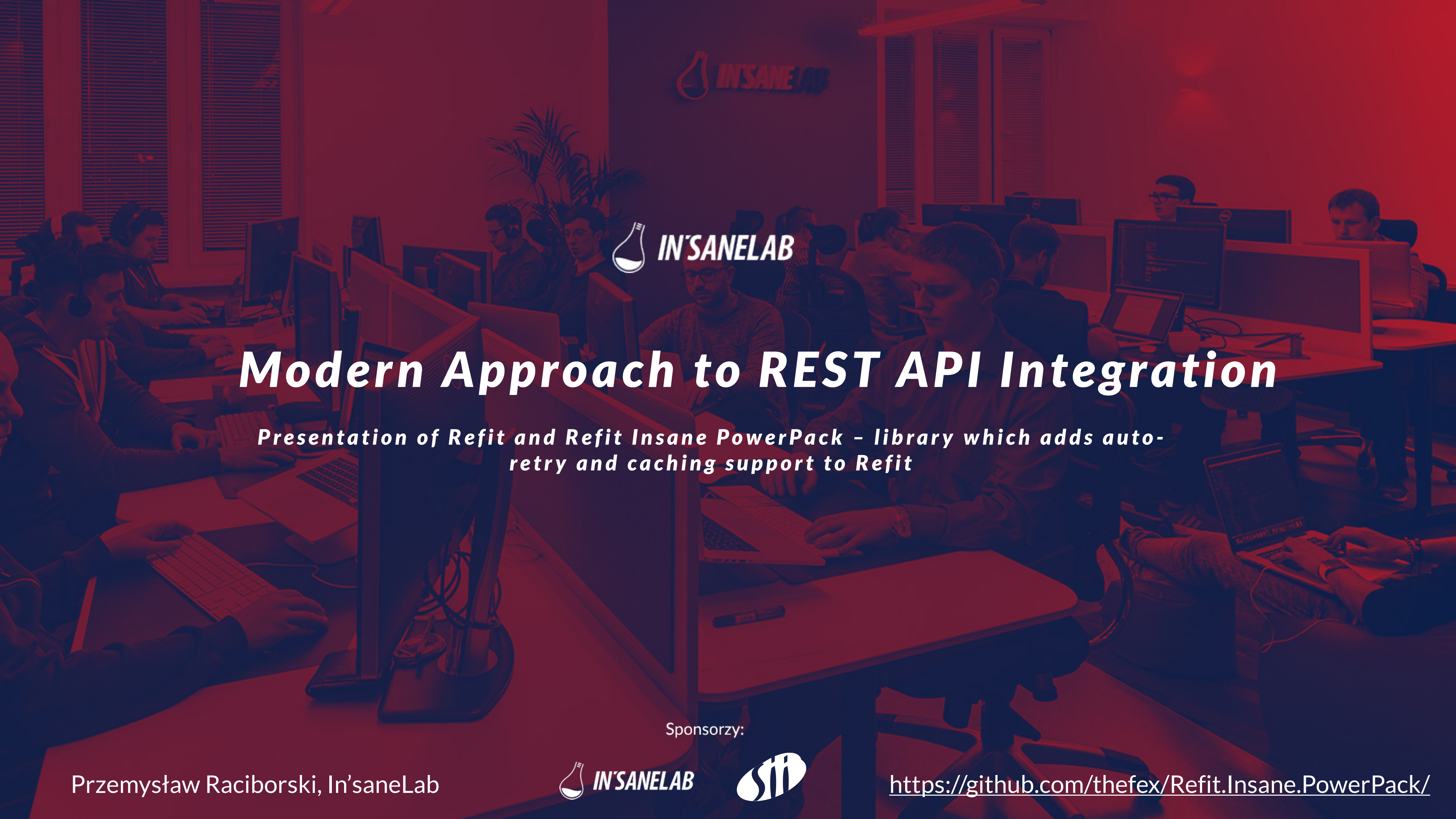


Modern Approach to REST API Integration

Presentation of Refit and Refit Insane PowerPack – library which adds auto-retry and caching support to Refit

Sponsorzy:

Przemysław Raciborski, In'saneLab

https://github.com/thefex/Refit.Insane.PowerPack/

WHAT IS REFIT?

Refit is an automatic type-safe REST library which converts your interfaces into fully integrated REST Connection module.

It has been created by Paul Betts - https://github.com/paulcbetts

Get it on nuget!

Install-Package Refit

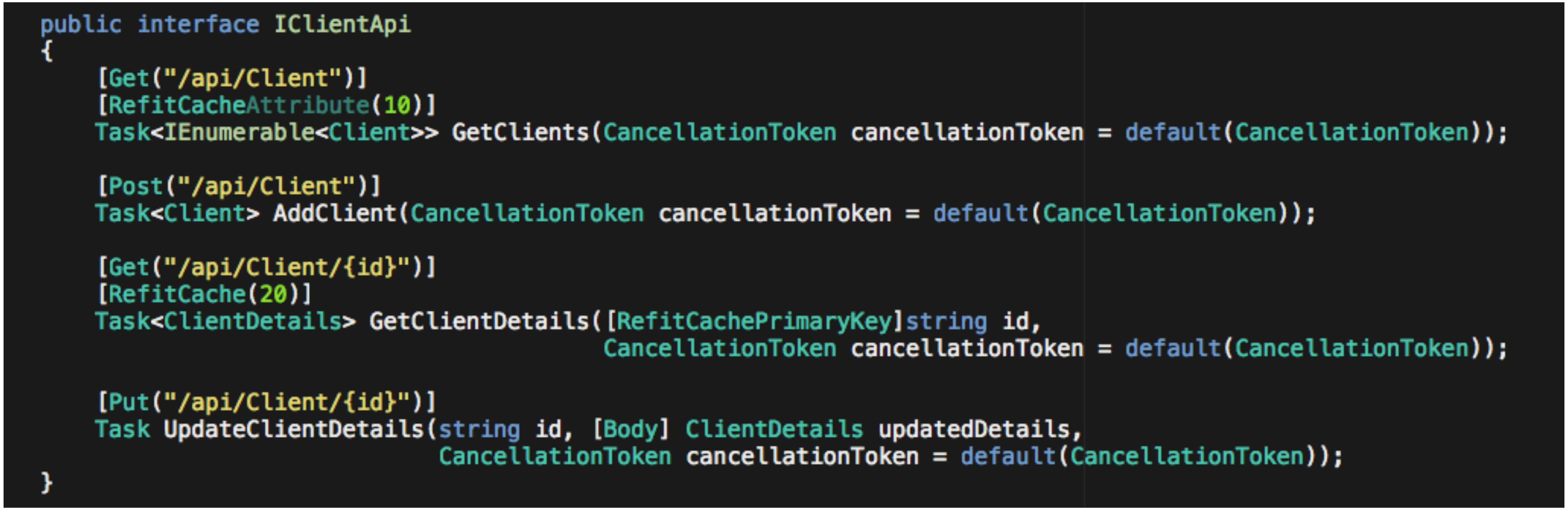
IF YOU ARE WRITING REST CONNECTION MODULE AND YOU ARE NOT USING REFIT...

...THEN YOU ARE ROBBING YOUR EMPLOYER

LET’S INTEGRATE SOME API WITH REFIT

For purpose of this presentation I have created simple API with few methods - http://apitestprezentacja.azurewebsites.net/swagger/

Using Refit, all what we have to do is describe our API methods using C# Interfaces and Refit Attributes which is self describing.



```
public interface IClientApi
{
    [Get("/api/Client")]
    [RefitCacheAttribute(10)]
    Task<IEnumerable<Client>> GetClients(CancellationToken cancellationToken = default(CancellationToken));

    [Post("/api/Client")]
    Task<Client> AddClient(CancellationToken cancellationToken = default(CancellationToken));

    [Get("/api/Client/{id}")]
    [RefitCache(20)]
    Task<ClientDetails> GetClientDetails([RefitCachePrimaryKey]string id,
                                         CancellationToken cancellationToken = default(CancellationToken));

    [Put("/api/Client/{id}")]
    Task UpdateClientDetails(string id, [Body] ClientDetails updatedDetails,
                             CancellationToken cancellationToken = default(CancellationToken));
}
```

WHAT EVERY <u>GOOD</u> HTTP API INTEGRATION SHOULD TAKE CARE OF?

HTTP Protocol by design does not guarantee that message will be delivered to the client. For example: Mobile Internet often miss packets, it might be slow or our API server might not process request in reasonable time.

Therefore we should prepare our apps to repeat requests that have failed for instance due to timeouts.

Besides that we can notice that some kind of data returned by our API does not change frequently.
To provide best user experience in our applications we should cache appropiate data and call our HTTP Service only when we predict/know that requested data might have changed.

IN’SANELAB INSANE WORK

REFIT.INSANE.POWERPACK LIBRARY

To make **yours** and our☺ life simpler we have created Refit extensions library, so called “Refit.Insane.PowerPack”.

We offer you:

- Attribute based ”Auto Request Retry” - when HTTP request fails we automatically resend it
- Attribute based automatic Caching support – add few [RefitCache**Attribute**s] and you get your responses cached!

HOW TO USE REQUESTS FAILURE AUTO-RETRY?

If you plan to use Auto-Retry feature add assembly attribute in one of your .cs file – [**RefitRetry**Attribute(retryCount)], ex.

[assembly:RefitRetryAttribute(3)]

As a second **RefitRetryAttribute** constructor argument you can pass HttpStatusCodes array (response status codes on which auto-retry feature will work).

HOW TO USE REFIT RESPONSE CACHING FEATURE?

1. Add **[RefitCache] attribute** before method definition in your Refit API interface. Make sure to append this attribute only to methods which have any return type – or you will get runtime exception.
   You can pass TimeSpan in **RefitCacheAttribute** constructor argument so your cache will be automatically invalidated after this time.

2. In case your **REST** Method takes any parameters (like resource id) append [**RefitCachePrimaryKeyAttribute**] next to this parameter.
   If you are using non primitive type as request parameter (like request custom model class) you have to override ".ToString()" method in this class. **ToString()** should return unique, cache primary key.

HOW TO USE REFIT RESPONSE CACHING FEATURE?

Sometimes you want to update **or** clear cache manually.

Imagine a case when you are having profile edit view in Mobile App.

You would use **RefitCache** to cache **GetProfileDetails** response.
Later on, you add “**UpdateProfileDetails**” method thus you expect that **GetProfileDetails** cache is either cleared or updated when you save profile details changes.

Due to technical limitations of C# Expressions and **Akavache** (which **Refit.Insane.PowerPack** caching uses) it was impossible to implement this feature in AOP (attribute based / aspect oriented programming) manner.

Nevertheless, I have exposed a ”Cache Service” class which gives you ability to update and clear cache of requested REST API method.

Simply, use: **RefitCacheService.Instance.UpdateCache/ClearCache** methods.

REMARKS ON IRESTSERVICE

All of the features in **Refit.Insane.PowerPack** library have been wrapped around **IRestService** interface.

If you are planning to use this library, simply inject instance of the **IRestService** instead of using **RestService.For<TApi>.**

Use **RestServiceBuilder** class to build instance of **IRestService** implementation.

HANDLING CUSTOM RESPONSE ON NON-OK/200 RESPONSE

Proper, well-designed REST API should make use of response HTTP Status codes other than 200. Therefore, we should be prepared to handling data contained for instance inside response with status code 403 (Forbidden).

IRestService, and in particular its implementation called **RefitRestService** has been designed to simplify this process.
Simply,

1. Inherit from **RefitRestService.** (you can pass **RefitRestService as RefitRestServiceBuilder.Build(..) parameter**)
2. Override three methods –
   CanPrepareResponse(ApiException fromApiException),
   PrepareResponse(ApiException fromApiException),
   PrepareResponse<TResult>(ApiException fromApiException)

   ApiException contains response body and status code that you can process further.

COMMON PITFALLS

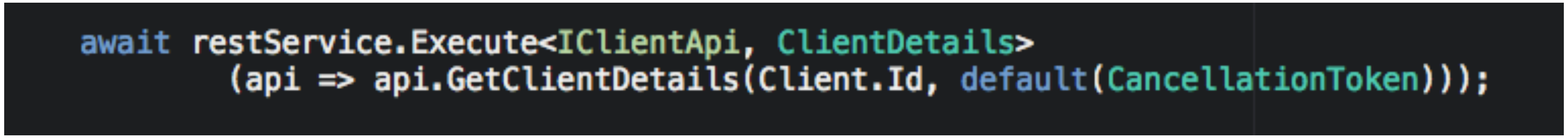
As our library strongly depends on C# **Expression<..>** analysis there are some catches that we are aware of.

On the next slides I have listed the most common problems.

COMMON PITFALLS – USING PROPERTIES AS ARGUMENTS

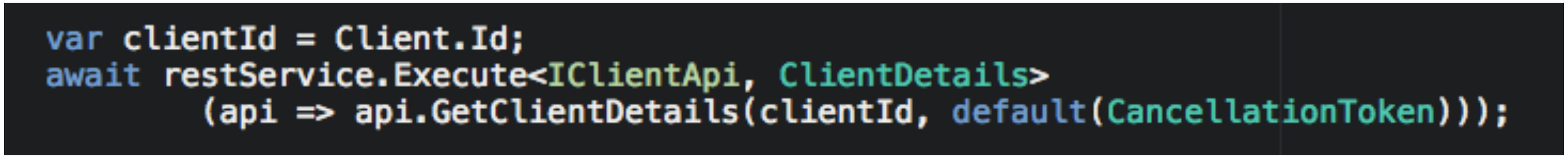
Currently, our expression analysis engine has some problems when you pass property as **Refit Interface** Method Call parameter.

Instead of:



```
    await restService.Execute<IClientApi, ClientDetails>
            (api => api.GetClientDetails(Client.Id, default(CancellationToken)));
```

Use:

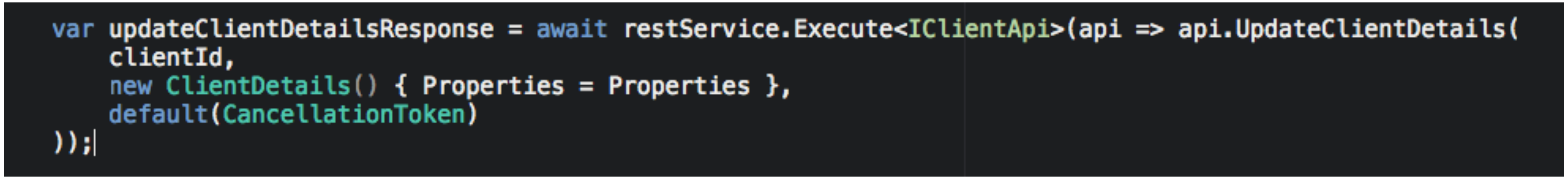


```
  var clientId = Client.Id;
  await restService.Execute<IClientApi, ClientDetails>
          (api => api.GetClientDetails(clientId, default(CancellationToken)));
```

COMMON PITFALLS – CREATING NEW OBJECT INSIDE ARGUMENT

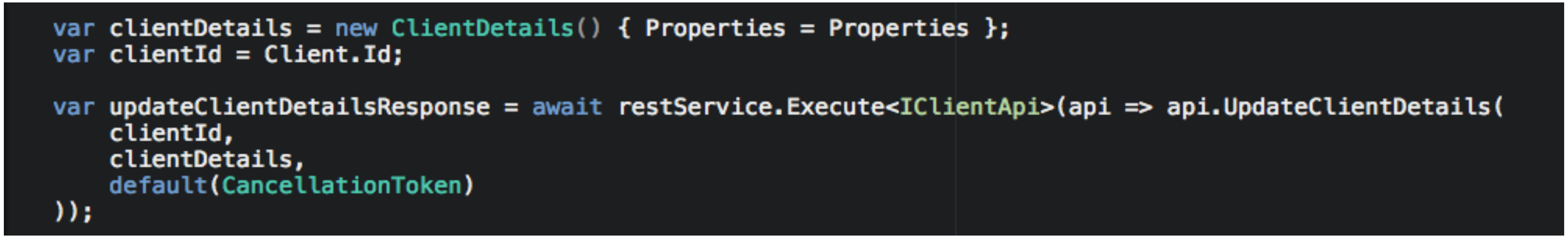
C# Expressions do not support creating new object inside „**Expression**” call.

Instead of:



```
var updateClientDetailsResponse = await restService.Execute<IClientApi>(api => api.UpdateClientDetails(
    clientId,
    new ClientDetails() { Properties = Properties },
    default(CancellationToken)
));
```

Use:



```
var clientDetails = new ClientDetails() { Properties = Properties };
var clientId = Client.Id;

var updateClientDetailsResponse = await restService.Execute<IClientApi>(api => api.UpdateClientDetails(
    clientId,
    clientDetails,
    default(CancellationToken)
));
```

COMMON PITFALLS

General rule of thumb is: if something does not work, try to move expression call argument to local variable.

THANKS FOR YOUR ATTENTION!

SEE YOU ON NEXT #XAMARINES!

PRZEMYSŁAW RACIBORSKI

IN'SANELAB XAMARINES