ReactJS - Memory

Introduction du sujet

Maintenant que vous êtes des experts en JavaScript, nous allons nous attaquer à la bibliothèque **ReactJs**.

Durant la semaine, vous allez devoir développer votre version du jeu memory.

Le jeu se compose de paires de cartes portant des illustrations identiques.

L'ensemble des cartes est mélangé, puis étalé face contre table.
Si le joueur retourne deux cartes identiques, elles restent retournées.

Dans le cas contraire, ces dernières sont remises face cachée. Le joueur a gagné lorsque toutes les cartes ont été retournées.

Vous êtes libre par la suite de mettre en place différents niveaux, un tableau des scores qui laisse sous-entendre le stockage de ces derniers dans une base de données avec nom du joueur ainsi que le score.

Nous allons commencer par l'installation de **ReactJs** et monter crescendo étape par étape pour la réalisation de ce projet. Voyez le comme un tutoriel dans lequel on prend le temps de poser les bases et l'on ne rush pas la vidéo en 45 min alors que cette dernière fait 1h30.

00 Installation

Dans ce premier job, nous allons nous attaquer à l'installation de **ReactJs**.

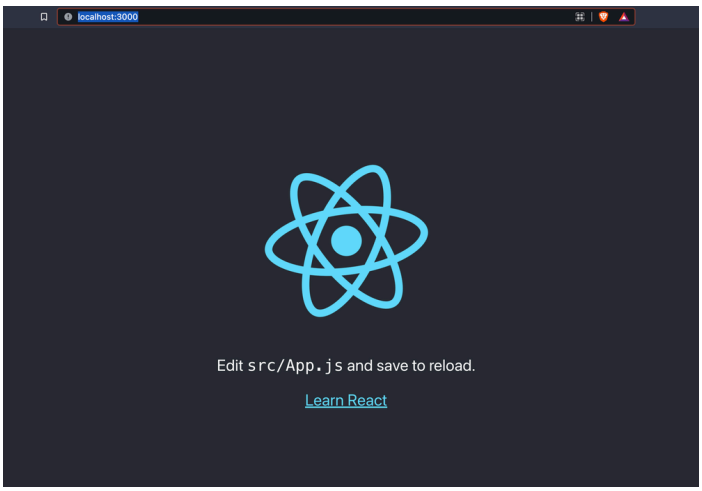
Il est important de noter que **JavaScript** est un langage qui évolue très rapidement, c'est pourquoi la documentation de **ReactJs** évolue aussi et vous trouverez des installations diverses et variées avec des frameworks tels que Next.js, Gatsby ...

Pour ce projet, nous resterons sur une installation **basique** de React **sans framework**.
La documentation vous présente l'installation en utilisant **npm**. Il est tout à fait possible d'utiliser **yarn** ou **vite** également.
https://fr.legacy.reactjs.org/docs/create-a-new-react-app.html

Une fois votre environnement installé, tester que tout fonctionne correctement avec la commande :

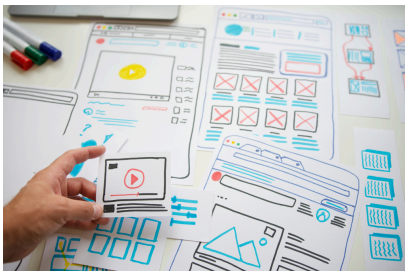
```
npm start
```

Si tout s'est bien passé, vous devriez avoir cette page qui s'ouvre dans le navigateur :



01 Conception

Avant de vous lancer dans le développement de votre application, il est nécessaire de faire un **peu de conception**.



Ce travail va nous permettre d'éviter certaines erreurs, mais aussi cela nous fera gagner du temps.

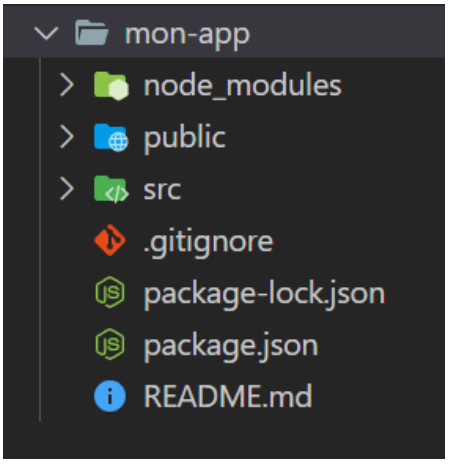
Commencer par réaliser un **wireframe**. Celui-ci va nous permettre de repérer les différents **composants** et facilitera la réalisation du projet.

Ensuite, réfléchissez à une charte graphique. Quelle palette de couleurs choisir ? Quelle police ?...

Pour finir, choisissez les images ou symboles pour les cartes.

02 Architecture

Maintenant que tout est clair dans vos têtes sur ce que le rendu final du projet devrait être, on peut s'attaquer à l'architecture du projet.



Après avoir utilisé la ligne de commande pour installer **ReactJs**, des dossiers et fichiers sont créés automatiquement.

Analysons cela ensemble.

Le dossier **node-module** est un dossier qu'il ne faut pas modifier. Il contient tout ce dont a besoin la librairie pour fonctionner.

Le dossier **public** contient entre autres un fichier **index.html**. Celui-ci est la porte d'entrée de **ReactJs**. Il n'y aura qu'un seul fichier HTML !

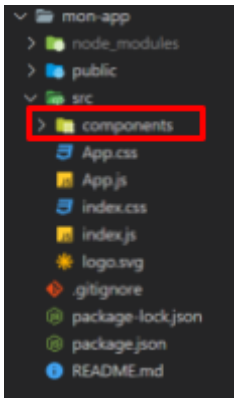
Le dossier **src** est The Dossier de **ReactJs**. Celui qui contient le fichier App.js qui charge le premier composant de votre application.

Le fichier **package.json** contient toutes les dépendances que nous ajoutons au projet.

Lors de l'installation, des fichiers se sont installés alors que nous n'en avons pas l'utilité. Simplifions tout de suite cela !

Supprimer les fichiers qui se trouvent dans le dossier **src**:

- reportWebVitals.js
- setupTests.js
- App.test.js



ReactJs offre une **flexibilité** sur l'organisation du projet, ce qui permet aux développeurs de choisir la méthode qui lui convient le mieux. Il existe deux approches :

- Regrouper les fichiers par fonctionnalité
- Regrouper les fichiers par type

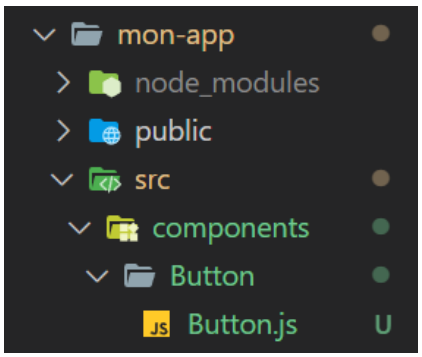
L'**organisation structurée** de votre projet permet de classer vos fichiers selon une logique **claire** et **définie**, ce qui facilite la compréhension pour toute l'équipe travaillant dessus ainsi que de rendre le projet plus facilement maintenable.

Pour ce projet, créer un dossier "**components**" qui comprendra tous les différents composants que l'on va créer.

03 Mise en place d'un composant

Dans **ReactJS**, **les composants** sont des éléments de base. Un composant est une fonction qui a un but particulier. Il peut contenir de la logique et/ou un élément de la page de votre application. Il vous est conseillé de créer vos propres composants génériques que vous pouvez ensuite réutiliser.

Créons notre premier composant, un bouton nommé "**Button**". Celui-ci sera **dynamique** et **réutilisable**.



Rendez-vous dans le dossier "**components**" , créer un sous-dossier "**Button**" et ajouter le fichier contenant le bouton. Par convention, le fichier se nomme comme le composant donc **Button.js**.

Ensuite, procédez comme suit :

- **Importer** ReactJS
- **Définir** le composant grâce à une fonction JavaScript et **oui un composant est simplement une fonction !** Cette fonction retourne obligatoirement un contenu à afficher
- **Rendre accessible** votre composant partout dans votre projet en l'exportant

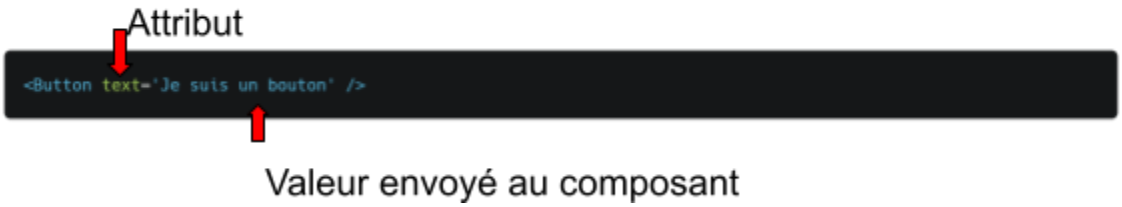
```jsx
import React from 'react';

function Button(props) {
    return (
        <button>{props.text}</button>
    )
}

export default Button;
```

Comment faire pour rendre notre bouton générique afin de pouvoir l'utiliser plusieurs fois ?

Reprenons le composant **Button**.

Comme vous l'avez compris, **Button** est une fonction et une fonction peut prendre des **paramètres**. Dans le cas d'un composant **ReactJs**, on appelle cela des **props** ou **propriétés**. Ces **props** permettent de passer des valeurs à notre composant pour pouvoir les utiliser.

Pour ajouter des props à notre composant, ajouter un attribut dans la balise de votre composant.



Par ce procédé, vous pourrez **récupérer** et **traiter** la valeur.

Voilà votre premier composant est créé ! Félicitations ! Mais comment l'afficher ?

Comme expliqué plus haut, le fichier **App.js** contenu dans le dossier "**src**" est le composant principal de votre application. C'est dans ce composant principal que nous allons ensuite importer les composants liés au développement de notre application.

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">

    </div>
  );
}

export default App;
```

Rendez-vous donc dans **App.js** et vider la div principale de son contenu.

Importer le composant fraîchement créé en haut du fichier et utilisez-le.

L'ordre dans lequel les composants seront placés dans **App** est l'ordre dans lequel ils seront affichés.

Toujours garder en tête que le code est lu de haut en bas.

04 L'état d'un composant

Un des grands principes de **ReactJS** est de ne **pas modifier** directement le DOM comme on pourrait le faire avec une page HTML et du JavaScript.

Ok, mais comment faire si je veux que mon DOM change après une action de l'utilisateur ?

En utilisant les **states** ou **états** en français. Un **composant** peut être considéré comme un ensemble de variables qui définit ce dernier, à un instant T. La modification de ces variables renvoie à nouveau le composant.
Pour utiliser les states, nous allons utiliser les **hooks** et en particulier le **useState**.

Créons un state dans un composant :

- **Importer** le module useState
- **Déclarer** une variable d'état et une fonction qui permet de mettre à jour le state.

Dans cet exemple, si le joueur clique sur un paragraphe et qu'il obtient un point supplémentaire, le useState mettrait à jour le composant sans recharger la page et sans nécessiter de modification du code.

```jsx
import React, {useState} from 'react';

function Score(props){

    const [ score , setScore ] = useState(0)

    // La fonction click met a jour le state du composant au click sur le p
    function click(){
        // En parametre de la fonction setScore on lui donne le score en ajoutant 1
        setScore( score + 1 )
    }

    return (
        // Les accolades permets la concatenation de la sting avec la variable score
        <p onClick={click}>Mon score est de {score}</p>
    )
}
export default Score;
```

05 Conclusion

Avec toutes les différentes étapes que nous avons détaillées tout au long de ce sujet, vous pouvez développer le jeu memory.

Garder à l'esprit que le composant **App** est le composant principal qui contient la logique du jeu.

Pour ce qui est des composants secondaires, il vous est imposé d'avoir au minimum :

- un composant **Title**
- un composant **Button**
- un composant **Card**

Vous êtes libre de faire plus de composants si vous en voyez l'utilité.

De même, concernant les fonctionnalités, vous devez avoir au minimum un bouton permettant de **relancer la partie**, une **animation sur le retournement**

des cartes et un **message de victoire** lorsque toutes les cartes ont été retournées.

Vous êtes libre aussi d'ajouter des fonctionnalités supplémentaires telles qu'un timer, une possibilité de Game Over. Un tableau de score ...

Prenez du plaisir à réaliser ce projet et essayez de pousser toujours plus loin.

"Bon chance"

Compétences visées

- ➔ Découverte Librairie ReactJs
- ➔ Découverte des composants
- ➔ Découverte des hooks

Rendu

Le projet est à rendre sur **https://github.com/prenom-nom/memory**. Votre projet sera évalué en présentation sans support à l'équipe pédagogique.

Base de connaissances

- ➔ https://fr.legacy.reactjs.org/
- ➔ https://react.dev/
- ➔ https://www.w3schools.com/react/