Asp.net core Web API микросервисы. Часть 1

Open API



# На этом уроке

На прошлом уроке мы рассмотрели как можно взаимодействовать сервисам между собой по протоколу Http. Мы написали специализированный клиент для сервиса агента сбора метрик и встроили его в наш сервис-менеджер сбора метрик. Каждый раз писать клиента к новому сервису затруднительно, к тому же, часто возникает потребность в поддержании клиентов для различных платформ. Для того, чтобы не приходилось это каждый раз делать заново, существует соглашение по API для сервисов под названием Open API. На этом уроке мы рассмотрим возможности, которые представляются при использовании Open API в построении API микросервисов.

# 

Оглавление

[На этом уроке](#_4wfp8dsrmxhk)

[Вступление](#_n93g6yblf23c)

[Open API](#_n9r8jonks8we)

[Спецификация Open API](#_x39p9fnx2rf9)

[Подключение Swagger](#_8ksdxaarmvim)

[Генерация клиента](#_wkuepipubce9)

[Практическое задание](#_tf1198bb3721)

[Список используемой литературы](#_54hssoocdbld)

[Дополнительные материалы](#_oreyj64p9zko)

# Вступление

На прошлом уроке мы написали клиента для сервиса агента сбора метрик. У нас была относительно простая задача, так как мы писали код для собственно-написанного сервиса и знали специфику его работы. Мы писали сервис и клиент на одном и том же языке, а также встраивали клиента в сервис, написанный на си шарпе. В современных микросервисных проектах не всегда представляется такая возможность и зачастую приходится реализовывать клиентов к сервисам сразу на нескольких языках. В нашем примере это может быть одновременно C# и JS. Чтобы легче реализовывать клиентов к сервисам программисты договорились между собой о правилах построения API сервисов. Считается хорошим тоном, если сервис поддерживает спецификацию Open API. На этом уроке мы познакомимся с этой спецификацией, подключим её к своим проектам и научимся в автоматическом режиме генерировать клиентов (да да, даже не писать самостоятельно) для своих сервисов.

# Open API

Swagger (OpenAPI) - это не зависящая от языка спецификация для описания REST API. Это позволяет как компьютерам, так и людям понимать возможности REST API без прямого доступа к исходному коду. Его основные цели:

* Свести к минимуму объем работы, необходимой для подключения несвязанных сервисов
* Сократить время, необходимое для написания пользовательской документации к сервису

Две основные реализации OpenAPI для .NET - это Swashbuckle и NSwag. На этом уроке мы рассмотрим Swashbuckle реализацию.

Есть некоторая путаница в терминологии. Прежде чем мы пойдем дальше, рассмотрим особенности именования технологий.

Проект Swagger был отдан в OpenAPI Initiative в 2015 году и с тех пор называется OpenAPI. Оба имени используются как синонимы. Однако “OpenAPI” относится к спецификации. Swagger относится к семейству коммерческих продуктов с открытым исходным кодом от компании SmartBear, которые работают со спецификацией OpenAPI. Последующие продукты с открытым исходным кодом, такие как OpenAPIGenerator, также относятся к семейству Swagger, несмотря на то, что SmartBear не выпускает их.

Резюмируя:

* OpenAPI - это спецификация.
* Swagger - это инструмент, использующий спецификацию OpenAPI. Например, OpenAPIGenerator и SwaggerUI.

## Спецификация Open API

Спецификация OpenAPI - это документ, описывающий возможности вашего API. Документ основан на XML и аннотациях атрибутов в контроллерах и моделях. Это основная часть OpenAPI и используется для управления такими инструментами, как SwaggerUI. По умолчанию он называется openapi.json. Данный файл генерируется инструментарием Swagger автоматически на основании атрибутов контроллеров, комментариев к ним, объектов запроса и ответа.

На основании спецификации Swagger умеет генерировать Web UI (сайт) для вашего сервиса. Пользовательский интерфейс Swagger предлагает веб-интерфейс, который предоставляет информацию сервисе, объектах запросов и ответов, возможность протестировать прямо из браузера запросы к сервису.. И Swashbuckle, и NSwag включают встроенную версию пользовательского интерфейса Swagger, поэтому его можно включить в наших приложениях с помощью вызова регистрации промежуточного программного обеспечения (middlware).

## Подключение Swagger

Подключим swagger к нашему проекту агенту сбора метрик. Для этого подключите через nuget пакет Swashbuckle.AspNetCore.

Теперь включим генерацию файла спецификации в методе ConfigureServices класса Startup

|  |
| --- |
| services.AddSwaggerGen()*;* |

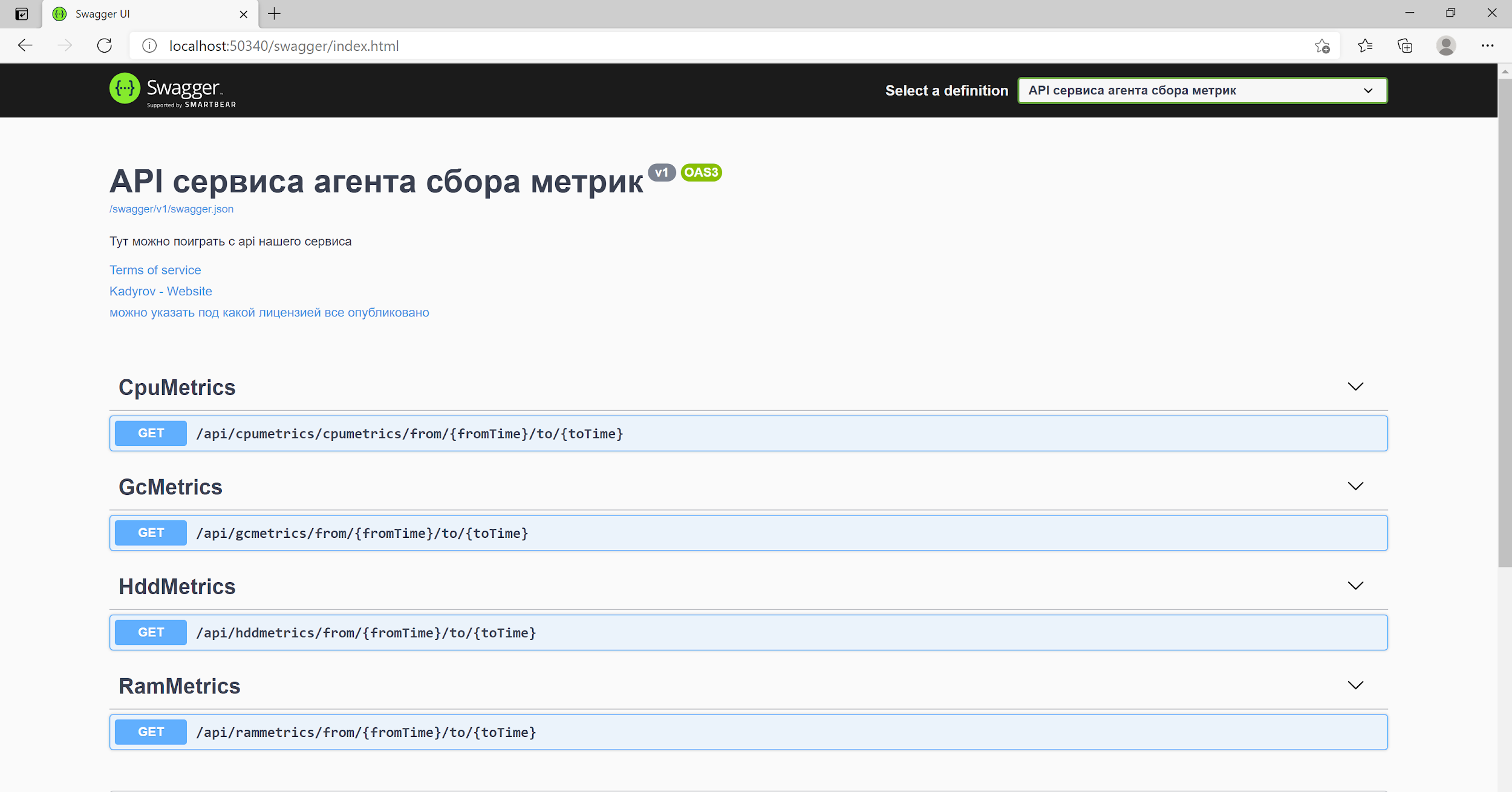
В методе Configure класса Startup следует внести следующие изменения

|  |
| --- |
| *// Включение middleware в пайплайн для обработки Swagger запросов.* app.UseSwagger(); *// включение middleware для генерации swagger-ui*  *// указываем Swagger JSON эндпоинт (куда обращаться за сгенерированной спецификацией* *// по которой будет построен UI).* app.UseSwaggerUI(c => {  c.SwaggerEndpoint("/swagger/v1/swagger.json", "API сервиса агента сбора метрик"); }); |

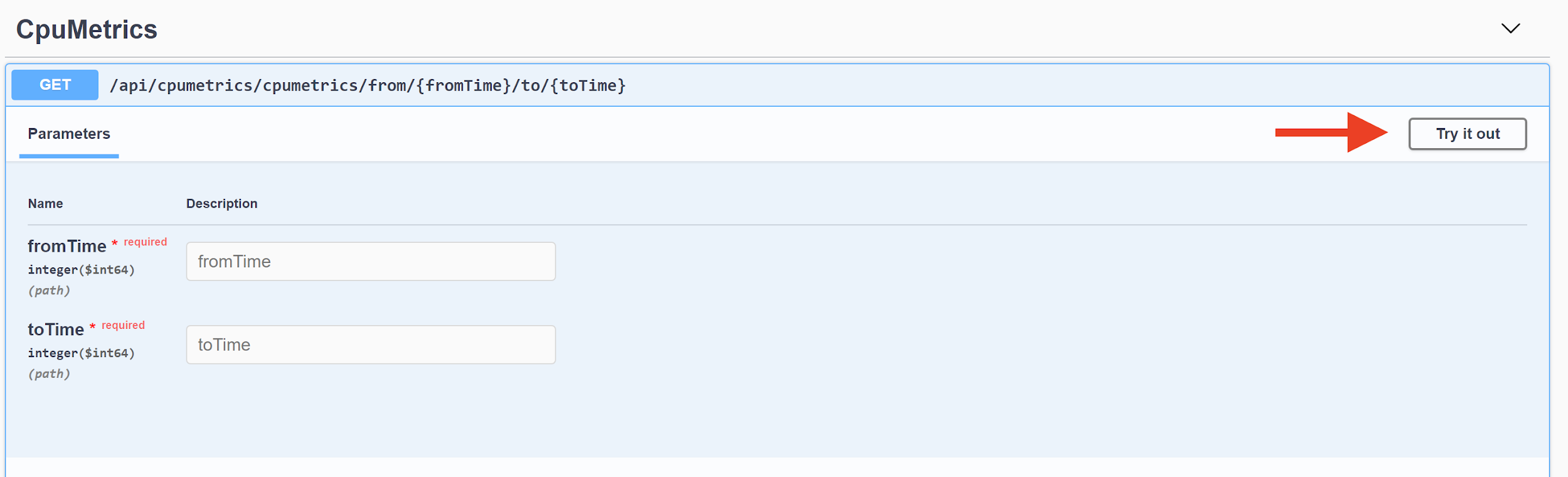
Добавим сразу описание на сгенерированную страницу. Для этого добавим дополнительной информации в регистрацию сервиса в методе ConfigureServices

|  |
| --- |
| services.AddSwaggerGen(c => {  c.SwaggerDoc("v1", new OpenApiInfo  {  Version = "v1",  Title = "API сервиса агента сбора метрик",  Description = "Тут можно поиграть с api нашего сервиса",  TermsOfService = new Uri("https://example.com/terms"),  Contact = new OpenApiContact  {  Name = "Kadyrov",  Email = string.Empty,  Url = new Uri("https://kremlin.ru"),  },  License = new OpenApiLicense  {  Name = "можно указать под какой лицензией все опубликовано",  Url = new Uri("https://example.com/license"),  }  }); }); |

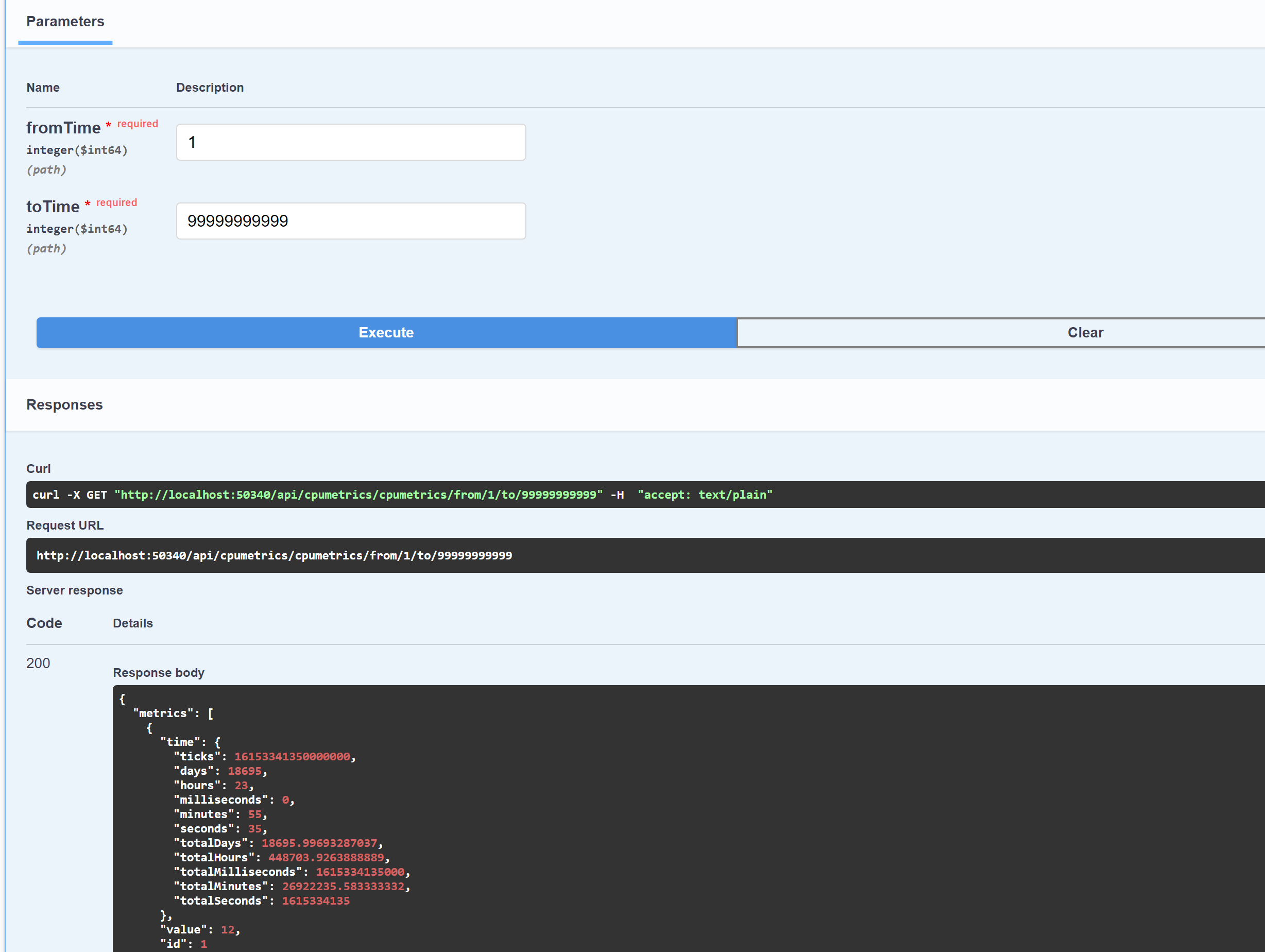
Попробуйте запустить наш сервис агент сбора метрик и перейти по адресу [http://localhost:порт/swagger](about:blank), где вместо порт укажите порт, на котором запустился ваш сервис.



У вас откроется страница с UI для API нашего сервиса. UI будет содержать все описания запросов, ответов, а также пути до методов контроллеров с указанием того, каким HTTP методом они могут быть вызваны. Напрямую из UI можно выполнить запрос, подставив необхоимые значения. Попробуйте выполнить запросы, учитывая, что выполняя предыдущие уроки, вы сделали автоматический сбор метрик. Для этого щелкните на шторку справа от названия метода, а затем нажмите на кнопку Try it out



Заполните данные для выполнения запроса и нажмите Execute Query. UI выполнит запрос с указанными вами параметрами и отправит их в наш сервис



В документацию можно добавить информацию прямо из xml комментариев к коду проекта. Посмотрим на краткое содержание контроллера сбора метрик и комментарии к его единственному методу

|  |
| --- |
| */// <summary>* */// Получает метрики CPU на заданном диапазоне времени* */// </summary>* */// <remarks>* */// Пример запроса:* *///* */// GET cpumetrics/from/1/to/9999999999* *///* */// </remarks>* */// <param name="fromTime">начальная метрка времени в секундах с 01.01.1970</param>* */// <param name="toTime">конечная метрка времени в секундах с 01.01.1970</param>* */// <returns>Список метрик, которые были сохранены в заданном диапазоне времени</returns>* */// <response code="201">Если все хорошо</response>* */// <response code="400">если передали не правильные параетры</response>*  [HttpGet("cpumetrics/from/{fromTime}/to/{toTime}")] public AllCpuMetricsResponse GetCpuMetrics([FromRoute] long fromTime, [FromRoute] long toTime) {  ... } |

Чтобы данные появились в автоматически сгенерированной документации и Swagger UI нужно включить генерацию xml документации в файле проекта сбора метрик. Необходимо добавить следующие строки в .csproj файл проекта агента сбора метрик

|  |
| --- |
| <PropertyGroup>  <GenerateDocumentationFile>true</GenerateDocumentationFile>  <NoWarn>$(NoWarn);1591</NoWarn> </PropertyGroup> |

В методе регистрации сервиса необходимо прописать путь до документации. Итого, регистрация сервиса, который будет генерировать файл спецификации Open API будет выглядеть следующим образом

|  |
| --- |
| services.AddSwaggerGen(c =>  {  c.SwaggerDoc("v1", new OpenApiInfo  {  Version = "v1",  Title = "API сервиса агента сбора метрик",  Description = "Тут можно поиграть с api нашего сервиса",  TermsOfService = new Uri("https://example.com/terms"),  Contact = new OpenApiContact  {  Name = "Kadyrov",  Email = string.Empty,  Url = new Uri("https://kremlin.ru"),  },  License = new OpenApiLicense  {  Name = "можно указать под какой лицензией все опубликовано",  Url = new Uri("https://example.com/license"),  }  });  *// Указываем файл из которого брать комментарии для Swagger UI*  var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";  var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);  c.IncludeXmlComments(xmlPath);  }); |

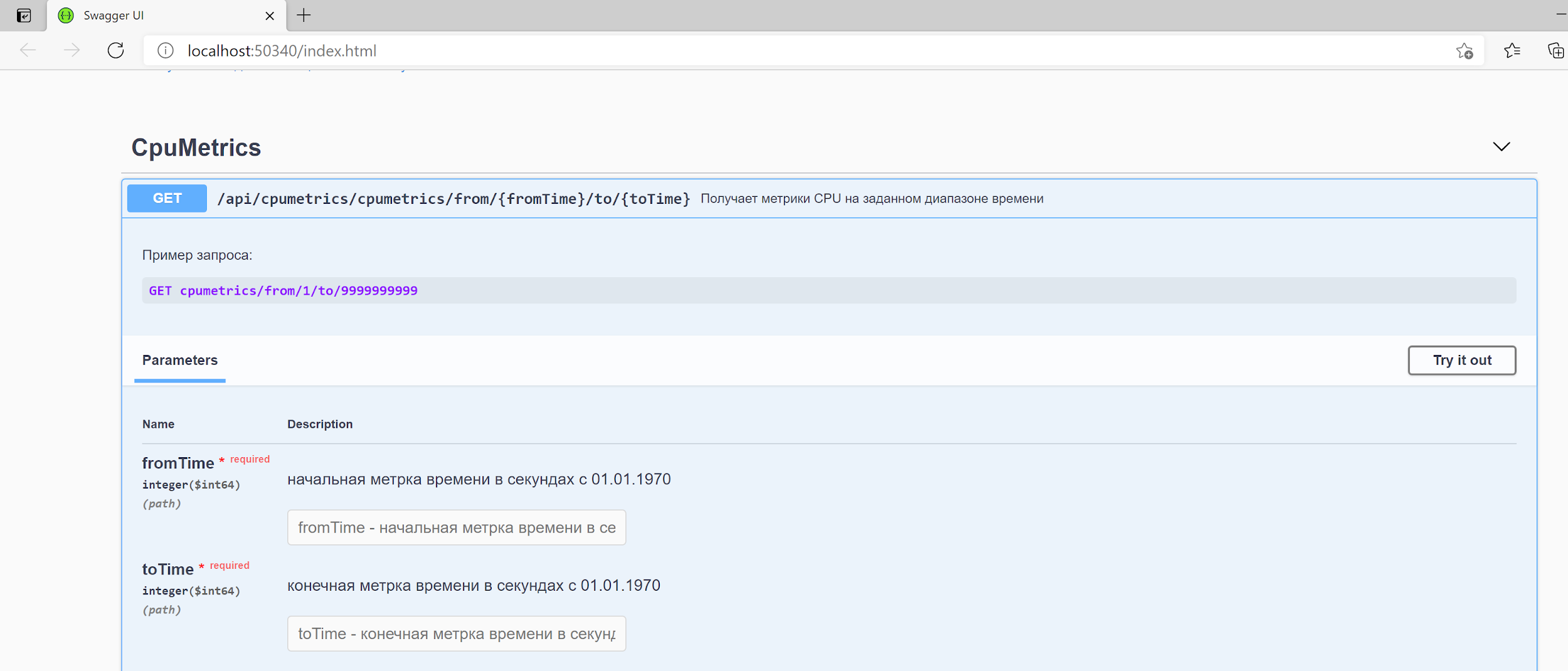
Не всегда удобно ходить по адресу /swagger. Давайте сделаем так, чтобы при запуске проекта мы сразу попадали на страницу Swagger UI. Для этого изменим регистрацию Swagger UI middlware на следующую

|  |
| --- |
| app.UseSwaggerUI(c => {  c.SwaggerEndpoint("/swagger/v1/swagger.json", "API сервиса агента сбора метрик")*;*  c.RoutePrefix = string.Empty*;* })*;* |

Дополнительно сообщим в launchSettings.json файле проекта, что мы не хотим переходить после запуска куда либо. Пусть нас сразу отправляет на страницу с документацией. Файл launchSettings.json необходимо привести к следующему виду

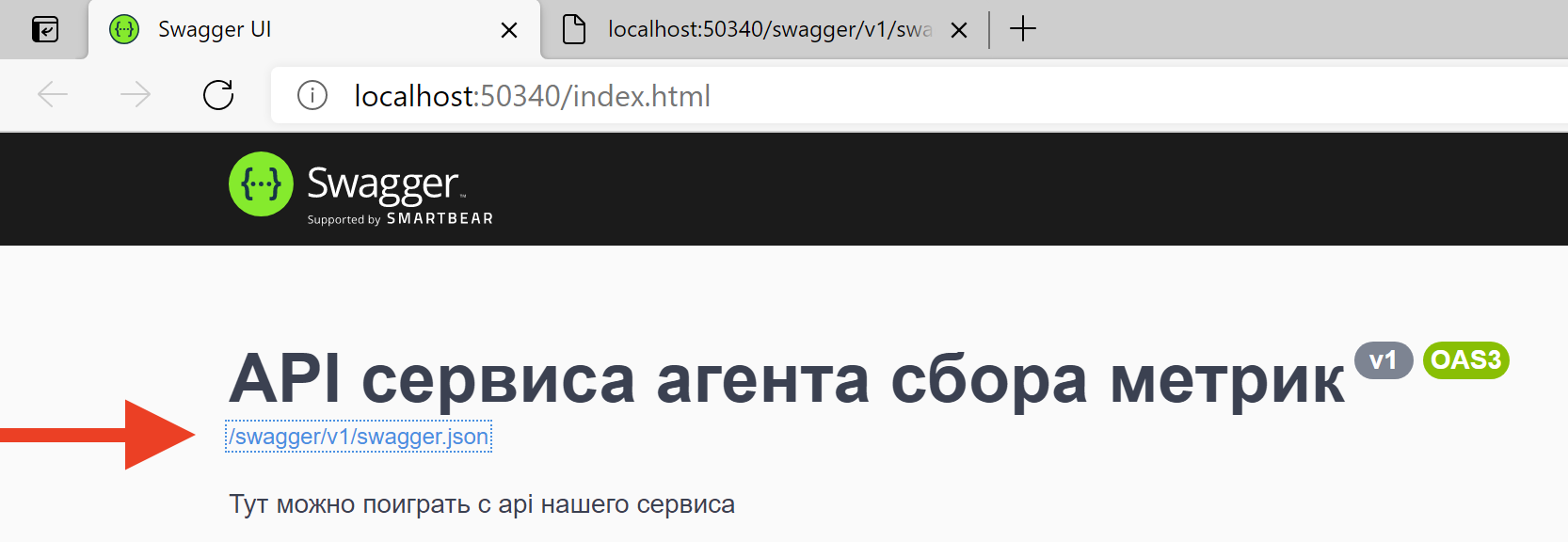
|  |
| --- |
| {  "$schema": "http://json.schemastore.org/launchsettings.json",  "iisSettings": {  "windowsAuthentication": false,  "anonymousAuthentication": true,  "iisExpress": {  "applicationUrl": "http://localhost:50340",  "sslPort": 0  }  },  "profiles": {  "IIS Express": {  "commandName": "IISExpress",  "launchBrowser": true,  "launchUrl": "",  "environmentVariables": {  "ASPNETCORE\_ENVIRONMENT": "Development"  }  }  } } |

Рассмотрим как в Swagger UI добавились комментарии из xml сгенерированной документации. Запустите отладку сервиса прямо из Visual Studio. Теперь перед вами сразу откроется документация к сервису

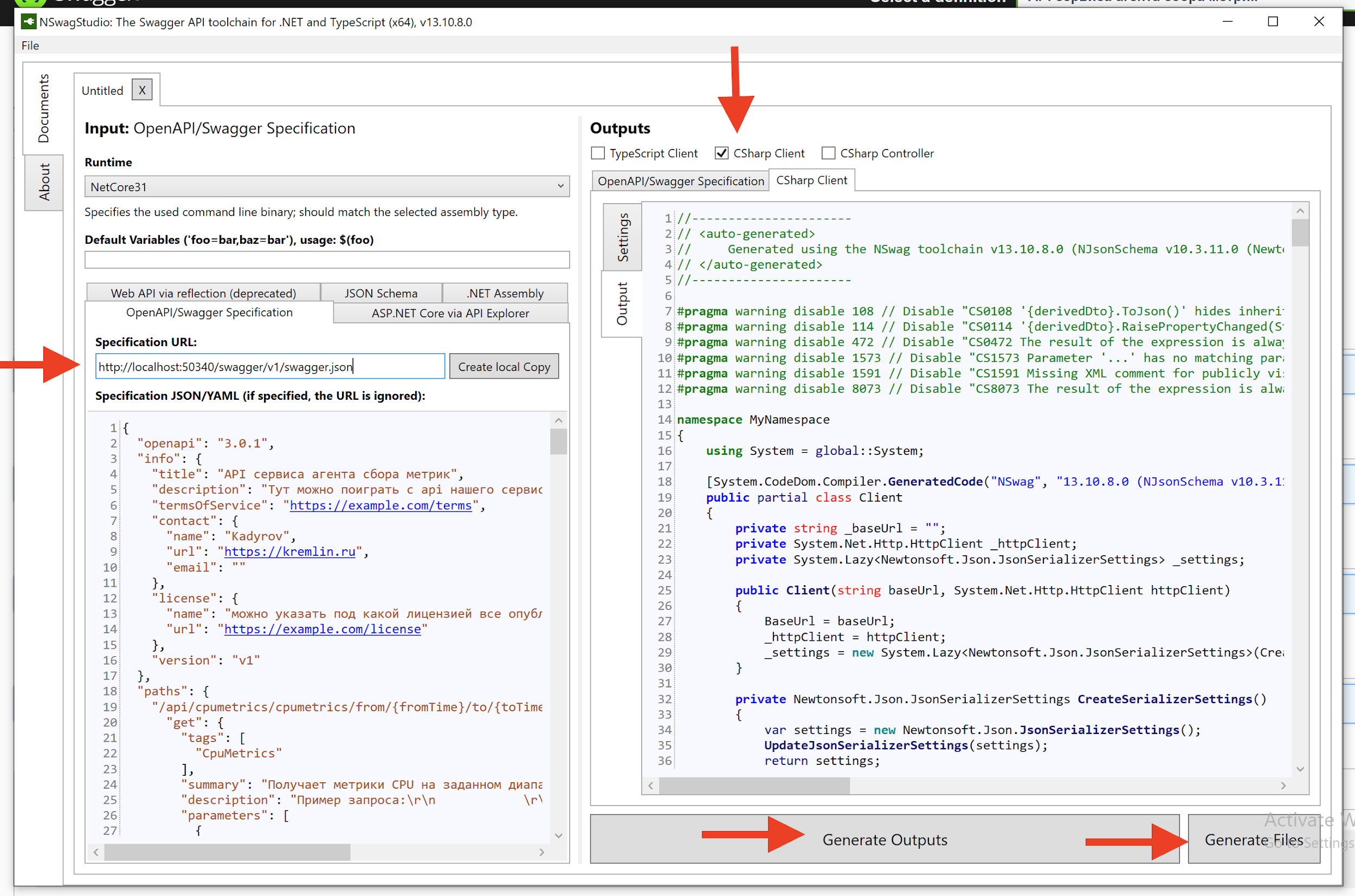


# Генерация клиента

По созданной автоматически Open API спецификации можно сгенерировать клиента автоматически. Для этого установите программу NSwag Studio как расказано в инструкции <https://github.com/RicoSuter/NSwag/wiki/NSwagStudio>. После этого запустите наш сервис в режиме отладки и перейдите на json описание файла (файл спецификацию Open API)



Скопируйте адрес, по которому лежит спецификация Open API из адресной строки браузера. Не закрывайте браузер, чтобы наш файл был доступен для программы NSwag Studio. Теперь запустите программу NSwag Studio



Интерфейс программы несколько перегружен, но он все же не очень сложный. Для начала вставьте адрес, по которому лежит спецификация Open API в строку Specification URL и нажмите кнопку Create local copy. Затем поставьте галочку CSharp Client как показано на рисунке. Тем самым мы сообщаем программе, что хотим, чтобы она сгенерировала клиента на языке C#. Последовательно нажмите кнопки Generate Outputs и Generate Files. Теперь можно воспользоваться автоматически сгенерированным клиентом непосредственно в коде менеджера метрик. Рассмотрим как он будет выглядеть. Для этого проставьте следующие галочки в настройках (список их длинный, постарайтесь найти)

1. В Operation Generation Mode установите SingleClientFromPathSegments
2. Установите галочку generate synchronous methods
3. Generate interfaces for client classes
4. В самом низу списка настроек укажите output директорию, в которую необходимо положить сгенерированный файл

Интерфейс клиента будет содержать примерно следующие методы

|  |
| --- |
| public partial interface IClient {  */// <summary>Получает метрики CPU на заданном диапазоне времени</summary>*  */// <param name="fromTime">начальная метрка времени в секундах с 01.01.1970</param>*  */// <param name="toTime">конечная метрка времени в секундах с 01.01.1970</param>*  */// <returns>Success</returns>*  */// <exception cref="ApiException">A server side error occurred.</exception>*  AllCpuMetricsResponse ApiCpumetricsCpumetricsFromTo(long fromTime, long toTime);   ... } |

Интерфейс можно зарегистрировать в контейнере как мы делали до этого с собственно написанным клиентом.

# Практическое задание

* Попробуйте подключить и настроить Swagger UI для менеджера метрик
* Попробуйте сгенерить клиент для агента с помощью Swagger, и наследовать MetricsAgentClient из прошлого урока от сгенеренного Client.

# Список используемой литературы

* <https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle>
* <https://swagger.io/tools/swagger-codegen/>
* <https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger>

# Дополнительные материалы

* <https://swagger.io/specification/>
* <https://www.openapis.org/>