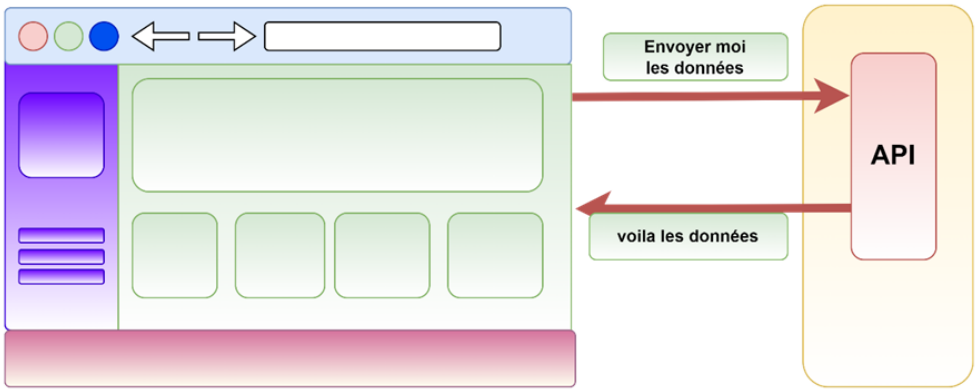
Consommation des API

Rappel Théorique :

1. Qu'est-ce qu'une API ?

- Une **API** (*Application Programming Interface*) est une **interface** qui permet à deux programmes ou systèmes de **communiquer et d'échanger des données**.
- Elle définit **des règles, des formats et des méthodes** (comme GET, POST, PUT, DELETE) que les applications utilisent pour **envoyer des requêtes** et **recevoir des réponses**.
- Une API agit comme un **pont** entre ton application (par exemple une app React) et une **source de données externe** (comme un serveur).
- Chaque ressource accessible via une API est identifiée par une **URL spécifique**, qu'on appelle un **endpoint**.
- Exemple : dans l'API *JSONPlaceholder*, l'endpoint `https://jsonplaceholder.typicode.com/todos` permet d'obtenir la **liste des tâches (todos)** que ton application peut ensuite afficher



2. Le Format JSON

Le **JSON** (*JavaScript Object Notation*) est le format de données standard des API Web. Il est utilisé parce qu'il est **facilement convertible** en objets JavaScript par React, ce qui est essentiel pour afficher les informations. En JSON, les données sont représentées sous forme de **paires clé–valeur**, entourées d'accolades { }.

Exemple :

```
{
  "id": 1,
  "title": "Apprendre React",
  "completed": false,
  "userId": 3
}
```

Ici :

- `"id"`, `"title"`, `"completed"` et `"userId"` sont les **clés**
- `1`, `"Apprendre React"`, `false` et `3` sont leurs **valeurs**

3. Les Méthodes HTTP

Les **méthodes HTTP** indiquent **l'action** que le client (comme une application React) veut effectuer sur les données d'un serveur via une API :

Methode	Rôle	Action (CRUD)
GET	Récupérer des données.	**Read** (Lire)
POST	Envoyer de nouvelles données.	**Create** (Créer)
PUT / PATCH	Modifier des données existantes.	**Update** (Mettre à jour)
DELETE	Supprimer des données.	**Delete** (Supprimer)

4. Axios et les Promesses

REACT n'est pas responsable pour consommer un API,c'est le rôle de développeur d'ajouter la partie du code responsable de cette operation (utiliser par exemple fetch,axios.....) .

Axios est la librairie que nous utilisons dans React pour faire ces requêtes. Une requête prend du temps (**asynchrone**). Axios gère cela avec les **Promesses** :

- **`axios.get(url)`** renvoie une Promesse.
- **`.then(reponse)`** s'exécute quand la requête est réussie.
- **`.catch(erreur)`** s'exécute en cas d'échec (erreur réseau, etc.).

II. Préparation : Axios et React Hooks

1. Installation d'Axios

Dans votre terminal :

```
npm install axios
```

2. Les Hooks Essentiels

Nous utiliserons deux hooks dans React :

- **`useState` :** Pour **stocker** les données reçues et le statut de chargement.
- **`useEffect` :** Pour déclencher la requête **une seule fois** au démarrage du composant (`[]`).

3. useEffect Hook :

`useEffect` est un **Hook React** qui permet d'exécuter du **code après le rendu du composant** (comme une action "secondaire" ou *effet*).

- Il est utilisé pour :
  - **Appeler une API** (récupérer des données)
  - **Mettre à jour le titre de la page**
  - **Gérer des timers**, des **abonnements**, ou du **localStorage**
- Structure de base :

```
useEffect(() => {
            // Code à exécuter après le rendu
      });
```

- Avec dépendances :

```
useEffect(() => {
      // Code exécuté quand "count" change
}, [count]);
```

- Le tableau `[count]` est appelé **tableau de dépendances**.

- L'effet s'exécute **uniquement** quand la variable `count` change.
- Si le tableau est **vide [ ]**, le code ne s'exécute **qu'une seule fois**, au **montage** du composant.

III. Exemple Applicable

Nous allons utiliser l'API de test **JSONPlaceholder** (URL de base : https://jsonplaceholder.typicode.com).

Exemple A : Lire les Données (GET)

Objectif : Afficher une liste de posts au démarrage du composant.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
function ExempleGet() {
  const [posts, setPosts] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // 1. Définir l'URL
    const API_URL = 'https://jsonplaceholder.typicode.com/posts?_limit=5';

    setLoading(true);

    // 2. Requête GET et gestion de la promesse
    axios.get(API_URL)
      .then(response => {
        // Succès : stocker les données (response.data)
        setPosts(response.data);
      })
      .catch(error => {
        // Échec : afficher l'erreur dans la console
        console.error("Erreur GET:", error);
      })
      .finally(() => {
        // Terminé : cacher l'indicateur de chargement
        setLoading(false);
      });
  },
```

```
 []);

  if (loading) return <p>Chargement des posts...</p>;

  return (
    <div>
      <h3>A. Exemple GET (Lire)</h3>
      {posts.map(post => (
        <div key={post.id}>
          <h4>{post.title}</h4>
          <p>{post.body.substring(0, 50)}...</p>
        </div>
      ))}
    </div>
  );
}
export default ExempleGet;
```

Exemple B : Créer des Données (POST)

Objectif : Créer et envoyer un nouveau post au serveur lors du clic sur un bouton.

```
// ... (Les imports useState et axios sont nécessaires)

function ExemplePost() {
  const [postMessage, setPostMessage] = useState("Cliquez pour créer un
post.");

  const handlePost = () => {
    const API_URL = 'https://jsonplaceholder.typicode.com/posts';

    // 1. Définir l'objet de données à envoyer
    const newPost = {
      title: 'Nouveau Post',
      body: 'text .....',
      userId: 99,
    };
    // changement de postMessage
    setPostMessage("Envoi en cours...");
```

```
    // 2. Requête POST : axios.post(URL, données)
    axios.post(API_URL, newPost)
      .then(response => {
        // Succès : Le serveur renvoie l'objet créé (avec son nouvel ID)

        console.log("Objet créé:", response.data);
      })
      .catch(error => {

        console.error("Erreur POST:", error);
      });
  };

  return (
    <div>
      <h3>B. Exemple POST (Créer)</h3>
      <button onClick={handlePost}>Envoyer un Nouveau Post</button>
      <p>{postMessage}</p>
    </div>
  );
}
export default ExemplePost;
```

Exemple C : Modifier des Données (PUT)

Objectif : Modifier complètement le post d'ID 1.

```
// ... (Les imports useState et axios sont nécessaires)

function ExemplePut() {

  const handlePut = () => {
    const POST_ID = 1;
    const API_URL = `https://jsonplaceholder.typicode.com/posts/${POST_ID}`;

    // 1. Définir les nouvelles données
    const updatedPost = {
      id: POST_ID,
      title: 'TITRE MODIFIÉ COMPLETEMENT PAR PUT',
      body: 'Le corps entier a été réécrit.',
```

```
      userId: 1,
    };


    // 2. Requête PUT : axios.put(URL/ID, nouvelles_données)
    axios.put(API_URL, updatedPost)
      .then(response => {
            console.log("Objet mis à jour:", response.data);
      })
      .catch(error => {
             console.error("Erreur PUT:", error);
      });
  };

  return (
    <div>
      <h3>C. Exemple PUT (Modifier)</h3>
      <button onClick={handlePut}>Modifier le Post ID 1</button>
          </div>
  );
}
export default ExemplePut;
```

Exemple D : Supprimer des Données (DELETE)

Objectif : Supprimer le post d'ID 3.

```
// ... (Les imports useState et axios sont nécessaires)

function ExempleDelete() {


  const handleDelete = () => {
    const POST_ID = 3;
    const API_URL = `https://jsonplaceholder.typicode.com/posts/${POST_ID}`;


    // 2. Requête DELETE : axios.delete(URL/ID)
    axios.delete(API_URL)
      .then(() => {
```

```
        // Succès : La réponse est souvent vide, on confirme la suppression
        console.log(“deleted”)
      })
      .catch(error => {
                console.error("Erreur DELETE:", error);
      });
  };

  return (
    <div>
      <h3>D. Exemple DELETE (Supprimer)</h3>
      <button onClick={handleDelete}>Supprimer le Post ID 3</button>
         </div>
  );
}
export default ExempleDelete;
```

Exercice 1 :

Objectif : Créer un composant React capable d'afficher une liste de tâches, de gérer leur état de complétion, et d'implémenter un filtrage simple basé sur la propriété userId (sans avoir à récupérer les noms d'utilisateurs).

1. La Source de Données (API)

Vous utiliserez le point de terminaison /todos de JSONPlaceholder.

- URL des tâches : https://jsonplaceholder.typicode.com/todos

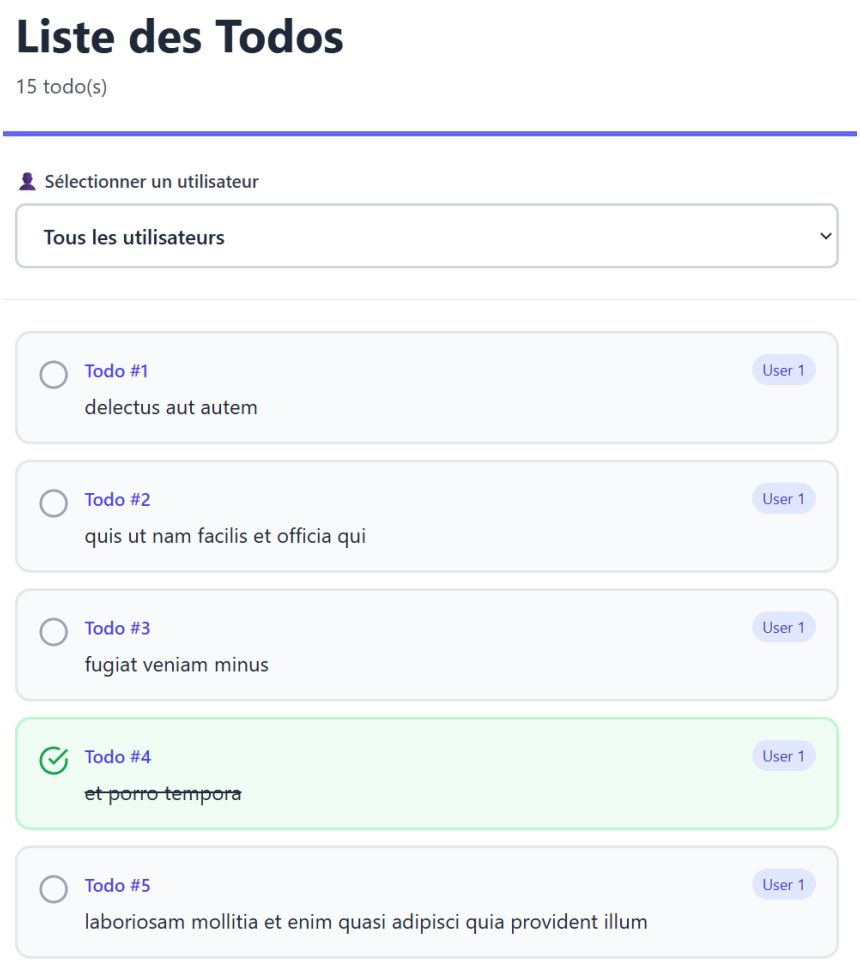
2. Étapes de Mise en Œuvre

Étape A : Récupération et Affichage (GET)

1. Au montage du composant, utilisez Axios (useEffect) pour effectuer une requête GET et récupérer toutes les tâches (/todos).
2. Stockez ces données dans un état (useState).
3. Affichez la liste des tâches (au moins le titre et l'ID de l'utilisateur) :
    a. Les tâches dont le champ completed est true doivent être visuellement distinguées (ex: couleur verte ou barrées).

Étape B : Le Filtrage par ID d'Utilisateur

1. Créez un menu déroulant (<select>) pour simuler la sélection d'un utilisateur. Les options seront :
    a. "Tous les utilisateurs" (pour afficher toutes les tâches).
    b. Les ID d'utilisateurs uniques présents dans les données récupérées (ex: 1, 2, 3, etc.).
2. Lorsque la sélection change, filtrez la liste des tâches affichées pour ne montrer que celles qui correspondent à l'ID utilisateur sélectionné (todo.userId).


Liste des Todos
15 todo(s)
Sélectionner un utilisateur
Tous les utilisateurs
Todo #1
User 1
delectus aut autem
Todo #2
User 1
quis ut nam facilis et officia qui
Todo #3
User 1
fugiat veniam minus
Todo #4
User 1
et porro tempora
Todo #5
User 1
laboriosam mollitia et enim quasi adipisci quia provident illum