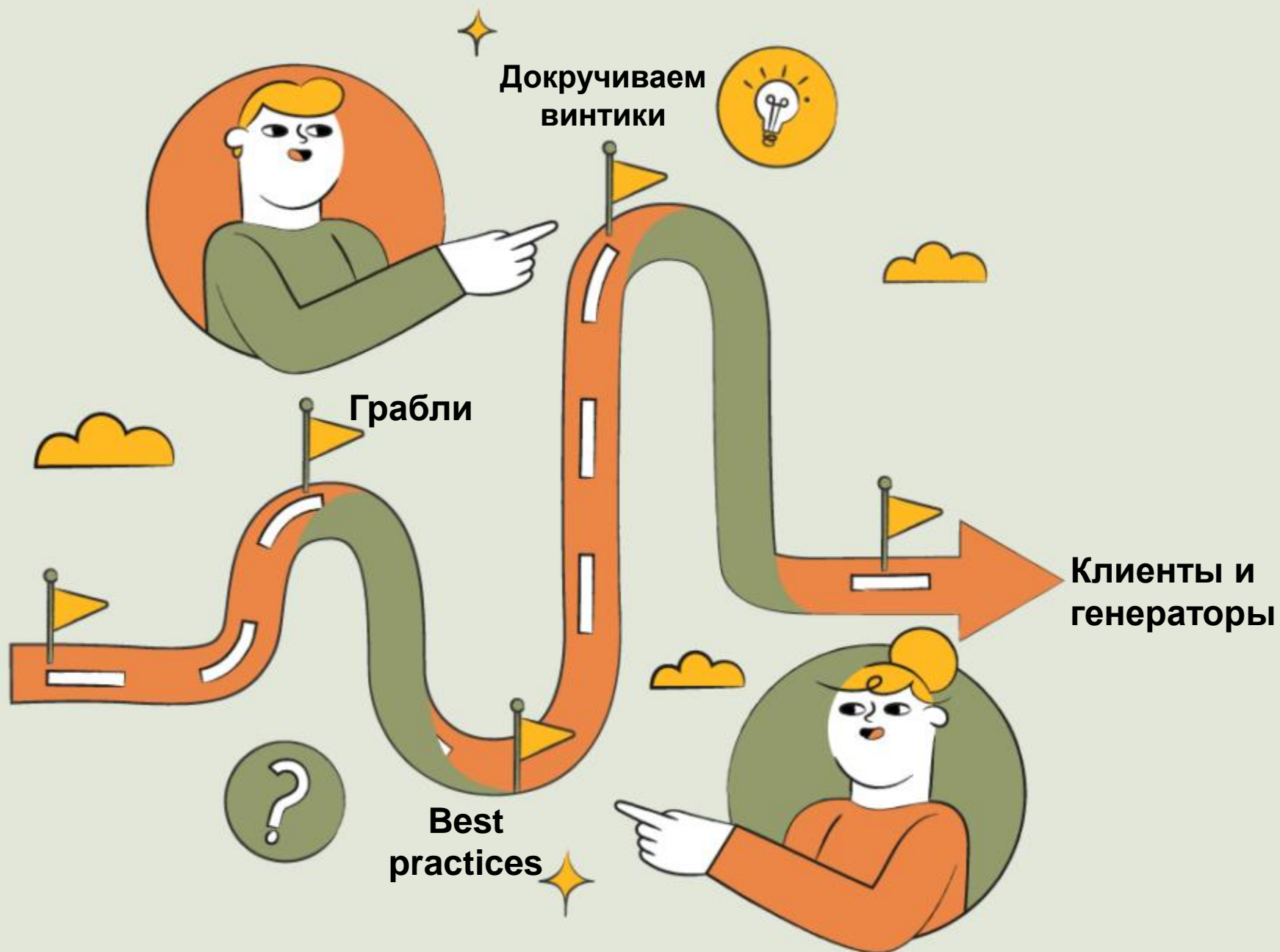


Abstract geometric lines in the top-left corner of the page, consisting of several thin black lines forming various polygons and intersecting each other.

REST API КЛИЕНТЫ ДЛЯ C#

Квашнин Артём

HttpClien
t



Задача:

1. Выполнить POST запрос.
2. Передать сереализованный объект в теле запроса.
3. Передать параметры в строке запроса.
4. Десереализовать ответ.

```
var queryString = new Dictionary<string, string>()
{
    { "foo", "bar" }
};

var requestUri = QueryHelpers.AddQueryString("resource/{id}", queryString);

using var response = await _httpClient.PostAsJsonAsync(requestUri, content);

return await response.Content.ReadFromJsonAsync<ResponseModel>();
```

Задача:

- 1.Выполнить POST запрос.
- 2.Передать сереализованный объект в теле запроса.
- 3.Передать аргументы.
- 4.Десереализовать ответ.
- 5.Обработать потенциальную ошибку.
- 6.Передать заголовок.

```
var queryString = new Dictionary<string, string>()
{
    { "foo", "bar" }
};

var jsonContent = JsonConvert.Create(content);
jsonContent.Headers.Add("X-custom-header", "Value");

var requestUri = QueryHelpers.AddQueryString("resource/{id}", queryString);

try
{
    using var response = await _httpClient.PostAsync(requestUri, jsonContent);

    return await response.Content.ReadFromJsonAsync<ResponseModel>();
}
catch (Exception e)
{
    // Handle
}
```

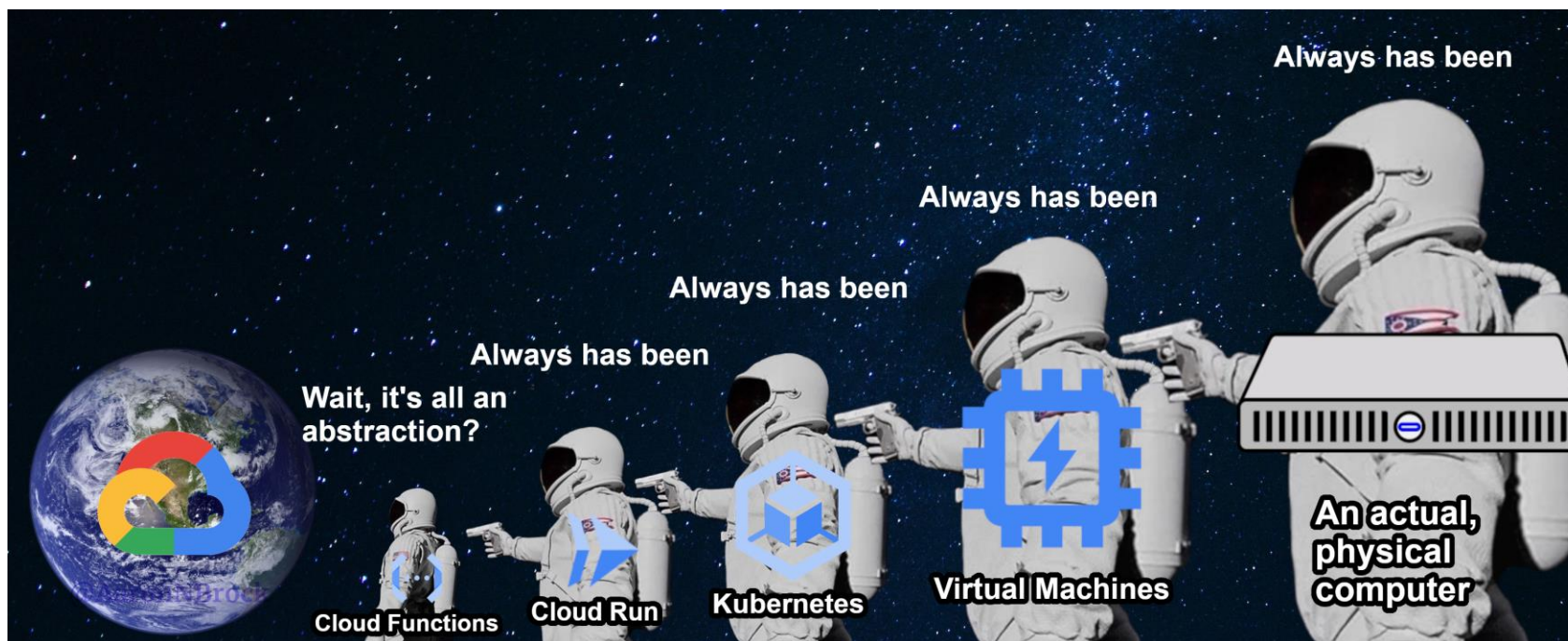
Задача:

- 1.Выполнить POST запрос.
- 2.Передать сереализованный объект в теле запроса.
- 3.Передать аргументы.
- 4.Десереализовать ответ.
- 5.Обработать потенциальную ошибку.
- 6.Передать заголовок.
- 7.Использовать XML, а не JSON.**

ЧТО НЕ ТАК С ЧИСТЫМ HTTPCLIENT?

- 1.Обработка исключений
- 2.Устойчивость к новым требованиям
- 3.Опечатки
- 4.Генераторы кода, не спасут
- 5.Грабли HttpClient

ЛЮБУЮ АРХИТЕКТУРНУЮ ПРОБЛЕМУ МОЖНО
РЕШИТЬ ДОБАВЛЕНИЕМ ДОПОЛНИТЕЛЬНОГО СЛОЯ
АБСТРАКЦИИ, КРОМЕ ПРОБЛЕМЫ БОЛЬШОГО
КОЛИЧЕСТВА АБСТРАКЦИЙ



ПРОЙДЁМСЯ ПО ГРАБЛЯМ!

ПРИВЕТ, ГРАБЛИ!
ЭТО СНОВА Я!



ЧТО ТЫТ HE TAK?

```
for (int i = 0; i < 10; i++)  
{  
    using (var client = new HttpClient())  
    {  
        using var result = await  
client.GetAsync("https://api.github.com");  
  
        Console.WriteLine(result.StatusCode);  
    }  
}
```

```
Администратор: Командная
C:\Users\eugen>netstat

Активные подключения

Имя      Локальный адрес      Внешний адрес      Состояние
TCP      127.0.0.1:53329      Eugene:53330      ESTABLISHED
TCP      127.0.0.1:53330      Eugene:53329      ESTABLISHED
TCP      192.168.0.112:55247  hem09s03-in-f14:https  TIME_WAIT
TCP      192.168.0.112:55249  hem09s03-in-f14:https  TIME_WAIT
TCP      192.168.0.112:55251  hem09s03-in-f14:https  TIME_WAIT
TCP      192.168.0.112:55252  hem09s03-in-f4:https   TIME_WAIT
TCP      192.168.0.112:55253  hem09s03-in-f14:https  TIME_WAIT
TCP      192.168.0.112:55254  hem09s03-in-f4:https   TIME_WAIT
TCP      192.168.0.112:55256  hem09s03-in-f4:https   TIME_WAIT
TCP      192.168.0.112:55258  hem09s03-in-f4:https   TIME_WAIT
TCP      192.168.0.112:55259  hem09s03-in-f14:https  TIME_WAIT
TCP      192.168.0.112:55260  hem09s03-in-f4:https   TIME_WAIT
TCP      192.168.0.112:55261  hem09s03-in-f14:https  TIME_WAIT
TCP      192.168.0.112:55262  hem09s03-in-f4:https   TIME_WAIT
TCP      192.168.0.112:55263  hem09s03-in-f14:https  TIME_WAIT
TCP      192.168.0.112:55264  hem09s03-in-f4:https   TIME_WAIT
TCP      192.168.0.112:55265  hem09s03-in-f14:https  TIME_WAIT
TCP      192.168.0.112:55266  hem09s03-in-f4:https   TIME_WAIT

C:\Users\eugen>
```

ОДИН HTTPCLIENT!?

```
public class RestClient
{
    private readonly HttpClient _httpClient = new();

    public async Task GetAsync()
    {
        await _httpClient.GetAsync("https://api.github.com");
    }
}
```



СХОДИМ В MSDN

HttpClient only resolves DNS entries when a connection is created.

DNS ЗАПИСИ

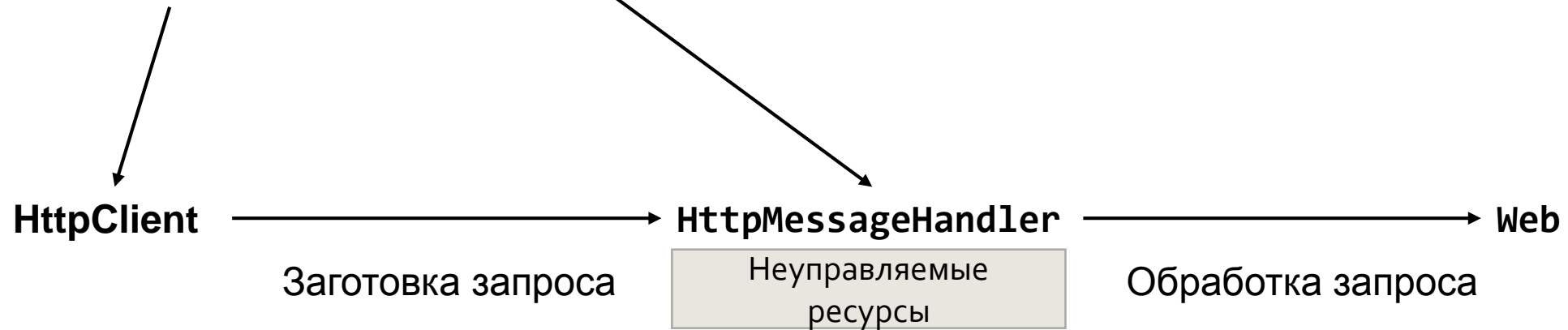
```
var handler = new SocketsHttpHandler
{
    PooledConnectionLifetime = TimeSpan.FromMinutes(15)
};

var sharedClient = new HttpClient(handler);
```

А КАК РАБОТАЕТ HTTPCLIENT?

```
public HttpClient() : this(new HttpClientHandler())
```

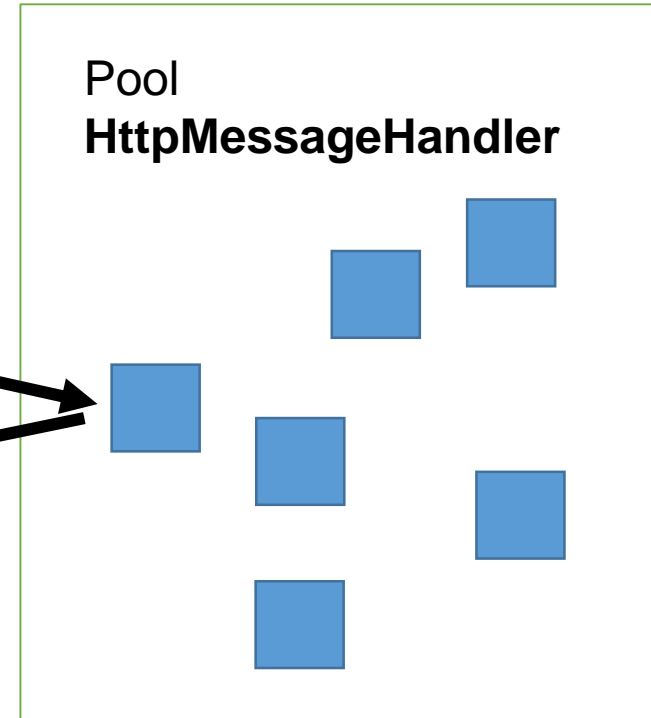
```
public HttpClient(HttpMessageHandler handler)
```



IHTTPCLIENTFACTORY

IHttpClientFactory

HttpClient



ИМЕНОВАННЫЕ КЛИЕНТЫ

```
services.AddHttpClient("github", c =>
{
    c.BaseAddress = new Uri("https://api.github.com");
    c.DefaultRequestHeaders.Add("User-Agent", "Sample");
});
```

```
var client = _IHttpClientFactory.CreateClient("github");
```

ТИПИЗИРОВАННЫЕ КЛИЕНТЫ

```
public class GitHubService
{
    public HttpClient Client { get; }

    public GitHubService(HttpClient client)
    {
        client.BaseAddress = new Uri("https://api.github.com");
        client.DefaultRequestHeaders.Add("User-Agent", "HttpClientFactory-Sample");

        Client = client;
    }
}

services.AddHttpClient<GitHubService>();
```

A 3A4EM DISPOSE?

```
public partial class HttpClient : HttpResponseMessageInvoker
```

```
public class HttpResponseMessageInvoker : IDisposable
```

A KAK РАБОТАЕТ DISPOSE?

```
public HttpClient(HttpMessageHandler handler, bool disposeHandler)
```

```
public class DefaultHttpClientFactory : IHttpClientFactory
{
    public HttpClient CreateClient(string name)
    {
        var httpMessageHandler = CreateOrGetHttpMessageHandler(name);
        return new HttpClient(httpMessageHandler, disposeHandler: false);
    }
}
```

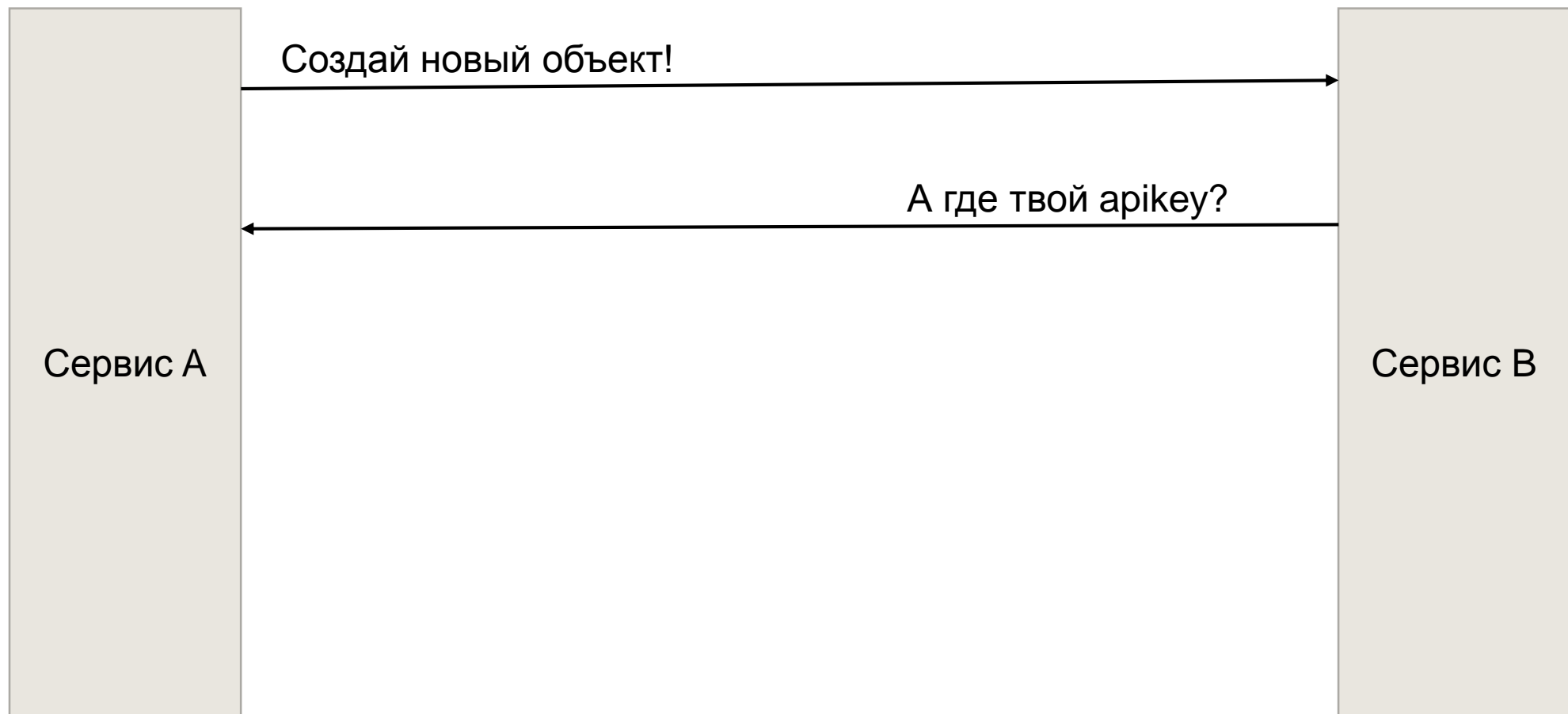
A KAK РАБОТАЕТ DISPOSE?

```
public HttpClient(HttpMessageHandler handler, bool disposeHandler)
```

```
public class DefaultHttpClientFactory : IHttpClientFactory
{
    public HttpClient CreateClient(string name)
    {
        var httpMessageHandler = CreateOrGetHttpMessageHandler(name);

        return new HttpClient(httpMessageHandler, disposeHandler: false);
    }
}
```

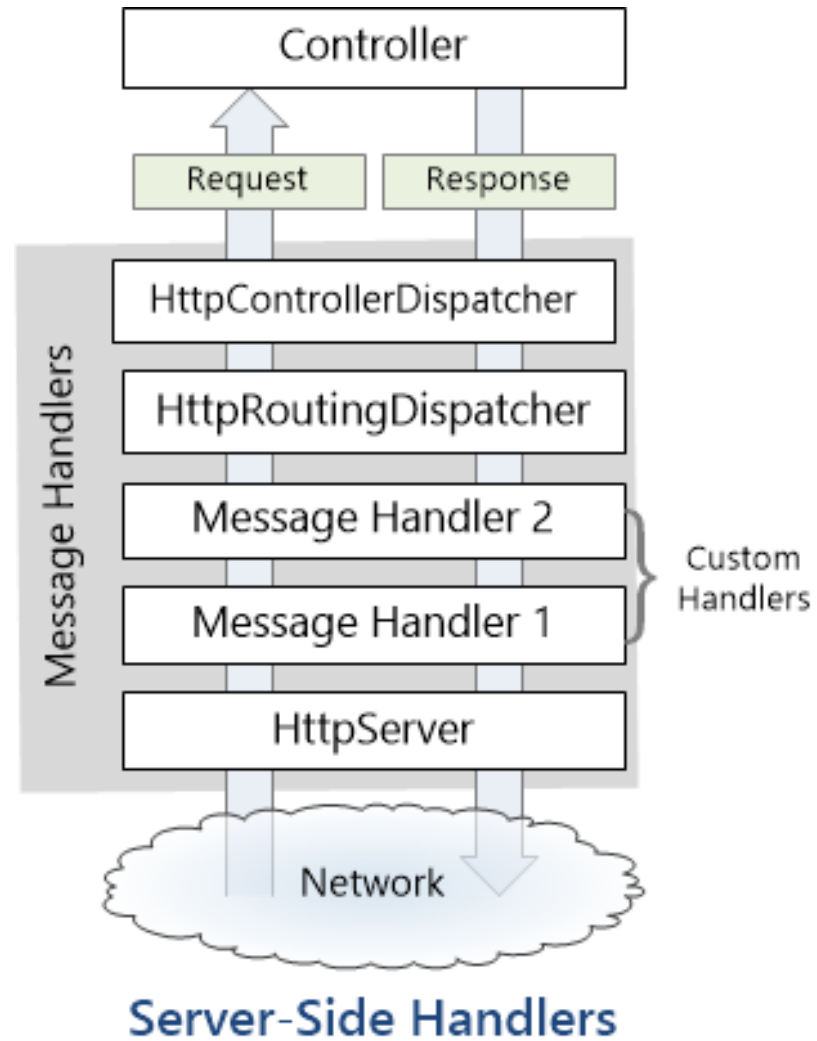
МИКРОСЕРВИСНОЕ ВЗАИМОДЕЙСТВИЕ



А КАК ПЕРЕДАТЬ API-KEY?

```
services.AddHttpClient<IBeaversServer_HttpClient,  
    BeaversServer_HttpClient>(x => x.DefaultRequestHeaders.Add("X-API-KEY", "my-api-key"));
```


HTTP MESSAGE HANDLERS



HTTP MESSAGE HANDLERS

```
public class ApiKeyHandler : DelegatingHandler
{
    private const string Header = "X-API-KEY";

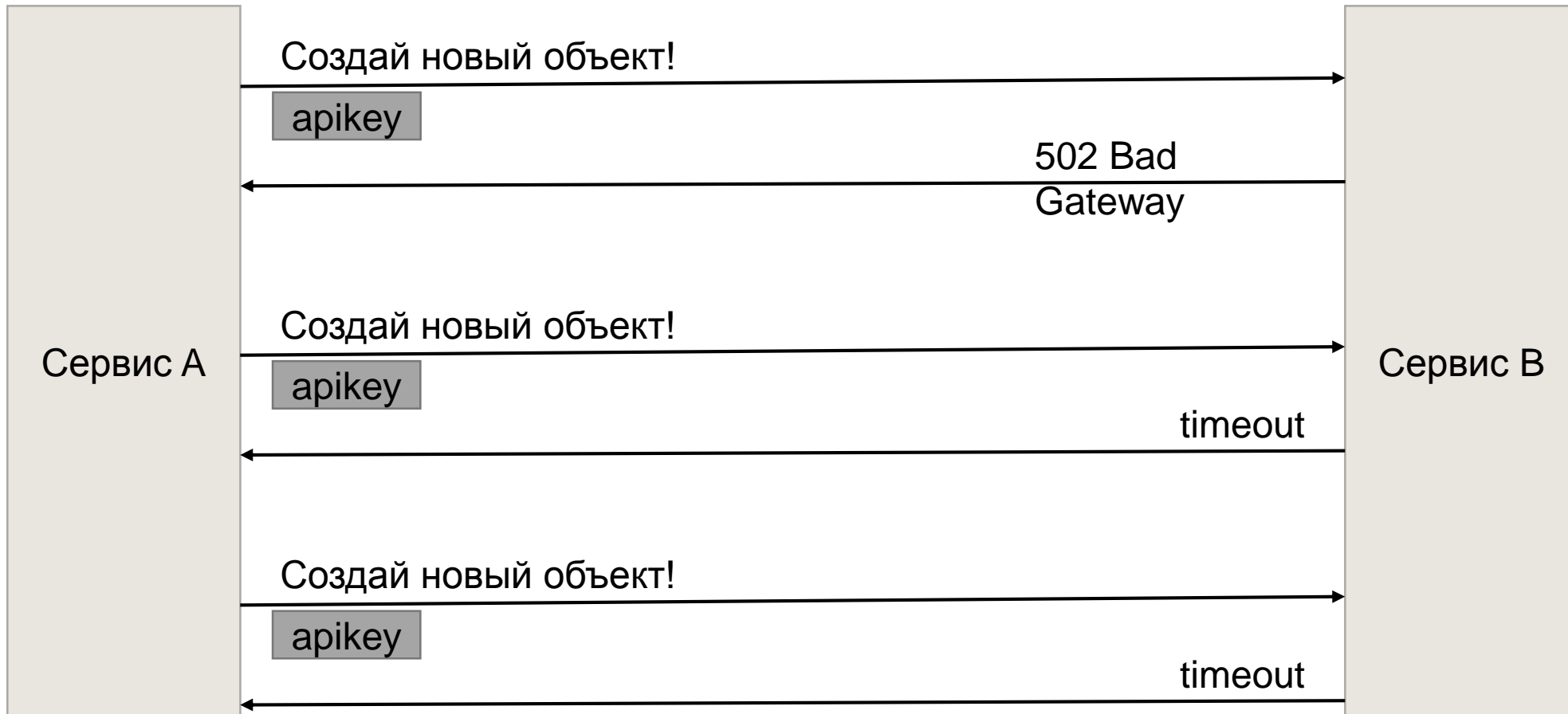
    protected override async Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
        CancellationToken cancellationToken)
    {
        if (!request.Headers.Contains(Header))
        {
            request.Headers.Add(Header, "my-api-key");
        }

        return await base.SendAsync(request, cancellationToken);
    }
}
```

HTTP MESSAGE HANDLERS

```
services.AddHttpClient<IBeaversServer_HttpClient,  
    BeaversServer_HttpClient>()  
    .AddHttpMessageHandler<ApiKeyHandler>();
```

МИКРОСЕРВИСНОЕ ВЗАИМОДЕЙСТВИЕ





POLLY

Retry policy

Если что-то
пошло не так,
то давайте
попробуем
ещё раз?

Circuit Breaker

Если сервис
недоступен,
зачем его
добивать?

Timeout policy

Ждём ответ от
сервиса, а его
всё нет



POLLY

```
var httpPolicy = HttpPolicyExtensions
    .HandleTransientHttpError()
    .OrResult(
        msg => msg.StatusCode == System.Net.HttpStatusCode.NotFound)
    .WaitAndRetryAsync(6,
        retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt)));

services.AddHttpClient<IGitHubClient>()
    .AddPolicyHandler(httpPolicy);
```

Client application/code

IHttpClientFactory

Dependency Injection



Controller or client code

ClientService
(i.e. CatalogService)

HttpClient
(Configured)

Policies



Delegating
Handlers

Pool

HttpMessageHandler
instances



Http
endpoint



СЕРВЕРНАЯ ЧАСТЬ


```
[HttpGet]  
public List<Beaver> GetAll()  
{  
    return Beavers;  
}
```

```
[HttpPut("feed")]
public void Feed([FromQuery] int id)
{
    var beaver = Beavers.SingleOrDefault(x => x.Id == id);

    if (beaver is not null)
    {
        beaver.Eat++;
    }
}
```

```
[HttpPost]
public int Create([FromBody] CreateBeaver beaver)
{
    var newBeaver = new Beaver()
    {
        Id = counter++,
        Name = beaver.Name,
        Eat = beaver.Eat
    };

    Beavers.Add(newBeaver);

    return newBeaver.Id;
}
```

```
[HttpDelete("{id:int}")]
public void Delete([FromRoute] int id)
{
    var beaver = Beavers.SingleOrDefault(x => x.Id == id);

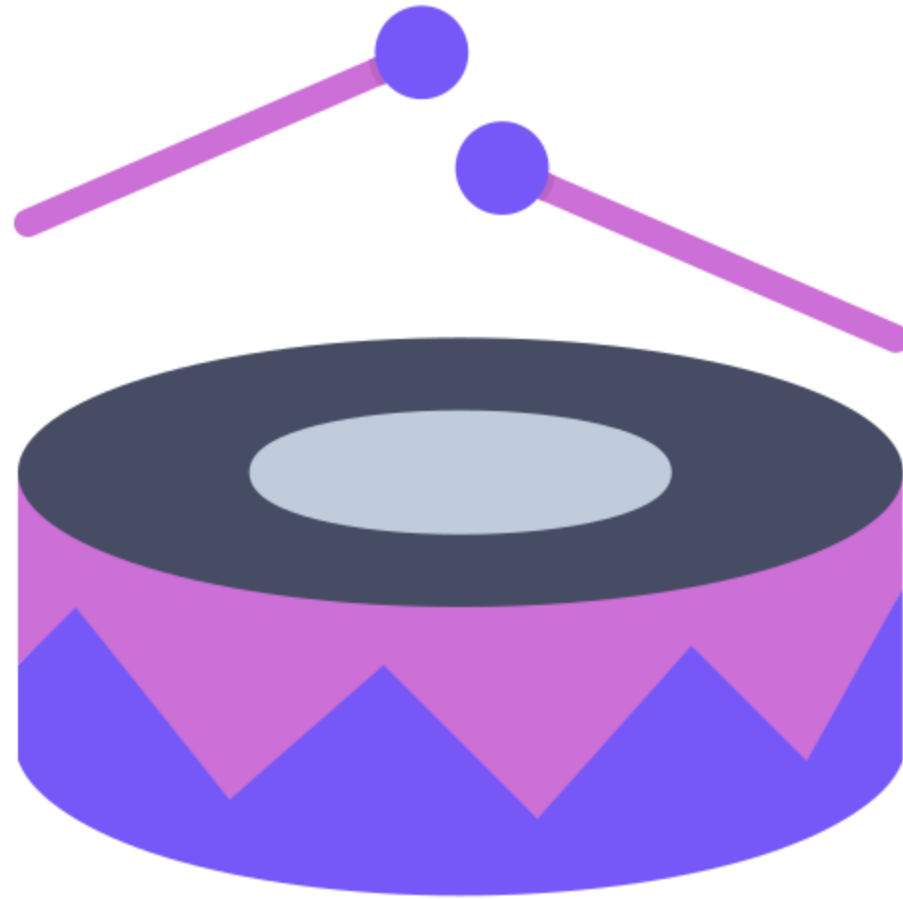
    if (beaver is not null)
    {
        Beavers.Remove(beaver);
    }
}
```



Напишем клиентскую часть



Начнём с самого популярного REST
API Client!





Чистый HttpClient!


```
public async Task<ICollection<Beaver>> GetAll()
{
    var beavers = await HttpClient
        .GetFromJsonAsync<ICollection<Beaver>>("beavers");

    return beavers;
}
```

```
public async Task<int> Create(CreateBeaver body)
{
    var response = await HttpClient
        .PostAsJsonAsync("beavers", body);

    var id = await response.Content
        .ReadFromJsonAsync<int>();

    return id;
}
```

```
public async Task Feed(int id)
{
    using var content = new FormUrlEncodedContent(
        new Dictionary<string, string>
        {
            { "id", id.ToString() },
        });

    await HttpClient.PutAsync("beavers/feed", content);
}
```

```
public async Task Delete(int id)
{
    await HttpClient.DeleteAsync($"beavers/{id}");
}
```

ОБРАБОТКА ОШИБОК

```
try
{
    HttpResponseMessage response = await client
        .GetAsync("http://ya.ru");
    // ...
}
catch (HttpRequestException e)
{
    // Handle exception.
}
```

HTTPCLIENT

Плюсы	Минусы
Дружит со всеми расширениями	
Поддерживается Microsoft	
Быстрый	
Не требует внешних зависимостей	

HTTPCLIENT

Плюсы	Минусы
Дружит со всеми расширениями	Нет возможности кастомизировать обработку ошибок
Поддерживается Microsoft	Многословен
Быстрый	Легко пройтись по граблям
Не требует внешних зависимостей	

API CLIENT LIBS

NSwag



Refit



Flurl



RestSharp



Kiota



RestEase





NSWAG

NSWAGStudio: The Swagger API toolchain for .NET

File

Documents

About

Swagger.nswag X

Input: Web API Assembly

JSON Schema | .NET Assembly

Swagger Specification | Web API Assembly

.NET Assembly

C:\Data\Projects\NSWag\src\NSWag.Demo.Web\bin\NSWag.Demo.Web.dll ... Load

Web API Controller

NSWag.Demo.Web.Controllers.PersonsController

☒ Use single controller

NSWag.Demo.Web.Controllers.PersonsController

Default Enum Handling

Choose 'Integer' if you use the default Json.NET serializer or 'String' if a global StringEnumConverter is registered.

Integer

☐ Flatten the inheritance hierarchy instead of using allOf to describe inheritance

Default URL Template

api/{controller}/{action}/{id}

Output

Swagger Specification | TypeScript Client | CSharp Client

Settings

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Net;
using System.Net.Http;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;

// Generated using the NSWag toolchain v1.17.5836.1609 (http://...)

namespace
{
    public partial class Client
    {
        public Client() : this("") { }

        public Client(string baseUrl)
        {
            BaseUrl = baseUrl;
        }

        partial void PrepareRequest(HttpClient request);

        partial void ProcessResponse(HttpClient request, HttpRe:

        public string BaseUrl { get; set; }

        /// <exception cref="SwaggerException">A server side er
        public async Task<string> XyzAsync(string data )
```

Generate Outputs



NSWAG

CSharp Client Settings

Namespace

Additional Namespace Usages (comma separated)

- ☐ Generate contracts output
- ☐ Generate record types
- ☒ Generate exception classes (when disabled, exception classes must be imported)

Exception class name (may contain the '{controller}' placeholder)

Client

- ☒ Generate Client Classes
- ☐ Suppress output of generated Client Classes

Operation Generation Mode

The {controller} placeholder of the Class Name is replaced by generated client name (depends on the OperationGenerationMode strategy).

Class Name

Client class access modifier

Methods with a protected access modifier to use in partial methods ('classname.methodname', comma separated)

- ☒ Use the base URL for the request



NSWAG

```
public partial interface IBeaversServer_NSswag
{
    /// <param name="cancellationToken">A cancellation token that can be used by
    other objects or threads to receive notice of cancellation.</param>
    /// <returns>Success</returns>
    /// <exception cref="ApiException">A server side error occurred.</exception>
    Task<ICollection<Beaver>> BeaversAllAsync(CancellationToken cancellationToken);
    ...
}
```



NSWAG

```
public partial class BeaversServer_NSswag : IBeaversServer_NSswag
{
    private string _baseUrl = "https://localhost:44376/";
    private System.Net.Http.HttpClient _httpClient;
    private System.Lazy<Newtonsoft.Json.JsonSerializerSettings> _settings;

    public BeaversServer_NSswag(System.Net.Http.HttpClient httpClient)
    {
        _httpClient = httpClient;
        _settings = new System.Lazy<JsonSerializerSettings>(CreateSerializerSettings);
    }
}
```



NSWAG

```
using (var request_ = new System.Net.Http.HttpRequestMessage())
{
    var json_ = Newtonsoft.Json.JsonConvert.SerializeObject(body, _settings.Value);
    var content_ = new System.Net.Http.StringContent(json_);
    content_.Headers.ContentType = MediaTypeHeaderValue.Parse("application/json");
    request_.Content = content_;
    request_.Method = new System.Net.Http.HttpMethod("POST");
    request_.Headers.Accept.Add(MediaTypeWithQualityHeaderValue.Parse("text/plain"));

    PrepareRequest(client_, request_, urlBuilder_);
    ...
}
```



NSWAG

```
var status_ = (int)response_.StatusCode;
if (status_ == 200)
{
    var objectResponse_ = await ReadObjectResponseAsync<ICollection<Beaver>>(response_, headers_,
cancellation_token).ConfigureAwait(false);
    if (objectResponse_.Object == null)
    {
        throw new ApiException("Response was null which was not expected.", status_,
objectResponse_.Text, headers_, null);
    }
    return objectResponse_.Object;
}
else
{
    var responseData_ = response_.Content == null ? null : await
response_.Content.ReadAsStringAsync().ConfigureAwait(false);
    throw new ApiException("The HTTP status code of the response was not expected (" + status_ + ").",
status_, responseData_, headers_, null);
}
```



NSWAG

Плюсы	Минусы
Генерация C#/TS клиента Все плюсы работа с HttpClient (Polly, middleware...) Настройка Json сериализатора	
Широкая кастомизация	
Нет внешних зависимостей	
Можно реализовать генерацию клиента при сборке проекте	



NSWAG

Плюсы	Минусы
Генерация C#/TS клиента Все плюсы работа с HttpClient (Polly, middleware...) Настройка Json сериализатора	Нет возможности кастомизировать обработку ошибок
Широкая кастомизация	Генерирует только то, что умеет (нет возможности вставить кастомные обработчики).
Нет внешних зависимостей	Тяжело читается.
Можно реализовать генерацию клиента при сборке проекте	

F FLURL

```
var person = await "https://api.com"
    .AppendPathSegment("person")
    .SetQueryParams(new { a = 1, b = 2 })
    .WithOAuthBearerToken("my_oauth_token")
    .PostJsonAsync(new
    {
        first_name = "Claire",
        last_name = "Underwood"
    })
    .ReceiveJson<Person>()
```

F FLURL

```
using (var httpTest = new HttpTest())
{
    // arrange
    httpTest.RespondWith("OK", 200);
    // act
    await client.CreatePersonAsync();
    // assert
    httpTest.ShouldHaveCalled("https://api.com/*")
        .WithVerb(HttpMethod.Post)
        .WithContentType("application/json");
}
```

F FLURL

httpTest

```
.ForCallsTo("*.api.com*", "*.test-api.com*")
.WithVerb("put", "PATCH") // or HttpMethod.Put, HttpMethod.Patch
.WithQueryParam("x", "a*") // value optional, wildcard supported
.WithQueryParams(new { y = 2, z = 3 })
.WithAnyQueryParam("a", "b", "c")
.WithoutQueryParam("d")
.WithHeader("h1", "f*o") // value optional, wildcard supported
.WithoutHeader("h2")
.WithRequestBody("*.something*") // wildcard supported
.WithRequestJson(new { a = "*", b = "hi" })
.With(call => true) // check anything on the FlurlCall
.Without(call => false) // check anything on the FlurlCall
.RespondWith("all conditions met!", 200);
```



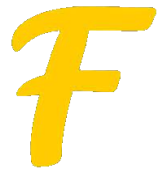
FLURL

```
public async Task<ICollection<Beaver>> GetAll()  
{  
    return await baseUrl  
        .GetJsonAsync<ICollection<Beaver>>();  
}
```

GET https://localhost:44376/beavers

```
public async Task<int> Create(CreateBeaver body)  
{  
    return await baseUrl  
        .PostJsonAsync(body)  
        .ReceiveJson<int>();  
}
```

POST https://localhost:44376/beavers



FLURL

```
public async Task Feed(int id)
{
    await baseUrl
        .AppendPathSegment("feed")
        .SetQueryParam("id", id)
        .PutAsync();
}
```

PUT <https://localhost:44376/beavers/feed?id=123>

```
public async Task Delete(int id)
{
    await baseUrl
        .AppendPathSegment(id)
        .DeleteAsync();
}
```

DELETE <https://localhost:44376/beavers/123>



FLURL

Плюсы	Минусы
Обработка ошибок	
Аутентификация	
Настройка Json сериализатора	
Поддержка Middleware	
Возможность тестирования API клиента	



FLURL

Плюсы	Минусы
Обработка ошибок	Нет генератора
Аутентификация	Нет возможности кастомизировать обработку ошибок
Настройка Json сериализатора	Polly
Поддержка Middleware	
Возможность тестирования API клиента	



RESTSHARP

```
var options = new RestClientOptions("https://api.twitter.com/1.1") {  
    Authenticator = new HttpBasicAuthenticator("username", "password")  
};  
var client = new RestClient(options);  
  
var request = new RestRequest("statuses/home_timeline.json");  
  
var timeline = await client.GetAsync<HomeTimeline>(request);
```



RESTSHARP

```
var params = new {  
    status = 1,  
    priority = "high",  
    ids = new [] { "123", "456" }  
};  
request.AddObject(params);
```

```
request.AddParameter("status", 1);  
request.AddParameter("priority", "high");  
request.AddParameter("ids", "123,456");
```

```
request.AddCookie("foo", "bar");
```



RESTSHARP

```
var client = new RestClient(  
    options,  
    configureSerialization: s => s.UseSerializer(() => new CustomSerializer());  
);
```

```
var client = new RestClient(  
    options,  
    configureSerialization: s => s.UseSystemTextJson(new JsonSerializerOptions {...})  
);
```

```
client.UseXmlSerializer();
```

```
var client = new RestClient(  
    options,  
    configureSerialization: s => s.UseNewtonsoftJson()  
);
```



RESTSHARP

```
var options = new RestClientOptions("http://example.com") {  
    Authenticator = new HttpBasicAuthenticator("username", "password")  
};
```

```
var options = new RestClientOptions("http://example.com") {  
    Authenticator = OAuth1Authenticator.ForRequestToken(consumerKey, consumerSecret)  
};
```

```
var options = new RestClientOptions("http://example.com") {  
    Authenticator = new JwtAuthenticator(myToken)  
};
```

```
var options = new RestClientOptions("http://example.com") {  
    Authenticator = new SuperAuthenticator()  
};
```






RESTSHARP

```
var response = await client.ExecutePostAsync<int>(request);
```

response.**Erro**

return

 ErrorException	Exception?
 ErrorMessage	string?
 ThrowIfError	RestResponse<...>

public Exception? **ErrorException** { get; set; }
Property in class **RestSharp.RestResponseBase**
The exception thrown during the request, if any



RESTSHARP

```
private readonly RestClient client = new("https://localhost:44376/");

public async Task<ICollection<Beaver>> GetAll()
{
    var request = new RestRequest("beavers");

    var response = await client.ExecuteGetAsync<ICollection<Beaver>>(request);

    return response.Data;
}

public async Task<int> Create(CreateBeaver body)
{
    var request = new RestRequest("beavers")
        .AddJsonBody(body);
    var response = await client.ExecutePostAsync<int>(request);

    return response.Data;
}
```



RESTSHARP

```
public async Task Feed(int id)
{
    var request = new RestRequest($"beavers/feed")
        .AddQueryParameter("id", id);

    await client.ExecutePutAsync(request);
}
```

```
public async Task Delete(int id)
{
    var request = new RestRequest($"beavers/{id}", Method.Delete);
    await client.ExecuteAsync(request);
}
```



OPENAPI GENERATOR

ActionScript, Ada, Apex, Bash, C, C# (.net 2.0, 3.5 or later, .NET Standard 1.3 - 2.1, .NET Core 3.1, .NET 5.0. Libraries: RestSharp, GenericHost, HttpClient), **C++** (Arduino, cpp-restsdk, Qt5, Tizen, Unreal Engine

4), **Clojure, Crystal, Dart, Elixir, Elm, Eiffel, Erlang, Go, Groovy, Haskell** (http-client, Servant), **Java** (Apache HttpClient 4.x, Apache HttpClient 5.x, Jersey2.x, OkHttp, Retrofit1.x, Retrofit2.x, Feign, RestTemplate, RESTEasy, Vertx, Google API Client Library for Java, Rest-assured, Spring 5 Web Client, MicroProfile Rest Client, Helidon), **Jetbrains HTTP**

Client, Julia, k6, Kotlin, Lua, N4JS, Nim, Node.js/JavaScript (ES5, ES6, AngularJS with Google Closure Compiler annotations, Flow types, Apollo GraphQL DataStore), **Objective-**

C, OCaml, Perl, PHP, PowerShell, Python, R, Ruby, Rust (hyper, reqwest, rust-server), **Scala** (akka, http4s, scalaz, sttp, swagger-async-httpclient, pekko), **Swift** (2.x, 3.x, 4.x, 5.x), **Typescript** (AngularJS, Angular (9.x - 17.x), Aurelia, Axios, Fetch, Inversify, iQuery, Nestjs, Node, redux-query, Rxjs), **Xojo, Zanier**

- Org.OpenAPITools
 - Dependencies
 - Api
 - C# BeaverApi.cs
 - Client
 - C# ApiClient.cs
 - C# ApiException.cs
 - C# ApiResponse.cs
 - C# ClientUtils.cs
 - C# Configuration.cs
 - C# ExceptionFactory.cs
 - C# FileParameter.cs
 - C# GlobalConfiguration.cs
 - C# IApiAccessor.cs
 - C# IAsynchronousClient.cs
 - C# IReadableConfiguration.cs
 - C# ISynchronousClient.cs
 - C# Multimap.cs
 - C# OpenAPIDateConverter.cs
 - C# RequestOptions.cs
 - C# RetryConfiguration.cs
 - C# WebRequestPathBuilder.cs
 - Model
 - C# AbstractOpenAPISchema.cs
 - C# Beaver.cs
 - C# CreateBeaver.cs
- Org.OpenAPITools.Test
 - Dependencies
 - Api
 - C# BeaverApiTests.cs
 - Model
 - C# BeaverTests.cs
 - C# CreateBeaverTests.cs

library=restsharp

Installed Packages in Org.OpenAPITools: 4

- Polly • 8.1.0 • [nuget.org](#)
- RestSharp • 110.2.0 • [nuget.org](#)
- JsonSubTypes • 2.0.1 • [nuget.org](#)
- Newtonsoft.Json • 13.0.3 • [VS Offline nuget.org](#)

library=httpclient

Installed Packages in Org.OpenAPITools: 3

- Polly • 8.1.0 • [nuget.org](#)
- JsonSubTypes • 2.0.1 • [nuget.org](#)
- Newtonsoft.Json • 13.0.3 • [VS Offline nuget.org](#)



RESTSHARP

Плюсы	Минусы
Генерация из Open Api спецификации	
Настройка Json сериализатора	
Аутентфикация из коробки	
Есть обработка ошибок	
Поддерживается AWS	



RESTSHARP

Плюсы	Минусы
Генерация из Open Api спецификации	Нет возможности кастомизировать обработку ошибок.
Настройка Json сериализатора	Тяжело настроить Polly
Аутентфикация из коробки	Нет middleware
Есть обработка ошибок	
Поддерживается AWS	



REFIT

```
public interface IGithubApi
{
    [Get("/users/{user}")]
    Task<User> GetUser(string user);
}
```

```
services
    .AddRefitClient<IGithubApi>()
    .ConfigureHttpClient(
        c => c.BaseAddress = new Uri("https://api.github.com"));
```



REFIT

```
[Get("/group/{id}/users")]  
Task<List<User>> GroupList([AliasAs("id")] int groupId, [AliasAs("sort")] string sortOrder);
```

```
GroupList(4, "desc");
```

/group/4/users?sort=desc



REFIT

```
[Get("/search/{**page}")]  
Task<List<Page>> Search(string page);
```

```
Search("admin/products");
```

```
/search/admin/products
```



REFIT

```
var githubApi = RestService.For<IGitHubApi>("https://api.github.com",  
    new RefitSettings {  
        ContentSerializer = new NewtonsoftJsonContentSerializer(  
            new JsonSerializerSettings {  
                ContractResolver = new SnakeCasePropertyNamesContractResolver()  
            }  
        ));
```



REFIT

```
[Headers("User-Agent: Awesome Octocat App")]  
public interface IGitHubApi  
{  
  
}
```

```
[Get("/users/{user}")]  
Task<User> GetUser(string user, [Header("Authorization")] string auth);
```




REFIT

```
try
{
    var result = await awesomeApi.GetFooAsync("bar");
}
catch (ValidationApiException ve)
{
    //exception handling
}
catch (ApiException e)
{
    //exception handling
}
```



REFIT

```
public interface IBeaversServer_Refit
{
    [Get("/beavers")]
    Task<ICollection<Beaver>> GetAll();

    [Post("/beavers")]
    Task<int> Create([Body] CreateBeaver body);

    [Put("/beavers/feed")]
    Task Feed([Query] int id);

    [Delete("/beavers/{id}")]
    Task Delete(int id);
}
```



REFIT

```
services.AddRefitClient<IBeaversServer_Refit>()  
    .ConfigureHttpClient(  
        c => c.BaseAddress = new Uri(beaversServerUrl))  
    .AddHttpMessageHandler<ApiKeyHandler>()  
    .AddPolicyHandler(httpPolicy);
```



REFITTER

```
refitter [path to OpenAPI spec file] --namespace "[Your.Namespace.Of.Choice.GeneratedCode]"
```



REFITTER

```
dotnet tool install --global Refitter
```

```
refitter [path] --namespace "[Your.Namespace.Of.Choice.GeneratedCode]"
```



REFITTER

```
dotnet add package  
Refitter.SourceGenerator
```

```
myclient.refitter
```

```
{  
  "openApiPath": "./swagger.json",  
  "namespace": "Your.Namespace.Of.Choice.GeneratedCode"  
}
```

```
<ItemGroup>  
  <AdditionalFiles Include="..\*.refitter" />  
  <Compile Include="..\Generated/*.cs" />  
</ItemGroup>
```



REFITTER

```
public partial interface IBeaversServer
{
    /// <returns>Success</returns>
    /// <throws cref="ApiException">Thrown when the request returns a non-success status code.</throws>
    [Headers("Accept: text/plain, application/json, text/json")]
    [Get("/beavers")]
    Task<ICollection<Beaver>> BeaversAll();

    /// <returns>Success</returns>
    /// <throws cref="ApiException">Thrown when the request returns a non-success status code.</throws>
    [Headers("Accept: text/plain, application/json, text/json")]
    [Post("/beavers")]
    Task<int?> BeaversPOST([Body] CreateBeaver body);

    /// <returns>A <see cref="Task"/> that completes when the request is finished.</returns>
    /// <throws cref="ApiException">Thrown when the request returns a non-success status code.</throws>
    [Put("/beavers/feed")]
    Task Feed([Query] int? id);

    /// <returns>A <see cref="Task"/> that completes when the request is finished.</returns>
    /// <throws cref="ApiException">Thrown when the request returns a non-success status code.</throws>
    [Delete("/beavers/{id}")]
    Task BeaversDELETE(int id);
}
```



GENERATE ASPNETCORE CLIENT

Generate Refit client from project

USAGE:

```
dotnet-generate-client [path to api.csproj] [OPTIONS]
```

EXAMPLES:

```
dotnet-generate-client MyApiProjectPath -o OutPath -n My.Client.Namespace
```




GENERATE ASPNETCORE CLIENT

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    [HttpGet]
    public async Task<IEnumerable<WeatherForecast>> Get()
    {
    }

    [HttpGet("{id}")]
    public async Task<WeatherForecast> Get(Guid id)
    {
        ...
    }
}
```



```
public interface IWeatherForecastApi
{
    [Get("/WeatherForecast")]
    Task<IEnumerable<WeatherForecast>> Get();

    [Get("/WeatherForecast/{id}")]
    Task<WeatherForecast> Get(Guid id);
}
```



GENERATE ASPNETCORE CLIENT

+Генерация клиента из проекта.

-Нужно запустить проект чтобы собрать все endpoint'ы.

-Только Asp.Net Core 6/7/8.

-Редкие обновления.



REFIT

Плюсы	Минусы
Генерация из Open Api спецификации	
Настройка Json сериализатора	
Есть обработка ошибок с возможностью настройки	
Аутентификация из коробки с возможностью настройки	
Клиент в виде интерфейса	
Легко встраивается Polly и Middleware	



REFIT

Плюсы	Минусы
Генерация из Open Api спецификации	Внешние зависимости
Настройка Json сериализатора	
Есть обработка ошибок с возможностью настройки	
Аутентификация из коробки с возможностью настройки	
Клиент в виде интерфейса	
Легко встраивается Polly и Middleware	



RESTEASE

```
public interface IGithubApi
{
    [Get("user")]
    Task<User> FetchUserAsync(int userid);
}
```



RESTEASE

```
public interface ISomeApi
{
    [Get("search")]
    Task<SearchResult> SearchBlogPostsAsync([QueryMap] IDictionary<string, string[]> filters);
}

var api = RestClient.For<ISomeApi>("https://api.example.com");
var filters = new Dictionary<string, string[]>()
{
    { "title", new[] { "bobby" } },
    { "tag", new[] { "c#", "programming" } }
};

var searchResults = await api.SearchBlogPostsAsync(filters);
```

GET <https://api.example.com/search?title=bobby&tag=c%23&tag=programming>



RESTEASE

```
public interface ISomeApi
{
    [Path("accountId")]
    int AccountId { get; set; }

    [Get("{accountId}/profile")]
    Task<Profile> GetProfileAsync();
}
```

```
var api = RestClient.For<ISomeApi>("https://api.example.com/user");
api.AccountId = 3;
```

```
var profile = await api.GetProfileAsync();
```

```
GET https://api.example.com/user/3/profile
```



RESTEASE

```
public interface IBeaversServer_RestEase
{
    [Get("/beavers")]
    Task<ICollection<Beaver>> GetAll();

    [Post("/beavers")]
    Task<int> Create([Body] CreateBeaver body);

    [Put("/beavers/feed")]
    Task Feed([Query] int id);

    [Delete("/beavers/{id}")]
    Task Delete([Path] int id);
}
```




RESTEASE

```
services.AddRestEaseClient<IBeaversServer_RestEase>()  
    .ConfigureHttpClient(  
        c => c.BaseAddress = new Uri(beaversServerUrl))  
    .AddHttpMessageHandler<ApiKeyHandler>()  
    .AddPolicyHandler(httpPolicy);
```



RESTEASE

Плюсы	Минусы
Генерация из Open Api спецификации	
Настройка Json сериализатора	
Есть обработка ошибок с возможностью настройки	
Аутентификация из коробки с возможностью настройки	
Клиент в виде интерфейса	
Легко встраивается Polly и Middleware	
Очень много кастомизации	



RESTEASE

Плюсы	Минусы
Генерация из Open Api спецификации	Внешние зависимости
Настройка Json сериализатора	Проигрывает по размерам комьюнити Refit
Есть обработка ошибок с возможностью настройки	
Аутентификация из коробки с возможностью настройки	
Клиент в виде интерфейса	
Легко встраивается Polly и Middleware	
Очень много кастомизации	



KIOTA

```
dotnet tool install --global Microsoft.OpenApi.Kiota
```

```
dotnet add package Microsoft.Kiota.Abstractions  
dotnet add package Microsoft.Kiota.Http.HttpClientLibrary  
dotnet add package Microsoft.Kiota.Serialization.Form  
dotnet add package Microsoft.Kiota.Serialization.Json  
dotnet add package Microsoft.Kiota.Serialization.Text  
dotnet add package Microsoft.Kiota.Serialization.Multipart
```

```
kiota generate -l CSharp -c Client -n KiotaPosts.Client -d ./posts-api.yml -o ./Client
```



KIOTA

```

Kiota
├── Beavers
│   ├── Feed
│   │   └── FeedRequestBuilder.cs
│   └── Item
│       ├── BeaversItemRequestBuilder.cs
│       └── BeaversRequestBuilder.cs
├── Models
│   ├── Beaver.cs
│   └── CreateBeaver.cs
├── .kiota.log
├── BeaversServer_Kiota.cs
└── kiota-lock.json
```



KIOTA

```
public class BeaversServer_Kiota : BaseRequestBuilder {  
    public BeaversRequestBuilder Beavers { get =>  
        new BeaversRequestBuilder(PathParameters, RequestAdapter);  
    }  
  
    public BeaversServer_Kiota(IRequestAdapter requestAdapter)  
        : base(requestAdapter, "{+baseurl}", new Dictionary<string, object>()) {  
        ApiClientBuilder.RegisterDefaultSerializer<JsonSerializationWriterFactory>();  
        ApiClientBuilder.RegisterDefaultSerializer<TextSerializationWriterFactory>();  
        ApiClientBuilder.RegisterDefaultSerializer<FormSerializationWriterFactory>();  
        ApiClientBuilder.RegisterDefaultDeserializer<JsonParseNodeFactory>();  
        ApiClientBuilder.RegisterDefaultDeserializer<TextParseNodeFactory>();  
        ApiClientBuilder.RegisterDefaultDeserializer<FormParseNodeFactory>();  
    }  
}
```



KIOTA

```
    /// <summary>The eat property</summary>
    public int? Eat { get; set; }
    /// <summary>The id property</summary>
    public int? Id { get; set; }
    /// <summary>The name property</summary>
#if NETSTANDARD2_1_OR_GREATER || NETCOREAPP3_1_OR_GREATER
    #nullable enable
        public string? Name { get; set; }
    #nullable restore
#else
        public string Name { get; set; }
#endif
```



KIOTA

```
public IDictionary<string, Action<IParseNode>> GetFieldDeserializers() {  
    return new Dictionary<string, Action<IParseNode>> {  
        {"eat", n => { Eat = n.GetIntValue(); } },  
        {"id", n => { Id = n.GetIntValue(); } },  
        {"name", n => { Name = n.GetStringValue(); } },  
    };  
}
```

```
public void Serialize(ISerializationWriter writer) {  
    _ = writer ?? throw new ArgumentNullException(nameof(writer));  
    writer.WriteIntValue("eat", Eat);  
    writer.WriteIntValue("id", Id);  
    writer.WriteStringValue("name", Name);  
}
```




KIOTA

```
var authProvider = new AnonymousAuthenticationProvider();
var adapter = new HttpClientRequestAdapter(authProvider);
adapter.BaseUrl = "https://localhost:44376";

var client = new BeaversServer_Kiota(adapter);

var bernarId = await client.Beavers.PostAsync(
    new RefClient.ApiClients.Kiota.Models.CreateBeaver
    {
        Eat = 10,
        Name = "Bernar"
    });
```



KIOTA

```
await client.Beavers.Feed.PutAsync(x => x.QueryParameters.Id = bernarId);
```

```
var beavers = await client.Beavers.GetAsync();
```

```
await client.Beavers[bernarId.ToString()].DeleteAsync();
```



known limitations #1

Open

20 tasks

baywet opened this issue on Dec 28, 2022 · 0 comments



baywet commented on Dec 28, 2022 · edited

Member



Hi everyone and thanks for trying our preview of the Microsoft Graph Ruby SDK built on our [next generation generator kiota](#).

This issue is here to list out all the limitations we're currently aware of, we'll try to tackle them as soon as possible.

Please go upvote the linked issues if this is something that's limiting your usage of the SDK, so we know which feature to prioritize.

List of limitations:

- ☐ [Binary bodies support](#)
- ☐ [support for additional status codes](#)
- ☐ [support for text serialization \(raw count\)](#)
- ☐ [enhanced doc comments](#)
- ☐ [Missing middleware for transient error retrial](#)
- ☐ [Missing middleware for redirection](#)
- ☐ ☒ [http/2 validation #8](#)
- ☐ ☒ [client factory defaults #7](#)
- ☐ ☒ [document support for http proxies #6](#)
- ☐ [tracing support](#)
- ☐ ☒ [enable request body gzip compression #9](#)
- ☐ ☒ [add page iterator #10](#)
- ☐ [No snippets present in the Graph Explorer/public documentation](#)
- ☐ ☒ [add support for batch requests #5](#)
- ☐ ☒ [add support for large file upload #3](#)
- ☐ [Backing store support for dirty tracking of changed properties](#)
- ☐ [Missing middleware for chaos \(unit testing\)](#)
- ☐ ☒ [change notifications types are missing #4](#)
- ☐ [Multi-part content](#)
- ☐ [support for continuous access evaluation](#)

Assign

No one assigned

Labels

enhancement

Project

None

Milestones

No milestones

Development

No branches

1 participant





КИОТА

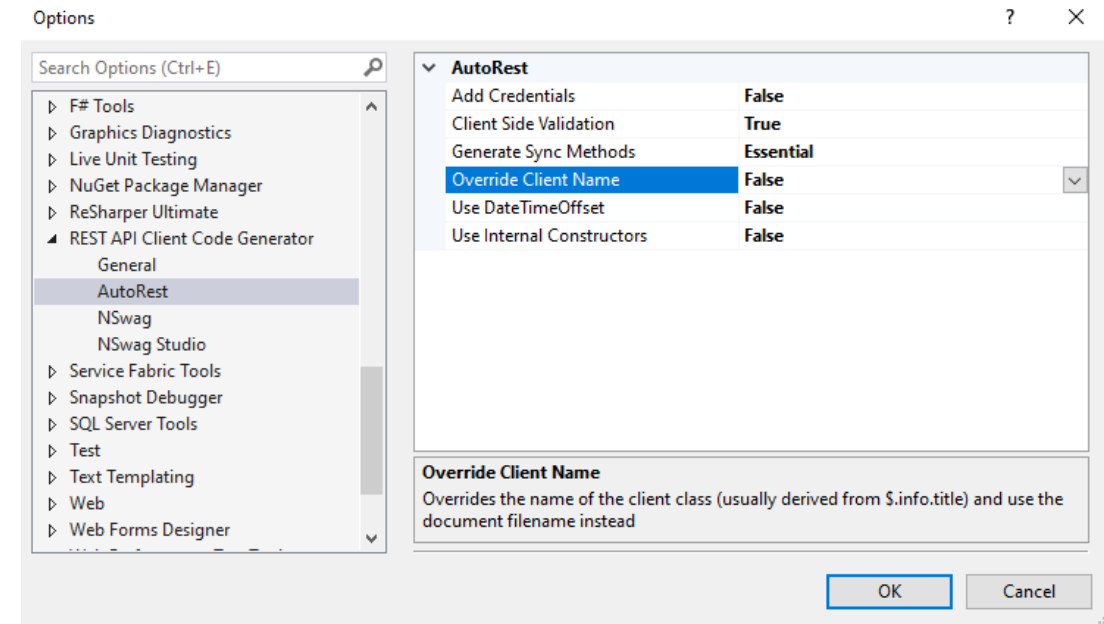
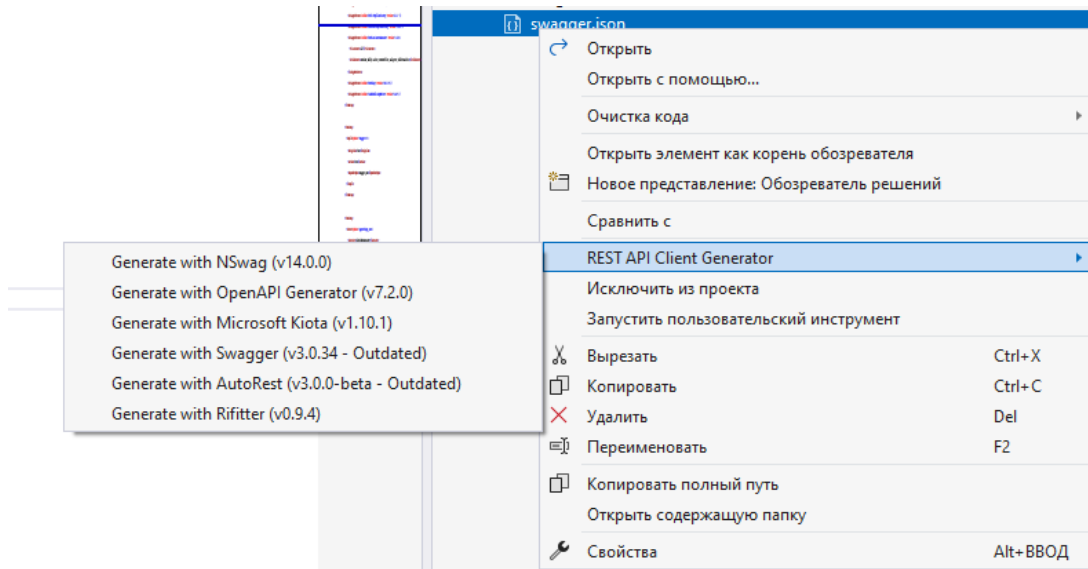
Плюсы	Минусы
Генерирует клиенты для C#, Go, Java, PHP, Python, Ruby, Shell, Swift, Typescript	
Поддерживается Microsoft	
Обработка ошибок	
Кастомизация Json сериализатора	



KIOTA

Плюсы	Минусы
Генерирует клиенты для C#, Go, Java, PHP, Python, Ruby, Shell, Swift, Typescript	Сложно встроить polly и middleware
Поддерживается Microsoft	Нет кастомной обработки ошибок
Обработка ошибок	Выглядит сырой
Кастомизация Json сериализатора	

REST API CLIENT CODE GENERATOR FOR VS 2022



	Flurl	NSwag	RestSharp	Refit	RestEase	Kiota
HttpClient	✓	✓	✓v107	✓	✓	✓
Генерация из Open API клиента	✗	✓	✓	✓	✓	✓
Polly	⚠	✓	⚠	✓	✓	⚠
Middleware	✓	✓	✗	✓	✓	⚠
Настраиваемый JSON сериализатор	✓	✓	✓	✓	✓	✓
Обработка ошибок	✓	✗	✓	✓	✓	✓
☆☆☆	4k	6.5k	9.4k	8k	1k	2.2k

[Kiota](#)



[RestSharp](#)



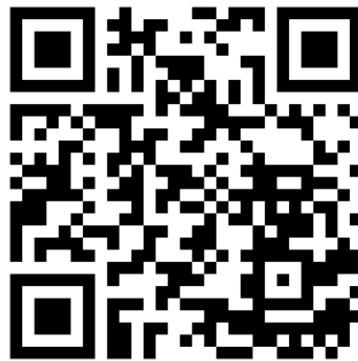
[RestEase](#)



[Flurl](#)



[Refit](#)



[NSwag](#)





Polly



Использование HttpClientFactory для реализации устойчивых HTTP-запросов



Выполнения HTTP-запросов с помощью HttpClientFactory в ASP.NET Core



Как использовать HttpClientFactory





СПАСИБО ЗА ВНИМАНИЕ!



Квашнин Артём

<https://t.me/the1kvin>