

TP : Réalisation d'une application CRUD avec React

Architecture des composants d'entreprise

Table des matières

1. Objectif du TP.....	2
2. Prérequis	2
3. Préparation de l'environnement de développement.....	2
1. Installer NODE.JS.....	2
2. Installation de Microsoft Visual Studio Code	4
4. Développement de l'application	5
1. Créer un nouveau projet.....	5
2. L'arborescence de l'application React.....	6
3. Installation de Bootstrap	7
4. Installation de la librairie AXIOS	7
5. Développement du service.....	7
6. Développement du composant CustomerList	8
7. Développement du composant CustomerComponent.....	9
8. Modification du fichier App.js.....	12
5. Tester votre application.....	13
Conclusion	14

1. Objectif du TP

- Réaliser une application CRUD (Create, Read, Update et Delete) avec la librairie React.
- Utiliser Axios pour exécuter les méthodes Get, Post, Put et Delete.

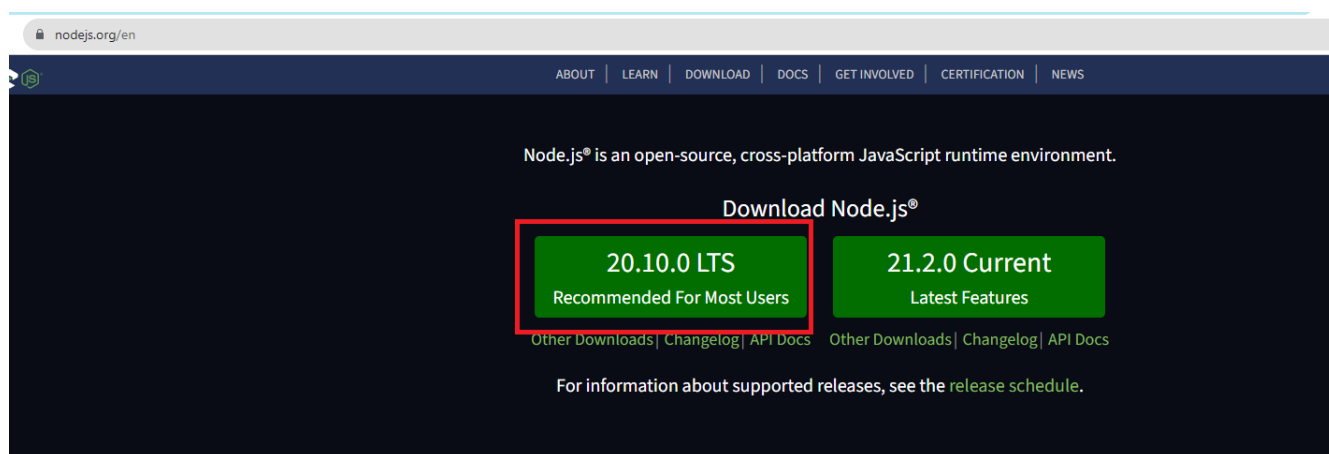
2. Prérequis

- Visual Studio Code (VSD);
- NODE JS ;
- Une connexion Internet pour permettre à NPM de télécharger les librairies javascript.

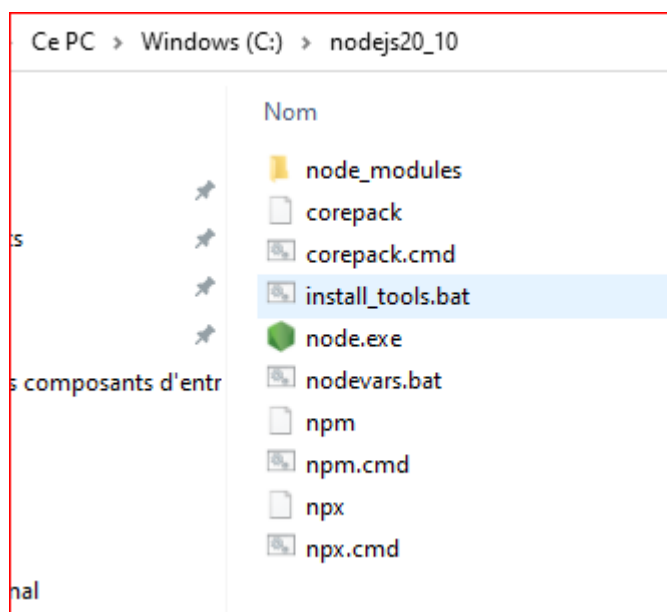
3. Préparation de l'environnement de développement

1. Installer NODE.JS

- Télécharger NodeJS via son site principal : <https://nodejs.org/> :



- Dans cet atelier, nous allons installer la version 20.10.0 LTS. Lancer l'exécutable une fois téléchargé et suivre les étapes pour installer NodeJS. Dans notre exemple, nous avons installé l'outil dans le dossier C:\nodejs20_10 :



- Remarquer que NPM (Node Package Manager) s'installe avec NodeJS.
- Vérifier si Node.JS et NPM ont été bien installés. Pour ceci, ouvrir votre terminal et exécuter les commandes suivantes :

```

CA Invite de commandes
Microsoft Windows [version 10.0.19045.3693]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\abbou>node -v
v20.10.0

C:\Users\abbou>npm -v
10.2.3

C:\Users\abbou>

```

- Ensuite, lancer la commande suivante pour mettre à jour npm :

```

C:\Users\abbou>npm i npm -g

added 1 package in 15s

28 packages are looking for funding
  run `npm fund` for details

C:\Users\abbou>npm -v
10.2.4

C:\Users\abbou>

```

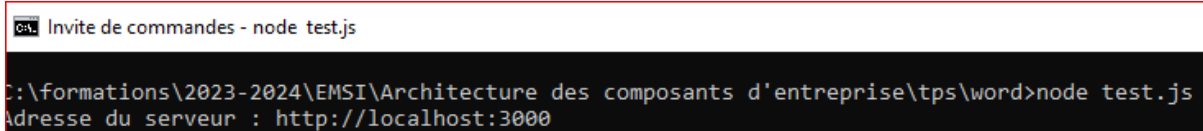
- Pour vérifier si Node.JS fonctionne, suivre les étapes suivantes :
 - ✓ Créer un fichier vide test.js
 - ✓ Ouvrir le fichier avec un éditeur de code (exemple bloc-notes)
 - ✓ Copier le code suivant :

```

const { createServer } = require('http');
//Creation du serveur
const server = createServer(
  (request, response) => {
    {
      response.writeHead(200, {'Content-Type': 'text/plain'});
      response.end('Hello World\n');
    }
  }
);
server.listen(3000, () => console.log(`Adresse du serveur : http://localhost:3000`));

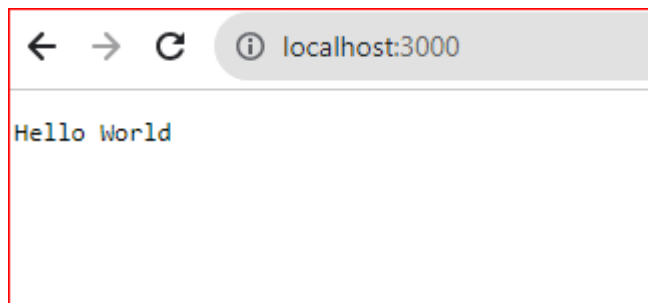
```

- Se positionner dans le dossier du fichier et lancer la commande suivante :



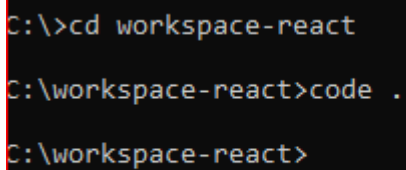
```
Invite de commandes - node test.js
C:\formations\2023-2024\EMSI\Architecture des composants d'entreprise\tps\word>node test.js
Adresse du serveur : http://localhost:3000
```

- Au niveau du navigateur, taper le lien <http://localhost:3000> et vérifier que le message Hello World est bien affiché :



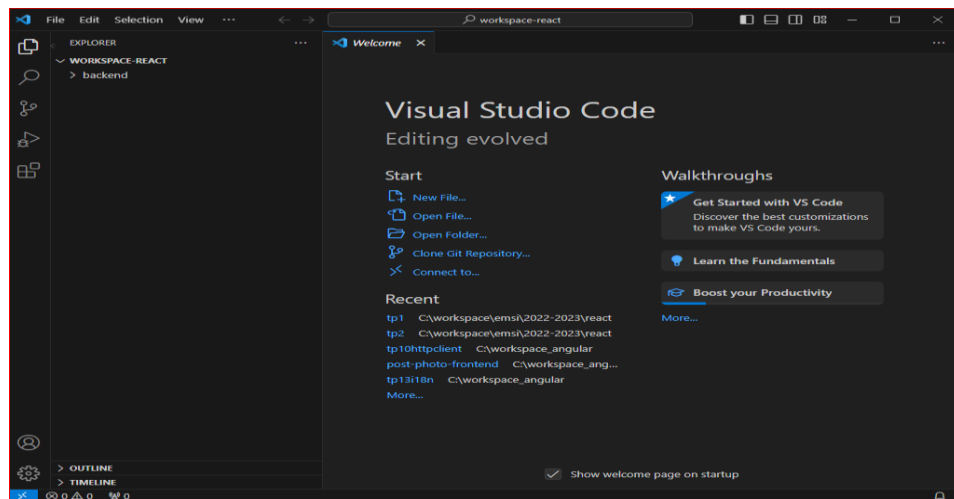
2. Installation de Microsoft Visual Studio Code

- Télécharger VSC de son site principal <https://code.visualstudio.com>. Dans cet atelier, nous allons utiliser la version 1.84.
- Créer le dossier c:\workspace-react se positionner dans ce dernier.
- Lancer la commande « **code .** » :



```
C:\>cd workspace-react
C:\workspace-react>code .
C:\workspace-react>
```

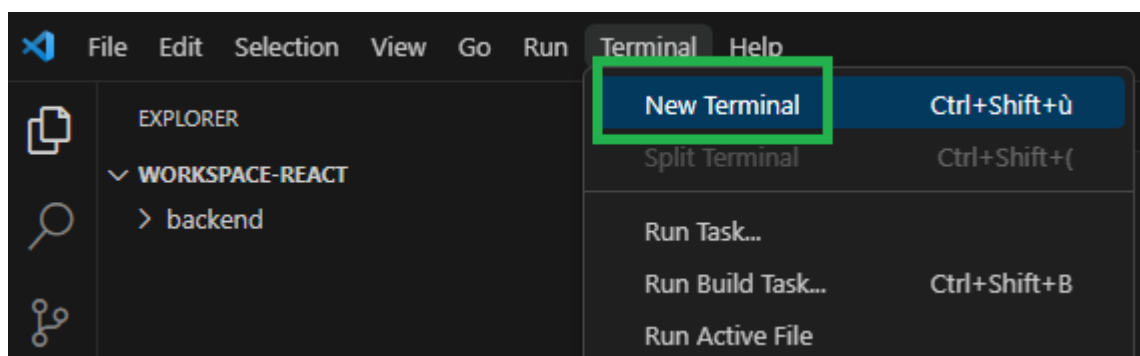
- L'interface de VSD sera affichée :



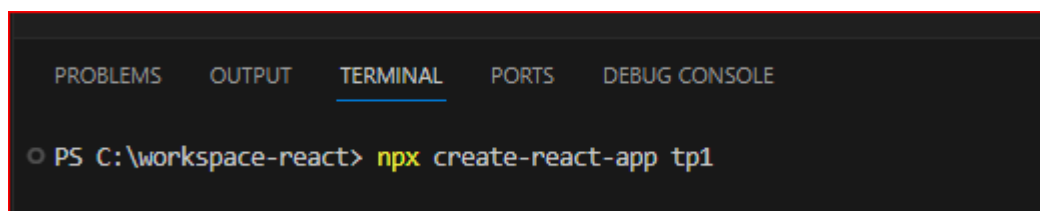
4. Développement de l'application

1. Créer un nouveau projet

- Dans VSD, ouvrir un terminal :



- Lancer la commande suivante :



Explications :

- ✓ **NPX** signifie **Node Package eXecute**. Il s'agit d'un exécuteur de packages NPM. Il permet aux développeurs d'exécuter n'importe quel package Javascript disponible sur le registre NPM sans même l'installer. NPX est installé automatiquement avec NPM version 5.2.0 et supérieure.
- ✓ Pour vérifier si NPX est installé ou non sur votre machine, vous pouvez exécuter la commande suivante sur le terminal :

```
C:\workspace-react>npx -v
10.2.4

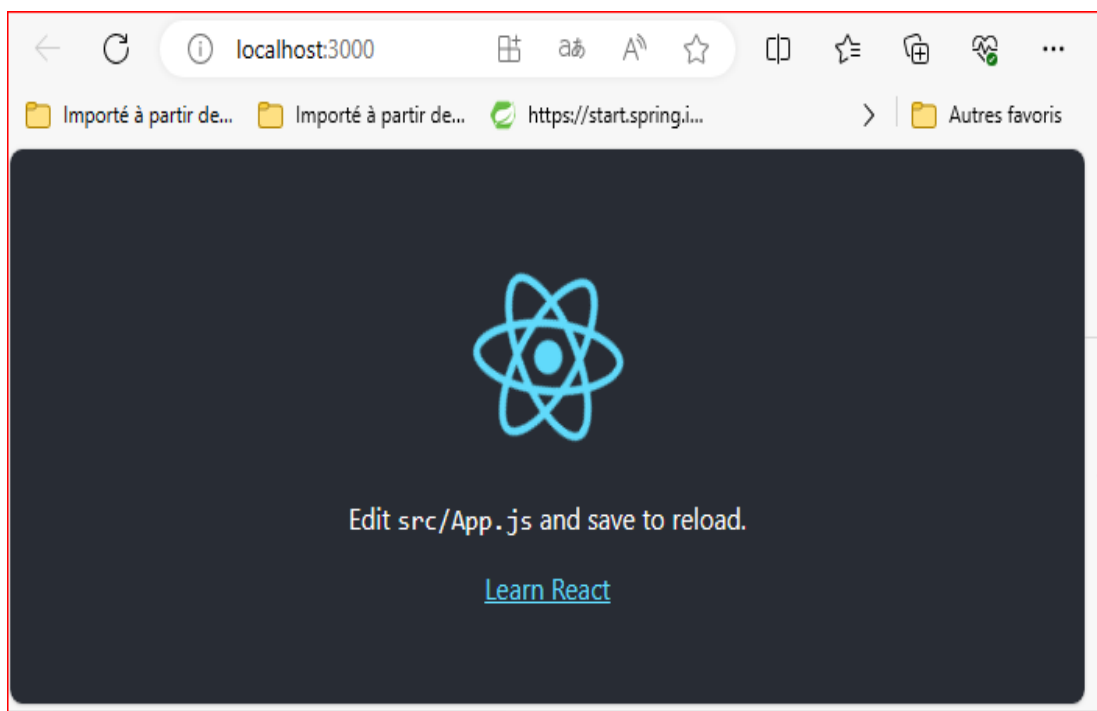
C:\workspace-react>
```

- ✓ **create-react-app** développé par Facebook (<https://create-react-app.dev>).
- ✓ **create-react-app** est un outil permettant la création d'une application SPA (Single Page Application) avec **React** sans aucune configuration.

- Une fois l'application tp1 est créée, se positionner dans le dossier tp1 et lancer la commande : **npm start** :

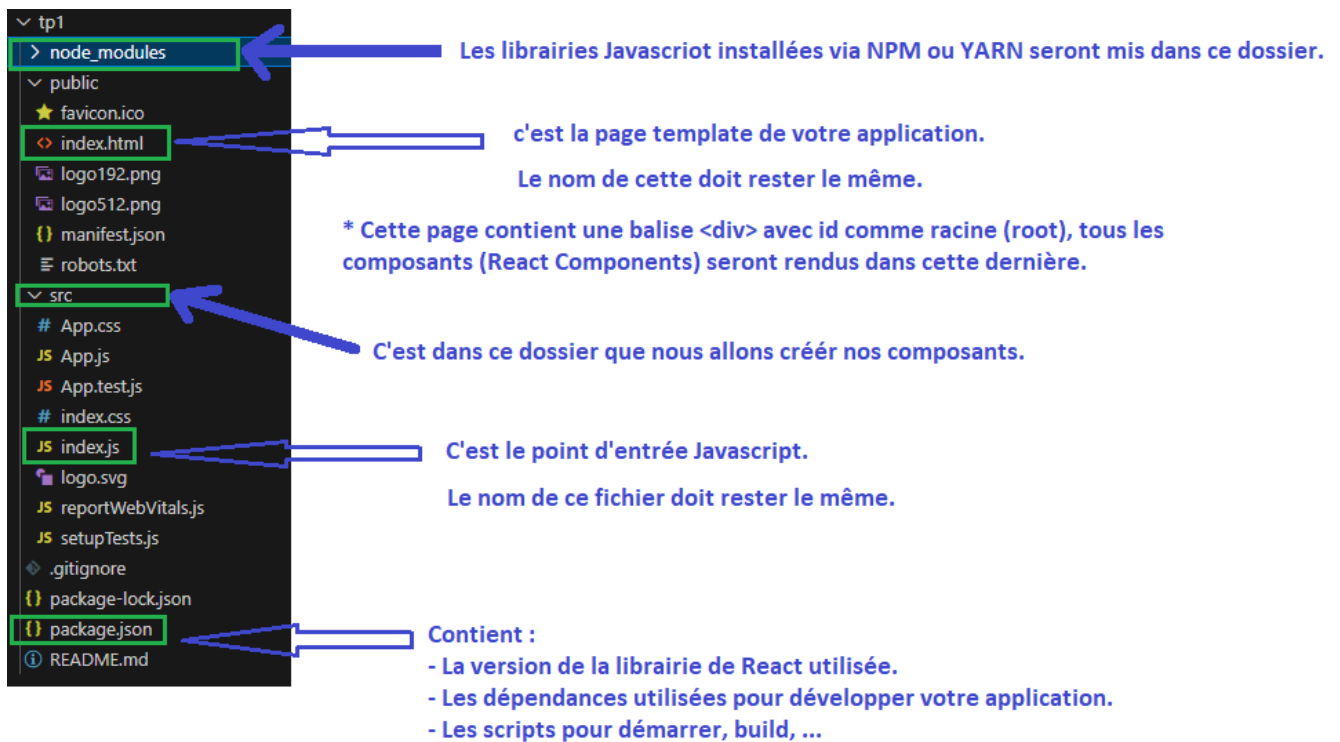
```
PS C:\workspace-react> cd .\tp1\
PS C:\workspace-react\tp1> npm start
```

Le navigateur s'ouvrira automatiquement comme illustré ci-après :



2. L'arborescence de l'application React

- L'imprime écran suivant explique l'arborescence de l'application React générée par l'outil **create-react-app** :



3. Installation de Bootstrap

- Pour ajouter Bootstrap à votre application React, lancer la commande suivante :

```
PS C:\workspace-react> cd tp1
PS C:\workspace-react\tp1> npm i bootstrap --save
```

- Ajouter la ligne suivante au niveau du fichier src/App.js :

```
import "bootstrap/dist/css/bootstrap.css";
```

4. Installation de la librairie AXIOS

- Pour consommer un SW Rest, il existe plusieurs librairies. A titre d'exemple :
 - ✓ Axios
 - ✓ Fetch
 - ✓ Superagent
 - ✓ React-axios
 - ✓ Use-http
 - ✓ React-request
- Dans notre atelier, nous allons utiliser **Axios**. Pour l'installer, lancer la commande suivante :

```
PS C:\workspace-react\tp1> npm add axios
```

5. Développement du service

- Créer le fichier src/api/axisConfig.js suivant :

```
import axios from "axios";
```



```
export default axios.create({
  baseURL: "http://localhost:8080/api/rest/customer",
});
```

NB : Le WS pour consulter la liste des clients est disponible moyennant l'URL :

<http://localhost:8080/api/rest/customer>.

6. Développement du composant CustomerList

- Créer le fichier src/components/CustomerList.jsx suivant :

```
import React from "react";

const CustomerList = ({ customers, editCustomer, deleteCustomer }) => {
  return (
    <table className="table table-hover mt-3" align="center">
      <thead className="thead-light">
        <tr>
          <th scope="col">N°</th>
          <th scope="col">Firstname</th>
          <th scope="col">Lastname</th>
          <th scope="col">Identity Ref</th>
          <th scope="col">Username</th>
          <th scope="col">Option</th>
        </tr>
      </thead>
      <tbody>
        {customers.map((customer, index) => {
          return (
            <tr>
              <td>{customer.id}</td>
              <td>{customer.firstname}</td>
              <td>{customer.lastname}</td>
              <td>{customer.identityRef}</td>
              <td>{customer.username}</td>
              <td>
                <button
                  type="button"
                  className="btn btn-warning"
                  onClick={() => editCustomer(customer)}
                >
                  Edit
                </button>
                <button
                  type="button"
                  className="btn btn-danger mx-2"
                  onClick={() => deleteCustomer(customer.identityRef)}
                >
                  Delete
                </button>
              </td>
            </tr>
          );
        })}
      </tbody>
    </table>
  );
};
```

```

                Delete
            </button>
        </td>
    </tr>
</tbody>
</table>
    );
  })}
</table>
);
};

export default CustomerList;

```

7. Développement du composant CustomerComponent

- Créer le fichier src/components/CustomerComponent.jsx suivant :

```

import { useState } from "react";
import api from "../api/axiosConfig";
import CustomerList from "../CustomerList";

const CustomerComponent = ({ load, customers }) => {
  /* state definition */
  const [id, setId] = useState("");
  const [identityRef, setIdentityRef] = useState("");
  const [firstname, setFirstname] = useState("");
  const [lastname, setLastname] = useState("");
  const [username, setUsername] = useState("");

  async function save(event) {
    event.preventDefault();
    if (id) {
      await api.put("/update/"+identityRef, {
        lastname: lastname,
        firstname: firstname,
        identityRef: identityRef,
        username: username
      });
    } else {
      await api.post("/create", {
        firstname: firstname,
        lastname: lastname,
        identityRef: identityRef,
        username: username
      });
    }
  }
}

```

```

alert("Customer Record Saved");
// reset state
setId("");
setFirstname("");
setLastname("");
setIdentityRef("");
setUsername("");
load();
}

async function editCustomer(customers) {
  setFirstname(customers.firstname);
  setLastname(customers.lastname);
  setIdentityRef(customers.identityRef);
  setUsername(customers.username);
  setId(customers.id);
}

async function deleteCustomer(id) {
  await api.delete("/delete/" + id);
  alert("Customer Details Deleted Successfully");
  load();
}

/* end handlers */

/* jsx */
return (
  <div className="container mt-4">
    <form>
      <div className="form-group my-2">
        <input
          hidden
          type="text"
          className="form-control"
          value={id}
          onChange={e => setId(e.target.value)}
        />
        <label>Lastname</label>
        <input
          type="text"
          className="form-control"
          value={lastname}
          onChange={e => setLastname(e.target.value)}
        />
      </div>
    </div>
  )

```

```

<div className="form-group mb-2">
  <label>Firstname</label>
  <input
    type="text"
    className="form-control"
    value={firstname}
    onChange={e => setFirstname(e.target.value)}
  />
</div>

<div className="row">
  <div className="col-4">
    <label>Identity Ref</label>
    <input
      type="text"
      className="form-control"
      value={identityRef}
      onChange={e => setIdentityRef(e.target.value)}
    />
  </div>
</div>

<div className="row">
  <div className="col-4">
    <label>Username</label>
    <input
      type="text"
      className="form-control"
      value={username}
      placeholder="Customers"
      onChange={e => setUsername(e.target.value)}
    />
  </div>
</div>

<div>
  <button className="btn btn-primary m-4" onClick={save}>
    Save
  </button>
</div>
</form>
<CustomerList
  customers={customers}
  editCustomer={editCustomer}
  deleteCustomer={deleteCustomer}
/>
</div>

```

```
);
};

export default CustomerComponent;
```

Explications :

- **useState** Hook (fonction) permet d'avoir des variables d'état dans les composants fonctionnels. Vous transmettez l'état initial à cette fonction et elle renvoie une variable avec la valeur de l'état actuel (pas nécessairement l'état initial) et une autre fonction pour mettre à jour cette valeur.

8. Modification du fichier App.js

- Modifier le fichier src/App.js comme suit :

```
import "bootstrap/dist/css/bootstrap.css";
import api from "../api/axiosConfig";
import { useEffect, useState } from "react";
import "../App.css";
import CustomerComponent from "../components/CustomerComponent";

function App() {
  const [customers, setCustomers] = useState([]);

  /* manage side effects */
  useEffect(() => {
    (async () => await load())();
  }, []);

  async function load() {
    const result = await api.get("/all");
    setCustomers(result.data);
  }

  return (
    <div>
      <h1 className="text-center">List Of customers</h1>
      <CustomerComponent load={load} customers={customers} />
    </div>
  );
}

export default App;
```

Explications :

- **useEffect** Hook permet d'effectuer des effets secondaires (*side effects*) (une action) dans les « *Function Component* ». Il n'utilise pas les méthodes de cycle de vie des composants disponibles dans les composants de classe. En d'autres termes, **Effects Hooks** sont équivalents aux méthodes de cycle de vie `ComponentDidMount()`, `ComponentDidUpdate()` et `ComponentWillUnmount()`.
- Les effets secondaires (*side effect*) ont des fonctionnalités communes que la plupart des applications Web doivent exécuter, telles que :
 - ✓ Mise à jour du DOM,
 - ✓ Récupérer et consommer des données depuis une API de serveur,
 - ✓ Traitement d'une *Subscription*, etc.

5. Tester votre application

- Accéder au lien <http://localhost:3000> :

List Of customers

LastName

Firstname

Identity Ref

Username

Customers

Save

N°	Firstname	Lastname	Identity Ref	Username	Option
1	FIRST_NAME1	LAST_NAME1	A100	user1	<div>EditDelete</div>
2	FIRST_NAME2	LAST_NAME2	A200	user2	<div>EditDelete</div>
4	FIRST_NAME8	LAST_NAME8	A800	user4	<div>EditDelete</div>
3	FIRST_NAME9	LAST_NAME9	A900	user3	<div>EditDelete</div>

- Ajouter un nouveau client en entrant les informations du client, à savoir : « *firstname* », « *lastname* », « *identity ref* », « *username* » et ensuite cliquer sur le bouton **Save**.
- Pour modifier un client, cliquer sur **Edit** et entrer les modifications :

Lastname
Ali

Firstname
ALAM

Identity Ref
A100

Username
user1

Save

N°	Firstname	Lastname	Identity Ref	Username	Option
1	FIRST_NAME1	LAST_NAME1	A100	user1	<div>EditDelete</div>

- Cliquer ensuite sur Save.
- Pour supprimer un client, cliquer sur Delete.

Conclusion

Le code source de cet atelier est disponible sur GITHUB :

<https://github.com/abbouformations/react-crud-example-axios.git>