Dans Angular, bien que le client HTTP natif (HttpClient) soit souvent utilisé, vous pouvez aussi utiliser **Axios** pour effectuer des requêtes HTTP. Voici comment créer un service HTTP avec **Axios** dans Angular :

**📌 Étape 1 : Installer Axios**

Dans votre projet Angular, installez **Axios** avec la commande suivante :

npm install axios

**📌 Étape 2 : Créer un Service Angular**

Générez un service Angular via la CLI :

ng generate service services/api

Cela crée un fichier api.service.ts dans src/app/services/.

**📌 Étape 3 : Implémenter Axios dans le Service**

Modifiez api.service.ts pour inclure **Axios** :

import { Injectable } from '@angular/core';

import axios, { AxiosInstance } from 'axios';

@Injectable({

providedIn: 'root'

})

export class ApiService {

private axiosInstance: AxiosInstance;

constructor() {

this.axiosInstance = axios.create({

baseURL: 'https://jsonplaceholder.typicode.com', // Remplacez par votre API

timeout: 5000,

headers: {

'Content-Type': 'application/json'

}

});

}

// GET Request

async getPosts() {

try {

const response = await this.axiosInstance.get('/posts');

return response.data;

} catch (error) {

console.error('Erreur lors de la récupération des posts', error);

throw error;

}

}

// POST Request

async createPost(postData: any) {

try {

const response = await this.axiosInstance.post('/posts', postData);

return response.data;

} catch (error) {

console.error('Erreur lors de la création du post', error);

throw error;

}

}

// PUT Request

async updatePost(id: number, postData: any) {

try {

const response = await this.axiosInstance.put(`/posts/${id}`, postData);

return response.data;

} catch (error) {

console.error('Erreur lors de la mise à jour du post', error);

throw error;

}

}

// DELETE Request

async deletePost(id: number) {

try {

await this.axiosInstance.delete(`/posts/${id}`);

return { message: 'Post supprimé avec succès' };

} catch (error) {

console.error('Erreur lors de la suppression du post', error);

throw error;

}

}

}

**📌 Étape 4 : Utiliser le Service dans un Composant**

Dans un composant Angular (app.component.ts par exemple), injectez et utilisez le service :

import { Component, OnInit } from '@angular/core';

import { ApiService } from './services/api.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

posts: any[] = [];

constructor(private apiService: ApiService) {}

ngOnInit() {

this.fetchPosts();

}

async fetchPosts() {

try {

this.posts = await this.apiService.getPosts();

} catch (error) {

console.error('Erreur lors du chargement des posts', error);

}

}

}

**📌 Étape 5 : Afficher les Données dans le Template**

Dans app.component.html :

<h1>Liste des Posts</h1>

<ul>

<li \*ngFor="let post of posts">

<strong>{{ post.title }}</strong>

<p>{{ post.body }}</p>

</li>

</ul>

**✅ Pourquoi utiliser Axios dans Angular ?**

✔️ Syntaxe simple et intuitive  
✔️ Gestion des promesses et async/await  
✔️ Support natif des requêtes JSON  
✔️ Meilleure gestion des erreurs et des timeouts

C'est tout ! 🎉 Vous avez maintenant un **service Angular basé sur Axios**. 🚀

Depuis Angular 16, **Signals** sont disponibles et offrent une manière réactive et performante de gérer l'état global dans une application Angular sans utiliser des solutions comme NgRx ou Akita.

## 📌 **Étape 1 : Créer le Service Global de State Management**

Dans Angular, on peut créer un service qui maintient un **state global** en utilisant des Signals.

Générez un service avec Angular CLI :

ng generate service services/global-state

Cela crée un fichier global-state.service.ts dans src/app/services/.

## 📌 **Étape 2 : Implémenter le Service avec Signals**

Modifiez global-state.service.ts pour utiliser **Signal** :

import { Injectable, signal, WritableSignal } from '@angular/core';

@Injectable({

providedIn: 'root',

})

export class GlobalStateService {

// Définition des Signaux

private \_counter: WritableSignal<number> = signal(0);

private \_username: WritableSignal<string> = signal('Invité');

// Exposition en tant que getters

get counter() {

return this.\_counter;

}

get username() {

return this.\_username;

}

// Méthodes pour modifier l'état

incrementCounter(): void {

this.\_counter.set(this.\_counter() + 1);

}

decrementCounter(): void {

this.\_counter.set(this.\_counter() - 1);

}

resetCounter(): void {

this.\_counter.set(0);

}

setUsername(newUsername: string): void {

this.\_username.set(newUsername);

}

}

## 📌 **Étape 3 : Utiliser le Service dans un Composant**

Dans un composant comme app.component.ts, injectez et utilisez le **GlobalStateService**.

🔹 **Injection du Service et usage des Signaux** :

import { Component } from '@angular/core';

import { GlobalStateService } from './services/global-state.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

constructor(public globalState: GlobalStateService) {}

changeUsername() {

this.globalState.setUsername('Utilisateur' + Math.floor(Math.random() \* 100));

}

}

## 📌 **Étape 4 : Affichage des Données dans le Template**

Dans app.component.html :

<h1>Gestion Globale d'État avec Signals</h1>

<p>Nom d'utilisateur : {{ globalState.username() }}</p>

<button (click)="changeUsername()">Changer le Nom</button>

<p>Compteur : {{ globalState.counter() }}</p>

<button (click)="globalState.incrementCounter()">➕ Augmenter</button>

<button (click)="globalState.decrementCounter()">➖ Diminuer</button>

<button (click)="globalState.resetCounter()">🔄 Réinitialiser</button>

## ✅ **Pourquoi Utiliser Signals pour le State Management ?**

✔ **Réactivité automatique** : mise à jour immédiate des composants liés  
✔ **Meilleure performance** : pas besoin de ChangeDetectionStrategy.OnPush  
✔ **Facilité d'utilisation** : pas de besoin de BehaviorSubject ou NgRx

🚀 **Avec cette implémentation, vous avez un service de gestion d’état global basé sur Angular Signals** ! 🎉