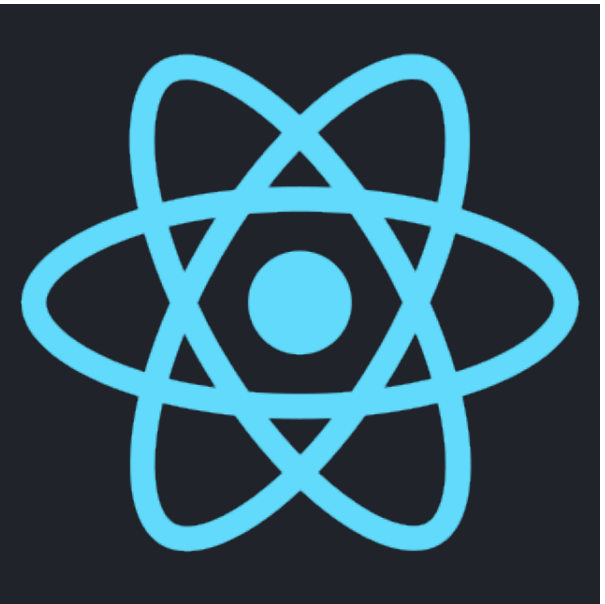
# Technologies Front-End

## Framework JS



### Module « React »

Made by Valentin AMATHIEU

# Routing

# SPA et navigation

» React permet, par défaut, de créer des Single Page Applications.

» Selon ce principe, une seule page est servie au client lorsque ce dernier souhaite accéder à une SPA.

» La page étant dynamisée en JavaScript, l'utilisateur peut interagir et utiliser cette dernière comme une application.

» Mais l'un des principes fondateur du Web est la possibilité de faire des liens hypertexte entre plusieurs pages, fournissant ainsi un système de navigation au client.

» En effet, il est préférable de distribuer le contenu au travers de plusieurs pages, plutôt que de tout mettre en une seule, dans un soucis de structuration et de lisibilité de notre application.

» L'utilisation de la balise HTML <a> classique ne sera pas possible pour une SPA.

» Elle entraînera, en effet, un rechargement complet de la page et une perte des éventuelles informations issues de la navigation de l'utilisateur…

# React-router-dom

» C'est ici qu'intervient react-router-dom.

» react-router-dom est une librairie tierce permettant de simuler un système de navigation (routing) côté client (sans appel http) et utilisant à l'API History du navigateur.

» Pour l’installer dans une application React existante, lancer la commande : npm install react-router-dom

# Le router

» Pour assurer une navigation côté client, l'application a besoin d'un router.

» Le router est l'élément logiciel dédié à l'interception des URLs de navigation et à la mise en correspondance de ces URLs (ou chemins) avec des contenus à afficher.

» Dans react-router-dom, il existe un composant Router assurant cette fonction.

» Ce composant doit contenir en children l'ensemble de tout ce qui concerne la navigation.

```
<Router>
    // ...
</Router>
```

# Les liens

» Pour naviguer vers une page, une URL de navigation peut être appelée.

» Les URLs de navigation peuvent être définies de manière programmatique ou via l'intermédiaire d'un lien.

» Un lien peut être créé grâce au composant Link, ainsi que sa propriété to.

```
<Router>
    <nav>
        <ul>
            <li>
                <Link to="/">Home</Link>
            </li>
            <li>
                <Link to="/about">About</Link>
            </li>
            <li>
                <Link to="/users">Users</Link>
            </li>
        </ul>
    </nav>
</Router>
```

# Les routes

» Une URLs de navigation ne sert à rien si elle ne correspond à aucun contenu affichable (un composant généralement).

» Cette correspondance est assurée par ce qu'on appelle une route.

» La librairie nous fournit un composant appelé Route ainsi que sa propriété path, qui est une représentation d'une URL.

» Le composant Route fournit plusieurs manières de représenter le contenu affichable, la plus simple étant de fournir ce contenu en tant que children de Route

# Les routes

» Exemple :

```
<Router>
    <nav>
        // ...
    </nav>
    <Route path="/about">
        <About />
    </Route>
    <Route path="/users">
        <Users />
    </Route>
    <Route path="/">
        <Home />
    </Route>
</Router>
```

# Correspondance des URLs et chemins

» Parfois, un chemin d'URL peut correspondre à plusieurs routes, car l'algorithme vérifie simplement si le path de la Route est "compris" dans l'URL.

» Ainsi, plusieurs routes n'ayant aucun rapport entre elles peuvent être affichées dans la page, alors qu'une seule est en vérité attendue.

» C'est notamment le cas dans l'exemple précédent !

» L'URL /about correspondra aux routes / et /about et l'URL /users à / et /users.

» Pour palier à ce problème, il est possible d'utiliser un élément qui décidera que la 1ère route à matcher un chemin d'URL, sera affichée et nulle autre.

» On appelle un tel élément un switch.

# Le switch

» Pour mettre en place un switch, il suffit d'utiliser le composant fournis Switch et de mettre l'ensemble des routes en tant que children de ce composant.

```
<Router>
    <nav>
        // ...
    </nav>
    <Switch>
        <Route path="/about">
            <About />
        </Route>
        <Route path="/users">
            <Users />
        </Route>
        <Route path="/">
            <Home />
        </Route>
    </Switch>
</Router>
```

# Paramètres d’URL

» Il est possible de définir des paramètres au sein des URLs.

» Une URL étant composée de segments, on peut faire en sorte que le chemin d'une route accepte des valeurs variables au niveau d'un ou plusieurs de ces segments.

» Ainsi toute URL possédant une structure similaire à ce chemin pourrait être matchée.

» Pour définir un paramètre dans une route, on utilise la notation :param, où param correspond au nom du paramètre.

# Paramètres d’URL

» Exemple :

```
<Link to="/recettes/salade"/>Salade</Link>
<Link to="/recettes/steak"/>Steak</Link>

// ...

<Route path="/recette/:repas" children={<Recette />} />
```

» Ici les 2 liens matchent la même route !

# Récupération des paramètres

» Pour récupérer le ou les paramètres de la route au sein du composant affiché, on peut utiliser le hook useParams.

» Ce hook renvoie un objet en faisant correspondre les noms des paramètres à leur valeur.

```
function Recette() {

    let { repas } = useParams();

    return (
        <div>
            <h1>Détail de la recette { repas }</h1>
        </div>
    )
}
```

# Navigation imbriquée

» Avec react-router-dom, il est possible d'imbriquer plusieurs navigations différentes

» Cela peut être pratique lorsque le composant d'une route propose lui-même une sous-navigation.

» Pour ce faire, il suffit de fournir de nouvelles routes au sein de la vue de ce composant (avec éventuellement un nouveau Switch).

» Il est possible de récupérer des informations concernant la route courante via le hook useRouteMatch.

» Ce hook renvoie notamment les propriétés url et path pouvant être exploités pour construire les liens et routes de la sous-navigation.

# Navigation imbriquée

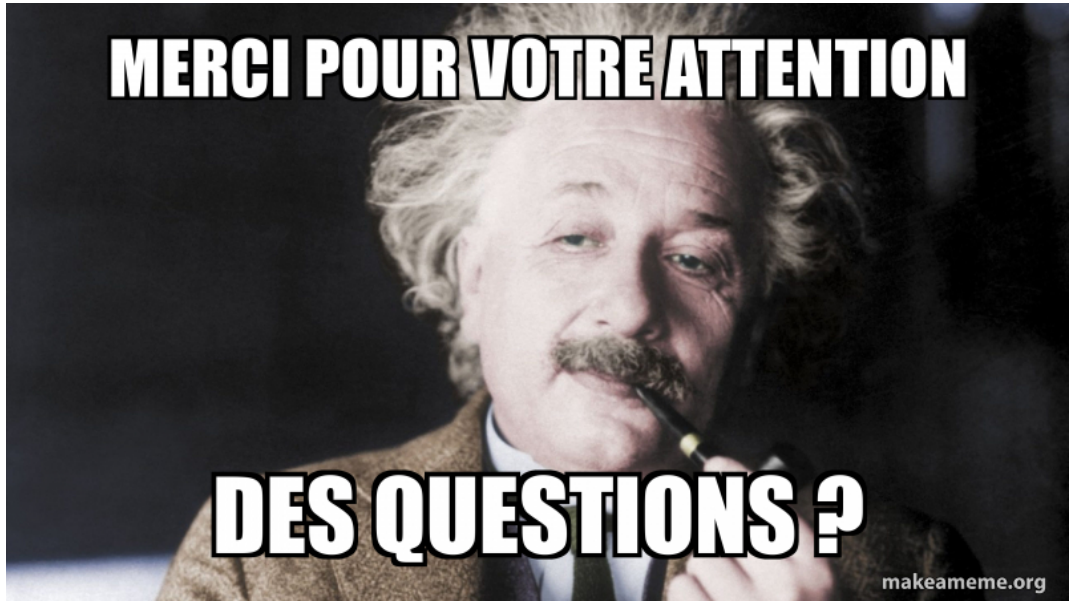
» Exemple :

```
function ListeRecette() {
    let match = useRouteMatch();

    return (
        <div>
            <ul>
                <li><Link to={`${match.url}/salade`}>Salade</Link></li>
                <li><Link to={`${match.url}/steak`}/>Steak</Link></li>
            </ul>
            <Switch>
                <Route path={`${match.path}/:repas`} children={<Recette/>}/>
                <Route path={match.path}>
                    <h3>Sélectionner un repas</h3>
                </Route>
            </Switch>
        </div>
    )
}
```

# Question Time

?



# Atelier Time

# Exercice
## « Webmarket »

# Fin du mon.. module !