Registering More Complex Services

Steve Gordon

.NET Engineer and Microsoft MVP

@stevejgordon www.stevejgordon.co.uk

Overview

IServiceCollection extension methods

Advanced registration requirements

- Review the options when using the built-in Microsoft container
- Explore where we reach its limitations

Improving registration code organization

Service Descriptors

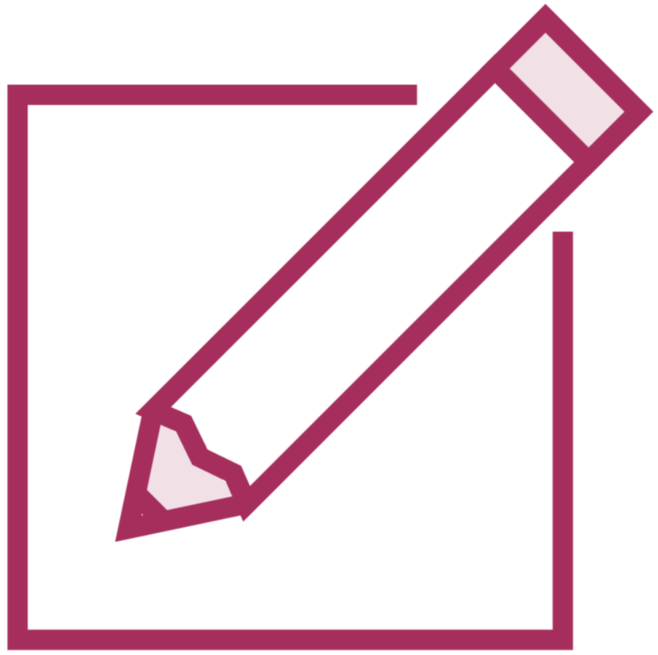
Service descriptors contain information about registered services.

ServiceDescriptor.cs

```
public class ServiceDescriptor
{
    // CONSTRUCTORS

    public Type ImplementationType { get; }
    public Type ServiceType { get; }
    public ServiceLifetime Lifetime { get; }
    public object ImplementationInstance { get; }
    public Func<IServiceProvider, object> ImplementationFactory { get; }
}
```

Service Descriptors



We rarely need to work directly with service descriptors

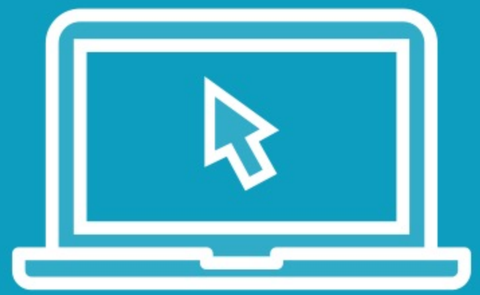
Created when we call (Try)AddTransient, (Try)AddScoped and (Try)AddSingleton

Some advanced scenarios require the use of service descriptors

Used internally by the service provider to resolve services

Add vs. TryAdd

Demo



Duplicated service registrations

Add vs. TryAdd behaviour

“The order of service registrations is generally not important”.

Steve Gordon – Earlier in this course!

The TryAdd extension methods only register a service if the service type does not already exist in the IServiceCollection.

TryAdd Extension Methods

The TryAdd methods are most convenient in complex applications

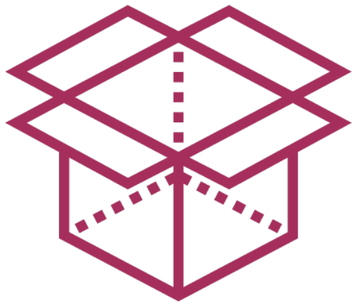
- Express intent more clearly
- Avoid accidental replacement of previously registered services

Safest to prefer the TryAdd methods by default

Registering an Interface Multiple Times

Multiple Registrations

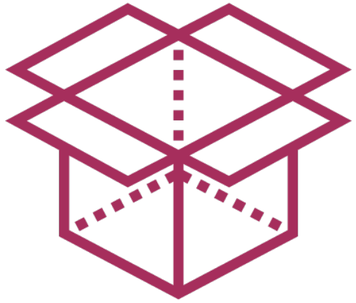
Dependency Injection
Container



Lifetime	Service Type	Implementation Type

Multiple Registrations

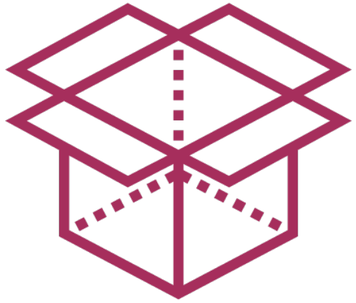
Dependency Injection Container



Lifetime	Service Type	Implementation Type
Singleton	IWeatherForecaster	AmazingWeatherForecaster

Multiple Registrations

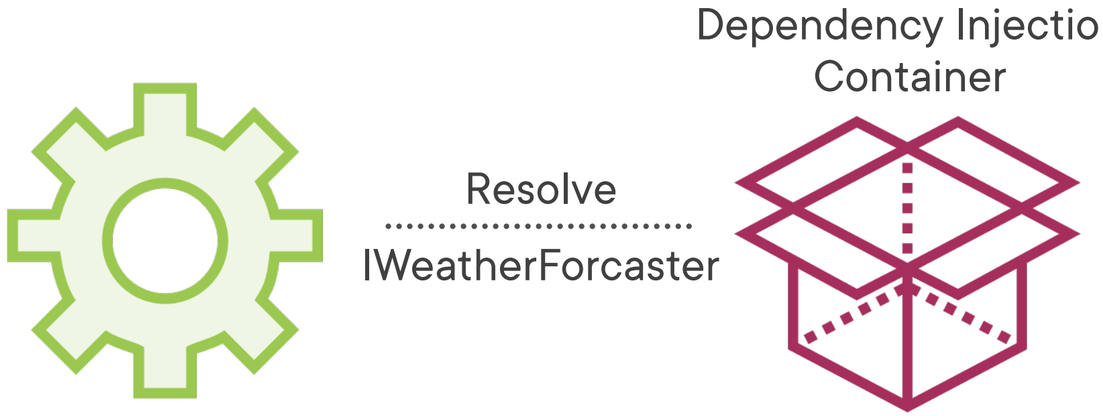
Dependency Injection Container



Lifetime	Service Type	Implementation Type
Singleton	IWeatherForecaster	AmazingWeatherForecaster
Singleton	IWeatherForecaster	RandomWeatherForecaster

Multiple Registrations

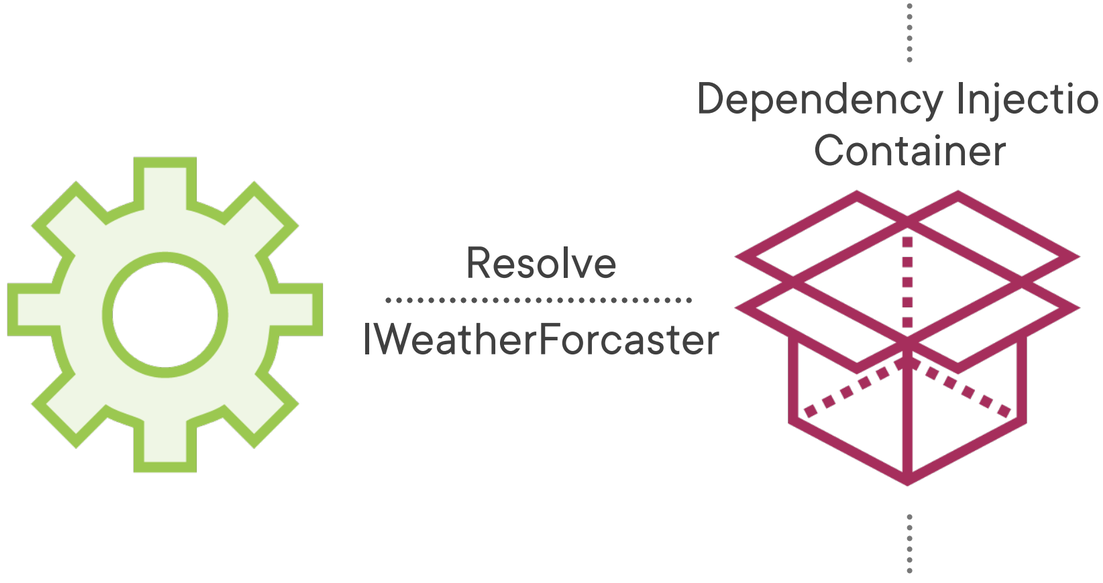
public SomeService(IWeatherForecaster weatherForecaster)



Lifetime	Service Type	Implementation Type
Singleton	IWeatherForecaster	AmazingWeatherForecaster
Singleton	IWeatherForecaster	RandomWeatherForecaster

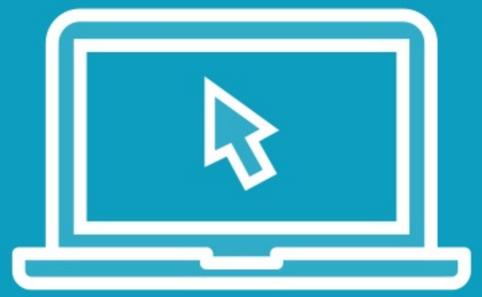
Multiple Registrations

public SomeService(IWeatherForecaster weatherForecaster)



Lifetime	Service Type	Implementation Type
Singleton	IWeatherForecaster	AmazingWeatherForecaster
Singleton	IWeatherForecaster	RandomWeatherForecaster

Demo



Replacing a service registration

Removing service registrations

Replace only supports removing the first registration of a service type from the IServiceCollection

Replace Method

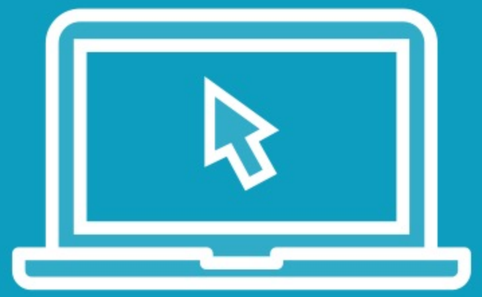
Looks for the first existing registration matching the service type

- If an entry is located, it is removed

A new registration for the service type is then added to the IServiceCollection

- Its specified implementation type will be used for the new registration

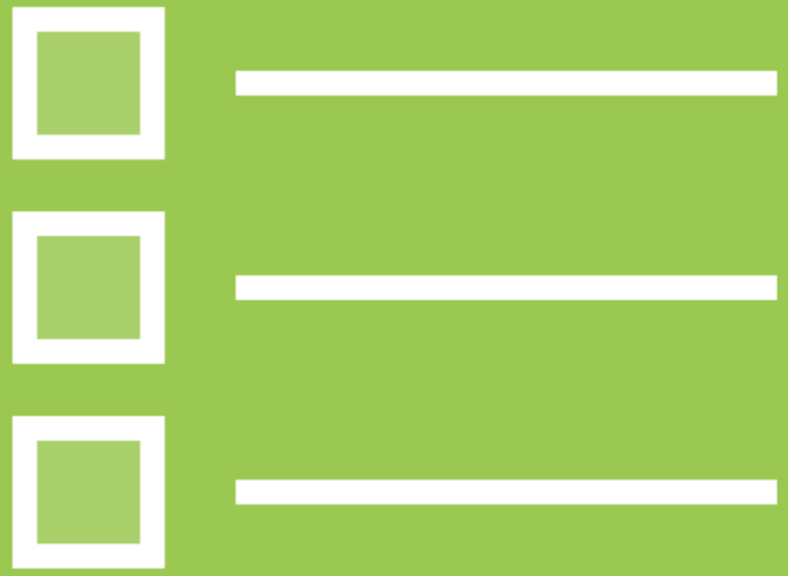
Demo



Why the container allows multiple service registrations

Implement a rule pattern by registering multiple implementations of an interface

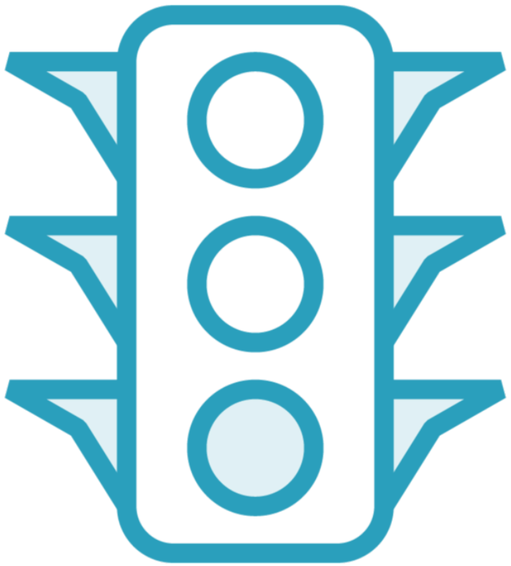
- Add new functionality with minimal changes to existing code



Requirement

Ensure that all bookings made by members of the tennis club comply with a set of defined rules.

IEnumerable<T> Dependencies



For IEnumerable<T> dependencies, the service provider resolves all registered implementations for the service type

This only applies when the parameter is typed as IEnumerable

The service provider does not resolve services if the parameter is an array or any other collection type

- IList
- ICollection
- etc.

Dependencies

ClubIsOpenRule

MaxBookingLengthRule

MaxPeakTimeBookingLengthRule

MemberBookingsMustNotOverlapRule

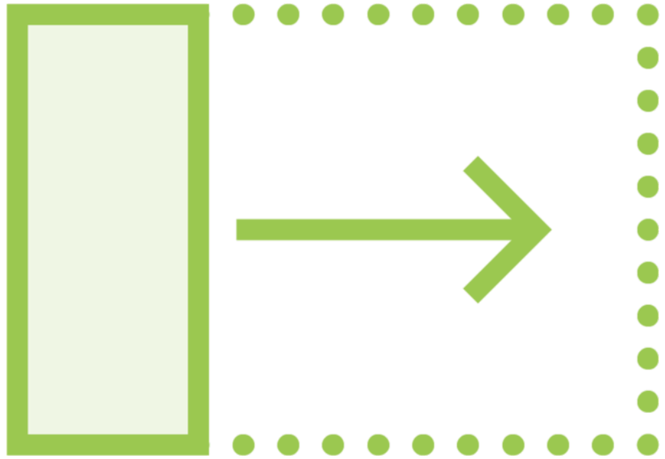
MemberCourtBookingsMaxHoursPerDayRule

IOptions<ClubConfiguration>

IOptions<BookingConfiguration>

ICourtBookingService

Simple Extensibility



Adding new rules

- Add new ICourtBookingRule implementation
- Register it with the IServiceCollection

No further changes required

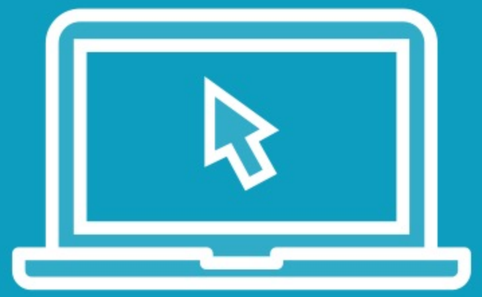
Applies separation of concerns and single responsibility

- Application is easier to maintain

Extension

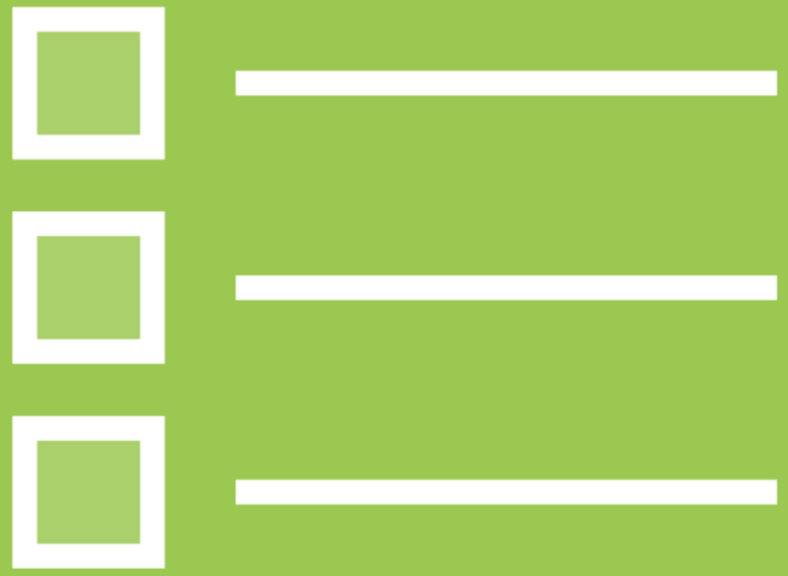
Add a new rule to ensure bookings are only valid if they are made for a date and time which is in the future.

Demo



What happens if the same implementation is added twice?

Safely registering multiple implementations with TryAddEnumerable

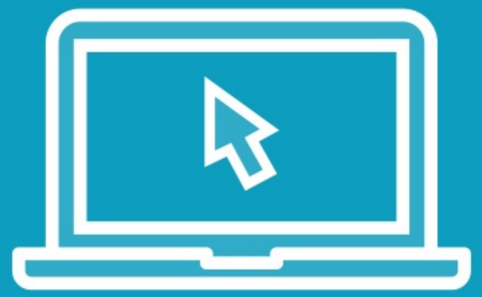


Requirement

Improve user experience by showing only valid time slots are selectable when making a court booking.

The Add methods are not idempotent.

Demo



Register services which provide an implementation factory

Discuss scenarios for applying implementation factories

Implementation Factories

Provide complete control over the creation of the implementation type.

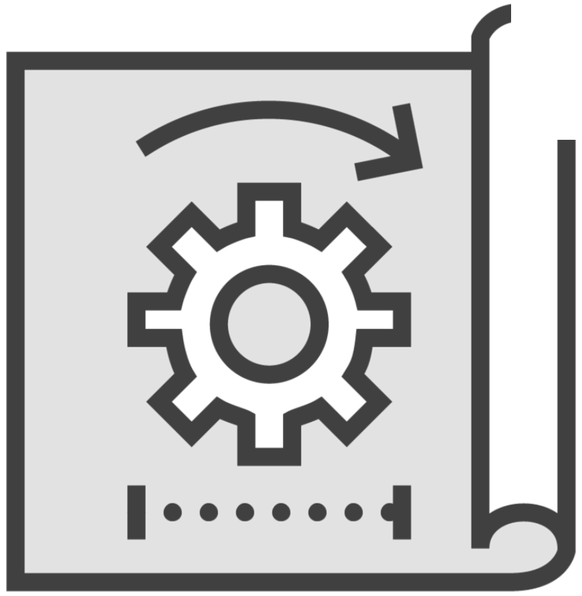
Implementation Factory Signature

```
public static void TryAddSingleton<TService>(
    this IServiceCollection services,
    Func<IServiceProvider, TService> implementationFactory)
        where TService : class;
```

Implementation Factory Signature

```
public static void TryAddSingleton<TService>(
    this IServiceCollection services,
    Func<IServiceProvider, TService> implementationFactory)
        where TService : class;
```

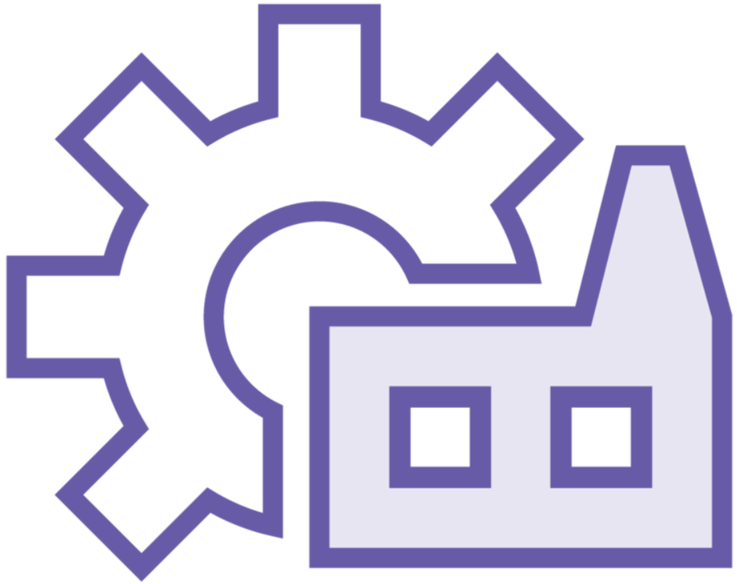
Implementation Factory Delegate



- **Invoked at runtime**
- **Has access to the built IServiceProvider**
- **May resolve services**
- **Responsible for returning a constructed instance of the service type**

Forwarding Registrations

Implementation Factories

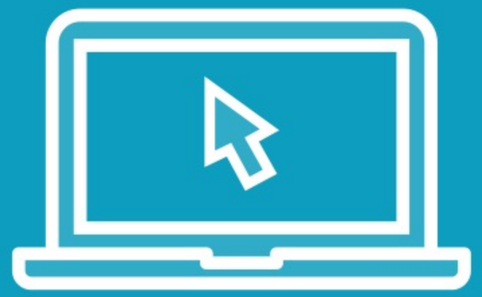


Apply in advanced scenarios

Prefer refactoring to support automatic creation of types by the service provider

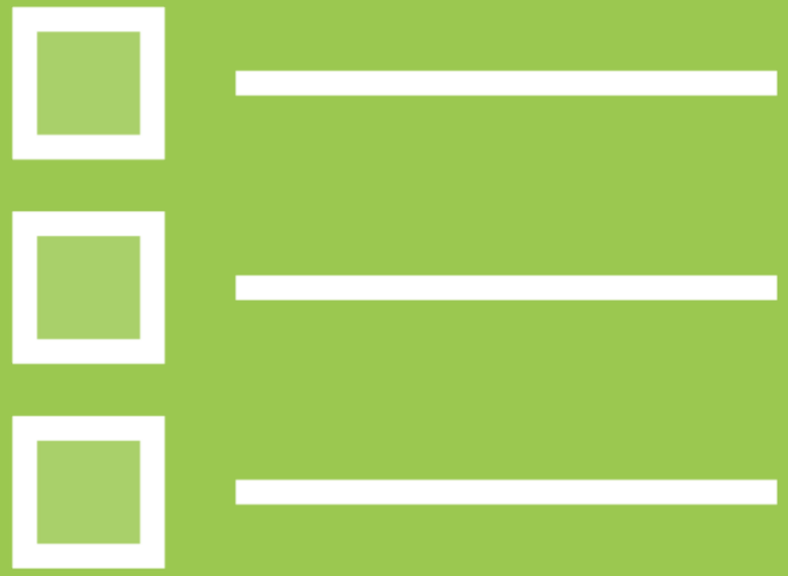
Legacy or third-party types may require developers to use implementation factories

Demo



Why register an implementation against multiple service types?

How to register an implementation multiple times

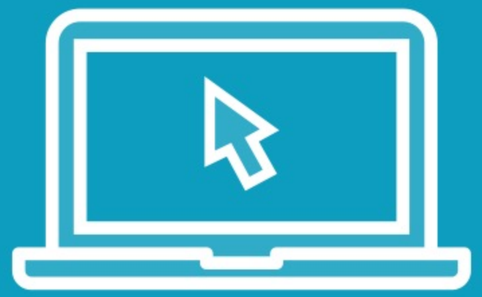


Requirement

Add a random greeting for logged in members when they begin making a booking.

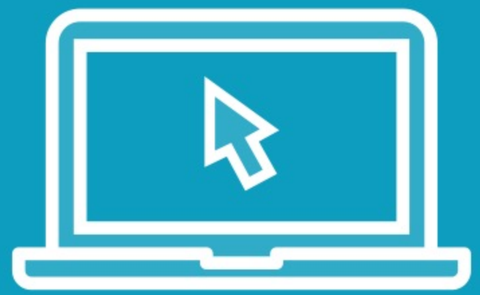
Any instances created outside of the dependency injection container are not automatically disposed of or released for garbage collection.

Demo



Registering open generic services

Demo



Improving the readability of registrations

Extension methods are particularly beneficial in libraries to ensure correct registration of all required services.

Up Next:
Injecting and Resolving Dependencies