

# TP - React.JS Suite

## I. Bilan du code précédent

### 1. *App.tsx*

```
// App.tsx
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  const [searchTerm, setSearchTerm] = useState('');
  const [sortOption, setSortOption] = useState("name");
  const [selectedHero, setSelectedHero] = useState<SuperHero | null>(null); // Nouvel état
  pour suivre le super-héros sélectionné

  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.id,
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);

  const filteredHeroes = heroes.filter((hero: SuperHero) => {
    return hero.name.toLowerCase().includes(searchTerm.toLowerCase());
  });

  const sortedHeroes = filteredHeroes.sort((a: SuperHero, b: SuperHero) => {
    if(sortOption === "name") {
      return a.name.localeCompare(b.name);
    } else {
      return a.id - b.id;
    }
  });
};
```

Le code React une application qui affiche une liste de super-héros. Elle dispose de fonctionnalités de recherche et de tri, ainsi que d'affichage des détails de chaque super-héros lorsqu'on clique dessus.

Voici une description plus détaillée des fonctionnalités de ce code :

1. **useState** : Cette application utilise plusieurs valeurs d'état gérées par le Hook useState. Ces états incluent :
  - **heroes** : contient un tableau de tous les super-héros chargés à partir des données JSON.
  - **searchTerm** : contient le terme de recherche entré dans la barre de recherche.
  - **sortOption** : contient l'option de tri sélectionnée par l'utilisateur.
  - **selectedHero** : contient les détails du super-héros sélectionné par l'utilisateur.
2. **useEffect** : Cette application utilise le Hook useEffect pour charger les super-héros à partir du fichier JSON lors du premier rendu du composant.
3. **Filtrage et tri des super-héros** : L'application filtre d'abord les super-héros en fonction du terme de recherche, puis elle trie le tableau filtré en fonction de l'option de tri choisie.
4. **Gestion des événements** : Cette application possède plusieurs gestionnaires d'événements :
  - Le champ de recherche a un gestionnaire **onChange** qui met à jour la valeur de **searchTerm** chaque fois que l'utilisateur change la valeur du champ.
  - Le sélecteur d'option de tri a également un gestionnaire **onChange** qui met à jour la valeur de **sortOption** lorsque l'utilisateur change l'option de tri.
  - Chaque super-héros a un gestionnaire **onClick** qui met à jour la valeur de **selectedHero** lorsque l'utilisateur clique sur le super-héros.
5. **Rendu conditionnel** : L'application utilise le rendu conditionnel pour afficher les détails du super-héros sélectionné seulement si **selectedHero** est non null.

## 2. *superHeroes.ts*

```
// SuperHeroes.ts
// On définit une classe 'SuperHero' qui va représenter chaque super-héros de notre fichier JSON.
export class SuperHero {
  id: number;
  name: string;
  idApi: number;
  slug: string | undefined;
  constructor(id: number, name: string, idApi: number, slug?: string) {
    this.id = id;
    this.name = name;
    this.idApi = idApi;
    this.slug = slug;
  }
}
```

C'est une définition de classe TypeScript pour une entité **SuperHero**.

Voici ce que fait chaque partie de ce code :

1. **Définition de la classe SuperHero** : une classe **SuperHero** avec quatre propriétés : **id**, **name**, **idApi** et **slug**. Chacune de ces propriétés représente un aspect différent d'un super-héros.
2. **Constructeur** : un constructeur pour la classe **SuperHero** qui prend quatre arguments : **id**, **name**, **idApi**, et **slug**. Ces arguments sont utilisés pour initialiser les propriétés de la classe.

### 3. *SuperHero.json*

```
[
  {
    "id": 1,
    "name": "Superman",
    "id-api": 644,
    "slug": "644-superman"
  },
  {
    "id": 2,
    "name": "Batman",
    "id-api": 69,
    "slug": "69-batman"
  },
  - - - - -
]
```

La classe **SuperHero** est utilisée dans le composant React pour créer des instances de super-héros à partir des données JSON. Chaque super-héros est ensuite utilisé pour afficher les informations sur le super-héros, y compris son nom et son slug, dans votre application.

Dans cette application React, chaque objet dans le tableau JSON est mappé à une nouvelle instance de **SuperHero** en utilisant le constructeur de cette classe. Ces instances sont ensuite utilisées pour afficher les détails des super-héros dans le composant React.

Le fichier **SuperHeroes.json** est un tableau JSON contenant des données sur différents super-héros. Chaque super-héros est un objet avec les attributs suivants :

1. **id** : un identifiant unique pour chaque super-héros dans votre application.
2. **name** : le nom du super-héros.
3. **id-api** : un identifiant unique pour chaque super-héros selon une API externe.
4. **slug** : un identifiant URL unique pour chaque super-héros, généralement utilisé pour créer des URLs propres pour le référencement.
- 5.

Ce fichier JSON est utilisé dans votre application React pour charger les données des super-héros. Dans le composant React, il est importé ces données et les utilisé pour créer des instances de la classe **SuperHero** définie précédemment. Ensuite, ces instances sont utilisées pour afficher les informations sur les super-héros dans votre application.

### 4. *Index.tsx*

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

ReactDOM.render(
  <App />,
  document.getElementById('root')
)
```

C'est l'entrée principale de l'application. Il utilise **ReactDOM.render** pour rendre le composant **App** dans l'élément DOM avec l'ID **root**. Voici ce que fait chaque partie de ce code :

1. **Imports** : Vous importez les bibliothèques **React** et **ReactDOM**, ainsi que le composant **App** de votre application.
2. **ReactDOM.render** : Vous utilisez **ReactDOM.render** pour rendre votre composant **App**. Ceci est typiquement fait une seule fois dans une application React, dans le fichier d'entrée de l'application.
3. **Sélecteur d'élément DOM** : **document.getElementById('root')** sélectionne l'élément avec l'ID **root** dans votre fichier HTML. C'est là que votre application React sera rendue.

## II.A FAIRE AVANT DE COMMENCER

Étape N°1 : SAUVERGARDER TOUTS VOS FICHIERS AVANT DE DEBUTER CE TP

## III. Afficher une « belle » liste

### 1. Les fonctions tri et recherche

Si vous avez trop de difficultés, les fonctions, de tri et de recherche peuvent ne plus être supportées dans les sections suivantes. Mais il faudra les implémenter néanmoins les remettre dans la dernière partie.

### 2. Mise en place à l'aide de bootstrap

Étape N°2 : Avant tout, mettez dans le head de index :

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSm07SnpJef0486qhLnuZ2cdeRh002iuK6FUUVM"
crossorigin="anonymous">
```

Étape N°3 : Puis juste avant </html> :

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy3Te4Bkz"
crossorigin="anonymous"></script>
```

Étape N°4 : Testez avec cette ligne dans App.tsx :

```
<button className="btn btn-primary">Bouton Bootstrap</button>
```

Bouton Bootstrap

### 3. Nouveau fichier json

Étape N°5 : Remplacez celui de la dernière fois par le nouveau que vous trouverez sur moodle.

Mais dans la définition de la classe, il manque deux champs : powerstats et apperance.  
powerstates est de type PowerStats et se définit ainsi dans SuperHero.ts

```
powerstats: PowerStats;
```

Quand à la classe PowerStats :

```
export interface PowerStats {
  intelligence: number;
  strength: number;
  speed: number;
  durability: number;
  power: number;
  combat: number;
}
```

Étape N°6 : Complétez la définition de la classe SuperHeros.

Maintenant

```
const heroesFromData = SuperHerosData.map((heroData: any) =>
```

ne fonctionne plus

Étape N°7 : Corrigez-le

## 4. Affichage des détails

Étape N°8 : Lorsqu'on sélectionne un personnage, les nouvelles informations sont affichées :

Id: 15  
Id API: 226  
Slug: 226-doctor-stran  
Intelligence: 100  
Force: 10  
Vitesse: 12  
Endurance: 84  
Pouvoir: 100  
Combat: 60



### 5. Affichage avec bootstrap

Étape N°9 : Commencez par afficher trois noms par ligne comme ceci :

Aquaman	Batman	Black Panther
Black Widow	Captain America	Doctor Strange
Hulk	Iron Man	Silver Surfer
Spider-Man	Supergirl	Superman
The Flash	Thor	Wolverine
Wonder Woman		

Aide :

1. `<div className="container">` : Il s'agit d'un conteneur Bootstrap qui enveloppe le contenu de la grille et ajoute une marge et un espacement appropriés.
2. `<div className="row">` : Il s'agit d'une rangée Bootstrap qui contient les colonnes des super-héros. Une rangée permet d'organiser les colonnes horizontalement.
3. une colonne Bootstrap de taille moyenne (**col-md-4**)
4. `" key={hero.id}` l'attribut **key** pour fournir une clé unique à chaque super-héros, ce qui est important pour optimiser le rendu et la performance de React.

Étape N°10 : Puis toutes les informations possibles



### Aquaman

Id API: 38

Slug: 38-aquaman

Intelligence: 81

Force: 85



### Batman

Id API: 69

Slug: 69-batman

Intelligence: 81

Force: 40



### Black Panther







Id API: 106

Slug: 106-black-panther

Intelligence: 88

Force: 16

Étape N°11 : Et pour terminer l'affichage :

 <p><b>Aquaman</b> Id API: 38 Slug: 38-aquaman Intelligence: <b>81</b> Force: <b>85</b> Vitesse: <b>79</b> Endurance: <b>80</b> Pouvoir: <b>100</b> Combat: <b>80</b></p>	 <p><b>Batman</b> Id API: 69 Slug: 69-batman Intelligence: <b>81</b> Force: <b>40</b> Vitesse: <b>29</b> Endurance: <b>55</b> Pouvoir: <b>63</b> Combat: <b>90</b></p>	 <p><b>Black Panther</b> Id API: 106 Slug: 106-black-panther Intelligence: <b>88</b> Force: <b>16</b> Vitesse: <b>30</b> Endurance: <b>60</b> Pouvoir: <b>41</b> Combat: <b>100</b></p>
 <p><b>Black Widow</b> Id API: 107 Slug: 107-black-widow Intelligence: <b>75</b> Force: <b>13</b> Vitesse: <b>33</b> Endurance: <b>30</b> Pouvoir: <b>36</b> Combat: <b>100</b></p>	 <p><b>Captain America</b> Id API: 149 Slug: 149-captain-america Intelligence: <b>69</b> Force: <b>19</b> Vitesse: <b>38</b> Endurance: <b>55</b> Pouvoir: <b>60</b> Combat: <b>100</b></p>	 <p><b>Doctor Strange</b> Id API: 226 Slug: 226-doctor-strange Intelligence: <b>100</b> Force: <b>10</b> Vitesse: <b>12</b> Endurance: <b>84</b> Pouvoir: <b>100</b> Combat: <b>60</b></p>

Étape N°12 : Lorsqu'on survole le cadre, la bordure change de couleur et devient bleu

## IV. Décomposition en composants

### 1. Principe

Il est possible de passer des données à un composant React via les Props ce qui va nous permettre de découper notre application.

En effet, lorsque votre application va commencer à devenir plus importante, nous ne pourrions pas mettre tout le code de l'application dans un seul composant : imaginez que notre composant app.tsx fasse 2000 lignes de code, après quelques jours sans avoir touché il serait très difficile de s'y retrouver.

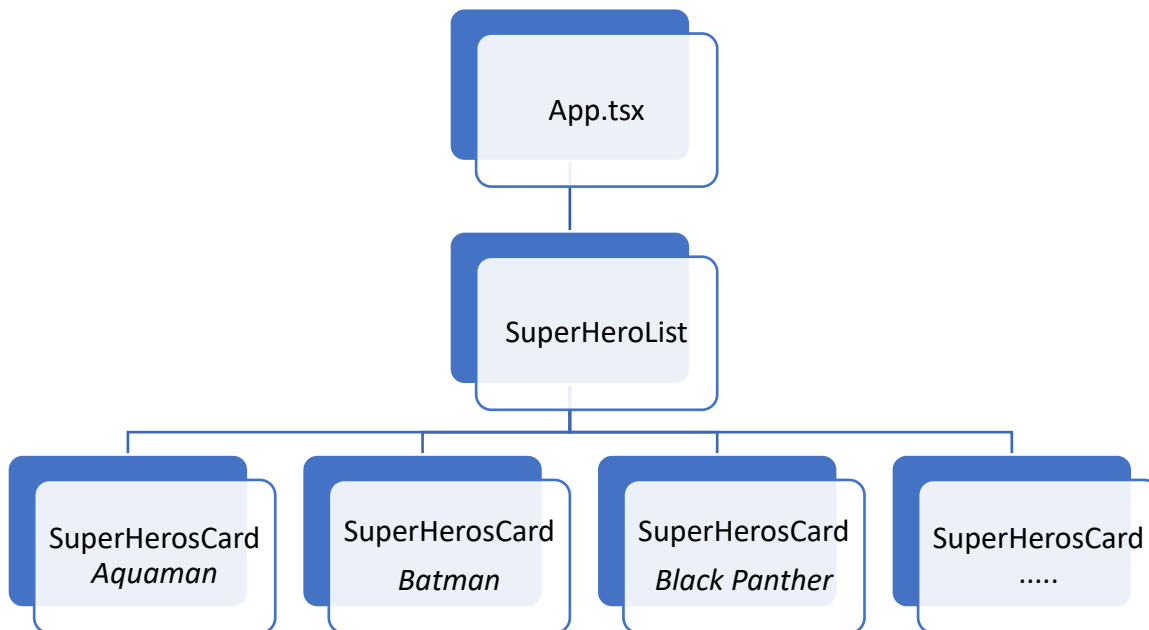
Il va donc falloir découper notre application en plusieurs composants plus petits.

En fait, les Props sont le moyen le plus simple de faire communiquer les composants de votre application entre eux afin qu'ils travaillent ensemble

Par exemple, nous allons imaginer deux composants SuperHeroList et SuperHeroCard : le premier a pour mission de récupérer une liste de personnage et ensuite de passer chaque Superhéros au composant SuperHeroCard sous la forme d'une Prop.

La mission du composant SuperHeroCard sera alors d'afficher le Superhéros à proprement parler. Notre application ressemblera alors à un arbre de composants comme ceci :





En italique, il s'agit de la propre que chaque composant file va recevoir.  
Via cette prop, le composant sera chargé d'afficher le personnage correspondant.

Dans notre schéma, le premier composant SuperHerosCard à gauche est chargé d'afficher les informations du SuperHero Aquaman et ainsi de suite pour les autres.

Pourquoi créer autant de composants pour afficher simplement une liste de SuperHéros ?  
Nous ne construisons pas une application pour demain ou après demain, mais pour le long terme.  
Si on est amené à travailler sur cette application dans trois mois, il faut que nous puissions comprendre notre code rapidement afin de pouvoir intervenir là où il y en a besoin.  
De plus, si nous ajoutons de nouvelles fonctionnalités et donc du code supplémentaire, nous pourrions organiser tout cela de la bonne façon.

## 2. Composant SuperHeroCard

Ce composant qui acceptera une seule prop : un SuperHero

Étape N°13 : Créez un nouveau dossier Components dans src, afin de placer notre composant superhero-card.tsx

Étape N°14 : Y mettre un composant REACT de base qui retourne simplement « Ce composant est chargé de visualiser un superhéros »

Étape N°15 : Complétez le fichier pour que le composant affiche le nom du support héros



Il faut vérifier que le type passé à notre composant est bien un SuperHero.

Étape N°16 : Importez la déclaration de la classe superheros

Étape N°17 : Définissez le type du prop pour l'utiliser avec FC

Aide :



	Web Avance	
	REACT.JS	

- Le type **SuperHeroCardProps** indique que le composant **SuperHeroCard** attend une prop nommée **hero** de type **SuperHero**.
- L'argument générique **<SuperHeroCardProps>** spécifie les types des props que le composant **SuperHeroCard** reçoit.

Maintenant, il nous reste à relier notre nouveau composant **SuperHeroList** au reste de l'application.

### 3. Composant **SuperHeroList**

Nous allons donc créer un composant parent dans un dossier pages. **SuperHeroList** aura pour rôle d'afficher la liste de nos personnages et utilisera le composant **SuperHeroCard** pour gérer plus facilement l'affichage de chaque composant.

Étape N°18 : Créez un fichier **superhero-list** dans un répertoire pages de **src**

Étape N°19 : En vous inspirant de **App.tsx**, écrivez **superhero-list.tsx** qui va, pour chaque super-héros, rendre le composant **SuperHeroCard** en passant l'objet **SuperHero** en tant que prop.

### 4. Nouveau **app.tsx**

C'est donc notre composant précédent qui sera chargé d'afficher chaque superhéros ainsi notre composant **app.tsx** à partir de maintenant, son rôle va simplement être d'appeler le composant **SuperHeroList** qui, lui, s'occupera d'afficher la liste des superhéros.

Étape N°20 : Modifiez **app.tsx** pour qu'il rende que **SuperHeroList**

## V. La navigation

### 1. Vérification

React ne possède pas nativement de système de navigation. Son rôle principal est de gérer l'interaction entre les composants et le DOM virtuel. Cependant, il existe une bibliothèque appelée **React Router DOM** qui permet d'ajouter un système de navigation à une application React. Cette bibliothèque facilite la gestion de la navigation dans le DOM du navigateur. Elle simule le comportement de navigation habituel d'un navigateur, vous permettant de naviguer dans votre application, de revenir en arrière et même de saisir directement une URL dans la barre d'adresse pour accéder à une page spécifique de votre application. En utilisant **React Router**, nous pouvons bénéficier d'une gestion de la navigation similaire à celle d'un navigateur classique sans avoir à la mettre en place manuellement.

Étape N°21 : Dans le fichier package.json, vérifiez que vous ailliez "react-router-dom": "^5.1.2", si ce n'est pas le cas, rajoutez cette ligne

## 2. Une deuxième composant

Nous allons donc créer un deuxième composant qui permettra d'afficher plus de détails d'un super-héros et lorsque que l'on cliquera sur un personnage depuis la liste.

L'URL sera /superheros/id où id est l'identifiant du personnage.

Il faut utiliser RouteComponentProps et link, voici un exemple :

```
import React, { FunctionComponent } from 'react';
import { RouteComponentProps, Link } from 'react-router-dom';

type Params = {
  id: string;
};

const UnePage: FunctionComponent<RouteComponentProps<Params>> = ({ match }) => {
  const { id } = match.params;

  return (
    <div>
      <h1>Page avec un paramètre</h1>
      <p>Paramètre : {id}</p>
      <Link to="/">Retour à la page d'accueil</Link>
    </div>
  );
};

export default UnePage;
```

### Explications :

La première ligne permet d'importer **RouteComponentProps** qui est un type fourni par **react-router-dom**. Il est utilisé pour définir le type des props passées au composant **UnePage** et comprend les informations de routage (comme les paramètres d'URL).

La deuxième ligne permet d'importer **Link** qui est un composant fourni par **react-router-dom** pour créer des liens vers d'autres pages dans votre application. Dans cet exemple, nous utilisons **<Link>** pour créer un lien de retour vers la page d'accueil.

Le composant **UnePage** utilise le type **RouteComponentProps** dans la définition de sa prop et reçoit les informations de routage dans la prop **match**. Nous utilisons également **Link** pour créer un lien vers la page d'accueil et affichons le paramètre **id** dans le rendu du composant.

Étape N°22 : Il faut peut-être nécessaire d'installer `npm install react-router-dom`

Étape N°23 : Dans le dossier page, créez un fichier `superhero-detail.tsx` et y mettre toutes les caractéristiques disponibles du super-héro. La récupération de l'id se fera comme précédemment et il est demandé de soigner la présentation

### 3. Mettre un système de routes

Soit un exemple :

```
import React from 'react';
import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom';

const Navigation = () => (
  <nav className="navbar navbar-expand-lg navbar-light bg-light">
    <div className="container-fluid">
      <Link className="navbar-brand" to="/">Home</Link>
      <div className="collapse navbar-collapse">
        <ul className="navbar-nav me-auto mb-2 mb-lg-0">
          <li className="nav-item">
            <Link className="nav-link" to="/page1">Page 1</Link>
          </li>
          <li className="nav-item">
            <Link className="nav-link" to="/page2">Page 2</Link>
          </li>
          <li className="nav-item">
            <Link className="nav-link" to="/page3">Page 3</Link>
          </li>
        </ul>
      </div>
    </div>
  </nav>
);

const App = () => (
  <Router>
    <Navigation />
    <Switch>
      <Route exact path="/page1" component={Page1} />
      <Route exact path="/page2" component={Page2} />
      <Route exact path="/page3" component={Page3} />
    </Switch>
  </Router>
);
```

```
);  
  
export default App;
```

Étape N°24 : Inspirez-vous de cet exemple pour mettre en place un système de navigation dans app.tsx

#### 4. *Push et history*

**history.push** est utilisé dans React pour changer de route et naviguer vers une nouvelle page programmatically. Voici comment vous pourriez l'utiliser :

```
import React from 'react';  
import { useHistory } from 'react-router-dom';  
  
function ExampleComponent() {  
  const history = useHistory();  
  
  const handleClick = () => {  
    history.push('/other-page');  
  };  
  
  return (  
    <button onClick={handleButtonClick}>  
      Go to other page  
    </button>  
  );  
}  
  
export default ExampleComponent;
```

Dans cet exemple, lorsque le bouton est cliqué, la fonction **handleButtonClick** est exécutée. Cette fonction appelle **history.push** avec le chemin de la page vers laquelle vous voulez naviguer. Cela a pour effet de changer la page actuellement affichée pour aller à **/other-page**

Étape N°25 : Dans la page de détail, lorsqu'on clique sur un cadre, le détail du personnage s'affiche

#### 5. *Page qui n'existe pas*

Pour rediriger les utilisateurs vers une page d'erreur lorsqu'ils essaient d'accéder à une page qui n'existe pas, vous pouvez utiliser le composant **<Switch>** et le composant **<Redirect>** de **react-router-dom** :

```
import React from 'react';  
import { BrowserRouter as Router, Route, Switch, Redirect } from 'react-router-dom';  
import HomePage from './HomePage';
```

```
import AboutPage from './AboutPage';
import ErrorPage from './ErrorPage';

function AppRouter() {
  return (
    <Router>
      <Switch>
        <Route exact path="/" component={HomePage} />
        <Route path="/about" component={AboutPage} />
        { /* Si aucune des routes ci-dessus ne correspond, nous redirigeons vers la page d'erreur */ }
        <Route path="/error" component={ErrorPage} />
        <Redirect to="/error" />
      </Switch>
    </Router>
  );
}

export default AppRouter;
```

Dans cet exemple, si l'utilisateur essaie d'accéder à une route qui n'est pas / ou /**about**, ils seront redirigés vers /**error** où le composant **ErrorPage** sera rendu.

Assurez-vous que le **<Redirect>** est la dernière instruction dans le **<Switch>** car les routes sont évaluées dans l'ordre et la première correspondance est choisie. Donc, si vous mettez le **<Redirect>** en premier, toutes les routes seront redirigées vers /**error**.

Étape N°26 :      Gérez cette erreur

## 6. Améliorations

Étape N°27 :      Si on met un identifiant inconnu, la page est en erreur. Modifiez ce comportement

Étape N°28 :      Lors du survol sur un super-héros, le cadre doit changer de couleur

Étape N°29 :      Dans la page de détail, ajoutez un lien de retour

---

## VI. Formulaire

---

Étape N°30 :      Ajoutez un formulaire qui soit effectue une recherche par nom soit par le niveau d'un ou de plusieurs pouvoirs.