**1. Create a Standalone Service**

TypeScript

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

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({ providedIn: 'root' })

export class ApiService {

private http = inject(HttpClient);

getData(): Observable<any[]> {

return this.http.get<any[]>('/api/endpoint');

}

// ... other HTTP methods

}

* **inject(HttpClient):** This is how you inject dependencies in standalone services.
* **providedIn: 'root':** This ensures the service is available throughout the application.

**2. Use the Service in a Standalone Component**

TypeScript

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

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

@Component({

selector: 'app-my-component',

standalone: true,

imports: [/\* Import any necessary modules here \*/],

template: `

<ul>

<li \*ngFor="let item of data">{{ item }}</li>

</ul>

`

})

export class MyComponent {

data: any[] = [];

constructor(private apiService: ApiService) {}

ngOnInit() {

this.apiService.getData().subscribe(

(response) => {

this.data = response;

},

(error) => {

console.error('Error fetching data:', error);

}

);

}

}

**Key Considerations:**

* **imports:** If your standalone component uses any other standalone components or directives, you need to import them in the imports array of the @Component decorator.
* **Error Handling:** Implement robust error handling using RxJS operators like catchError.
* **Loading States:** Display loading indicators while data is being fetched.
* **Data Transformation:** Use RxJS operators to transform and manipulate the data before displaying it.

**Example with bootstrapApplication:**

TypeScript

import { bootstrapApplication } from '@angular/platform-browser';

import { AppComponent } from './app/app.component';

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

import { HttpClientModule } from '@angular/common/http';

bootstrapApplication(AppComponent, {

providers: [importProvidersFrom(HttpClientModule)]

});

This example demonstrates how to bootstrap an application with standalone components and provide the necessary dependencies, such as HttpClientModule.

By following these guidelines, you can effectively use services with standalone components in Angular 17 and beyond, creating modular and maintainable applications.