



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.

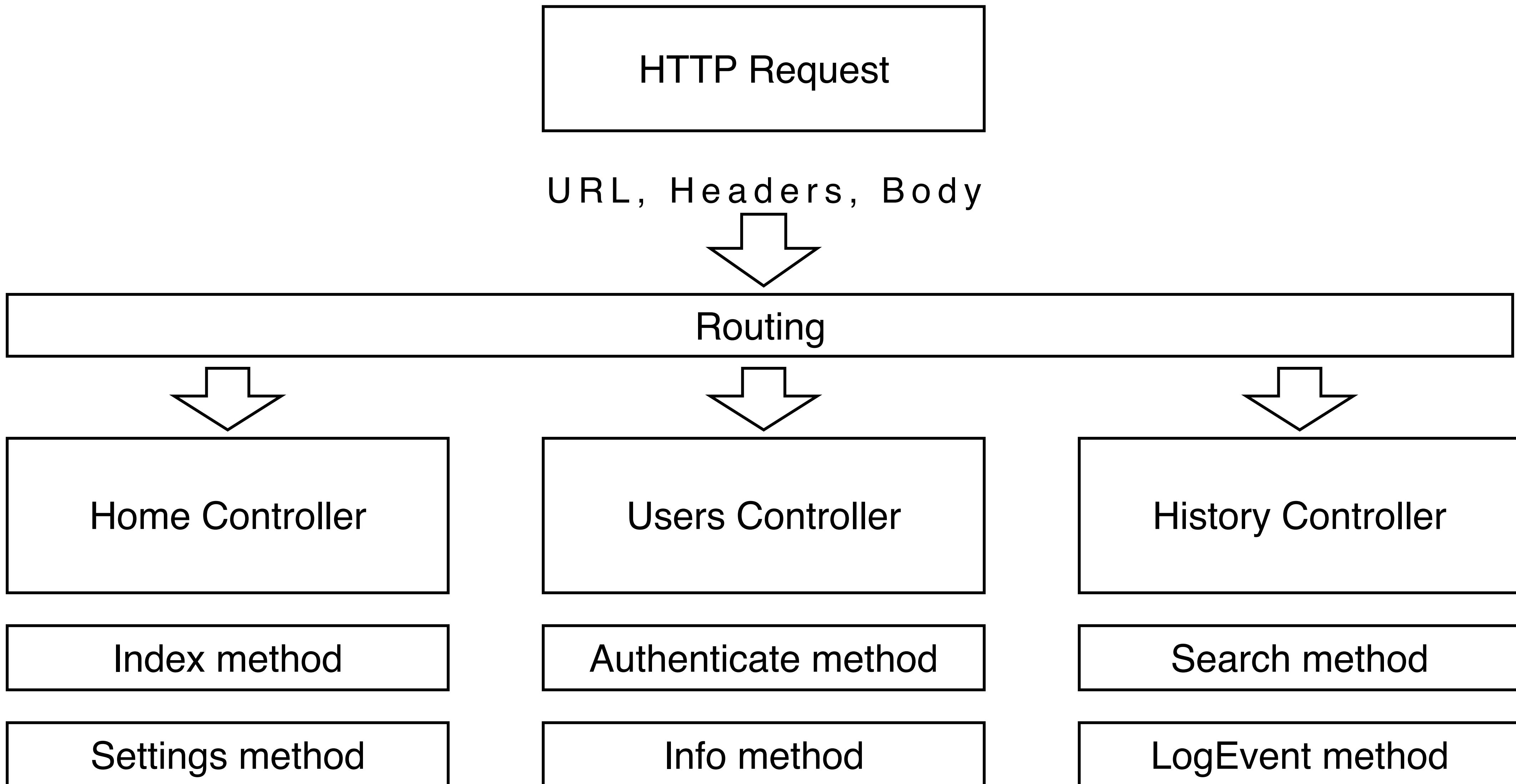
Basics

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

The screenshot shows the Brackets IDE interface with a dark theme. The top menu bar includes 'Brackets', 'Fichier', 'Modifier', 'Rechercher', 'Affichage', and 'Navigateur'. On the left, a sidebar lists files: 'foundation.css', 'product-image.php' (which is the active tab), 'single-product.php', 'add-to-cart.php', 'price.php', 'product-image.php', 'related.php', 'review.php', 'sale-flash.php', 'share.php', 'short-description.php', 'tabs.php', 'title.php', and 'up-sells.php'. The main editor area displays the following PHP code:

```
26             //Get the Thumbnail
27             $src = wp_get_attachment_image_src($attachment_id, 'single_product_small_thumbnail_size');
28             $src_small = wp_get_attachment_image_src($attachment_id, 'single_product_small_thumbnail_size');
29             $src_title = get_post_meta($attachment_id, '_shop_thumbnail_title', true);
30             ?>
31         <?php
32         <div class="slide easyzoom" style="display: flex; align-items: center;">
33             
34         </div>
35         <?php } else { echo '<div>' . $src['url'] . '</div>';
36     </div>`} ?>
37     <?php
38
39     if ( $attachment_id ) {
40         $loop = 0;
41         $columns = apply_filters( 'single_product_columns', 3 );
42         foreach ( $attachments as $attachment ) {
43             $classes = array( 'zoom' );
44             if ( $loop == 0 || $loop % $columns == 0 ) {
45                 $classes[] = 'first';
46             }
47             if ( ( $loop + 1 ) % $columns == 0 ) {
48                 $classes[] = 'last';
49             }
50             $image_link = wp_get_attachment_url( $attachment_id );
51             if ( ! $image_link )
52                 continue;
53             $image      = wp_get_attachment_image( $attachment_id, apply_filters( 'single_product_small_thumbnail_size', 'shop_thumbnail' ) );
54             $image_class = esc_attr( implode( ' ', $classes ) );
55             $image_title = esc_attr( get_the_title( $attachment_id ) );
56
57             printf( '<div class="slide easyzoom"><a href="%s" title="%s"><span>%s</span><div class="zoom-button large icon-expand tip-top hide-for-small" title="Zoom"></div></a></div>', wp_get_attachment_url( $attachment_id ), get_post( $attachment_id )->post_title, wp_get_attachment_image( $attachment_id, apply_filters( 'single_product_large_thumbnail_size', 'shop_single' ) ) );
58
59             $loop++;
60         }
61     }
62
63     <?php
64     <?php
65     <?php
66     <?php
67     <?php
68     <?php
69     <?php
70     <?php
71     <?php
72     <?php
73     <?php
74     <?php
```

At the bottom, status bars show 'Ligne 53, colonne 65 - 1281 lignes', 'INS PHP', and 'Espaces 4'.



Dependency Injection



The image shows a person's hands working on a laptop keyboard. A smartphone is connected to the laptop via a USB cable. In the background, a monitor displays the Android Studio interface, specifically the code editor for a Java file named `Video.java`. The code is related to dependency injection, showing methods like `startPollingForUpdates` and `stopPollingForUpdates` which interact with an `AlarmManager`.

```
public void startPollingForUpdates(@NotNull String uid) {
    Timber.d("startPollingForUpdates(), with this uid: %s", uid);
    long interval = 1000 * 60; // check every minute
    long start = System.currentTimeMillis() + interval;
    Intent i = LiveEventsService.getCheckLiveEventIntent(this, uid);
    pi = PendingIntent.getService(this, 0, i, 0);
    alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, start, interval, pi);
}

public void stopPollingForUpdates() {
    Timber.d("stopPollingForUpdates()");
    if (pi != null) {
        AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
```

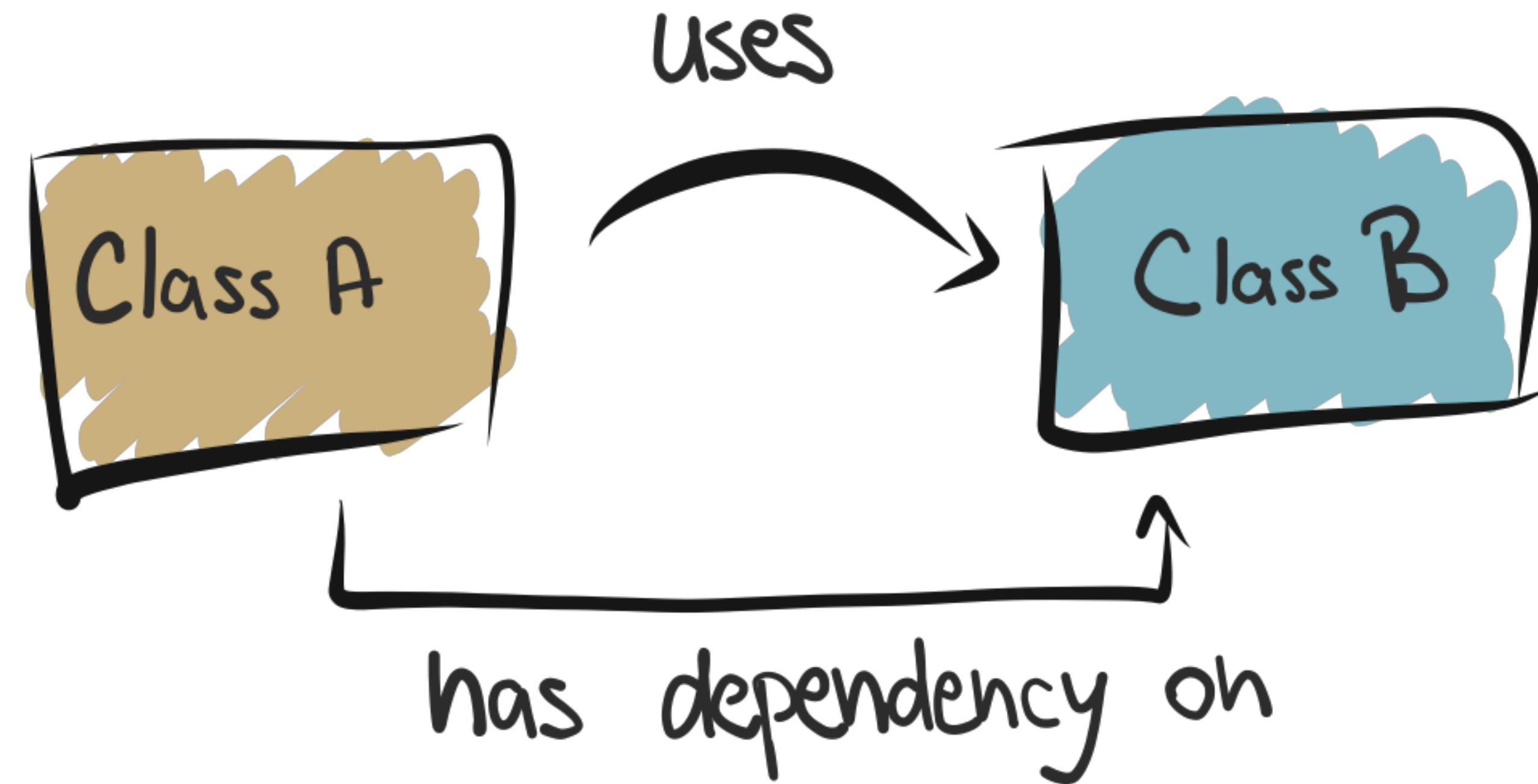
Inversion of Control

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

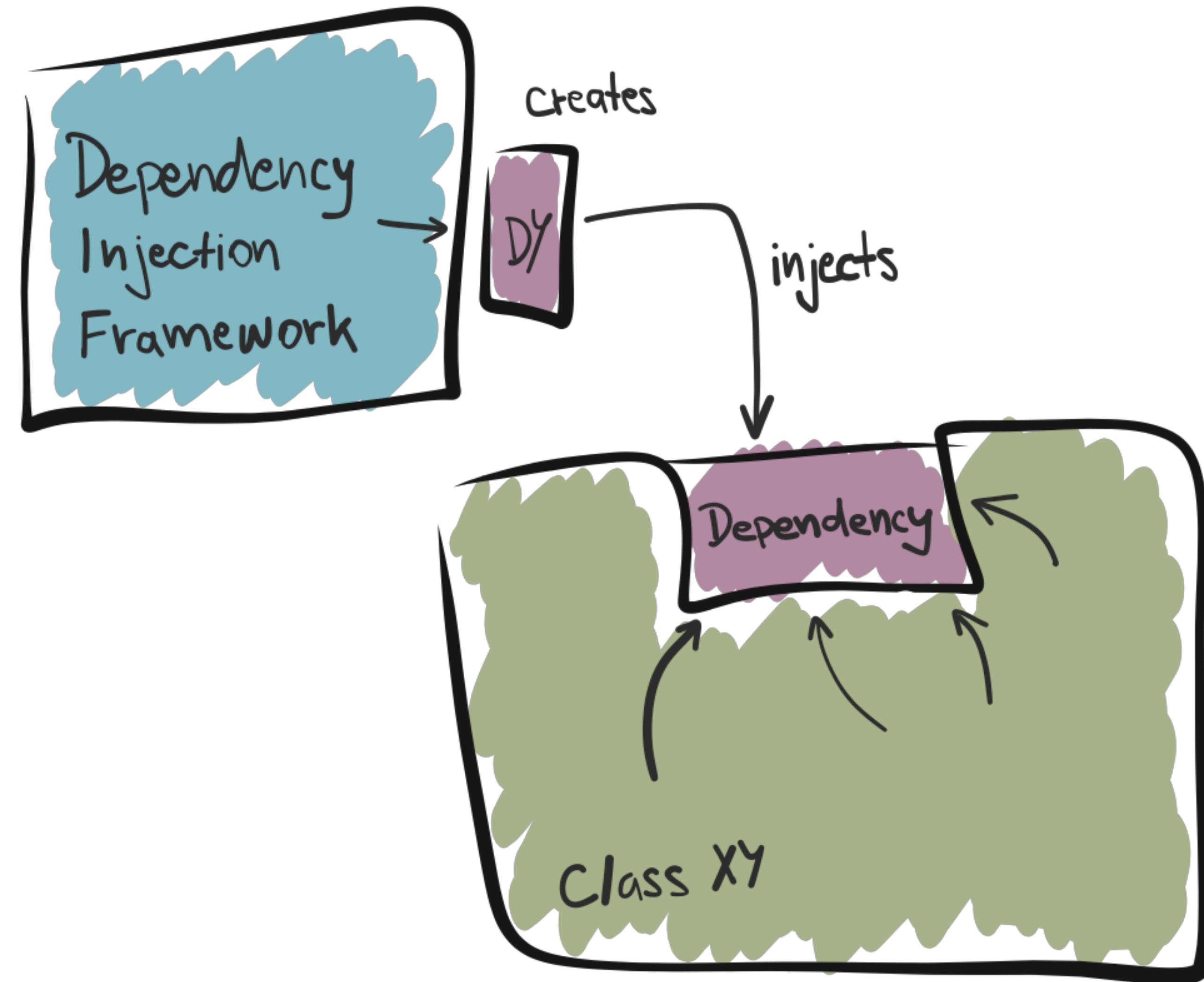
Dependency



Dependency Injection Framework

- ④ Creates the objects
- ④ Resolves dependencies
- ④ 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



The image shows a person working on a laptop, with a smartphone connected via a USB cable. The laptop screen displays Android Studio with code related to service lifetime. The code includes methods for starting and stopping polling for updates, and managing alarm managers. The background is a blurred office environment.

```
public void startPollingForUpdates(@NotNull String uid) {
    Timber.d("startPollingForUpdates(), with this uid: %s", uid);
    long interval = 1000 * 60; // check every minute
    long start = System.currentTimeMillis() + interval;
    Intent i = LiveEventsService.getCheckLiveEventIntent(this, uid);
    pi = PendingIntent.getService(this, 0, i, 0);
    AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, start, interval, pi);
}

public void stopPollingForUpdates() {
    Timber.d("stopPollingForUpdates()");
    if (pi != null) {
        AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
```

Scopes and Lifetime

Each registered service has a lifetime associated with it.

- ④ Transient
- ④ Scoped
- ④ 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.

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



Q & A

