First Steps in WebAPI

LUKÁŠ VAVREK

Outline

Routing

Dependency Injection

Service Lifetime


Routing

Basics

Apps should choose a basic and descriptive routing scheme so that URLs are readable and meaningful. Traditional default route:

```
{controller=Home}/{action=Index}/{id?}
```

In WebAPI, all actions must have Route attribute.

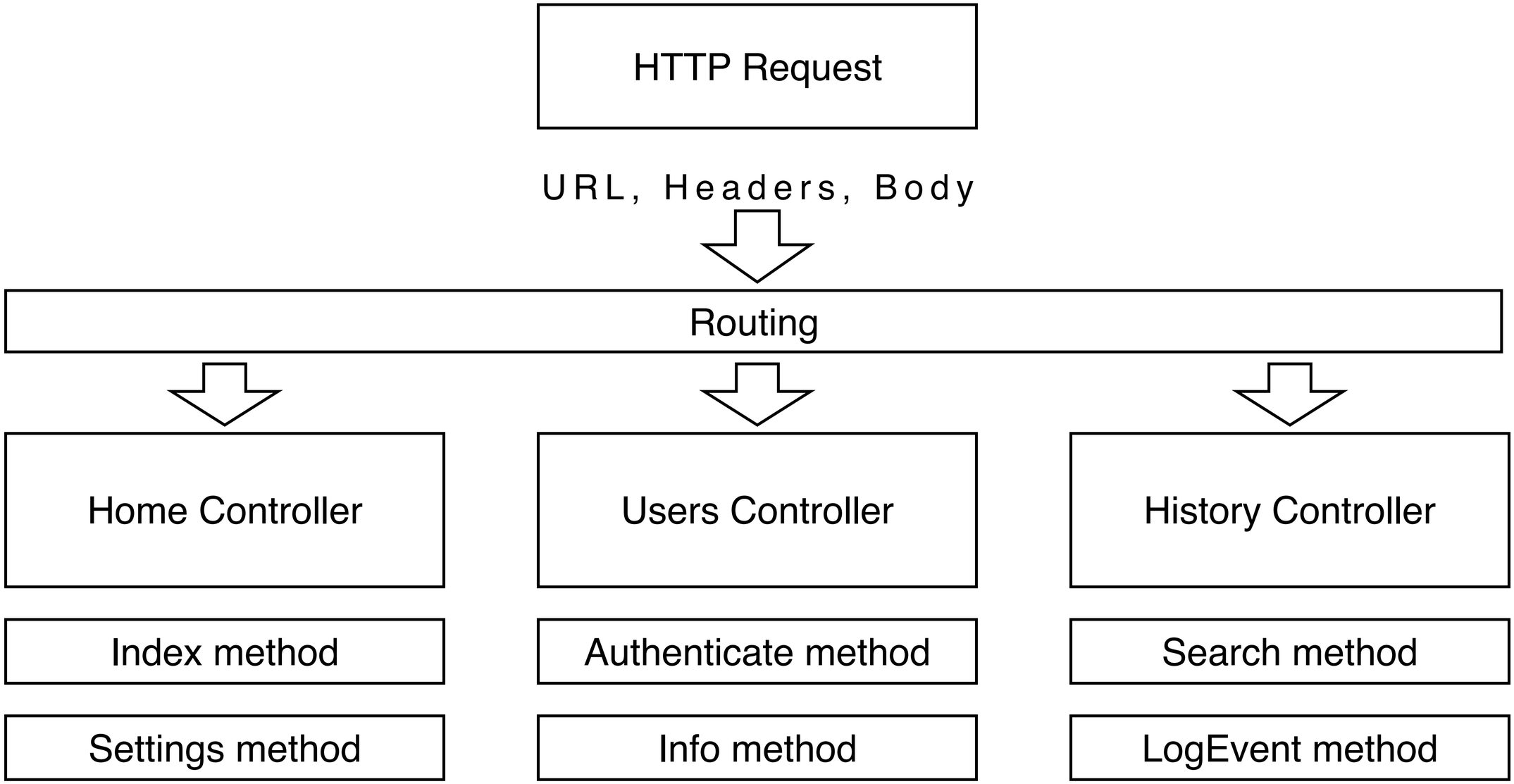
Basics

Web APIs should use attribute routing to model the app's functionality as a set of resources where operations are represented by HTTP verbs.

Many operations on the same logical resource will use the same URL.

Routing needs to be carefully designed.

URL matching



Route attribute

```
[ApiController]
public class HomeController : ControllerBase
{
  [HttpGet]
  [Route("Hello/World")]
  public string Index() => "Ahoj Index.";
}
```

Route attribute

```
[HttpGet]
[Route(“Hello")]
public string HelloSomeone([FromQuery] string name)
    => $"Hello {name}";
```

Route attribute

```
[HttpGet]
[Route("Hello/{name}")]
public string HelloName(string name) => $"Hello {name}";
```

Route attribute

```
[HttpGet]
[Route("HelloNumber/{number:int}")]
public string HelloNumber(int number) => $"Hello n.{number}";
```

DEMO

Routing

Web API


Dependency Injection

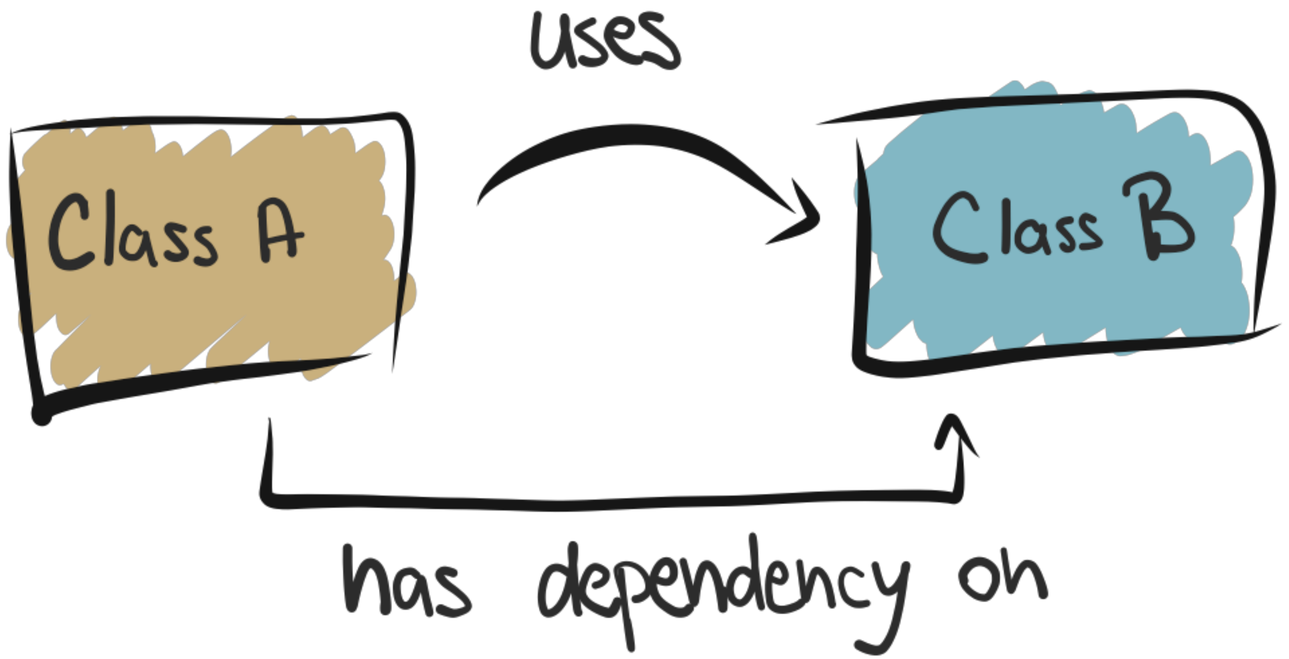
Inversion of Control

A class should not configure its dependencies statically but should be configured by some other class from outside.

A class should depend on abstraction and not upon concretions.

S.O.L.**I**.D. principles

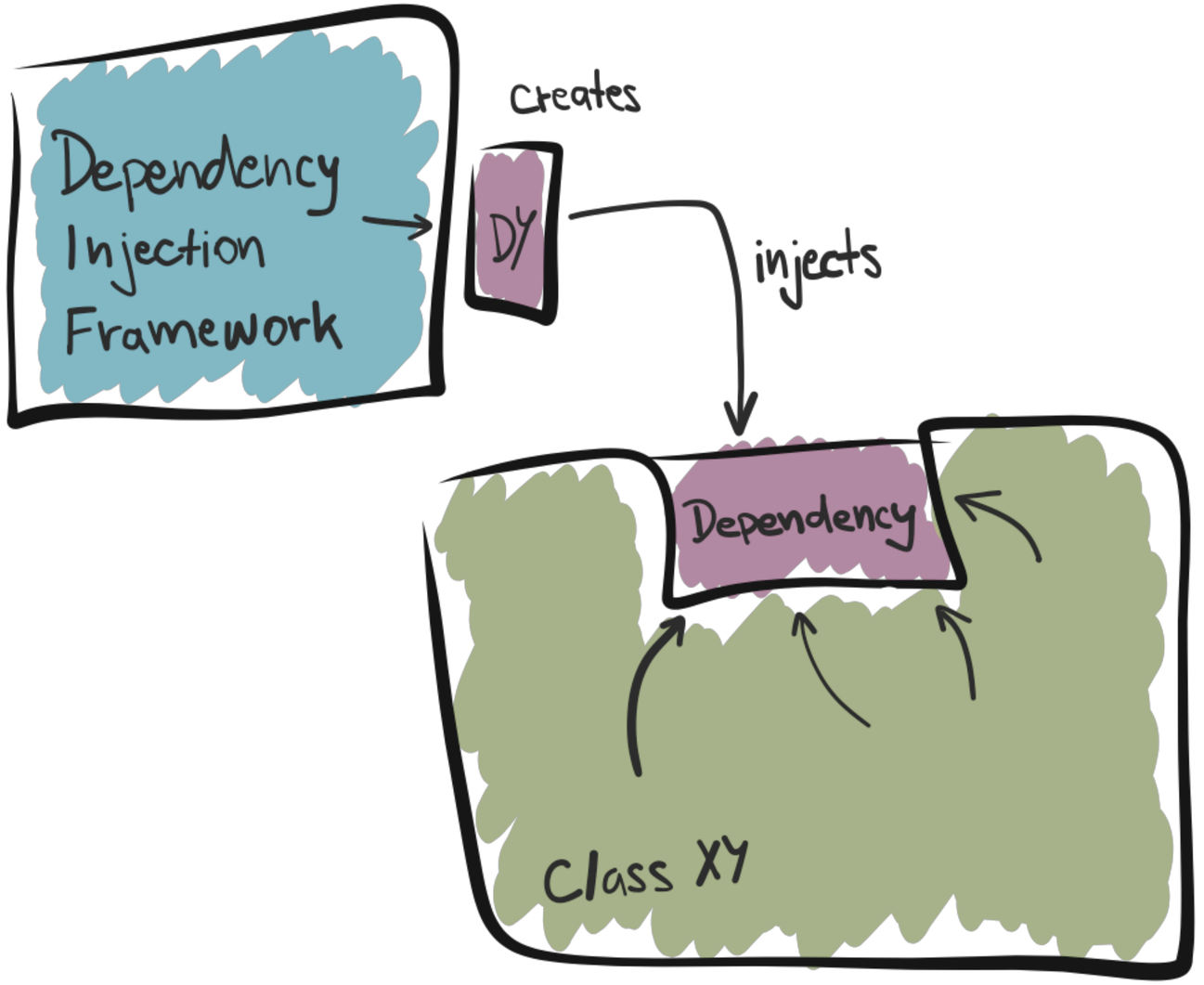
Dependency



Dependency Injection Framework

- Creates the objects
- Resolves dependencies
- Injects dependencies into another objects

DI Framework



Hardcoded Dependency

```
public interface INewsService {}
public class NewsService : INewsService {}

public class Client
{
    private NewsService _newsService;

    public Client()
    {
        _newsService = new NewsService();
    }
}
```

Injected Dependency

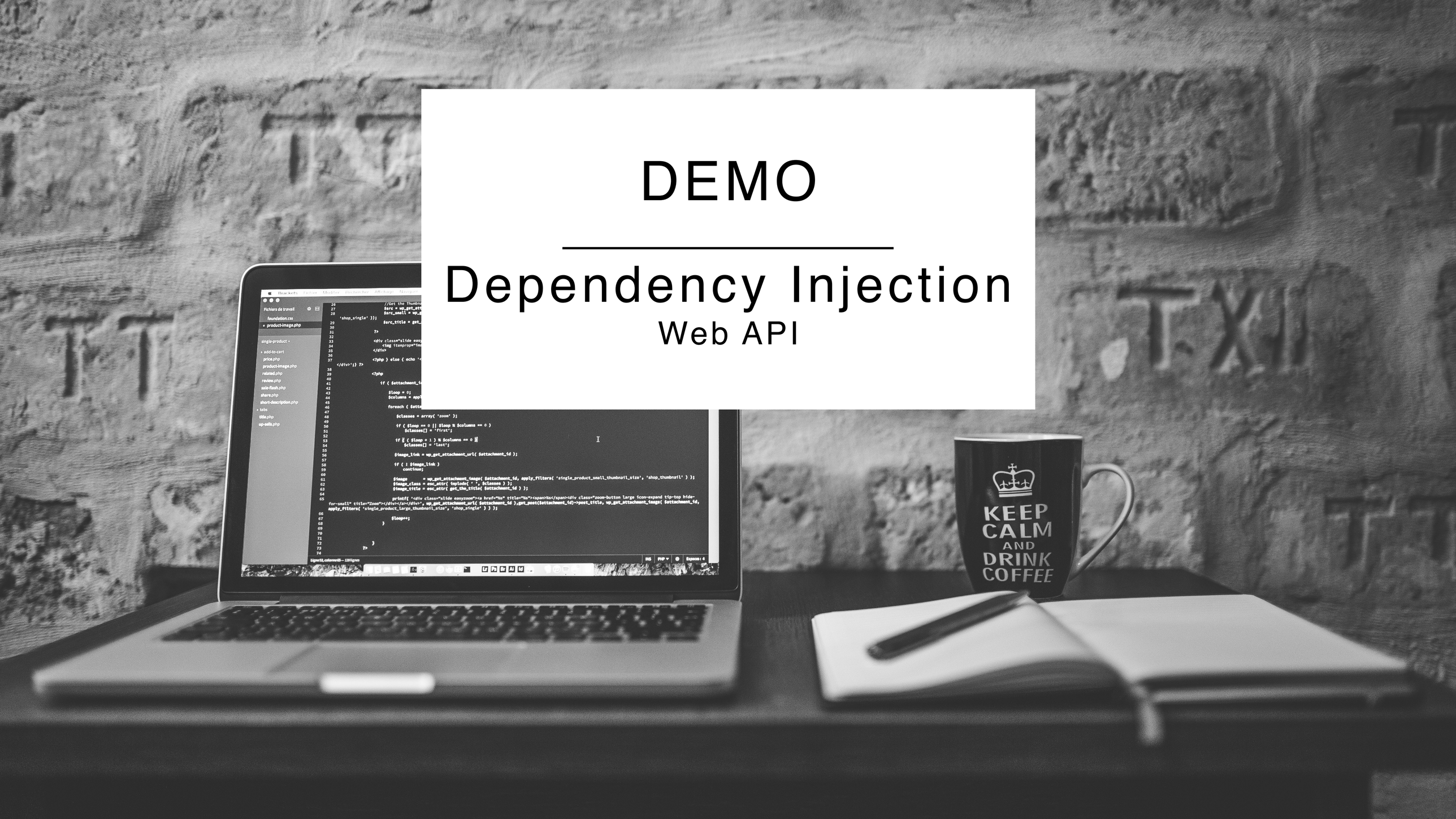
```
public interface INewsService {}
public class NewsService : INewsService {}

public class Client
{
    private NewsService _newsService;

    public Client(NewsService newsService)
    {
        _newsService = newsService;
    }
}
```

Abstract Dependency

```
public interface INewsService {}
public class NewsService : INewsService {}

public class Client
{
    private INewsService _newsService;

    public Client(INewsService newsService)
    {
        _newsService = newsService;
    }
}
```

Registering Dependencies

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IMyDependency, MyDependency>();
    services.AddTransient<IOperationTransient, Operation>();
    services.AddScoped<IOperationScoped, Operation>();
    services.AddSingleton<IOperationSingleton, Operation>();
    services.AddSingleton<IOperationSingletonInstance>(new
Operation(Guid.Empty));

    // OperationService depends on each of the other Operation
types.
    services.AddTransient<OperationService, OperationService>();
}
```



DEMO

Dependency Injection

Web API


Service Lifetime

Scopes and Lifetime

Each registered service has a lifetime associated with it.

- Transient
- Scoped
- Singleton

Transient

Transient lifetime services are created each time they’re requested. This lifetime works best for lightweight, stateless services.

Scoped

Scoped lifetime services are created once per request. With each request being processed, new scope is created.

Service Lifetime

Singleton

Singleton lifetime services are created the first time they're requested. Every subsequent request uses the same instance.

If the app requires singleton behaviour, allowing the service container to manage the service's lifetime is recommended. Don't implement the singleton design pattern and provide user code to manage the object's lifetime in the class.

DEMO

Services Lifetime

Web API


A 3
B 3
A 4
B 4
Q & A