

# Cours Inf3522 - Développement d'Applications N tiers

## Lab 6 : Démarrer avec React

Ce lab décrit l'environnement de développement et les outils nécessaires pour React, qui sont nécessaires pour commencer le développement frontend. Nous allons créer une application React simple, d'abord en téléchargeant la librairie afin de travailler en local, puis en utilisant l'outil create-react-app, développé par Facebook.

### Exercice 1

Tout d'abord, vous devez obtenir une copie de la bibliothèque React. Il existe différentes façons de le faire. Optons pour la plus simple qui ne nécessite aucun outil spécial et qui vous permettra d'apprendre et de coder rapidement.

Créez un dossier pour tout le code du cours à un emplacement où vous pourrez le retrouver et créez un sous dossier react pour séparer le code de la bibliothèque React.

```
mkdir ~/front_tiers
```

```
mkdir ~/front_tiers/react
```

Ensuite, vous devez ajouter trois fichiers : le premier est React lui-même, le deuxième est l'extension ReactDOM et le troisième est la bibliothèque Babel qui traduit le JavaScript actuel (et le JSX) en JavaScript classique compatible avec les anciens navigateurs. Vous pouvez récupérer les dernières versions des trois depuis l'hébergeur unpkg.com, comme ceci (assurez-vous de vous positionner dans le sous dossier react de front\_tiers :

```
curl -L https://unpkg.com/react/umd/react.development.js > react.js
```

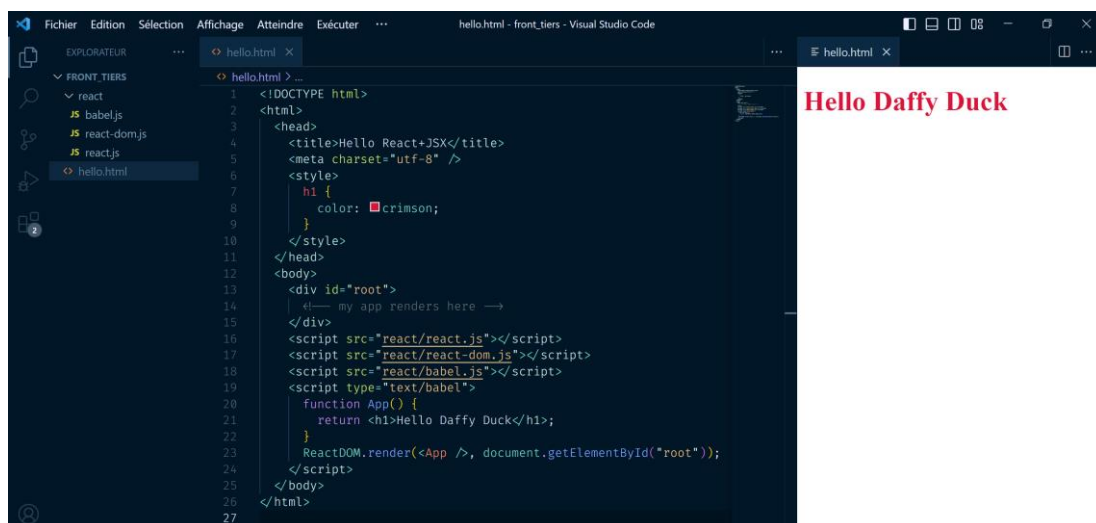
```
curl -L https://unpkg.com/react-dom/umd/react-dom.development.js > react-dom.js
```

```
curl -L https://unpkg.com/babel-standalone/babel.min.js > babel.js
```

Notez que React n'impose aucune structure de répertoire ; vous êtes libre de vous déplacer vers un répertoire différent ou de renommer react.js comme bon vous semble.

Vous n'êtes pas obligé de télécharger les bibliothèques, vous pouvez les utiliser directement depuis unpkg.com, mais les avoir localement permet de pouvoir apprendre n'importe où et sans avoir besoin d'une connexion internet.

Commençons par une page simple dans votre répertoire de travail (~/front\_tiers/hello.html) :



```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Hello React+JSX</title>
5     <meta charset="utf-8" />
6     <style>
7       h1 {
8         color: crimson;
9       }
10    </style>
11  </head>
12  <body>
13    <div id="root">
14      <!-- my app renders here -->
15    </div>
16    <script src="react/react.js"></script>
17    <script src="react/react-dom.js"></script>
18    <script src="react/babel.js"></script>
19    <script type="text/babel">
20      function App() {
21        return <h1>Hello Daffy Duck</h1>;
22      }
23      ReactDOM.render(<App />, document.getElementById("root"));
24    </script>
25  </body>
26 </html>
27
```

## Exercice 2

Pour tout développement sérieux et déploiement, en dehors du prototypage ou des tests JSX, vous devez mettre en place un processus de construction (build process).

La communauté JavaScript et son écosystème offrent de nombreuses options en matière de développement et de processus de construction. L'une des approches les plus simples et courantes est d'utiliser l'utilitaire Create-React-App (CRA), donc allons avec cela.

### Create-React-App

Create-React-App (CRA) est un ensemble de scripts Node.js et de leurs dépendances qui vous facilitent la tâche en mettant en place tout ce dont vous avez besoin pour démarrer rapidement. Tout d'abord, vous devez installer Node.js.

CRA dispose également d'une excellente documentation sur <https://create-react-app.dev/>.

### Node.js

Pour installer Node.js, rendez-vous sur <https://nodejs.org> et téléchargez l'installateur correspondant à votre système d'exploitation. Suivez les instructions de l'installateur et c'est terminé. Vous pouvez maintenant profiter des services fournis par l'utilitaire en ligne de commande npm (Node Package Manager).

Pour vérifier, saisissez ceci dans votre terminal :

```
npm --version
```

Même si vous avez déjà Node.js installé, il est conseillé de l'installer à nouveau pour vous assurer d'avoir la dernière version.

### Hello CRA

Vous pouvez installer CRA et l'avoir disponible localement pour les projets futurs. Cependant, cela signifie le mettre à jour de temps en temps. Une approche encore plus pratique consiste à utiliser l'utilitaire npx fourni avec Node.js. Il vous permet d'exécuter (d'où le "x") des scripts de packages Node. Ainsi, vous pouvez exécuter le script CRA une fois, il télécharge et exécute la dernière version, configure votre application, puis disparaît. La prochaine fois que vous aurez besoin de démarrer un autre projet, vous l'exécutez à nouveau sans vous soucier des mises à jour.

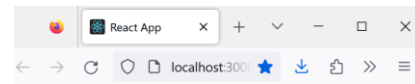
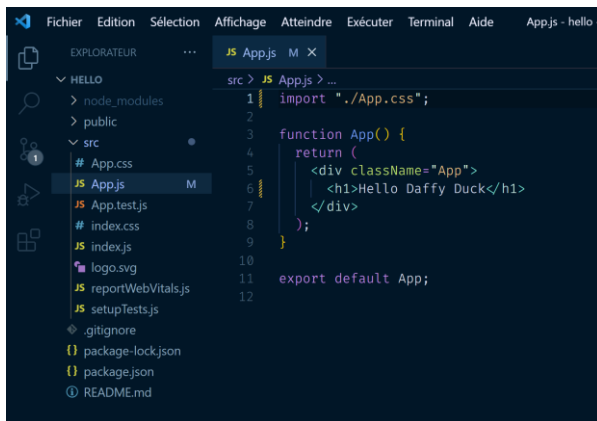
Pour commencer et simplement tester, créez un répertoire temporaire et exécutez CRA :

```
mkdir ~/front_tiers/test  
cd ~/front_tiers/test  
npx create-react-app hello
```

Laissez une ou deux minutes pour que le processus se termine et vous serez accueilli par un message de succès/bienvenue puis faire ces deux commandes :

```
cd hello  
npm start
```

Après l'exécution, arrêter le serveur et modifier le fichier App.js qui se trouve dans le dossier src comme ci-dessous :



**Hello Daffy Duck**

Le fichier `package.json` situé dans le répertoire racine de l'application contient diverses configurations concernant l'application (<https://create-react-app.dev/> dispose d'une documentation détaillée). Une des parties de la configuration concerne les dépendances, telles que React et React-DOM. Ces dépendances sont installées dans le dossier `node_modules` à la racine de l'application.

Les dépendances présentes là-bas sont destinées au développement et à la construction de l'application, pas au déploiement. De plus, elles ne doivent pas être incluses si vous partagez le code de votre application avec des amis, des collègues ou la communauté open-source. Par exemple, si vous souhaitez partager cette application sur GitHub, vous ne devez pas inclure le dossier `node_modules`. Lorsqu'une autre personne souhaite contribuer ou si vous souhaitez contribuer à une autre application, vous installez les dépendances localement.

Vous pouvez également essayer maintenant. Supprimez le dossier `node_modules` en entier. Ensuite, allez à la racine de l'application et saisissez :

```
$ npm i
```

Le "i" signifie "install" (installer). De cette manière, toutes les dépendances répertoriées dans votre `package.json` ainsi que leurs dépendances sont installées dans un nouveau répertoire `node_modules` créé.

**Explorons un peu le code généré par CRA et remarquons quelques éléments.**

### Indices

Dans `public/index.html`, vous trouverez la page d'index HTML classique, la racine de tout ce qui est rendu par le navigateur. C'est là que `<div id="root">` est défini, l'endroit où React va rendre votre composant de niveau supérieur et tous ses enfants.

Le fichier `src/index.js` est l'entrée principale de l'application du point de vue de React. Remarquez la partie supérieure :

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
```

## JavaScript : Modernisé

À mesure que vous avancez vers des applications plus complexes avec plusieurs composants, vous avez besoin d'une meilleure organisation. Vous avez besoin de modules. Les modules permettent de découper les différentes parties fonctionnelles de votre application en fichiers gérables. En général, vous avez une relation un-à-un : une préoccupation, un module.

Le modèle général pour un module est le suivant : déclarez les dépendances en haut, exportez à la fin, mettez en œuvre le "contenu" entre les deux. En d'autres termes, ces trois tâches :

- Requérir/importer les dépendances
- Fournir une API sous la forme d'une fonction/d'une classe/d'un objet
- Exporter l'API

Pour un composant React, le modèle pourrait ressembler à ceci :

```
import React from 'react';
import MyOtherComponent from './MyOtherComponent';
```

```
function MyComponent() {
  return <div>Hello</div>;
}
```

```
export default MyComponent;
```

Encore une fois, une convention qui peut s'avérer utile est la suivante : un module exporte un seul composant React.

Avez-vous remarqué la différence dans l'importation de React par rapport à MyOtherComponent : de 'react' et de './MyOtherComponent' ? Ce dernier ressemble à un chemin de répertoire, et c'est bien le cas - vous indiquez au module de récupérer la dépendance à partir d'un emplacement de fichier relatif au module, tandis que le premier extrait une dépendance d'un endroit partagé (node\_modules).

## CSS

Dans src/index.js, vous pouvez voir comment le CSS est traité comme un autre module :

```
import './index.css';
```

Le fichier src/index.css devrait contenir des styles généraux, tels que body, html, etc., des styles applicables à l'ensemble de la page.

En plus des styles applicables à toute l'application, vous avez besoin de styles spécifiques pour chaque composant. Dans le cadre de la convention consistant à avoir un fichier CSS (et un fichier JS) par composant React, il est judicieux d'avoir MyComponent.css contenant uniquement les styles liés à MyComponent.js et rien d'autre.

Pendant qu'il existe de nombreuses autres façons d'écrire du CSS, restons simples et old school, tout ce qui fonctionnera simplement dans le navigateur sans aucune transpilation.