Marvel - Version 0.4.0

Alexandre GIRARD - Conseiller en Nouvelles Technologies - alexandre.girard@maif.fr

Gestion de la navigation dans l'application

- Installation et configuration de **React Router**, librairie tierce permettant de gérer la navigation
- Mise en place de la navigation dans l'application Marvel
  - définir les routes
  - définir les composants à afficher en fonction de la route
- Définir le layout de l'application
  - header, footer, menu, sidebar, contenu principal...

git

A vous de faire le nécessaire pour travailler correctement avec git et générer la version 0.4.0 de l'application.

- Rappel:
  - On ne travaille pas directement sur la branche principale.
  - On utilise des branches de fonctionnalités pour travailler sur des fonctionnalités spécifiques.
  - On commit régulièrement de manière atomique et avec des messages de commit explicites.
  - On ne commit pas du code rendant l'application inutilisable.
  - On ne merge pas une branche sans avoir testé le code.

git

Nous verrons plus tard comment sécuriser nos branches de développement et de production. Pour l'instant c'est à nous de faire attention à ne pas commettre d'erreurs.

Création des pages de l'application

- Création des pages de l'application
  - Home
  - Characters
  - Contact
  - About

Création des pages de l'application (suite)

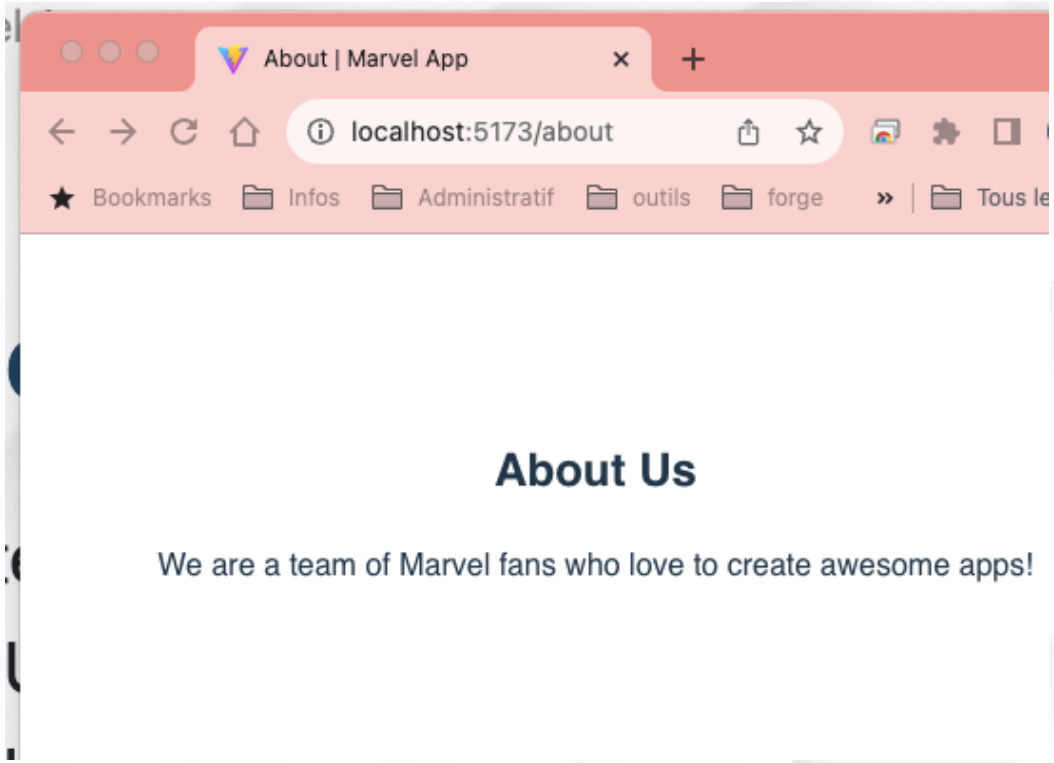
- Les pages sont des composants React
  - définis dans le dossier `src/pages`
  - importés dans le composant principal `App`

Page About

Créez le composant `AboutPage` dans le dossier `src/pages` et ajoutez le contenu suivant:

- Utilisation de **document.title** permettant de redéfinir le titre de la page
- Un en-tête **h2**
- Un paragraphe

Pour tester cette page, modifier le fichier App.jsx pour utiliser le composant **AboutPage** (et supprimer l'affichage actuel).

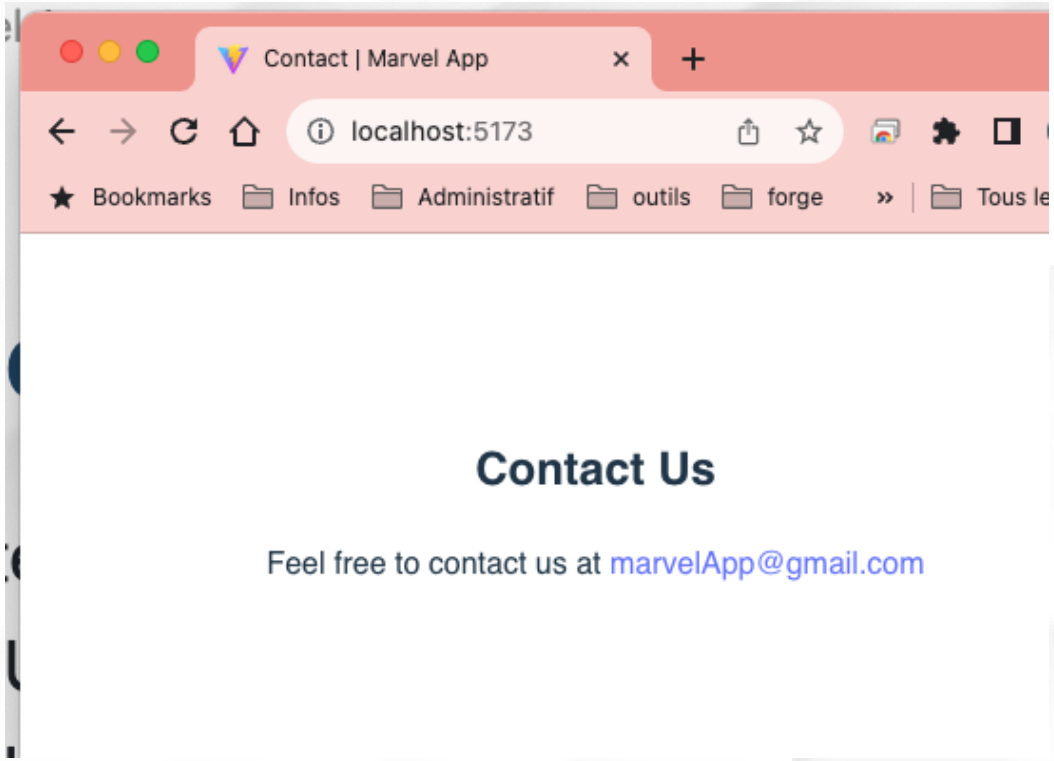


Page Contact

Créez le composant `ContactPage` dans le dossier `src/pages` et ajoutez le contenu suivant:

- Utilisation de **document.title** permettant de redéfinir le titre de la page
- Un en-tête **h2**
- Un message de contact avec un lien de type **mailto**

Pour tester cette page, modifier le fichier App.jsx pour afficher la page **Contact**.

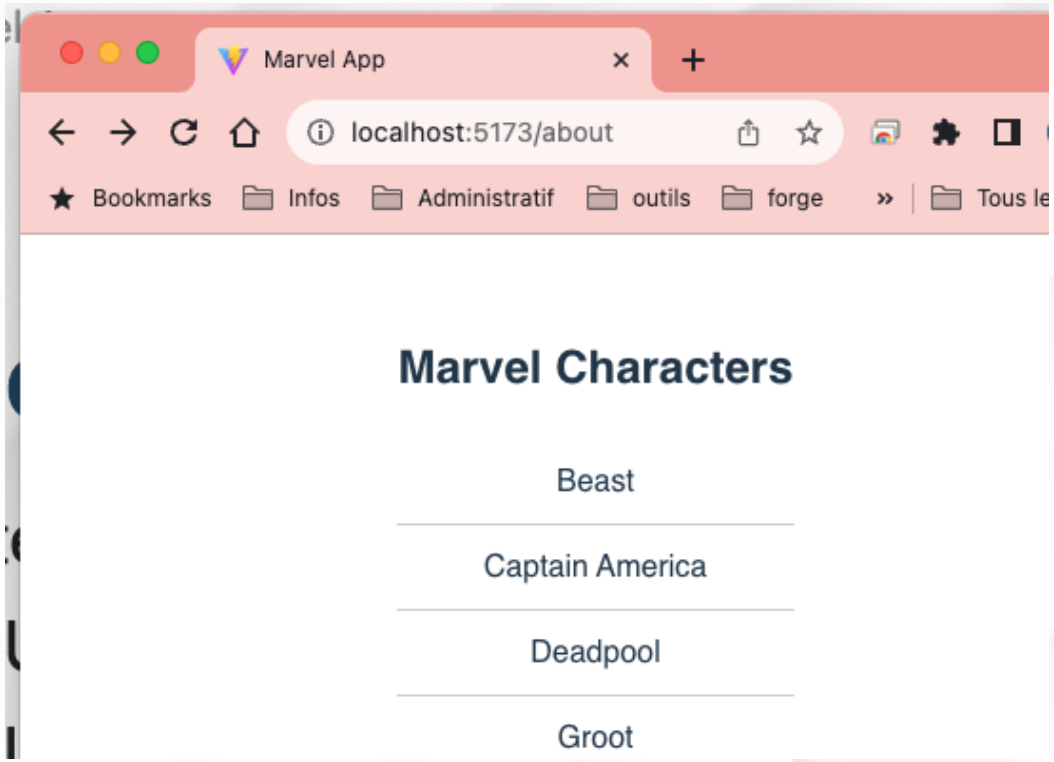


Page Characters

Créez le composant `CharactersPage` dans le dossier `src/pages` et ajoutez le contenu suivant:

- Utilisation de **document.title** permettant de redéfinir le titre de la page
- Le contenu qui était affiché sur la page d'accueil, attention au chemin d'accès aux imports

Pour tester cette page, modifier le fichier App.jsx pour afficher la page **Characters**.



Page Home

Nous n'avons pas de contenu spécifique pour la page Home, nous afficherons le contenu de la page Characters. Nous traiterons la page Home plus tard.

Problématique - Gestions des routes

- Comment gérer la navigation dans une application React ?
- Comment définir les routes de l'application ?
- Comment afficher les composants en fonction de la route ?

React Router

React Router est une librairie tierce permettant de gérer la navigation dans une application React, voir React Router

Son objectif est de permettre une navigation fluide et dynamique entre les différentes pages de l'application sans recharger la page.

Installation de la librairie:

```
npm install react-router
```

React Router - layout

React Router permet de définir un layout pour l'application, c'est-à-dire les éléments qui seront affichés sur toutes les pages de l'application et ainsi ne pas dupliquer le code dans chaque page.

La partie principale de l'application sera affichée grâce à la propriété `children` du composant.

Le layout de l'application est généralement composé de trois parties :

- **Header** : en-tête de l'application
  - navigation entre les pages
- **Main** : contenu principal de l'application
- **Footer** : pied de page de l'application

```
const Layout = ({ children }) => {
    return (
        <>
            <header>
                <h1>Marvel App</h1>
                <nav>
                    <a href="/">Home</a>
                    <a href="/about">About</a>
                    <a href="/contact">Contact</a>
                </nav>
            </header>
            <main>
                {children}
            </main>
            <footer>
                <p>Marvel App - 2025</p>
            </footer>
        </>
    );
};

export default Layout;
```

React Router - routes

Il est ensuite nécessaire de définir les routes de l'application, c'est-à-dire les URL qui permettront d'accéder aux différentes pages de l'application.

Dans un premier temps, nous allons définir les routes directement dans le fichier `App.jsx`.

Les composants **BrowserRouter**, **Routes** et **Route** de **React Router** permettent de définir les routes de l'application.

```
import './App.css'
import { BrowserRouter, Route, Routes } from "react-router";
import AboutPage from './pages/AboutPage';
import CharactersPage from './pages/CharactersPage';
import ContactPage from './pages/ContactPage';
import Layout from './Layout';

function App() {
  return (
    <>
      <Layout>
        <BrowserRouter>
          <Routes>
            <Route path="/" element={<CharactersPage />} />
            <Route path="/about" element={<AboutPage />} />
            <Route path="/contact" element={<ContactPage />} />
          </Routes>
        </BrowserRouter>
      </Layout>
    </>
  );
}

export default App;
```

React Router - routes

Le problème de cette approche est que le composant `App` risque de devenir rapidement très complexe si l'on ajoute de nombreuses routes.

On mélange de plus des notions de navigation avec la structure de l'application.

React Router propose une autre approche plus simple et plus propre pour définir les routes de l'application grâce au composant `RouterProvider` et à la fonction `createBrowserRouter`.

React Router - routes

RouterProvider est un composant qui permet de fournir le routeur à l'application, il doit être placé au plus haut niveau de l'application.

Il va gérer la navigation entre les différentes pages de l'application et afficher le composant correspondant à la route.

Il s'appuie sur un routeur créé avec la fonction `createBrowserRouter`. Nous pouvons définir les routes dans un fichier séparé (`routes.js`) pour ne pas alourdir le composant `App`.

```
import './App.css'
import { createBrowserRouter, RouterProvider } from "react-router";
import routes from './routes';

// router to navigate through the app
const router = createBrowserRouter(routes);

function App() {
  return (
    <>
      <RouterProvider router={router} />
    </>
  );
}

export default App;
```

```
import AboutPage from './pages/AboutPage';
import CharactersPage from './pages/CharactersPage';
import ContactPage from './pages/ContactPage';
import Layout from './Layout';

// routes of the application
const routes = [
  {
    path: "/",
    Component: Layout,
    children: [
      {
        // main page
        index: true,
        Component: CharactersPage
      },
      {
        // about page
        path: "/about",
        Component: AboutPage
      },
      {
        // contact page
        path: "/contact",
        Component: ContactPage
      },
    ],
  },
]

export default routes;
```

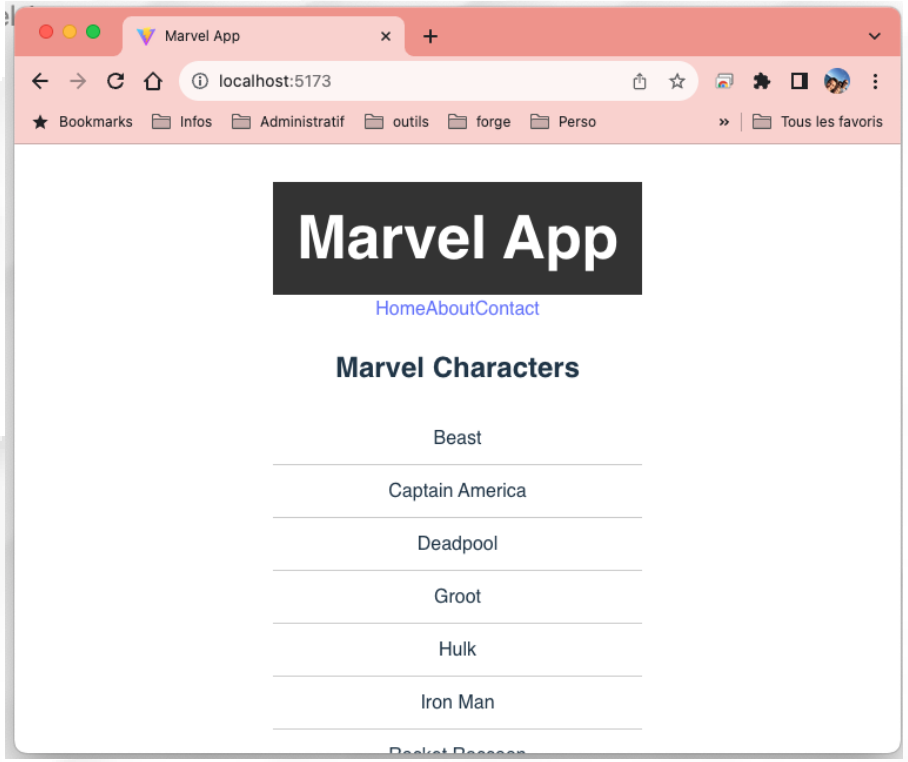
React Router - routes (suite)

Nous pouvons maintenant accèder à chaque page de l'application en fonction de l'URL.

Nous n'avons pas encore géré le cas d'une page inexistante, nous verrons cela plus tard.

React Router

Nous avons maintenant une application avec une gestion de la navigation grâce à **React Router**.



React Router - NavLinks

- Nous avons utilisé des balises `a` pour les liens de navigation, ce qui n'est pas optimal, on perd un des aspects de **React Router** dans l'optimisation de la navigation et on recharge la page à chaque clic.
- Nous allons remplacer ces balises par des composants `NavLink` de **React Router**
  - Permet de gérer la navigation sans recharger la page
  - Permet de gérer les classes CSS pour les liens actifs

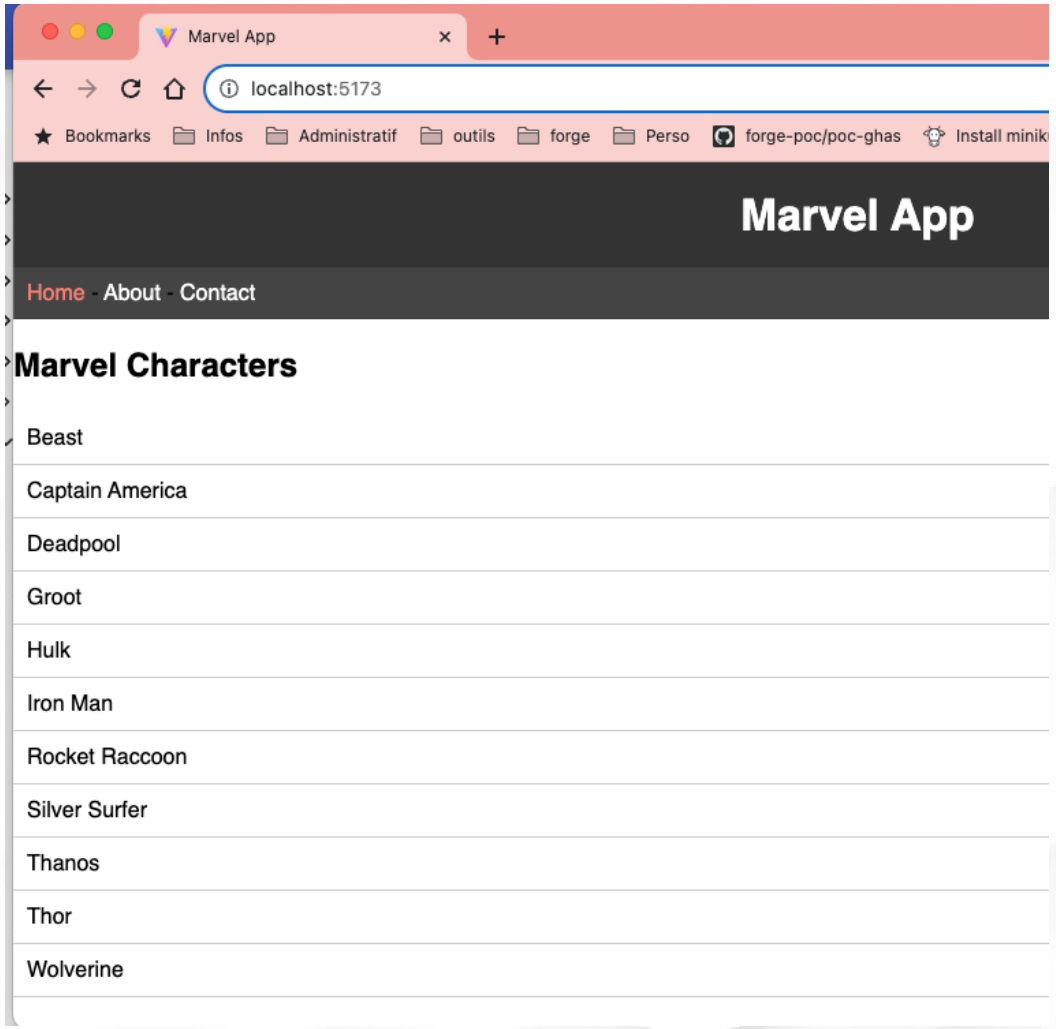
```
import { NavLink, Outlet } from "react-router";

const Layout = () => {
    return (
        <>
            <header>
                <h1>Marvel App</h1>
                <nav>
                    <NavLink to="/">Home</NavLink>-
                    <NavLink to="/about">About</NavLink>-
                    <NavLink to="/contact">Contact</NavLink>-
                </nav>
            </header>
            <main>
                <Outlet />
            </main>
            <footer>
                <p>Marvel App - 2025</p>
            </footer>
        </>
    );
};

export default Layout;
```

Ajout de css pour améliorer l'affichage

- Remplacer le contenu du fichier `App.css` afin d'améliorer l'affichage de l'application
- Télécharger le fichier App.css



React Router - gestion des pages inexistantes

Nous allons maintenant gérer le cas d'une page inexistante, c'est-à-dire une URL qui ne correspond à aucune route définie dans l'application.

Pour cela, nous allons créer une page `NotFoundPage` qui sera affichée lorsque l'utilisateur accède à une URL non définie.

```
const NotFoundPage = () => {
    // change the title of the page
    document.title = "404 - Page Not Found";

    return (
        <div>
            <h2>404 - Page Not Found</h2>
            <p>The page you are looking for does not exist.</p>
        </div>
    );
};

export default NotFoundPage;
```

```
import AboutPage from './pages/AboutPage';
import CharactersPage from './pages/CharactersPage';
import ContactPage from './pages/ContactPage';
import Layout from './Layout';
import NotFoundPage from './pages/NotFoundPage';

// routes of the application
const routes = [
  {
    path: "/",
    Component: Layout,
    children: [
      {
        // main page
        index: true,
        Component: CharactersPage
      },
      {
        // about page
        path: "/about",
        Component: AboutPage
      },
      {
        // contact page
        path: "/contact",
        Component: ContactPage
      },
      {
        // 404 page
        path: "*",
        Component: NotFoundPage
      }
    ],
  },
]

export default routes;
```

Fin de la version 0.4.0

- Nous avons ajouté la gestion de la navigation dans l'application
- Nous avons créé les pages de l'application
- Nous avons créé le layout de l'application
- Nous avons utilisé **React Router** pour gérer la navigation
- Nous avons utilisé les composants `NavLink` pour gérer la navigation sans recharger la page

Fin de la version 0.4.0 (suite)

L'application est maintenant fonctionnelle, nous avons ajouté les fonctionnalités de base pour une application web permettant de naviguer entre les différentes pages de l'application de manière fluide.

Il est maintenant temps de générer une nouvelle version de l'application et de l'emmener jusqu'à la production.

Faites le nécessaire pour générer la version 0.4.0 de l'application.

git - Etat final