IOC Container

Singleton : Uygulama ilk ayağa kalktığı anda, servisin tek bir instance'ı oluşturularak memory'de tutulur ve daha sonrasında her servis çağrısında bu instance gönderilir.

Scoped : Her request için tek bir instance yaratılmasını sağlayan lifetime seçeneğidir. Request cycle'ı tamamlanana kadar gerçekleşen servis çağrılarında daha önce oluşturulan instance gönderilir. Daha sonra yeni bir request başladığında farklı bir instance oluşturulur.

Transient : Her servis çağrısında yeni bir instance oluşturulur. Bağlayıcılığı en az olan lifetime seçeneğidir.

IOC Container yapısını bulduğum güzel bir örnekle açıklayacağım. Önce projeye 3 farklı lifetime seçeneğini de ekliyoruz.

```
1 reference
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddTransient<ITransientService, TransientService>();
    services.AddScoped<IScopedService, ScopedService>();
    services.AddSingleton<ISingletonService, SingletonService>();

    services.AddMvc();
    services.AddControllersWithViews();
}
```

Daha sonra 3 tane servis oluşturuyoruz. Servislerde getGuid() metodunu tanımlıyoruz. Bu servislerdeki methodlar geri bir Guid nesnesi dönderiyor.

```
using System;

namespace iocContainer.Services
{
    6 references
    public interface ISingletonService : IService { }

    2 references
    public class SingletonService : ISingletonService
    {
        private string _guid;

        0 references
        public SingletonService()
        {
            _guid = Guid.NewGuid().ToString();
        }

        7 references
        public string getGuid()
        {
            return _guid;
        }
    }

}
```

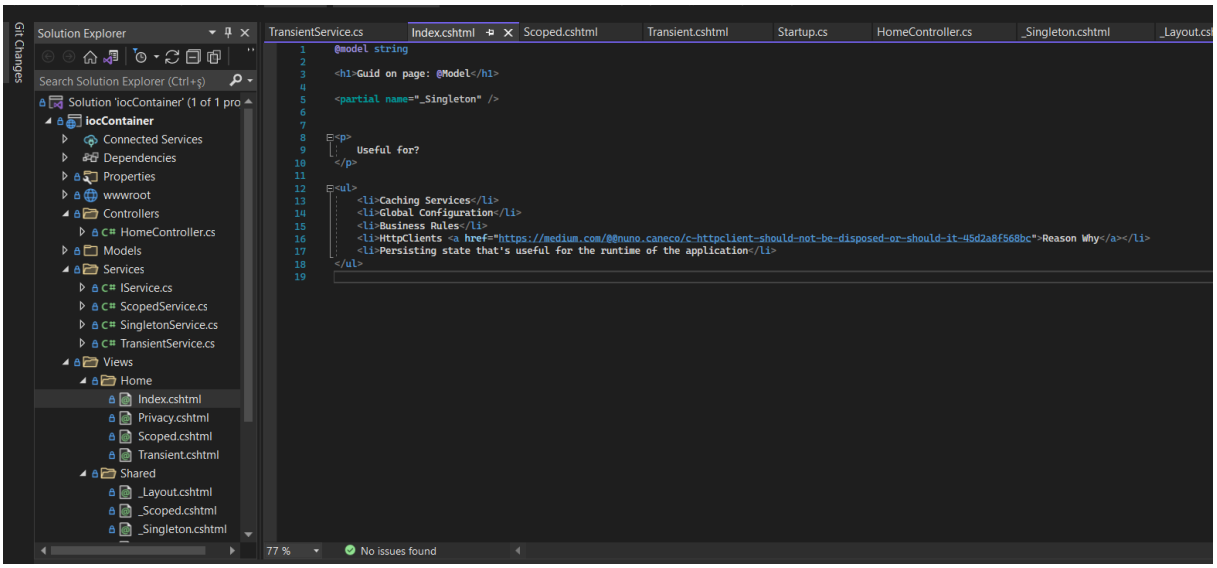
```
namespace iocContainer.Services
{
    6 references
    public interface IScopedService : IService { }

    2 references
    public class ScopedService : IScopedService
    {
        private string _guid;

        0 references
        public ScopedService()
        {
            _guid = Guid.NewGuid().ToString();
        }

        7 references
        public string getGuid()
        {
            return _guid;
        }
    }
}
```

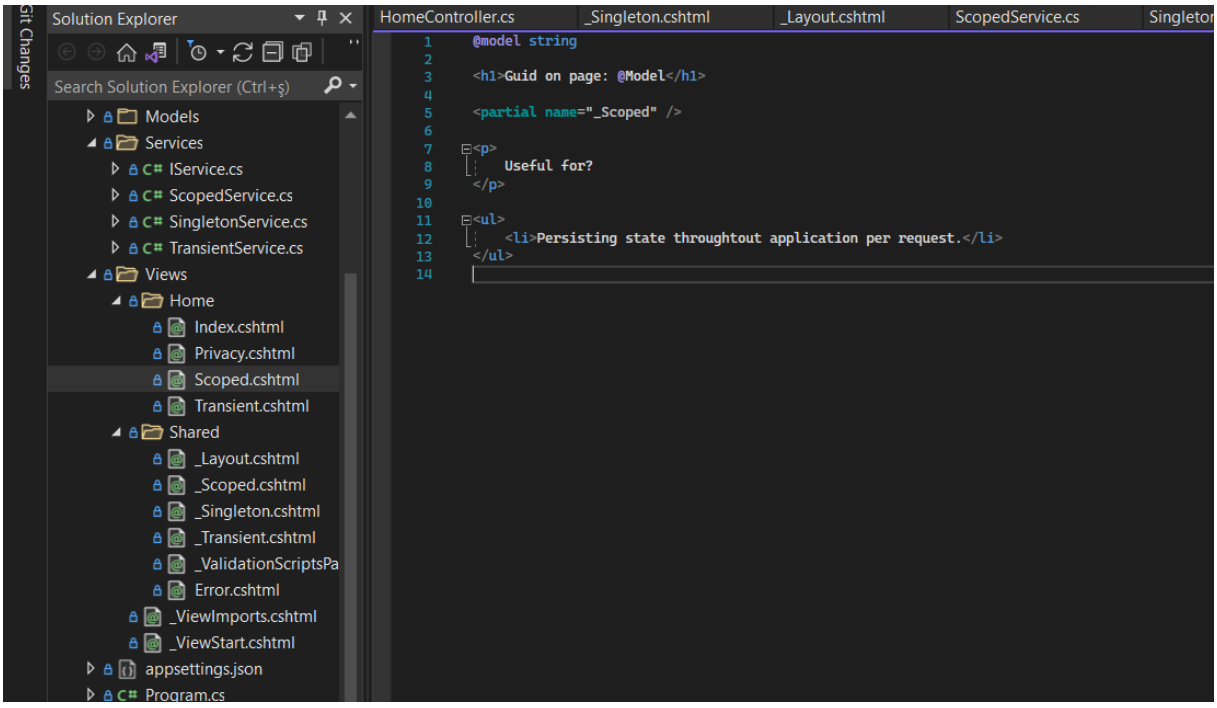
```
using System;

namespace iocContainer.Services
{
    6 references
    public interface ITransientService : IService { }

    2 references
    public class TransientService : ITransientService
    {
        private string _guid;

        0 references
        public TransientService()
        {
            _guid = Guid.NewGuid().ToString();
        }

        7 references
        public string getGuid()
        {
            return _guid;
        }
    }
}
```

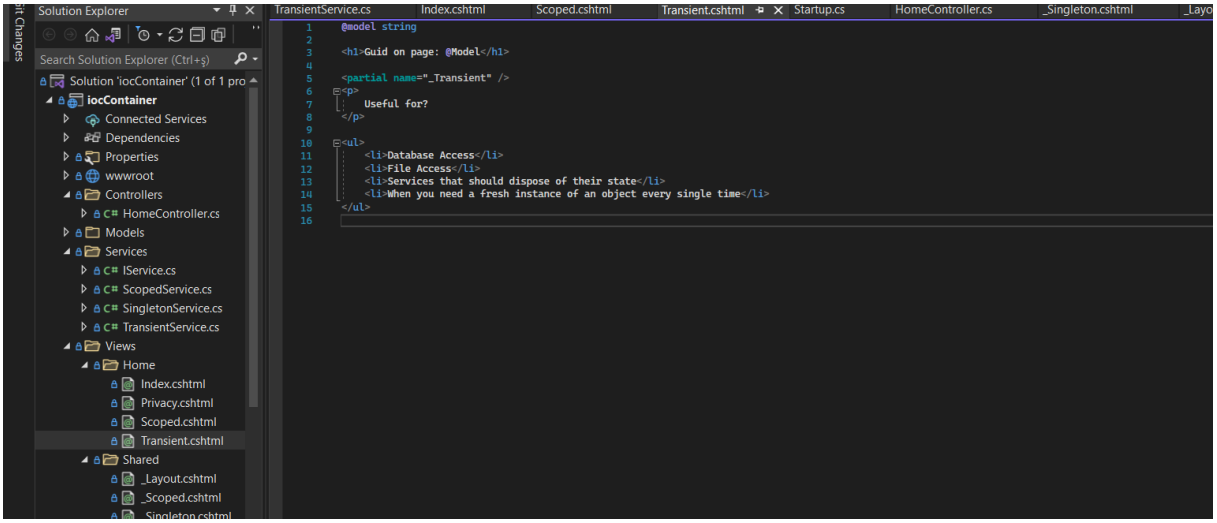
HomeController constructorında servislerimizi ioc container'da belirttiğimiz lifetime seçenekleri ile atama yapıyoruz. Oluşturduğumuz view'lara getGuid() metodundan dönen Guid nesnelerini gönderiyoruz.

```
public class HomeController : Controller
{
    private ISingletonService _singleton;
    private IScopedService _scoped;
    private ITransientService _transient;

    0 references
    public HomeController(
        ISingletonService singleton,
        IScopedService scoped,
        ITransientService transient)
    {
        _singleton = singleton;
        _scoped = scoped;
        _transient = transient;
    }

    0 references
    public IActionResult Index()
    {
        return View("Index", _singleton.getGuid());
    }

    0 references
    public IActionResult Scoped()
    {
        return View("Scoped", _scoped.getGuid());
    }

    0 references
    public IActionResult Transient()
    {
        return View("Transient", _transient.getGuid());
    }
}
}
```

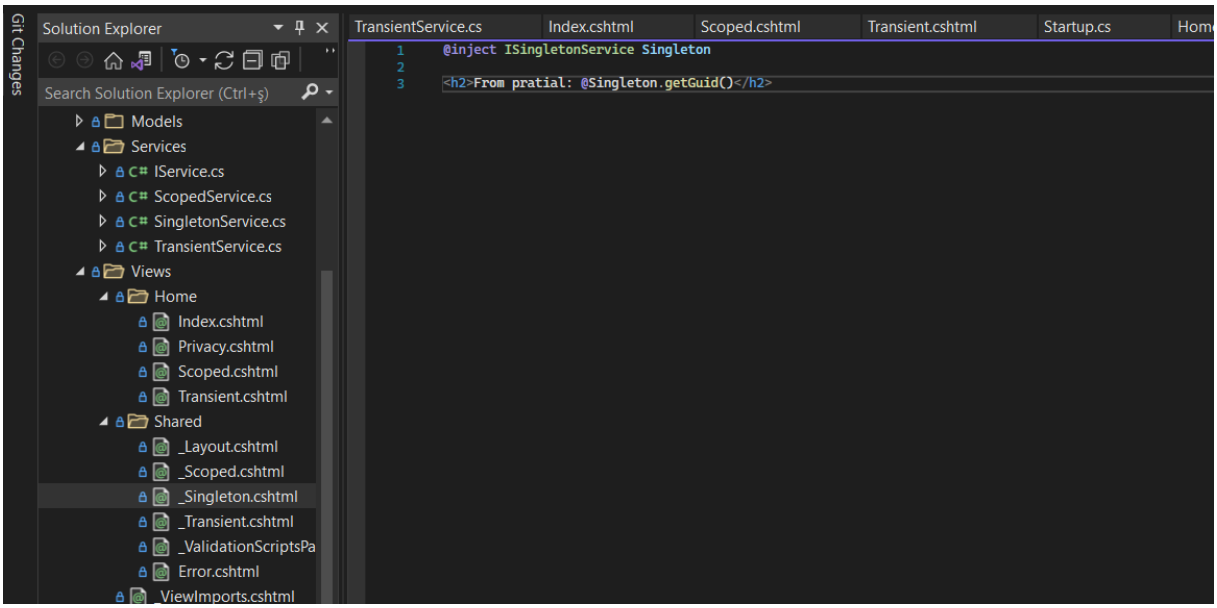
Alttaki görüntülerde 3 adet view var. Her view in içinde ise birer adet partial view var.

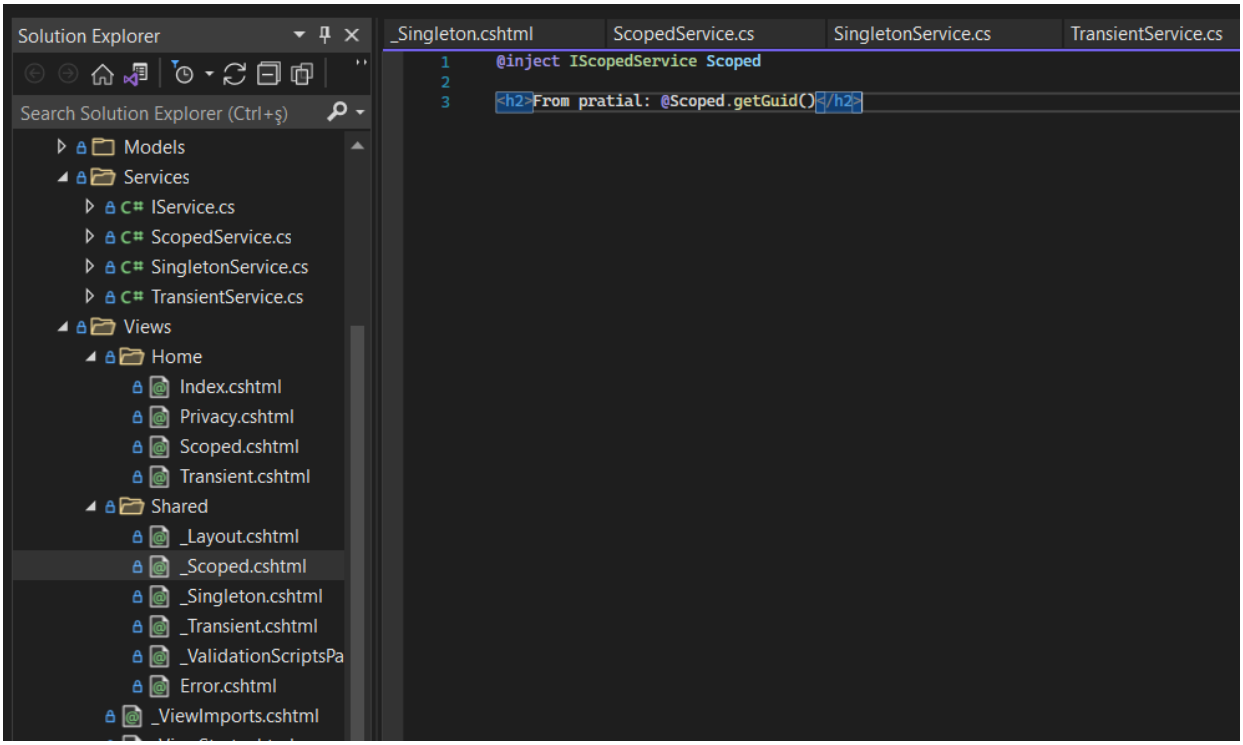

Solution Explorer
Search Solution Explorer (Ctrl+ş)
Solution 'iocContainer' (1 of 1 pro
iocContainer
Connected Services
Dependencies
Properties
wwwroot
Controllers
HomeController.cs
Models
Services
IService.cs
ScopedService.cs
SingletonService.cs
TransientService.cs
Views
Home
Index.cshtml
Privacy.cshtml
Scoped.cshtml
Transient.cshtml
Shared
_Layout.cshtml
_Scoped.cshtml
_Singleton.cshtml
TransientService.cs
Index.cshtml
Scoped.cshtml
Transient.cshtml
Startup.cs
HomeController.cs
_Singleton.cshtml
@model string
<h1>Guid on page: @Model</h1>
<partial name="_Singleton" />
Useful for?
Caching Services
Global Configuration
Business Rules
HttpClients
Reason Why
Persisting state that's useful for the runtime of the application
77 %
No issues found

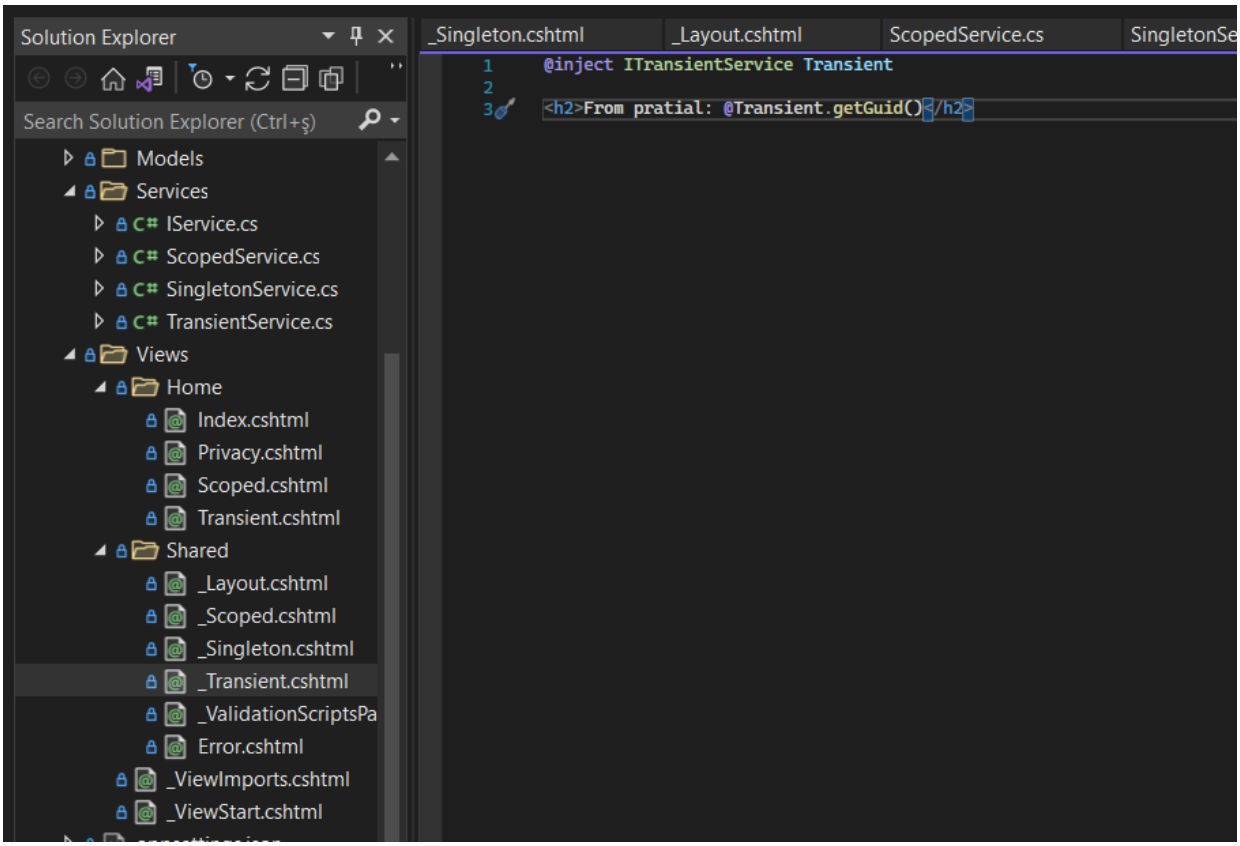

Solution Explorer
Search Solution Explorer (Ctrl+ş)
Models
Services
IService.cs
ScopedService.cs
SingletonService.cs
TransientService.cs
Views
Home
Index.cshtml
Privacy.cshtml
Scoped.cshtml
Transient.cshtml
Shared
_Layout.cshtml
_Scoped.cshtml
_Singleton.cshtml
_Transient.cshtml
_ValidationScriptsPa
Error.cshtml
_ViewImports.cshtml
_ViewStart.cshtml
appsettings.json
Program.cs
HomeController.cs
_Singleton.cshtml
_Layout.cshtml
ScopedService.cs
@model string
<h1>Guid on page: @Model</h1>
<partial name="_Scoped" />
Useful for?
Persisting state throughtout application per request.


Solution Explorer
Search Solution Explorer (Ctrl+ş)
Solution 'iocContainer' (1 of 1 pro
iocContainer
Connected Services
Dependencies
Properties
wwwroot
Controllers
HomeController.cs
Models
Services
IService.cs
ScopedService.cs
SingletonService.cs
TransientService.cs
Views
Home
Index.cshtml
Privacy.cshtml
Scoped.cshtml
Transient.cshtml
Shared
_Layout.cshtml
_Scoped.cshtml
TransientService.cs
Index.cshtml
Scoped.cshtml
Transient.cshtml
Startup.cs
HomeController.cs
_Singleton.cshtml
@model string
<h1>Guid on page: @Model</h1>
<partial name="_Transient" />
Useful for?
Database Access
File Access
Services that should dispose of their state
When you need a fresh instance of an object every single time

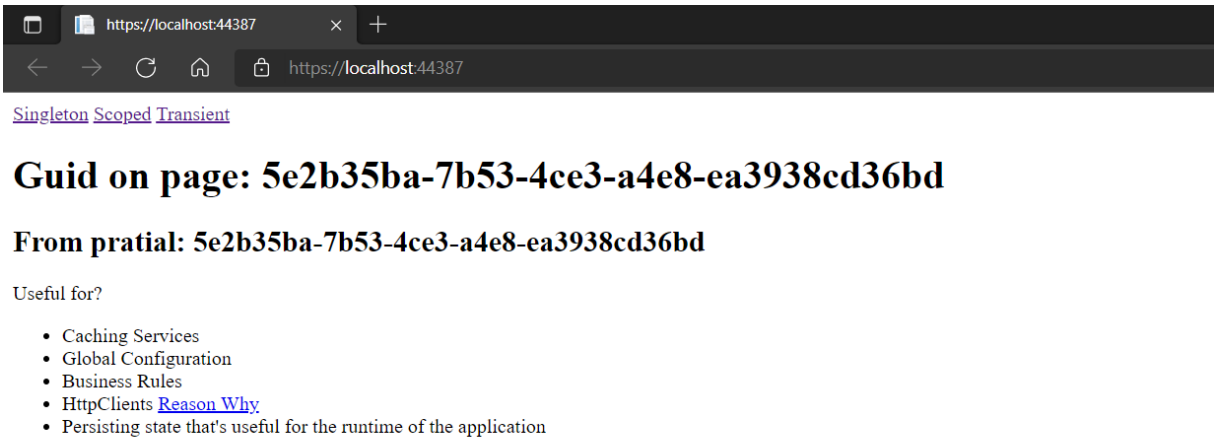
Oluşturulan partial view'ler alttaki çıktılardadır.



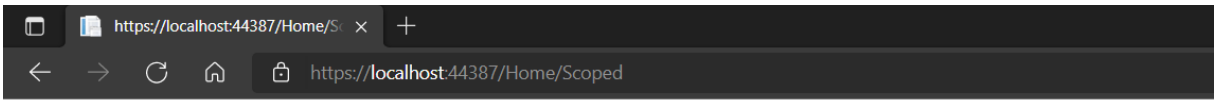



Solution Explorer
Search Solution Explorer (Ctrl+ş)
Models
Services
IService.cs
ScopedService.cs
SingletonService.cs
TransientService.cs
Views
Home
Index.cshtml
Privacy.cshtml
Scoped.cshtml
Transient.cshtml
Shared
_Layout.cshtml
_Scoped.cshtml
_Singleton.cshtml
_Transient.cshtml
_ValidationScriptsPa
Error.cshtml
_ViewImports.cshtml
_ViewStart.cshtml
_Singleton.cshtml
_Layout.cshtml
ScopedService.cs
SingletonSe
1 @inject ITransientService Transient
2
3 <h2>From pratial: @Transient.getGuid()</h2>

Uygulamanın sonucunda ise şu sonuçlara ulaşmaktayız.



Singleton ile oluşturulduğunda tek bir guid nesnesi oluşturuluyor. Uygulama ömrü boyunca isteklere bu Guid nesnesi yanıt vermektedir. Hem sayfa oluşturulurken controller dan gelen değer, hemde partial view de servis çağırımından gelen değer aynı. Sayfayı yenilememize rağmen guid nesnesi değişmiyor. Başta oluşturulan Guid nesnesi uygulama süresi boyunca gelen requestler de kullanılıyor.



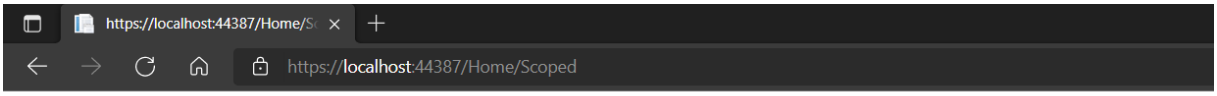
Singleton Scoped Transient

Guid on page: ef1d99d8-3c47-47c2-97c8-5d5909494674

From pratial: ef1d99d8-3c47-47c2-97c8-5d5909494674

Useful for?

- Persisting state throughtout application per request.



Singleton Scoped Transient

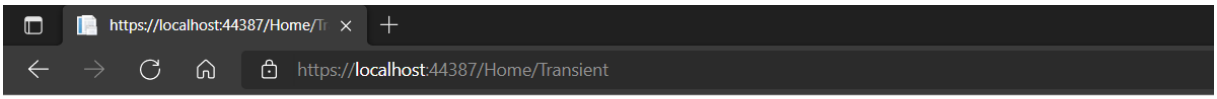
Guid on page: 4613e47f-5946-4a9c-9f8a-179280e9c5ad

From pratial: 4613e47f-5946-4a9c-9f8a-179280e9c5ad

Useful for?

- Persisting state throughtout application per request.

Scoped ile oluşturduğumuz da her request için bir Guid nesnesi oluşturuluyor. Hem controller dan gelen Guid nesnesi, hem de Partial view de servis çağrımından gelen Guid nesnesi aynı. Fakat sayfayı yenilediğimizde yeni request oluşmasından dolayı tekrar bir Guid nesnesi oluşturuluyor. Yani bir request cycle için bir adet nesne oluşturuluyor ve o cycle boyunca isteklere o nesne veriliyor.



Singleton Scoped Transient

Guid on page: 377bd50a-6daf-4289-8362-122f24343749

From pratial: 1ea5ced4-a2fc-403f-895a-d238925e2da6

Useful for?

- Database Access
- File Access
- Services that should dispose of their state
- When you need a fresh instance of an object every single time

Son seçeneğimiz olan transient seçeneğinde ise request sonucunda bir Guid nesnesi oluşturuluyor. Partial view de servis çağrımında ise yeni bir Guid nesnesi oluşturuyor. Sayfayı her yenileyişimiz de tekrar 2 farklı Guid nesnesi oluşturuyor.