**Example: Displaying Data from an Observable**

TypeScript

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

import { Observable, of } from 'rxjs';

@Component({

selector: 'app-async-pipe-example',

template: `

<p>Data: {{ data$ | async }}</p>

`

})

export class AsyncPipeExampleComponent {

data$: Observable<string>;

constructor() {

// Simulate an asynchronous operation (e.g., fetching data from a service)

this.data$ = of('Data from Observable').pipe(delay(2000));

}

}

In this example:

* data$: An Observable that emits a string after a 2-second delay.
* {{ data$ | async }}: The async pipe subscribes to the data$ Observable and displays the emitted value in the template.
* When the Observable emits a value, the template is automatically updated.

**Key Benefits of the Async Pipe:**

* **Simplified Subscription Management:** The async pipe handles the subscription and unsubscription automatically, preventing memory leaks.
* **Improved Readability:** The template remains cleaner and more concise.
* **Automatic Updates:** The view is automatically updated whenever the Observable emits a new value.

**Example with HTTP Request:**

TypeScript

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

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

@Component({

selector: 'app-http-example',

template: `

<p>Data: {{ data$ | async }}</p>

`

})

export class HttpExampleComponent {

data$: Observable<any>;

constructor(private http: HttpClient) {

this.data$ = this.http.get('/api/data');

}

}

In this example, the data$ Observable is obtained from an HTTP request using the HttpClient. The async pipe handles the HTTP request subscription and updates the view with the fetched data.

**Key Considerations:**

* The async pipe is designed to work with Observables and Promises.
* It automatically unsubscribes when the component is destroyed, preventing memory leaks.
* Use the async pipe judiciously to avoid unnecessary re-renders.

By effectively using the async pipe, you can simplify your Angular templates, improve component performance, and write more maintainable and reactive applications.