Cours Inf3522 - Développement d'Applications N tiers

Lab_10 : Consommation d'une API REST avec React

Ce lab explique l'accès réseau avec React. C'est une compétence vraiment importante dont nous avons besoin dans la plupart des applications React. Nous apprendrons les promesses, qui rendent le code asynchrone plus propre et plus lisible. Pour l'accès réseau, nous utiliserons les bibliothèques fetch et axios.

Utilisation des promesses

La manière traditionnelle de gérer une opération asynchrone consiste à utiliser des fonctions de rappel (callbacks) pour le succès ou l'échec de l'opération. Une de ces fonctions de rappel est appelée (succès ou échec), en fonction du résultat de l'appel. L'exemple suivant montre l'idée d'utilisation d'une fonction de rappel :

```
function doAsyncCall(success, failure) {
 // Faire un appel API
 if (REUSSI)
   success(resp);
 else
   failure(err);
}

function success(response) {
 // Faire quelque chose avec la réponse
}

function failure(error) {
 // Gérer l'erreur
}
doAsyncCall(success, failure);
```

Une promesse est un objet qui représente le résultat d'une opération asynchrone. L'utilisation de promesses simplifie le code lors de l'exécution d'appels asynchrones. Les promesses sont non bloquantes.

Une promesse peut se trouver dans l'un des trois états suivants :

• **En attente (pending)** : État initial

• **Accomplie (fulfilled)** : Opération réussie

• **Rejetée (rejected)** : Opération échouée

Avec les promesses, nous pouvons effectuer des appels asynchrones si l'API que nous utilisons prend en charge les promesses. Dans l'exemple suivant, l'appel asynchrone est effectué et lorsque la réponse est renvoyée, la fonction de rappel à l'intérieur de la méthode **then** est exécutée en prenant la réponse comme argument :

```
doAsyncCall()
.then(response => // Faire quelque chose avec la réponse)
```

La méthode **then** renvoie une promesse. Vous pouvez enchaîner plusieurs instances de **then** ensemble, ce qui signifie que vous pouvez exécuter plusieurs opérations asynchrones les unes après les autres :

```
doAsyncCall()
        .then(response => // Obtenir des données à partir de la réponse)
        .then(data => // Faire quelque chose avec les données)
```

Vous pouvez également ajouter une gestion des erreurs aux promesses en utilisant catch():

```
doAsyncCall()
        .then(response => // Obtenir des données à partir de la réponse)
        .then(data => // Faire quelque chose avec les données)
        .catch(error => console.error(error))
```

Il existe une manière plus moderne de gérer les appels asynchrones qui implique **async/await**, introduit dans ECMAScript 2017. Pour utiliser **async/await**, vous devez définir une fonction asynchrone qui peut contenir des expressions **await**. L'exemple suivant montre un appel asynchrone contenant **async/await**. Comme vous pouvez le voir, vous pouvez écrire le code de manière similaire au code synchrone :

```
doAsyncCall = async () => {
 const response = await fetch('http://someapi.com');
 const data = await response.json();
 // Faire quelque chose avec les données
}
```

Pour la gestion des erreurs, vous pouvez utiliser **try...catch** avec **async/await**, comme le montre l'exemple suivant :

```
doAsyncCall = async () => {
 try {
  const response = await fetch('http://someapi.com');
  const data = await response.json();
  // Faire quelque chose avec les données
 } catch (err) {
  console.error(err);
 }
}
```

Maintenant, nous pouvons commencer à apprendre sur l'API **fetch**, que nous pouvons utiliser pour effectuer des requêtes dans nos applications React.

Utilisation de l'API fetch

Avec l'API fetch, vous pouvez effectuer des requêtes web. L'idée de l'API fetch est similaire à celle de l'XMLHttpRequest traditionnel, mais l'API fetch prend également en charge les promesses, ce qui la rend plus simple à utiliser. Vous n'avez pas besoin d'installer de bibliothèques supplémentaires si vous utilisez fetch.

L'API fetch fournit une méthode **fetch()** qui a un argument obligatoire : le chemin de la ressource que vous appelez. Dans le cas d'une requête web, il s'agira de l'URL du service. Pour un simple appel de méthode GET, qui renvoie une réponse JSON, la syntaxe est la suivante :

```
fetch('http://someapi.com')
        .then(response => response.json())
        .then(data => console.log(data))
        .catch(error => console.error(error))
```

Pour utiliser une autre méthode HTTP, telle que POST, vous devez la définir dans le deuxième argument de la méthode fetch(). Le deuxième argument est un objet dans lequel vous pouvez définir plusieurs paramètres de requête. Le code source suivant effectue la requête en utilisant la méthode POST :

```
fetch('http://someapi.com', {method: 'POST'})
        .then(response => response.json())
        .then(data => console.log(data))
        .catch(error => console.error(error));
```

Vous pouvez également ajouter des en-têtes (headers) dans le deuxième argument. L'appel fetch() suivant contient l'en-tête **'Content-Type':'application/json'** :

```
fetch('http://someapi.com', {
        method: 'POST',
        headers: {'Content-Type':'application/json'}
})
        .then(response => response.json())
        .then(data => console.log(data))
        .catch(error => console.error(error));
```

Si vous devez envoyer des données encodées en JSON dans le corps de la requête, la syntaxe est la suivante :

```
fetch('http://someapi.com', {
        method: 'POST',
        headers: {'Content-Type':'application/json'},
        body: JSON.stringify(data)
})
        .then(response => response.json())
        .then(data => console.log(data))
        .catch(error => console.error(error));
```

L'API fetch n'est pas la seule façon d'effectuer des requêtes dans une application React. Il existe également d'autres bibliothèques que vous pouvez utiliser. Dans la prochaine section, nous apprendrons comment utiliser l'une de ces bibliothèques populaires : **axios**.

Utilisation de la bibliothèque axios

Vous pouvez également utiliser d'autres bibliothèques pour les appels réseau. Une bibliothèque très populaire est axios (https://github.com/axios/axios), que vous pouvez installer dans votre application React avec npm :

```
npm install axios
```

Vous devez ajouter la commande d'importation suivante dans votre composant React avant de l'utiliser :

import axios from 'axios';

La bibliothèque axios présente certains avantages, tels que la transformation automatique des données JSON. Le code suivant montre un exemple d'appel effectué avec axios :

```
axios.get('http://someapi.com')
        .then(response => console.log(response))
        .catch(error => console.log(error));
```

La bibliothèque axios possède ses propres méthodes d'appel pour les différentes méthodes HTTP. Par exemple, si vous souhaitez effectuer une requête POST et envoyer un objet dans le corps, axios fournit la méthode axios.post :

```
axios.post('http://someapi.com', { newObject })
        .then(response => console.log(response))
        .catch(error => console.log(error));
```

Maintenant, nous sommes prêts à examiner des exemples pratiques impliquant les appels réseau avec React.

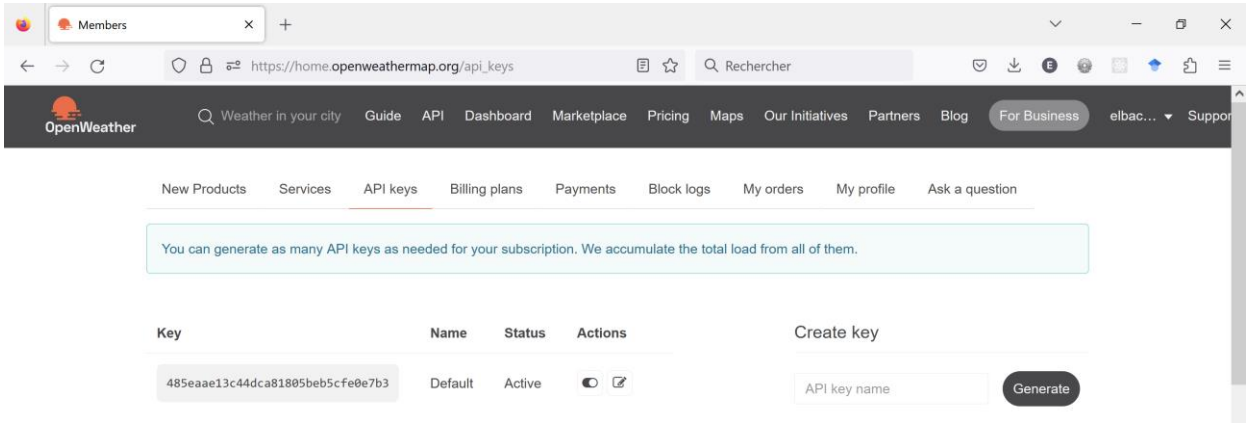
Exemple pratique

Dans cette partie, nous passerons en revue un exemple d'utilisation d'une API REST dans votre application React.

API OpenWeatherMap

Tout d'abord, nous créerons une application React qui affiche la météo actuelle à **Ziguinchor**. Ces données météorologiques seront récupérées depuis OpenWeatherMap (https://openweathermap.org/).

Vous devez vous inscrire auprès d'OpenWeatherMap pour obtenir une clé API. Nous utiliserons un compte gratuit car cela suffit à nos besoins. Une fois inscrit, accédez aux informations de votre compte pour trouver l'onglet "API Keys". Vous y trouverez la clé API dont vous avez besoin pour votre application météo React :



Créons une nouvelle application React avec **create-react-app** :

1. Ouvrez un terminal sous Windows ou Terminal sous macOS/Linux et tapez la commande suivante :

npx create-react-app weatherapp

2. Accédez au dossier weatherapp :

cd weatherapp

3. Démarrez votre application avec la commande suivante :

npm start

4. Ouvrez votre dossier de projet avec VS Code et ouvrez le fichier App.js dans l'éditeur. Supprimez tout le code à l'intérieur de la balise ** **. Votre code source devrait maintenant ressembler à ceci :

```
import React, { useState } from 'react';
import './App.css';
function App() {
 const [temp, setTemp] = useState('');
 const [desc, setDesc] = useState('');
 const [icon, setIcon] = useState('');
 const [isReady, setReady] = useState(false);
 return (
  <div className="App">
  </div>
 );
}

export default App;
```

5. Tout d'abord, nous devons ajouter les états nécessaires pour les données de réponse. Nous afficherons la température, la description et l'icône météo dans notre application. Nous devons donc définir trois valeurs d'état. Nous ajouterons également un état booléen pour indiquer l'état du chargement de la requête :

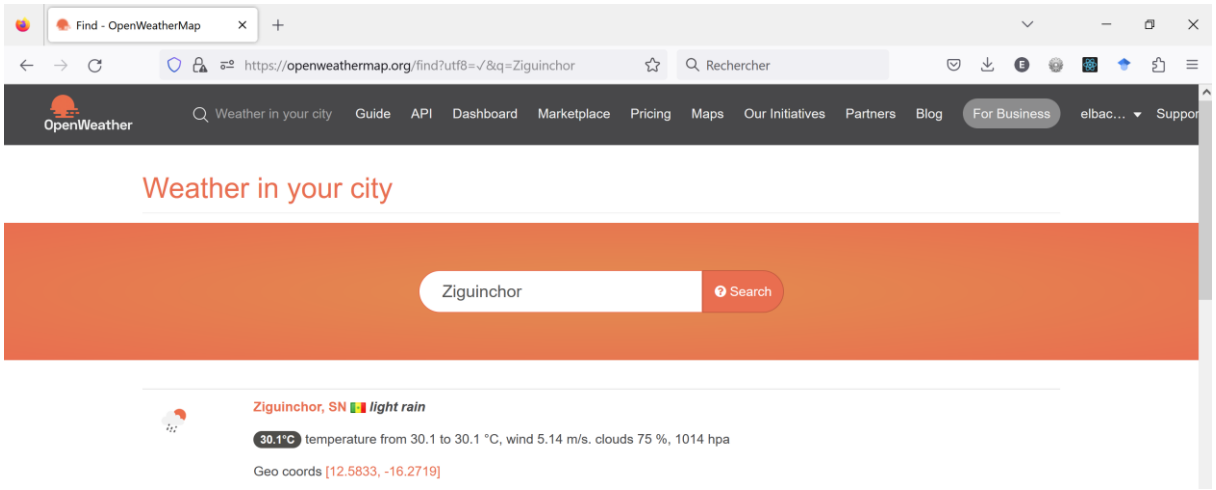
```
import React, { useState } from 'react';
import './App.css';
function App() {
 const [temp, setTemp] = useState('');
 const [desc, setDesc] = useState('');
 const [icon, setIcon] = useState('');
 const [isReady, setReady] = useState(false);
 return (
  <div className="App">
  </div>
 );
}
export default App;
```

Lorsque vous utilisez une API REST, vous devez inspecter la réponse pour pouvoir obtenir les valeurs à partir des données JSON. Dans l'exemple suivant, vous pouvez voir l'adresse qui renvoie les

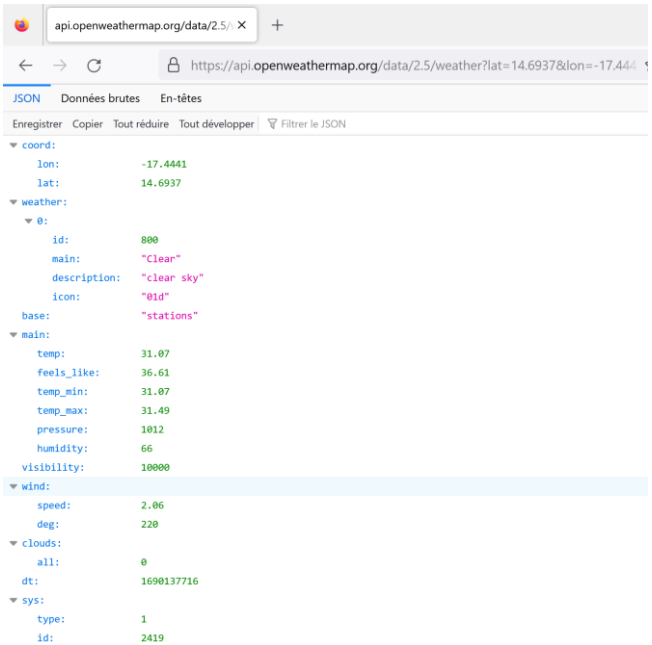
conditions météorologiques actuelles pour Ziguinchor. Si vous copiez cette adresse dans un navigateur, vous pourrez voir les données JSON de la réponse.

https://api.openweathermap.org/data/2.5/weather?lat=14.6937&lon=-17.4441&appid=485eaae13c44dca81805beb5cfe0e7b3&units=metric

NB : j'ai affiché la météo pour Dakar, faut rechercher dans OpenWeather la ville de Ziguinchor et récupérer les coordonnées de Ziguinchor :



À partir de la réponse, vous pouvez voir que la température (temp) peut être obtenue en utilisant main.temp. Ensuite, vous pouvez voir que la description (description) et l'icône (icon) se trouvent à l'intérieur du tableau weather, qui ne contient qu'un seul élément, et que nous pouvons y accéder en utilisant weather[o].description et weather[o].icon.

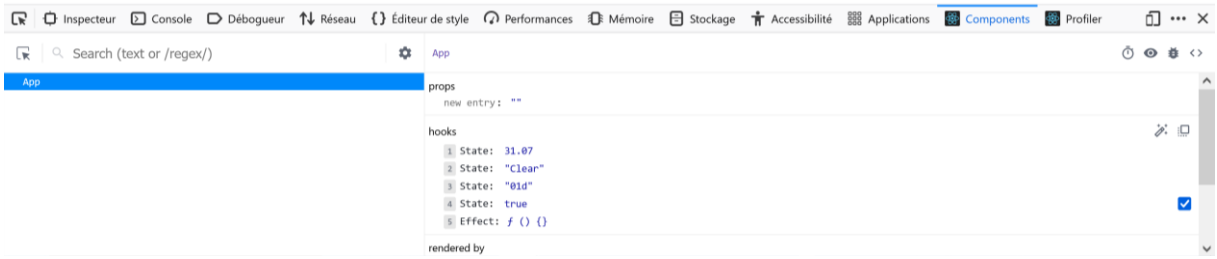


Le recours à l'API REST est effectué à l'aide de fetch dans la fonction useEffect, en utilisant un tableau vide comme deuxième argument. Ainsi, la requête est effectuée une seule fois, après le premier rendu. Après une réponse réussie, nous enregistrons les données météorologiques dans l'état (state) et nous changeons l'état isReady à true. Une fois que les valeurs de l'état ont été modifiées, le composant sera réaffiché. Nous mettrons en œuvre l'instruction return dans la prochaine étape.

6. Le code source suivant est destiné à la fonction useEffect, qui exécute la fonction fetch une seule fois après le premier rendu :

```
React.useEffect(() => {
fetch('https://api.openweathermap.org/data/2.5/weather?lat=14.6937&lon=-
17.4441&appid=VOTRE_CLE&units=metric ')
        .then(result => result.json())
        .then(jsonresult => {
         setTemp(jsonresult.main.temp);
         setDesc(jsonresult.weather[0].main);
         setIcon(jsonresult.weather[0].icon);
         setReady(true);
        })
        .catch(err => console.error(err))
}, [])
```

7. Une fois que vous avez ajouté la fonction useEffect, la requête est exécutée après le premier rendu. Vous pouvez vérifier que tout s'est déroulé correctement en utilisant l'outil de développement React. Ouvrez votre application dans un navigateur et ouvrez l'onglet "Component" de l'outil de développement React. Maintenant, vous pouvez voir que les états ont été mis à jour avec les valeurs de la réponse :



Vous pouvez également vérifier que l'état de la requête est "200 OK" dans l'onglet "Network". Enfin, nous mettons en œuvre l'instruction return pour afficher les valeurs météorologiques. Nous utilisons ici le rendu conditionnel ; sinon, nous aurons une erreur car le code de l'image n'est pas présent dans le premier appel de rendu et le téléchargement de l'image échouera.

8. Pour afficher l'icône météo, nous devons ajouter http://openweathermap.org/img/wn/ avant le code de l'icône et @2x.png après le code de l'icône. Ensuite, nous pouvons définir l'URL de l'image concaténée comme attribut src de l'élément img. La température (temperature) et la description (description) sont affichées dans l'élément de paragraphe (p). L'entité HTML °C montre le symbole degré Celsius :

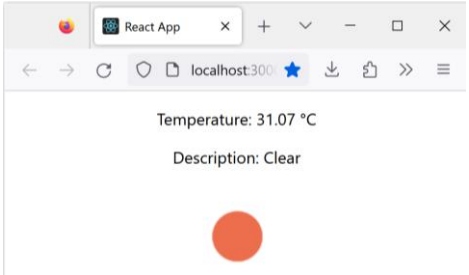
```
if (isReady) {
 return (
  <div className="App">
   <p>Température : {temp} °C</p>
   <p>Description : {desc}</p>
   <img src= {`http://openweathermap.org/img/wn/${icon}@2x.png`} alt="Icône météo" />
  </div>
 );
} else {
 return <div>Chargement en cours...</div>;
```

```
}
```

9. Le code complet de App.js est donné ci-dessous :

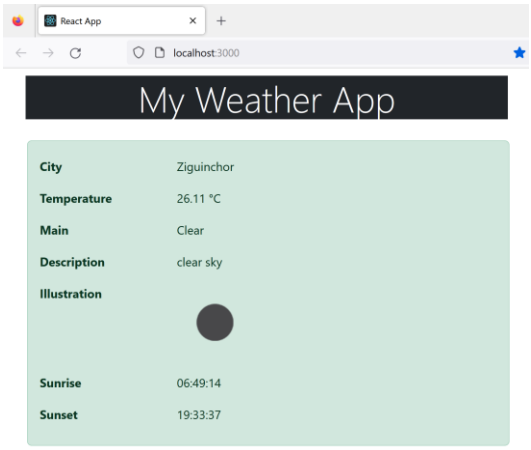
```
import React, { useState } from "react";
import "./App.css";
function App() {
 const [temp, setTemp] = useState("");
 const [desc, setDesc] = useState("");
 const [icon, setIcon] = useState("");
 const [isReady, setReady] = useState(false);
 React.useEffect(() => {
  fetch(
   "https://api.openweathermap.org/data/2.5/weather?lat=14.6937&lon=-
17.4441&appid=VOTRE_CLE&units=metric"
  )
   .then(result => result.json())
   .then(jsonresult => {
    setTemp(jsonresult.main.temp);
    setDesc(jsonresult.weather[0].main);
    setIcon(jsonresult.weather[0].icon);
    setReady(true);
   })
   .catch(err => console.error(err));
 }, []);
 if (isReady) {
  return (
   <div className="App">
    <p>Temperature: {temp} °C</p>
    <p>Description: {desc}</p>
    <img
     src={`http://openweathermap.org/img/wn/${icon}@2x.png`}
     alt="Weather icon"
    />
   </div>
  );
 } else {
  return <div>Loading...</div>;
 }
}
export default App;
```

10. Votre application devrait maintenant être prête et devrait ressembler à ceci :



Exercice 1

Modifiez l'exemple précédent pour afficher plus d'informations sur la météo de Ziguinchor comme suit :



Pour l'interface de votre application, je m'attends à mieux de votre part ☹.

Exercice 2

Ajoutez une zone de texte dans laquelle récupérer la longitude et la latitude comme suit :

NB : Le thème de l'interface s'adapte à la température.