**Understanding providedIn in Angular Services**

The providedIn property in Angular services determines the scope and lifecycle of the service within an application. Here's a breakdown of the commonly used values:

**providedIn: 'root'**

* **Scope:** Application-wide singleton.

[1. What is the purpose of providedIn in Angular services? - GeeksforGeeks](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

[[Source icon](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/#:~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application.)](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

[www.geeksforgeeks.org](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

* **Behavior:** The service is registered at the root injector level.

[1. What is the purpose of providedIn in Angular services? - GeeksforGeeks](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

[[Source icon](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/#:~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application.)](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

[www.geeksforgeeks.org](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

* **Benefits:**
  + Easily accessible from any component, service, or module.
  + Only one instance of the service exists throughout the application.

[1. What is the purpose of providedIn in Angular services? - GeeksforGeeks](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

[[Source icon](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/#:~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application.)](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

[www.geeksforgeeks.org](https://www.geeksforgeeks.org/what-is-the-purpose-of-providedin-in-angular-services/" \l ":~:text=The%20providedIn%20property%20is%20part,decorator%20when%20defining%20a%20service.&When%20set%20to%20'root'%20%2C,available%20throughout%20the%20entire%20application." \t "_blank)

* + Ideal for shared services like configuration, logging, or data access.

**providedIn: 'any'**

* **Scope:** Module-level singleton.
* **Behavior:** The service is registered at the NgModule injector level.
* **Benefits:**
  + Creates a new instance of the service for each NgModule.
  + Useful for services that need to be isolated within specific modules.

[1. Angular providedIn vs providers arrays | by PRATAP ADITYA - Medium](https://medium.com/@pratap-aditya/angular-providedin-vs-providers-arrays-17b475f03cbb" \l ":~:text=Use%20the%20ProvidedIn%20root%20option,shall%20see%2C%20can%20cause%20problems." \t "_blank)

[[Source icon](https://medium.com/@pratap-aditya/angular-providedin-vs-providers-arrays-17b475f03cbb#:~:text=Use%20the%20ProvidedIn%20root%20option,shall%20see%2C%20can%20cause%20problems.)](https://medium.com/@pratap-aditya/angular-providedin-vs-providers-arrays-17b475f03cbb" \l ":~:text=Use%20the%20ProvidedIn%20root%20option,shall%20see%2C%20can%20cause%20problems." \t "_blank)

[medium.com](https://medium.com/@pratap-aditya/angular-providedin-vs-providers-arrays-17b475f03cbb" \l ":~:text=Use%20the%20ProvidedIn%20root%20option,shall%20see%2C%20can%20cause%20problems." \t "_blank)

**providedIn: 'platform'**

* **Scope:** Platform-level singleton.
* **Behavior:** The service is registered at the platform injector level, shared across multiple applications on the same page.
* **Benefits:**
  + For services that need to be shared between different Angular applications on the same page.

[1. multiple NG9 angular elements consuming a service as a singleton using providedIn:platform · Issue #40301 - GitHub](https://github.com/angular/angular/issues/40301" \l ":~:text=According%20to%20angular%20documentation%20for,all%20applications%20on%20the%20page." \t "_blank)

[[Source icon](https://github.com/angular/angular/issues/40301#:~:text=According%20to%20angular%20documentation%20for,all%20applications%20on%20the%20page.)](https://github.com/angular/angular/issues/40301" \l ":~:text=According%20to%20angular%20documentation%20for,all%20applications%20on%20the%20page." \t "_blank)

[github.com](https://github.com/angular/angular/issues/40301" \l ":~:text=According%20to%20angular%20documentation%20for,all%20applications%20on%20the%20page." \t "_blank)

* + Rarely used in most applications.

**Key Points**

* providedIn: 'root' is the most common choice for shared services.
* providedIn: 'any' is useful for creating module-specific services.
* providedIn: 'platform' is for very specific use cases where services need to be shared across multiple applications on the same page.
* The providedIn property is a powerful tool for managing service lifecycle and dependency injection in Angular.

**Example:**

TypeScript

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

@Injectable({

providedIn: 'root' // or 'any', 'platform' depending on your needs

})

export class MyService {

constructor(private injector: Injector) {}

// Service logic

}

**Understanding providers: [{provide:ErrorHandler, useExisting:GlobalErrorHandler}]**

**This line of code in your Angular application's NgModule is configuring a custom global error handler.**

**Breakdown**

* **providers**: This array property within an NgModule is used to register services and other dependencies.
* **{provide: ErrorHandler, useExisting: GlobalErrorHandler}**: This object within the providers array specifies a custom provider.
  + **provide: ErrorHandler**: Indicates that we're overriding the default Angular ErrorHandler service.
  + **useExisting: GlobalErrorHandler**: Specifies that the GlobalErrorHandler class (which you've defined elsewhere) should be used as the implementation for the ErrorHandler.

**What it Does**

By including this line in your NgModule, you're telling Angular to use your custom GlobalErrorHandler class instead of the default error handler whenever an error occurs within your application. This allows you to:

* **Centralize error handling:** Handle errors in one place instead of scattering try-catch blocks throughout your code.
* **Custom error logic:** Implement custom logic for logging, reporting, or displaying error messages.
* **Error recovery:** Potentially implement retry logic or fallback mechanisms.

**Example:**

TypeScript

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

@Injectable({

providedIn: 'root'

})

export class GlobalErrorHandler implements ErrorHandler {

handleError(error)

[1. github.com](https://github.com/AshMuUTN/Testology_Angular" \t "_blank)

[github.com](https://github.com/AshMuUTN/Testology_Angular" \t "_blank)

{

// Custom error handling logic here

console.error('An error occurred:', error);

// You can log the error, display a user-friendly message, etc.

}

}

Use code [with caution.](/faq#coding)

TypeScript

import { NgModule, ErrorHandler } from '@angular/core';

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

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

[1. javascript.plainenglish.io](https://javascript.plainenglish.io/angular-regime-series-tree-shaking-technique-3dc07f5e85a1" \t "_blank)

[javascript.plainenglish.io](https://javascript.plainenglish.io/angular-regime-series-tree-shaking-technique-3dc07f5e85a1" \t "_blank)

import { GlobalErrorHandler } from './global-error-handler';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule

],

providers: [

{ provide:

[1. github.com](https://github.com/MichaelOdumosu57/sentry-angular-multi-provider-issue" \t "_blank)

[github.com](https://github.com/MichaelOdumosu57/sentry-angular-multi-provider-issue" \t "_blank)

ErrorHandler, useExisting: GlobalErrorHandler }

],

bootstrap: [AppComponent]

})

export class AppModule { }

Use code [with caution.](/faq#coding)

**Additional Considerations:**

* **Error Types:** You might want to handle different error types differently (e.g., HTTP errors, uncaught exceptions).
* **Error Logging:** Consider using a logging service to log errors for analysis.
* **User Experience:** Display informative error messages to the user without exposing sensitive details.
* **Performance:** Be mindful of performa

**Equivalent to providedIn: 'root' in Angular**

**Providing a service in the providers array of the root module (typically AppModule) is equivalent to using providedIn: 'root' in the service's @Injectable decorator.**

**Understanding the Equivalence**

Both methods achieve the same goal: making a service available as a singleton throughout the entire Angular application.

**Example using providers array:**

TypeScript

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

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

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

[1. auth0.com](https://auth0.com/blog/angular-2-ngmodules/" \t "_blank)

[auth0.com](https://auth0.com/blog/angular-2-ngmodules/" \t "_blank)

import { MyService } from './my.service';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule

],

providers: [MyService],

bootstrap: [AppComponent]

})

export class AppModule

[1. stackoverflow.com](https://stackoverflow.com/questions/76899241/confusion-with-dependency-injection-and-provider-hierarchy-in-angular" \t "_blank)

[stackoverflow.com](https://stackoverflow.com/questions/76899241/confusion-with-dependency-injection-and-provider-hierarchy-in-angular" \t "_blank)

{ }

Use code [with caution.](/faq#coding)

**Equivalent using providedIn: 'root':**

TypeScript

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

@Injectable({

providedIn: 'root'

})

export class MyService {

// ...

}

Use code [with caution.](/faq#coding)

**Key Points**

* Both methods create a single instance of the service accessible throughout the application.
* providedIn: 'root' is generally preferred for better readability and maintainability.
* Using providers array might be necessary in older Angular versions or specific scenarios.

**When to Use Which**

* **providedIn: 'root':** Recommended for most cases as it's cleaner and more explicit.
* **providers array:** Can be used for more complex scenarios or when working with older Angular versions.

**In conclusion,** while both methods achieve the same result, providedIn: 'root' is the preferred approach in modern Angular applications due to its simplicity and clarity.

**Would you like to explore other ways to provide services in Angular or discuss specific use ca**

**Non-Singleton Services in Angular**

By default, services in Angular are singletons, meaning there's only one instance shared across the entire application. However, there are scenarios where you might need a non-singleton service, where each component has its own instance.

[1. [Proposal] Defining non-singleton services or component's service · Issue #18015 · angular/angular - GitHub](https://github.com/angular/angular/issues/18015" \l ":~:text=Create%20a%20new%20decorator%20for,be%20created%20for%20each%20injection." \t "_blank)

[[Source icon](https://github.com/angular/angular/issues/18015#:~:text=Create%20a%20new%20decorator%20for,be%20created%20for%20each%20injection.)](https://github.com/angular/angular/issues/18015" \l ":~:text=Create%20a%20new%20decorator%20for,be%20created%20for%20each%20injection." \t "_blank)

[github.com](https://github.com/angular/angular/issues/18015" \l ":~:text=Create%20a%20new%20decorator%20for,be%20created%20for%20each%20injection." \t "_blank)

**Creating a Non-Singleton Service**

To create a non-singleton service, you need to provide it at the component level rather than in the providers array of the NgModule.

**Example:**

TypeScript

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

@Component({

selector: 'app-my-component',

template: '',

providers: [MyService]

})

export class MyComponent {

constructor(private myService: MyService) {}

}

Use code [with caution.](/faq#coding)

In this example, MyService is provided at the component level, meaning each instance of MyComponent will have its own instance of MyService.

**When to Use Non-Singleton Services**

* **Component-specific data:** When you need to store data that is specific to a component and its children.
* **Avoiding shared state:** To prevent unintended side effects from shared state.
* **Testing isolation:** To isolate components for testing purposes.

**Important Considerations**

* **Performance:** Creating multiple instances of a service can impact performance, especially for large applications. Use non-singleton services judiciously.
* **Dependency Injection:** Be aware of how dependency injection works in Angular and how it affects the lifecycle of services.
* **Component Hierarchy:** Non-singleton services are scoped to the component and its children. Changes made in one instance won't affect other instances.

**Additional Notes**

* **providedIn: 'any':** While this option can be used to create non-singleton services, it's generally not recommended as it can lead to unexpected behavior and performance issues.
* **Lazy Loading:** Be careful when using non-singleton services with lazy loading modules.

By understanding the concept of non-singleton services and their implications, you can effectively manage the lifecycle and scope of your services in Angular applications.

**Would you like to explore specific use cases or delve deeper into the implicati**

**Understanding Angular Error Handling**

Angular provides mechanisms to handle errors at different levels:

1. **HTTP Errors:**
   * HttpErrorResponse is used to handle errors from HTTP requests.
   * catchError operator can be used in RxJS pipelines to intercept errors.
2. **Global Error Handling:**
   * ErrorHandler can be injected and overridden to provide custom error handling.
3. **Component-Level Error Handling:**
   * try...catch blocks can be used within components.

**Example: Handling HTTP Errors**

TypeScript

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

import { catchError, throwError } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class

[1. github.com](https://github.com/marceta99/MatchDevelopers" \t "_blank)

[github.com](https://github.com/marceta99/MatchDevelopers" \t "_blank)

MyService {

constructor(private http: HttpClient) {}

getData(): Observable<any> {

return this.http.get<any>('https://api.example.com/data')

[1. www.adaface.com](https://www.adaface.com/blog/angular-interview-questions/" \t "_blank)

[www.adaface.com](https://www.adaface.com/blog/angular-interview-questions/" \t "_blank)

.pipe(

catchError((error: HttpErrorResponse) => {

// Handle the error here, e.g., log to console, display error message

console.error('An error occurred:', error.error);

return throwError(() => new Error('Something bad happened'));

})

);

}

}

Use code [with caution.](/faq#coding)

**Key Points:**

* Angular doesn't have a built-in errorOption property.
* Use HttpErrorResponse for HTTP-related errors.
* Employ catchError operator to handle errors in observables.
* Implement custom error handling logic based on your application's requirements.

**If you encounter an error message mentioning something similar to errorOption, please provide more context about the error, the code snippet, and the library or framework you're using.**

By understanding these concepts, you can effectively handle errors in your Angular applications and provide a better user experience.

**Would you like to explore specific error handling scenarios or delve deeper into any of these concepts?**

Sources and related content

[github.com](https://github.com/marceta99/MatchDevelopers" \t "_blank)

**Understanding the Component Lifecycle**

**Imagine a component as a person going through a day.**

* **Birth (ngOnInit):** When the person wakes up (component is created), they start their day (component is initialized).
* **Daily Routine (ngOnChanges):** Throughout the day, the person might change clothes (component's input properties change).
* **Checking In (ngDoCheck):** The person might check their schedule or messages (Angular checks for changes).
* **Lunch Time (ngAfterContentInit):** After eating lunch (content projected into the component), the person is ready for the afternoon.
* **Afternoon Activities (ngAfterContentChecked):** The person might check their phone or emails (content projected into the component is checked).
* **Work Time (ngAfterViewInit):** After reaching the office (component's view is ready), the person starts working.
* **Checking Emails (ngAfterViewChecked):** The person might check emails again (component's view is checked).
* **Going Home (ngOnDestroy):** At the end of the day, the person goes home (component is destroyed).

**Lifecycle Hooks in Action**

These are specific moments in a component's life where you can write code to do something. For example:

* **Fetching data:** In ngOnInit, you can fetch data from a server.
* **Updating the UI:** In ngOnChanges, you can update the UI based on changed input properties.
* **Performing cleanup:** In ngOnDestroy, you can unsubscribe from observables or clear timers.

**Tracking Changes**

* **ngOnChanges:** This is like noticing when someone changes their clothes. You can react to this change.

TypeScript

import { Component, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({

selector: 'app-product-card',

template: `

<p>Product name: {{ productName }}</p>

`

})

export class ProductCardComponent implements OnChanges {

@Input() productName: string;

ngOnChanges(changes: SimpleChanges) {

console.log('Product name changed:', changes.productName.currentValue);

}

}

Use code [with caution.](/faq#coding)

**Lifecycle Hooks and Template Access**

* **ngAfterViewInit:** This is when you can access elements in the component's HTML.
* **ViewChild:** This is like getting a reference to a specific child element.

TypeScript

import { Component, ViewChild, AfterViewInit, ElementRef } from '@angular/core';

@Component({

selector: 'app-product-list',

template: `

<ul>

<li #firstProduct>Product 1</li>

<li>Product 2</li>

</ul>

`

})

export class ProductListComponent implements AfterViewInit {

@ViewChild('firstProduct') firstProductElement: ElementRef;

ngAfterViewInit() {

console.log(this.firstProductElement.nativeElement.textContent); // Access the text content of the first product

}

}

Use code [with caution.](/faq#coding)

**Remember:**

* These are just a few examples. There are many more ways to use lifecycle hooks and ViewChild.
* Understanding the component lifecycle helps you write efficient and well-structured Angular code.
* By using these tools, you can create dynamic and interactive user interfaces.

Would you like to learn about specific use cases or delve deeper into any of these concepts?

**Upload an image**

This prompt requires an image that you need to add. Tap the image button to upload an image.

Got it

**Need a little help with this prompt?**

Power up your prompt and Gemini will expand it to get you better results

Got it

Gemini may display inaccurate info, including about people, so double-check its responses. [Your privacy & Gemini Apps Opens in a new window](https://support.google.com/gemini?p=privacy_notice)



**The Lifecycle of a Component**

Imagine a component as a new employee in a company. The employee goes through various stages from joining the company to leaving. This is similar to the lifecycle of a component.

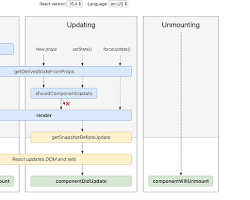
**Lifecycle Hooks as Milestones**

* **ngOnInit:** This is like the first day of the job. The employee is introduced to the team and starts their work.
* **ngOnChanges:** This is like receiving a new project or task. The employee adjusts their work accordingly.
* **ngDoCheck:** This is like a regular check-in with the manager to see if anything has changed.
* **ngAfterContentInit:** This is when the employee receives their office setup and starts using it.
* **ngAfterContentChecked:** This is like a quick check after setting up the office to ensure everything is in place.
* **ngAfterViewInit:**

This is when the employee's workspace is fully set up and they can start working on projects.

* **ngAfterViewChecked:** This is like a quick check after setting up the workspace to ensure everything is working correctly.
* **ngOnDestroy:** This is when the employee resigns from the company.

**Visualizing the Lifecycle**

[Opens in a new window [](https://betterprogramming.pub/learn-reacts-lifecycle-methods-through-an-analogy-9fc82b990ece)betterprogramming.pub](https://betterprogramming.pub/learn-reacts-lifecycle-methods-through-an-analogy-9fc82b990ece)

component lifecycle diagram with analogies

**In-Depth Explanation with Examples**

* **ngOnInit:**
  + Fetch initial data from a server.
  + Subscribe to observables.
  + Set up initial values for properties.
* **ngOnChanges:**
  + Update component state based on input property changes.
  + Perform calculations or transformations.
* **ngDoCheck:**
  + Rarely used. It's for custom change detection logic, but Angular's default change detection is usually sufficient.
* **ngAfterContentInit:**
  + Access projected content (content from parent component).
* **ngAfterContentChecked:**
  + Rarely used. It's for custom checks after content projection.
* **ngAfterViewInit:**
  + Access child components or DOM elements.
  + Set up DOM manipulations.
* **ngAfterViewChecked:**
  + Rarely used. It's for custom checks after view rendering.
* **ngOnDestroy:**
  + Unsubscribe from observables.
  + Clear timers or intervals.
  + Perform cleanup actions.

**Example: A Product Card Component**

Imagine a product card component that displays product details.

* **ngOnInit:** Fetch product data from a server.
* **ngOnChanges:** Update product details if the product input changes.
* **ngAfterViewInit:** Set up a click event listener on the "Add to Cart" button.
* **ngOnDestroy:** Unsubscribe from any observables used for fetching data.

By understanding the component lifecycle and using these hooks effectively, you can create more efficient and maintainable Angular applications.

Content is what is passed as children usually to be projected at some <ng-content> element of a component.  
View is the template of the current component.

https://stackoverflow.com/questions/66515574/debugging-angular-app-in-vscode-is-not-working-again