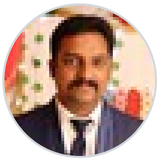
Instructor example



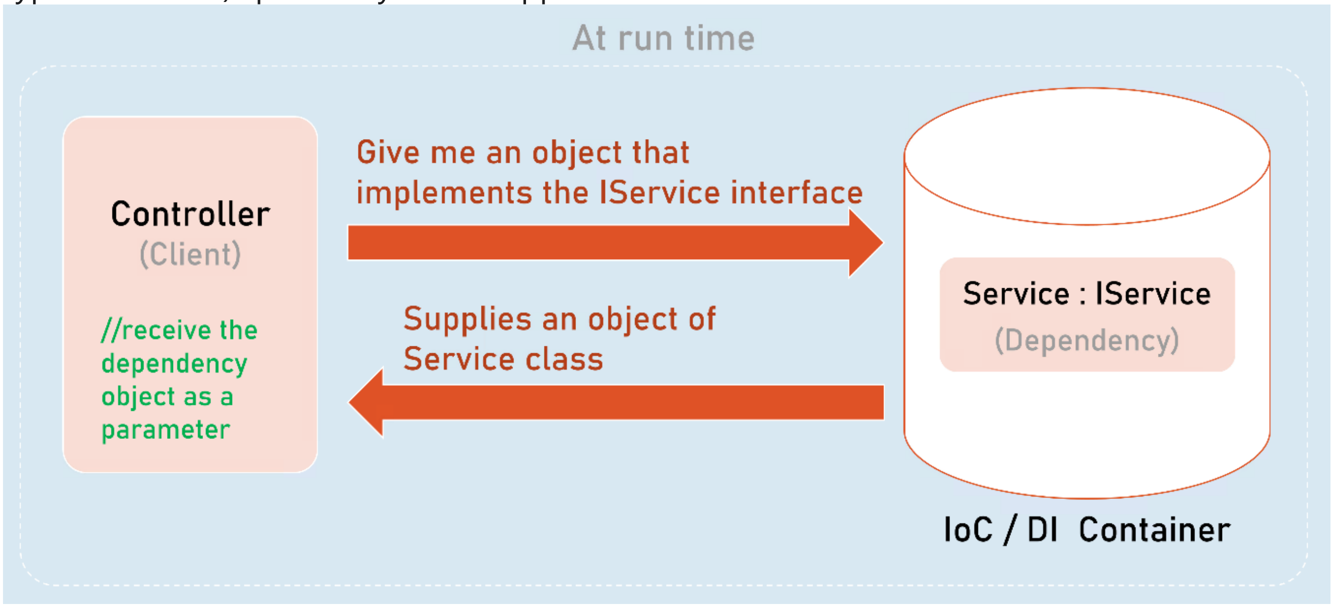
Web University by Harsha Vardhan

Explain how dependency injection works in ASP.NET Core?

ASP.NET Core injects instances of dependency classes by using the built-in IoC (Inversion-of-Control) container. This container is represented by the IServiceProvider interface.

The types (classes) managed by the container are called services. We first need to register them with the IoC container in the Startup class.

ASP.NET Core supports two types of services, namely framework and application services. Framework services are a part of ASP.NET Core framework such as ILoggerFactory, IHostingEnvironment, etc. In contrast, a developer creates the application services (custom types or classes) specifically for the application.


At run time
Controller
(Client)
//receive the dependency object as a parameter
Give me an object that implements the IService interface
Supplies an object of Service class
Service : IService
(Dependency)
IoC / DI Container

“ASP.NET Core has dependency injection to manage services; are you aware of the different lifetimes? What are they, and what does each mean?”

ASP.NET Core has three lifetimes of Singleton, Scoped, and Transient.

1. **Singleton:** ASP.NET Core services container will create services registered as a singleton only once for the duration of the application’s lifetime. Singletons are helpful for expensive services or services with little to no internal state.

2. **Scoped:** As the name suggests, with regard to scoped services, they are created within a scope. The scope is typically the lifetime of an HTTP request, but not necessarily

always. I, as a developer, might create custom scopes in code, but anyone should be careful to use this technique sparingly.

3. **Transient:** Finally, ASP.NET Core creates transient services when a dependent instance asks for them. I might consider registering dependencies as transient as the "safest" approach to creating dependencies as there's no chance for contention, race conditions, or deadlocks. Still, it can also come at the expense of performance and resource utilization.

Each lifetime has its use, and it depends on the dependency we are registering.

What are the benefits of Dependency Injection?

DI helps to implement decoupled architecture where you change at one place and changes are reflected at many places.

Dependency injection is basically providing the objects that an object needs (its dependencies) instead of having it construct them itself. When using dependency injection, objects are given their dependencies at run time rather than compile time (car manufacturing time).

- It allows your code to be more loosely coupled because classes do not have hard-coded dependencies
- Decoupling the creation of an object (in other words, separate usage from the creation of an object)
- Making isolation in unit testing possible/easy. It is harder to isolate components in unit testing without dependency injection.
- Explicitly defining dependencies of a class
- Facilitating good design like the single responsibility principle (SRP) for example
- Promotes Code to an interface, not to implementation principle
- Enabling switching/ability to replace dependencies/implementations quickly (Eg: DbLogger instead of ConsoleLogger)

What is IoC (DI) Container?

A Dependency Injection container, sometimes, referred to as DI container or IoC container, is a framework that helps with DI. It "creates" and "injects" dependencies for us automatically.

What is Inversion of Control?

Inversion of control is a broad term but for a software developer it's the most commonly described as a pattern used for decoupling components and layers in the system.

It inverses the control by shifting the control to IoC container.

For example, say your application has a text editor component and you want to provide spell checking. Your standard code would look something like this:

```
public class TextEditor {
 private SpellChecker checker;

 public TextEditor() {
  this.checker = new SpellChecker();
 }
}
```

What we've done here creates a dependency between the TextEditor and the SpellChecker. In an IoC scenario we would instead do something like this:

```
public class TextEditor {
 private IocSpellChecker checker;

 public TextEditor(IocSpellChecker checker) {
   this.checker = checker;
 }
}
```

You have inverted control by handing the responsibility of instantiating the spell checker from the TextEditor class to the caller.

```
SpellChecker sc = new SpellChecker; // dependency
TextEditor textEditor = new TextEditor(sc);
```

How do you create your own scopes in asp.net core?

We can create child scopes by using IServiceScopeFactory.

```
//controller
public ControllerName(IServiceScopeFactory serviceScopeFactory)
{
  _serviceScopeFactory = serviceScopeFactory;
}

[Route("route-path")]
public IActionResult ActionMethod()
{
 using (IServiceScope scope = _serviceScopeFactory.CreateScope())
 {
  IService service = scope.ServiceProvider.GetRequiredService<IService>();
  //call service methods here
 }
 return View();
}
```

How do you inject a service in view?

We can do that by using @inject directive in razor view.

Eg:

`@inject IService service`

Why you prefer Autofac over built-in Microsoft DI?

Autofac is an IoC container for .NET. It manages the dependencies between classes so that applications stay easy to change as they grow in size and complexity. Autofac is the most popular DI/IoC container for ASP.NET core.

Though the default Microsoft DI may offer enough functionality, there is a certain limitations like resolving a service with some associated Metadata, Named/Keyed services, Aggregate Services, Multi-tenant support, lazy instantiation, and much more. As the system grows you might need such features, and Autofac gives you all these features.

What exception do you get when a specific service that you injected, can’t be found in the IoC container?

I’ll get an “InvalidOperationException” with error message “Unable to resolve service for type 'type’ while attempting to activate ‘class_name'.

I’ll get above exception when I injected a service class into another service or controller; but the service that I injected is not added to the IoC container at application startup.