



Secteur Tertiaire Informatique
Filière étude - développement

Développement Web et Web Mobile

ReactJS

Développer des scripts clients dans une page web grâce à un Framework

Travaux Pratiques 2

1. ToDoList

Cet exercice va vous permettre de mettre en application les principes de React. Vous serez guidé en début de travail puis, de plus en plus autonome. Au fur à mesure de l'avancement, vous devez renseigner la fiche de synthèse afin de mémoriser les syntaxes et les points abordés.

Vous avez à votre disposition les fichiers suivants :

- *static.html* : code html permettant l'affichage des tâches,
- *todo.css* : code css de l'application,
- *addTask.html* : code html pour la création d'une tâche,
- *initialData.js* : code permettant d'initialiser la liste des tâches,
- *getTasks.js* : code permettant de renseigner les tâches à partir d'une API.

Créer l'application React avec cette commande dans la console:

`npx create-react-app todolist`

En ouvrant **App.js**, nous voyons que la syntaxe générée est celle des composants « fonction ». Dans tout cet exercice, nous garderons cette syntaxe.

Copier la partie <section> du code html (static.html) dans la fonction return() du fichier App.js (renommer class en className car class est un mot réservé en React)

Vous devez avoir cet affichage en lançant l'application (npm start):

Liste de tâches

- Ranger la vaisselle ☒
- Répondre appel d'offres ☒
- Signer contrat ☒
- Ranger le salon ☒

[ListCompletedAdd](#)

Installer Bootstrap et react-icons:

`npm install bootstrap react-icons jquery popper.js -save`

react-icons est une bibliothèque d'icônes qui va permettre d'améliorer l'Interface Utilisateur.

Importer bootstrap dans index.js :

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Importer les icônes nécessaires dans App.js :

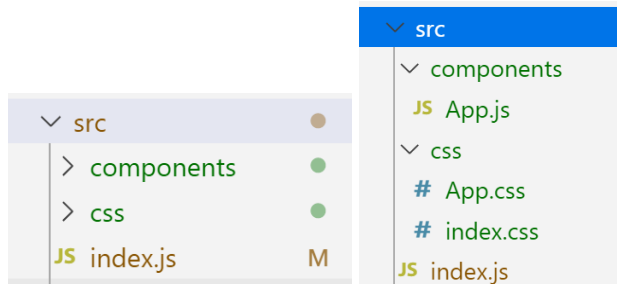
```
import { FaListAlt, FaCheckSquare, FaPlusSquare, FaTrash } from 'react-icons/fa'
```

Remplacer le texte par les icônes dans le html :

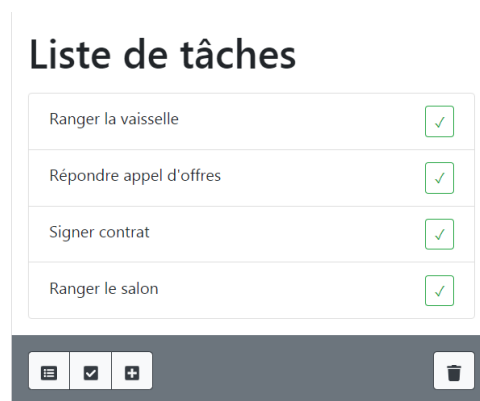
```
<div className="btn-group">
  <a href="#" className="btn btn-outline-dark bg-light"><FaListAlt /></a>
  <a href="#" className="btn btn-outline-dark bg-light">< FaCheckSquare /></a>
  <a href="#" className="btn btn-outline-dark bg-light"><FaPlusSquare /></a>
</div>
<button className="btn btn-outline-dark bg-light"><FaTrash /></button>
```

Structurer le code :

Pour le moment tous les fichiers sont dans le dossier *src*. Modifier la structure de l'application pour que les composants soient sauvegardés dans un dossier *components*, les css dans un dossier *css*. Vous devez avoir cette structure :



Il faut donc modifier les liens vers les fichiers .js et .css. Vérifier ensuite, que votre code fonctionne toujours, vous devez obtenir cet écran :



Import et export des modules :

En ouvrant le fichier index.js, vous pouvez voir que l'on importe le composant *App* :

```
import App from './components/App';
```

Ceci fonctionne parce que l'on a autorisé l'export dans le module *App.js* :

```
export default App;
```

A chaque fois que l'on créera un nouveau module, il faudra **autoriser** l'export pour pouvoir utiliser l'import dans un autre module.

Nota Bene : Bien respecter la syntaxe des imports/exports.

Découper les composants pour obtenir des petits composants simples :

L'objectif maintenant est de créer des composants simples imbriqués entre eux.

On va créer les composants :

- **ToDoList** : liste des tâches,
- **NavBar** : barre de navigation.

Composant ToDoList :

Créer le fichier `Todolist.js` qui se chargera du code suivant initialement codé dans `App.js`

```
<h1 className="m-3">Liste de tâches</h1>
<ul className="list-group m-3">
  <li className="list-group-item d-flex align-items-center">
    Ranger la vaisselle
    <button className="btn btn-sm ml-auto btn-outline-success">&#x2713;</button>
  </li>
  <li className="list-group-item d-flex align-items-center">
    Répondre appel d'offres
    <button className="btn btn-sm ml-auto btn-outline-success">&#x2713;</button>
  </li>
  <li className="list-group-item d-flex align-items-center">
    Signer contrat
    <button className="btn btn-sm ml-auto btn-outline-success">&#x2713;</button>
  </li>
  <li className="list-group-item d-flex align-items-center">
    Ranger la salon
    <button className="btn btn-sm ml-auto btn-outline-success">&#x2713;</button>
  </li>
</ul>
```

Attention, penser au **fragment** dans le fichier `ToDoList.js`

Composant NavBar :

Et le fichier `Navbar.js` qui se chargera du code suivant initialement codé dans `App.js`. Ce code gère les boutons de navigation :

```
<footer className="d-flex justify-content-between bg-secondary p-3" id="mainFooter">
  <div className="btn-group">
    <a href="#" className="btn btn-outline-dark bg-light"><FaListAlt /></a>
    <a href="#" className="btn btn-outline-dark bg-light">< FaCheckSquare /></a>
    <a href="#" className="btn btn-outline-dark bg-light"><FaPlusSquare /></a>
  </div>
  <button className="btn btn-outline-dark bg-light"><FaTrash /></button>
</footer>
```

Ces deux fichiers sont enregistrés dans le dossier *components*.

Vous allez créer ici deux nouveaux composants. Prenez exemple sur `App.js` et `index.js` pour la syntaxe des composants.

Mettre à jour tous les imports et les exports pour avoir le même rendu.

Ceci va vous permettre de naviguer dans les différents fichiers et apprendre à vous repérer dans l'arborescence du projet.

Dans le fichier `App.js`, on aura plus que l'appel à ces deux composants :

```
<section id="todo">
  <h1 className="m-3">Liste de tâches</h1>
  <ToDoList />
  <NavBar />
</section>
```

Vous devez toujours avoir le même écran affichant la liste des tâches à ce stade du développement.

Création des tâches :

Récupérer le code html du fichier addtask.html et créer le composant **AddTask** à partir des fichiers qui vous ont été fournis et incorporer le dans le dossier *components* de votre application. Ce composant permettra de créer des tâches.

2. Gestions des routes :

Pour pouvoir gérer les routes dans react, il faut importer react-router-dom :

```
npm install react-router-dom --save
```

Ensuite, il faut importer les composants que l'on veut utiliser dans notre fichier App.js :

```
import { BrowserRouter, Switch, Route } from 'react-router-dom'
```

Ceci permet de faire des liens vers les composants désirés en cliquant sur les boutons. Remplacer les deux composants ToDoList et NavBar par ces lignes :



```
<BrowserRouter>
  <Switch>
    <Route path="/add-task" component={AddTask} />
    <Route path="/:filter?" component={ToDoList} />
  </Switch>
  <NavBar />
</BrowserRouter>
```

Dans le composant NavBar, créer les liens de navigation :

```
<div className="btn-group">
  <NavLink to="/" className="btn btn-outline-dark bg-light"><FaListAlt /></NavLink>
  <NavLink to="/completed" className="btn btn-outline-dark bg-light">< FaCheckSquare /></NavLink>
  <NavLink to="/add-task" className="btn btn-outline-dark bg-light"><FaPlusSquare /></NavLink>
</div>
```

Et importer le composant NavLink :

```
import { NavLink } from 'react-router-dom'
```

Ici, vous voyez que les deux boutons  sont validés quand on clique sur . Ceci est dû à la route qui est identique « / ». Rajouter l'option `exact={true}` sur les différents NavLink pour corriger ce bug.

Appeler le formateur pour avoir des explications sur ces lignes de code si nécessaire afin de bien renseigner le document de synthèse 02-react-synthese.doc.

Amélioration du style de l'application:

Copier le fichier css : todo.css fourni dans le dossier css et l'importer dans le fichier index.js.

Site dynamique:

L'étape suivante consiste à rendre notre site dynamique. Pour cela, nous allons intégrer un fichier de données qui contient les premières tâches de notre ToDoList.

Dans le composant *App*, nous allons récupérer ces données qui sont dans le fichier initialData.js donné en ressources.

Copier le fichier ressource dans un dossier *data* dans *src*

Importer ce fichier dans App.js :

```
import initialData from '../data/initialData'
```

Si vous ouvrez App.js, vous voyez que la route «/» est associée au composant `ToDoList`.

App va envoyer en props à `ToDoList` le tableau de données `initialData`.

Pour cela, remplacer la route

```
<Route path="/:filter?" component={ToDoList} />
```

par :

```
<Route path="/:filter?" render={(props) => <ToDoList {...props} tasks={initialData} />} />
```

Vous reconnaissez ici l'utilisation du **spread operator** de JavaScript vu précédemment (ou à voir si ce n'est pas le cas).

Le composant App ajoute à toutes les props du composant `ToDoList`, la props `tasks` contenant le tableau de données.

Maintenant, il reste à afficher le contenu du tableau à la place des quatre tâches présentes en dur dans le fichier *ToDoList.js*

Créer un nouveau composant **ToDo** qui va se charger d'afficher une tâche en recevant la *props* de *ToDoList* :

```
<li className="list-group-item d-flex align-items-center">
  {props.task.name}
  <button className="btn btn-sm ml-auto btn-outline-success">&#x2713;</button>
</li>
```

Modifier le code de *ToDoList* de manière à parcourir la props **tasks** contenant les tâches du tableau *initialData*. Pour chaque ligne du tableau, appeler le composant `ToDo` avec, en props, la tâche à afficher.


Je vous laisse écrire le code permettant de faire cela, s'aider de l'exercice « **6. Tableaux** » sur les listes, faits précédemment (attention, utiliser la syntaxe des composants « fonctions »).

Filtrer les données à afficher

Vous avez pu remarquer que dans la route du composant App, vous avez un filtre :

```
<Route path="/:filter?" render={(props) => <ToDoList {...props} tasks={initialData} />} />
```

A ce filtre sont associées plusieurs propriétés, dans votre navigateur, si vous cliquez sur votre composant "ToDoList", vous apercevez toutes les props créées par ce filtre :

En cliquant sur le premier bouton :



En cliquant sur le deuxième bouton ☒ qui ne doit afficher que les tâches réalisées :



Nous allons donc maintenant prendre en compte ce filtre dans notre code. La props *match* permet de filtrer les données, nous allons la récupérer dans le composant `ToDoList`.

```
function ToDoList (props)
{...}
```

Une fois la props *match* récupérée, il faut tester si elle est égale à « completed » (filtre présent dans l'url sur le deuxième bouton) pour savoir s'il faut afficher toutes les tâches ou seulement les tâches réalisées. Vous pouvez utiliser un switch pour cela.

Pour bien comprendre ce qu'il faut faire, revoir les données définissant l'objet `Tasks` dans *initialData.js*.

Voici comment filtrer les « task » à l'aide du switch (Attention, syntaxe fonctionnelle des composants react)

```
function ToDoList(props) {
  let filteredTasks

  switch (props.match.params.filter){
    case 'completed':
      A completer ....
      break;
    default:
      filteredTasks = props.tasks
  }
}
```

La variable *filteredTasks* va contenir :

- uniquement les tâches « completed » dans le premier cas du switch,
- toutes les tâches dans le deuxième cas.

Pour compléter la ligne manquante, voir la fonction [Array.prototype.filter\(\)](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/L_op%C3%A9rateur_conditionnel) de JavaScript.

Ensuite, afficher « Aucune tâche à afficher » s'il n'y a aucune tâche à afficher, sinon, afficher la liste des tâches réalisées. Pour cela, l'opérateur ternaire est très pratique en testant la longueur du tableau *filteredTasks* :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/L_op%C3%A9rateur_conditionnel

Valider les tâches

Maintenant, nous allons activer le bouton qui permet de valider les tâches effectuées :

Ranger la vaisselle



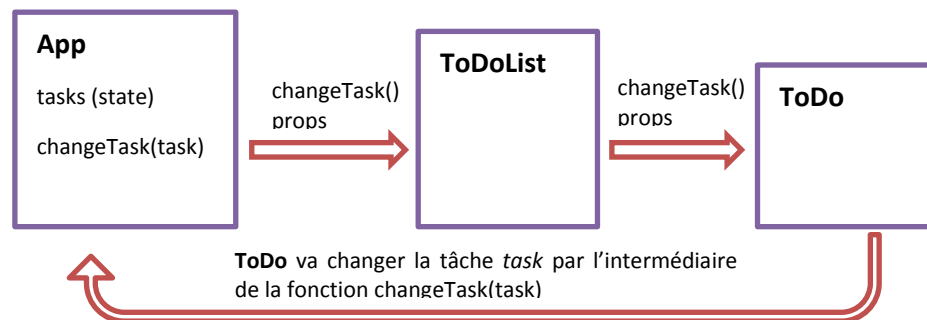
Il faut déjà le rendre actif. Il est géré dans le fichier `ToDo.js`. Rajouter l'option **onClick** sur le bouton afin d'appeler la fonction **changeCompleted()** qui va inverser la valeur du champ **completed** de la tâche.

```
onClick={changeCompleted}
```

et définir la fonction (on utilise ici la notation **arrow** de JavaScript : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions):

```
const changeCompleted = () => {  
  ...  
}
```

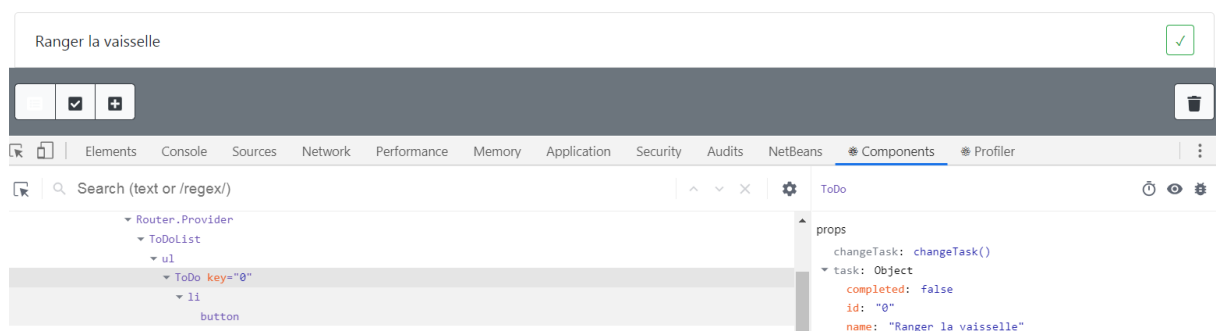
Nous devons ici, modifier la liste des tâches (tasks) et mettre à jour l'affichage automatiquement. C'est ce que permet de faire les **states** de React. Comme nous utilisons la syntaxe composant « fonction », nous allons utiliser les **hooks** (voir document 02-react.pdf paragraphe 3). La liste des tâches et la fonction qui va modifier cette liste sont définies dans le composant **App**.



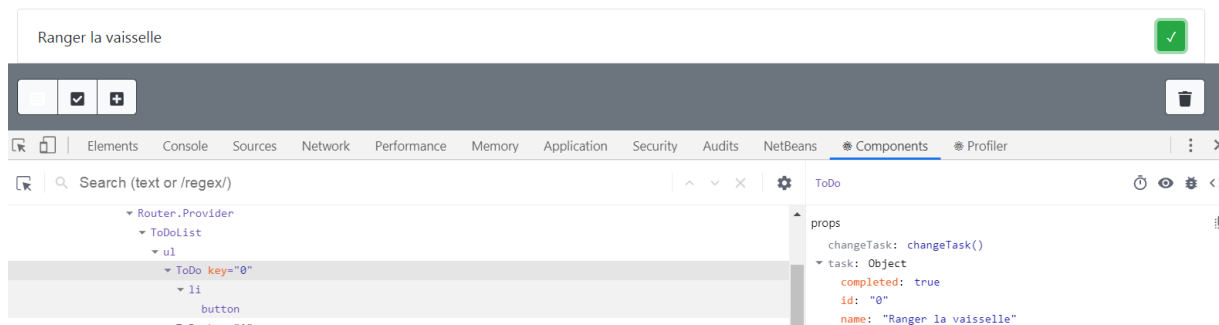
La fonction `changeCompleted()` de `ToDo` doit faire deux choses :

- inverser l'attribut `completed` de la tâche,
- envoyer la tâche modifiée à la fonction `changeTask(task)` pour mettre à jour la liste (hooks).

A ce stade, vous devez avoir, avant le clic sur le bouton :



Et, après le clic sur le bouton :



Il reste à modifier l’affichage en fonction du clic sur le bouton :

```
<button className={"btn btn-sm ml-auto " + (props.task.completed ? 'btn-success' : 'btn-outline-success')} onClick ... .. . . .>&#x2713;</button>
```

Créer des tâches:

Les tâches créées ont un id. Il existe une dépendance qui permet de générer des id uniques. Vous pouvez l’installer à l’aide de la console :

```
npm install uuid --save
```

Vous pourrez ensuite l’utiliser dans vos .js en appelant la fonction `uuid` après avoir fait l’import dans le fichier : `import uuid from 'uuid'`

Comme pour la mise à jour des tâches complétées, il faut modifier la liste dans le composant **App** à partir du composant **AddTask**.

On peut rajouter une tâche à notre tableau de tâches en utilisant le **spread operator** :

```
const [tasks, setTasks] = useState(initialData);
...
setTasks([...tasks, newTask]);
```

Une fois la tâche créée, on peut rediriger vers la liste des tâches :

```
props.history.push('/')
```

Activer le bouton

Ce bouton permet de supprimer les tâches réalisées. Utiliser la fonction *filter* de JavaScript pour cela.

3. Utilisation d’une API pour charger les tâches à effectuer :

Comme vu dans le cours (seq8904-se32008-ua3-02-react.pdf – paragraphe 6), nous allons utiliser « axios » pour charger la liste des tâches à partir d’une API.

```
import axios from 'axios';
```

L’API se trouve à l’adresse :

<http://jsonplaceholder.typicode.com/todos/>

Nous pouvons faire une sélection par user :

<http://jsonplaceholder.typicode.com/todos/?userId=1>

En ouvrant cette URL dans un navigateur, vous pouvez afficher la réponse de l'API.

Nous allons charger les données dans la méthode `componentDidMount()`.

```
axios
    // The API we're requesting data from
    .get("http://jsonplaceholder.typicode.com/todos/?userId=1")
    // Once we get a response, we'll map the API endpoints to our props
    .then(response => {
        console.log(response);

... A faire, mettre à jour le state contenant les tâches à exécuter

        });

... A faire, parcourir response et mettre à jour les objets tâches.
    })

    }
)

// We can still use the `.catch()` method since axios is promise-based
.catch(error => this.setState({ error, isLoading: false }));
```

Une fois les charges téléchargées, on doit obtenir cet affichage :

Liste de tâches

delectus aut autem	<input type="checkbox"/>
quis ut nam facilis et officia qui	<input type="checkbox"/>
fugiat veniam minus	<input type="checkbox"/>
et porro tempora	<input checked="" type="checkbox"/>
laboriosam mollitia et enim quasi adipisci quia provident illum	<input type="checkbox"/>
qui ullam ratione quibusdam voluptatem quia omnis	<input type="checkbox"/>
illo expedita consequatur quia in	<input type="checkbox"/>
quo adipisci enim quam ut ab	<input checked="" type="checkbox"/>
molestiae perspiciatis ipsa	<input type="checkbox"/>
illo est ratione doloremque quia maiores aut	<input checked="" type="checkbox"/>