcul() {

    const { op } = useParams();
    const [searchParams] = useSearchParams();

    const value1 = searchParams.get("value1");
    const value2 = searchParams.get("value2");

    const map = {
        plus: '+',
        moins: '-',
        fois: '*',
        div: '/'
    }

    return (
        <div>
            <h2>Calcul</h2>
            <p>{value1} {map[op]} {value2} = {eval(`${value1} ${map[op]} ${value2}`)}</p>
        </div>
    );
}
```

React.js

Définir une route pour le composant `Personne` dans `routes.js`

```
<Route path="/personne" element={<Personne />} />
```

React.js

Définir une route pour le composant `Personne` dans `routes.js`

```
<Route path="/personne" element={<Personne />} />
```

Dans `Menu.js`, ajoutons un lien vers le composant `Personne`.

```
const Menu = () => {
    return (
        <nav>
            <Link to="/">Home</Link> |
            <Link to="/about">About</Link> |
            <Link to="/compteur">Compteur</Link> |
            <Link to="/primeur">Primeur</Link> |
            <Link to="/personne">Personne</Link>
        </nav>
    );
};
```

React.js

Dans `Personne.js`, définissons un lien avec paramètre vers `PersonneDetails`

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

export default function Personne() {
    const personnes = [
        { id: 1, nom: 'Wick', prenom: 'John', age: 45 },
        { id: 2, nom: 'Dalton', prenom: 'Jack', age: 40 },
        { id: 3, nom: 'Dupont', prenom: 'Sophie', age: 30 }
    ];
    return (
        <div>
            <ul>
                {
                    personnes.map((elt) =>
                        <li key={elt.id}>
                            <Link to={`/personne/${elt.id}`}>

                                {elt.nom} {elt.prenom}
                            </Link>
                        </li>
                    )
                }
            </ul>
        </div>
    );
}
```

React.js

Dans `About.js`, construisons un lien avec paramètres vers `Adresse`

```
<Link to={`/adresse?ville=${ville}&codePostal=${cp}`}>
    Visitez Montpellier
</Link>
```

React.js

Dans `About.js`**, construisons un lien avec paramètres vers** `Adresse`

```
<Link to={`/adresse?ville=${ville}&codePostal=${cp}`}>
    Visitez Montpellier
</Link>
```

Sans oublier de déclarer les constantes `ville` **et** `cp`

```
const ville = "Montpellier"
const cp = "34000"
```

React.js

Exercice

- Créez un nouveau composant `Tableau`
- Déclarez le tableau `const numbers = [2, 3, 8, 1]`
- Définissez une route `tableau/:id`
  `id` étant l'indice de l'élément dans `numbers` à afficher dans le template
- Ajoutez deux liens `Suivant` et `Précédent` qui permettent de naviguer respectivement sur l'élément suivant et précédent de `numbers`
- Les deux liens `Suivant` et `Précédent` doivent permettre une navigation circulaire

React.js

Solution possible

```
import { Link, useParams } from "react-router-dom";

export default function Tableau() {

    const numbers = [2, 3, 8, 1]
    const { id } = useParams();

    const precedent = (Number(id) - 1 + numbers.length) % numbers.length
    const suivant = (Number(id) + 1) % numbers.length

    return (
        <div>
            <h2>Tableau : {numbers}</h2>
            <Link to={`/tableau/${precedent}`}>
                Précédent
            </Link>
            <p>{numbers[id]}</p>
            <Link to={`/tableau/${suivant}`}>
                Suivant
            </Link>
        </div>
    );
}
```

React.js

Ajoutons le bouton suivant dans le `render` de `About.js`

```
<button onClick={gotoPersonne}>
    Consultez la fiche de Sophie
</button>
```

React.js

Ajoutons le bouton suivant dans le `render` de `About.js`

```
<button onClick={gotoPersonne}>
    Consultez la fiche de Sophie
</button>
```


Objectif

- En cliquant sur le bouton, la fonction `gotoPersonne()` sera exécutée.
- `gotoPersonne()` doit nous rediriger vers le composant `PersonneDetails`.

React.js

Pour rediriger vers le composant `PersonneDetails`, on utilise lae hook `useNavigate`

```
const navigate = useNavigate();
```

React.js

Pour rediriger vers le composant `PersonneDetails`, on utilise lae hook `useNavigate`

```
const navigate = useNavigate();
```

On utilise aussi passer comme paramètre un objet spécifiant le `path`

```
const gotoPersonne = () => {
    navigate('/personne/3')
}
```

React.js

Pour rediriger vers le composant `PersonneDetails`, on utilise lae hook `useNavigate`

```
const navigate = useNavigate();
```

On utilise aussi passer comme paramètre un objet spécifiant le `path`

```
const gotoPersonne = () => {
    navigate('/personne/3')
}
```


On peut aussi passer un objet

```
const gotoPersonne = () => {
    navigate({ pathname: '/personne/3' })
}
```

React.js

Pour naviguer vers la page précédente de l'historique

```
<button onClick={() => navigate(-1)}>
    Page précédente
</button>
```

React.js

Pour naviguer vers la page précédente de l'historique

```
<button onClick={() => navigate(-1)}>
    Page précédente
</button>
```

Pour naviguer vers la page suivante de l'historique

```
<button onClick={() => navigate(1)}>
    Page suivante
</button>
```

React.js

Exercice (à refaire avec la redirection)

- Reprenez le composant `Tableau`.
- Ajoutez deux boutons `Suivant` et `Précédent` qui permettent de naviguer respectivement sur l'élément suivant et précédent de `numbers`.
- Les deux boutons `Suivant` et `Précédent` doivent permettre une navigation circulaire.

React.js

On peut rediriger vers un chemin existant

```
<Route path="/personne" element={<Personne />} />
<Route path="/person" element={<Navigate to="/
   personne" />} />
```

React.js

On peut rediriger vers un chemin existant

```
<Route path="/personne" element={<Personne />} />
<Route path="/person" element={<Navigate to="/
   personne" />} />
```


Explication

En demandant la route `/person` :

- `person` sera remplacé par `personne` dans la barre d'adresse,
- Le composant `Personne` sera affiché.

React.js

Redirection vs alias

- La redirection remplace le chemin demandé par un autre auquel un composant est associé.
- L'alias permet d'associer plusieurs chemin à un même composant sans que l'un remplace l'autre dans la barre d'adresse.

React.js

Exemple avec deux alias

```
<Route path="/about" element={<About />} />
<Route path="/about-us" element={<About />} />
```

React.js

Exemple avec deux alias

```
<Route path="/about" element={<About />} />
<Route path="/about-us" element={<About />} />
```


Explication

- Les alias `/about` et `/about-us` pointent tous les deux vers le composant `About`.
- Cela signifie que l'un ou l'autre affichera le même contenu.

React.js

Objectif

Afficher en gras ou italic l'élément actif du menu.

React.js

Dans `Menu.js`**, remplaçons** `Link` **par** `NavLink`

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

const Menu = () => {
    return (
        <nav>
            <NavLink to="/">Home</NavLink> |
            <NavLink to="/about">About</NavLink> |
            <NavLink to="/compteur">Compteur</NavLink> |
            <NavLink to="/primeur">Primeur</NavLink> |
            <NavLink to="/personne">Personne</NavLink>
        </nav>
    );
};

export default Menu;
```

React.js

Dans `Menu.js`**, remplaçons** `Link` **par** `NavLink`

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

const Menu = () => {
    return (
        <nav>
            <NavLink to="/">Home</NavLink> |
            <NavLink to="/about">About</NavLink> |
            <NavLink to="/compteur">Compteur</NavLink> |
            <NavLink to="/primeur">Primeur</NavLink> |
            <NavLink to="/personne">Personne</NavLink>
        </nav>
    );
};

export default Menu;
```

Vérifiez dans le **DevtTools** du navigateur que l'élément actif du menu a la classe `active`.

React.js

Définissons la classe `active` **dans** `Menu.css`

```
a.active {
  font-weight: bold;
}
```

React.js

Dans `Menu.js`**, importons** `Link` `Menu.css`

```
import { NavLink } from "react-router-dom";
import './Menu.css';

const Menu = () => {
    return (
        <nav>
            <NavLink to="/">Home</NavLink> |
            <NavLink to="/about">About</NavLink> |
            <NavLink to="/compteur">Compteur</NavLink> |
            <NavLink to="/primeur">Primeur</NavLink> |
            <NavLink to="/personne">Personne</NavLink>
        </nav>
    );
};

export default Menu;
```

React.js

Dans `Menu.js`**, importons** `Link` `Menu.css`

```
import { NavLink } from "react-router-dom";
import './Menu.css';

const Menu = () => {
    return (
        <nav>
            <NavLink to="/">Home</NavLink> |
            <NavLink to="/about">About</NavLink> |
            <NavLink to="/compteur">Compteur</NavLink> |
            <NavLink to="/primeur">Primeur</NavLink> |
            <NavLink to="/personne">Personne</NavLink>
        </nav>
    );
};

export default Menu;
```

Vérifiez que l'élément actif du menu s'affiche en gras.

React.js

Pour un lien avec paramètre, on utilise `className`

```
const Menu = () => {

    return (
        <nav>
            <NavLink to="/">Home</NavLink> |
            <NavLink to="/about">About</NavLink> |
            <NavLink to="/compteur">Compteur</NavLink> |
            <NavLink to="/personne">Personne</NavLink> |
            <NavLink to="/primeur">Primeur</NavLink> |
            <NavLink
                to="/tableau/0"
                className={() =>
                    window.location.pathname.startsWith('/tableau') ? 'active' : undefined
                }
            >
                Tableau
            </NavLink>
        </nav>
    );
};
```

React.js

Pour un lien avec paramètre, on utilise `className`

```
const Menu = () => {

    return (
        <nav>
            <NavLink to="/">Home</NavLink> |
            <NavLink to="/about">About</NavLink> |
            <NavLink to="/compteur">Compteur</NavLink> |
            <NavLink to="/personne">Personne</NavLink> |
            <NavLink to="/primeur">Primeur</NavLink> |
            <NavLink
                to="/tableau/0"
                className={() =>
                    window.location.pathname.startsWith('/tableau') ? 'active' : undefined
                }
            >
                Tableau
            </NavLink>
        </nav>
    );
};
```

Vérifiez que le lien `Tableau` du menu s'affiche en gras quelle que soit la valeur du paramètre.

React.js

Créons un composant `NotFound.js` avec le contenu suivant

```
const NotFound = () => {
    return <h2>404 - Page Not Found</h2>;
};

export default NotFound;
```

React.js

Créons un composant `NotFound.js` avec le contenu suivant

```
const NotFound = () => {
    return <h2>404 - Page Not Found</h2>;
};

export default NotFound;
```

Ajoutons la route suivante dans `routes.js` à la dernière position

```
<Route path="*" element={<NotFound />} />
```

React.js

Créons un composant `NotFound.js` avec le contenu suivant

```
const NotFound = () => {
    return <h2>404 - Page Not Found</h2>;
};

export default NotFound;
```

Ajoutons la route suivante dans `routes.js` à la dernière position

```
<Route path="*" element={<NotFound />} />
```

Allez à `http://localhost:3000/x` et vérifiez le rendu suivant :
404 - Page Not Found

React.js

Pour afficher la route demandée dans le message d'erreur, on utilise le hook `useLocation`

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

const NotFound = () => {
    const location = useLocation();

    return (
        <div>
            <h1>404 - Page Not Found</h1>
            <p>The page "{location.pathname}" does not exist.</p>
        </div>
    );
};

export default NotFound;
```

React.js

Pour gérer les métadonnées d'une application (titre...), deux solutions possibles

- `react-helmet`,
- `react-helmet-async` pour une version compatible avec le rendu côté serveur (**SSR**).

React.js

Installons le module `react-helmet`

```
npm install react-helmet
```

React.js

Installons le module `react-helmet`

```
npm install react-helmet
```

Utilisons ensuite `Helmet` **pour modifier le titre de la page (**`NotFound.js`**)**

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

const NotFound = () => {
    const location = useLocation();

    return (
        <div>
            <Helmet>
                <title>Page 404</title>
            </Helmet>
            <h1>404 - Page Not Found</h1>
            <p>The page "{location.pathname}" does not exist.</p>
        </div>
    );
};

export default NotFound;
```

React.js

Pour tester

- visitez la route `/about` et vérifiez que le `title` affiché est `React App`
- visitez la route `/x` et vérifiez que le `title` affiché est `Page 404`

React.js

Désinstallons le module `react-helmet`

```
npm install react-helmet
```

React.js

Désinstallons le module `react-helmet`

```
npm install react-helmet
```

Et installons le module `react-helmet-async`

```
npm install react-helmet-async
```

React.js

Enveloppons notre application dans un `HelmetProvider`

```
const AppRoutes = () => {
    return (
        <HelmetProvider>
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/about" element={<About />} />
                <Route path="/about-us" element={<About />} />
                <Route path="/adresse" element={<Adresse />} />
                <Route path="/primeur" element={<Primeur />} />
                <Route path="/compteur" element={<Compteur />} />
                <Route path="/tableau/:id" element={<Tableau />} />
                <Route path="/calcul/:op" element={<Calcul />} />
                <Route path="/personne" element={<Personne />} />
                <Route path="/person" element={<Navigate to="/personne" />} />
                <Route path="/personne/:id" element={<PersonneDetails />} />
                <Route path="*" element={<NotFound />} />
            </Routes>
        </HelmetProvider>
    );
};

export default AppRoutes;
```

React.js

Rien à changer dans `NotFound.js`

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

const NotFound = () => {
    const location = useLocation();

    return (
        <div>
            <Helmet>
                <title>Page 404</title>
            </Helmet>
            <h1>404 - Page Not Found</h1>
            <p>The page "{location.pathname}" does not exist.</p>
        </div>
    );
};

export default NotFound;
```

React.js

Pour tester

- visitez la route `/about` et vérifiez que le `title` affiché est `React App`
- visitez la route `/x` et vérifiez que le `title` affiché est `Page 404`

React.js

`react-helmet`

- API intuitive et bien documentée
- Facile à installer et à utiliser pour des applications simples
- Moins efficace pour les applications à grande échelle
- Peut entraîner des problèmes de mémoire et des comportements imprévisibles avec **SSR**

`react-helmet-async`

- Optimisé pour les applications avec **SSR**
- Plus performant pour les applications complexes et à grande échelle
- Nécessite une légère configuration supplémentaire (envelopper l'application dans `HelmetProvider`)
- Moins intuitif pour les développeurs habitués à `react-helmet` sans contexte asynchrone

React.js

Lazy loading ou **On-demand loading**

- Chargement du composant à la demande (pas au démarrage de l'application)
- Utilise l'import avec les promesses (**ES2020**)

React.js

Pour commencer

- Allez dans le **DevTools** du navigateur, cliquez sur l'onglet `Réseau` et notez le nombre de requêtes, le nombre d'octets transférés et le temps de chargement (tout se trouve en bas du **DevTools**).
- Remplacez tous les imports statiques par des imports utilisant les promesses et refaites l'opération précédente (item 1).

Dans `routes.js`, utilisons les promesses pour réaliser le lazy-loading

```
import { lazy, Suspense } from "react";

const About = lazy(() => import('../views/About'));
const Adresse = lazy(() => import('../views/Adresse'));
const Personne = lazy(() => import('../views/Personne'));
const Primeur = lazy(() => import('../views/Primeur'));
const Compteur = lazy(() => import('../views/Compteur'));
const NotFound = lazy(() => import('../views/NotFound'));

const AppRoutes = () => {
    return (
        <Suspense fallback={<div>Loading...</div>}>
            <HelmetProvider>
                <Routes>
                    <Route path="/" element={<Home />} />
                    <Route path="/about" element={<About />} />
                    <Route path="/about-us" element={<About />} />
                    <Route path="/adresse" element={<Adresse />} />
                    <Route path="/primeur" element={<Primeur />} />
                    <Route path="/compteur" element={<Compteur />} />
                    <Route path="/personne" element={<Personne />} />
                    <Route path="/person" element={<Navigate to="/personne" />} />
                    <Route path="/personne/:id" element={<PersonneDetails />} />
                    <Route path="*" element={<NotFound />} />
                </Routes>
            </HelmetProvider>
        </Suspense>
    );
};
export default AppRoutes;
```

React.js

Explication

- La fonction `lazy()` est utilisée pour importer des composants de manière dynamique. Elle accepte une callback qui utilise import() pour charger le module.
- Le composant `<Suspense>` est utilisé pour entourer les routes qui utilisent des composants chargés de manière paresseuse.
- Il accepte une `prop` (`fallback`) qui définit ce qui doit être affiché pendant le chargement des composants (comme un indicateur de chargement).

React.js

Explication

- La fonction `lazy()` est utilisée pour importer des composants de manière dynamique. Elle accepte une callback qui utilise import() pour charger le module.
- Le composant `<Suspense>` est utilisé pour entourer les routes qui utilisent des composants chargés de manière paresseuse.
- Il accepte une `prop` (`fallback`) qui définit ce qui doit être affiché pendant le chargement des composants (comme un indicateur de chargement).

Pour tester, pensez à vider le cache et à effectuer une actualisation forcée.

React.js

Lazy loading : avantages

- **Amélioration des performances** : En retardant le chargement de ressources non essentielles, on réduit le temps de chargement initial de la page.
- **Optimisation des ressources** : En ne chargeant que les ressources nécessaires au fur et à mesure que l'utilisateur fait défiler la page ou interagit avec elle, on optimise l'utilisation des ressources système, notamment la puissance du processeur et la mémoire.
- **Économie de bande passante** : En retardant le chargement des ressources non essentielles, cela peut être particulièrement bénéfique pour les utilisateurs sur des connexions Internet lentes ou limitées.
- ...

React.js

Lazy loading : inconvénients

- **Complexité du code** : Gérer le chargement asynchrone des ressources nécessite une écriture de code plus détaillée et peut être difficile à comprendre.
- **Problème de référencement et de classement** : Les moteurs de recherche peuvent avoir des difficultés à indexer correctement le contenu chargé de manière asynchrone car les robots d'exploration peuvent ne pas attendre le chargement des ressources différées.
- **Compatibilité du navigateur** : Bien que la plupart des navigateurs modernes prennent en charge le lazy loading, des problèmes de compatibilité peuvent survenir avec des versions plus anciennes ou moins courantes.
- ...

React.js

Objectif

- Mieux structurer l'application avec les routes imbriquées
- Particulièrement utiles pour les pages avec des sous-sections ou des sous-pages qui doivent être rendues dans le même layout ou composant parent.
- Quelques cas d'utilisation : navigation en étapes, tableaux de bord...

React.js

Objectif

- Mieux structurer l'application avec les routes imbriquées
- Particulièrement utiles pour les pages avec des sous-sections ou des sous-pages qui doivent être rendues dans le même layout ou composant parent.
- Quelques cas d'utilisation : navigation en étapes, tableaux de bord...

Démarche

- Créer trois composants : `Vehicule`, `Voiture` et `Camion`.
- Définir les routes `/vehicule`, `/vehicule/voiture` et `/vehicule/camion`.
- Utiliser `Outlet` dans `Vehicule` pour indiquer l'emplacement des composants associées aux routes imbriquées.

React.js

Contenu de `Camion.js`

```
const Camion = () => {
    return <h2>Camion</h2>;
};

export default Camion;
```

React.js

Contenu de `Camion.js`

```
const Camion = () => {
    return <h2>Camion</h2>;
};

export default Camion;
```


Contenu de `Voiture.js`

```
const Voiture = () => {
    return <h2>Voiture</h2>;
};

export default Voiture;
```

React.js

Contenu de `Vehicule.js`

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

const Vehicule = () => {
    return (
        <div>
            <h2>Vehicule</h2>
            <Outlet />
        </div>
    );
};

export default Vehicule;
```

React.js

Définissons les routes imbriquées dans `routes.js`

```
const AppRoutes = () => {
    return (
        <Suspense fallback={<div>Loading...</div>}>
            <HelmetProvider>
                <Routes>
                    <Route path="/" element={<Home />} />
                    <Route path="/about" element={<About />} />
                    <Route path="/about-us" element={<About />} />
                    <Route path="/vehicule" element={<Vehicule />} >
                        <Route path="voiture" element={<Voiture />} />
                        <Route path="camion" element={<Camion />} />
                    </Route>
                    <Route path="/adresse" element={<Adresse />} />
                    <Route path="/primeur" element={<Primeur />} />
                    <Route path="/compteur" element={<Compteur />} />
                    <Route path="/tableau/:id" element={<Tableau />} />
                    <Route path="/calcul/:op" element={<Calcul />} />
                    <Route path="/personne" element={<Personne />} />
                    <Route path="/person" element={<Navigate to="/personne" />} />
                    <Route path="/personne/:id" element={<PersonneDetails />} />
                    <Route path="*" element={<NotFound />} />
                </Routes>
            </HelmetProvider>
        </Suspense>
    );
};

export default AppRoutes;
```

React.js

Remarque

Ne préfixez pas les routes imbriquées par `/`.