

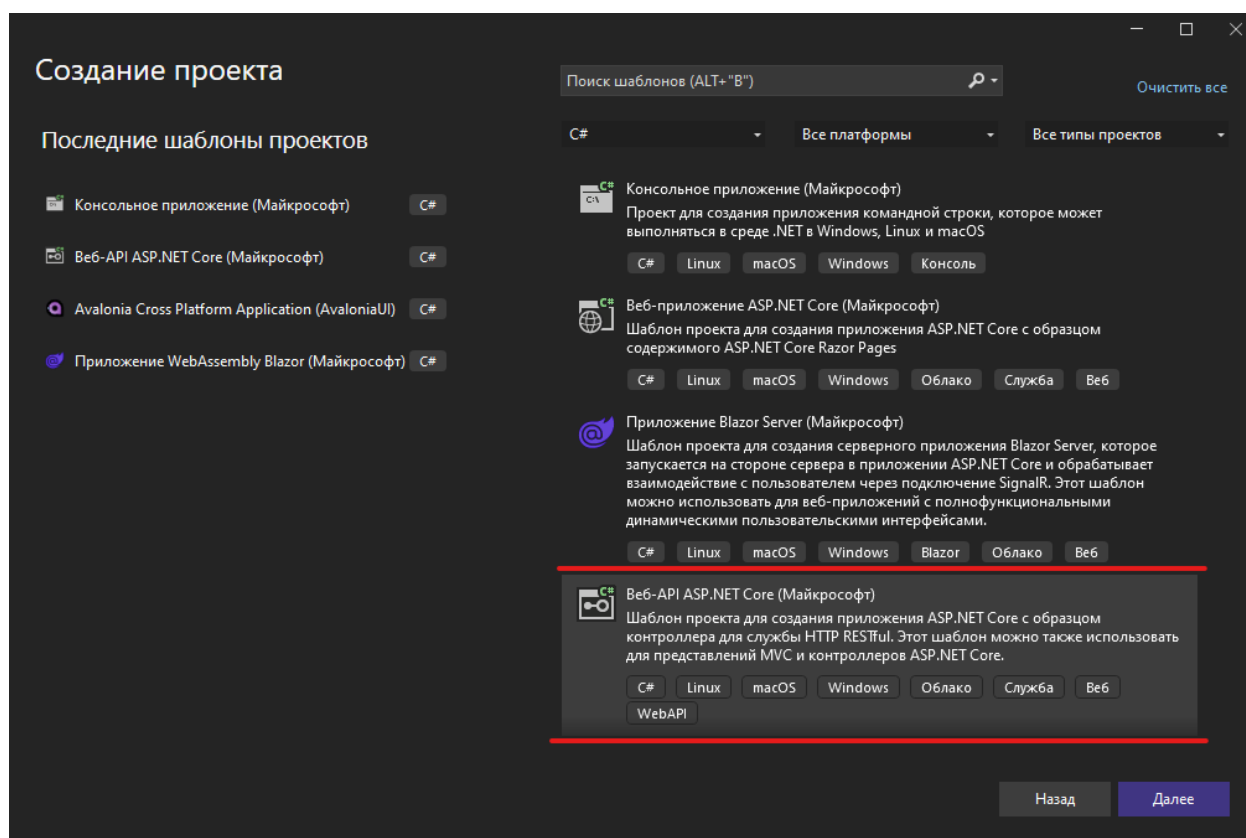
Создание первого API.

Взаимодействие с API.

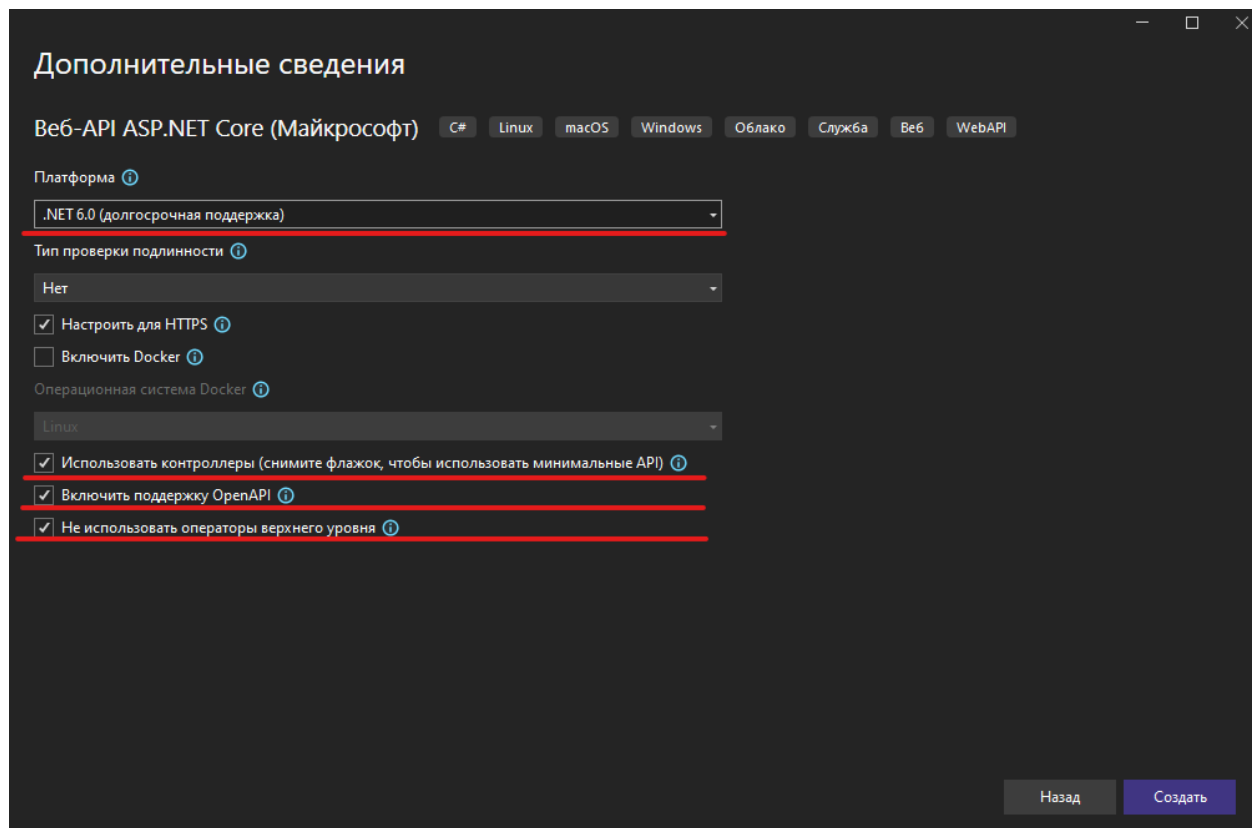
В данной практической работе мы создадим простейший API для доступа и обработки данных и на практике рассмотрим работу с ним.

Создание проекта API

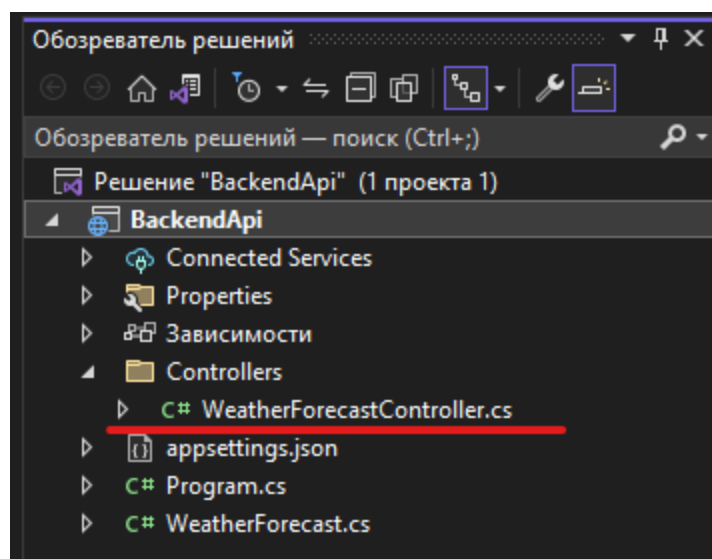
Откройте Visual Studio 2022 и при создании проекта выберите следующий шаблон Web API.



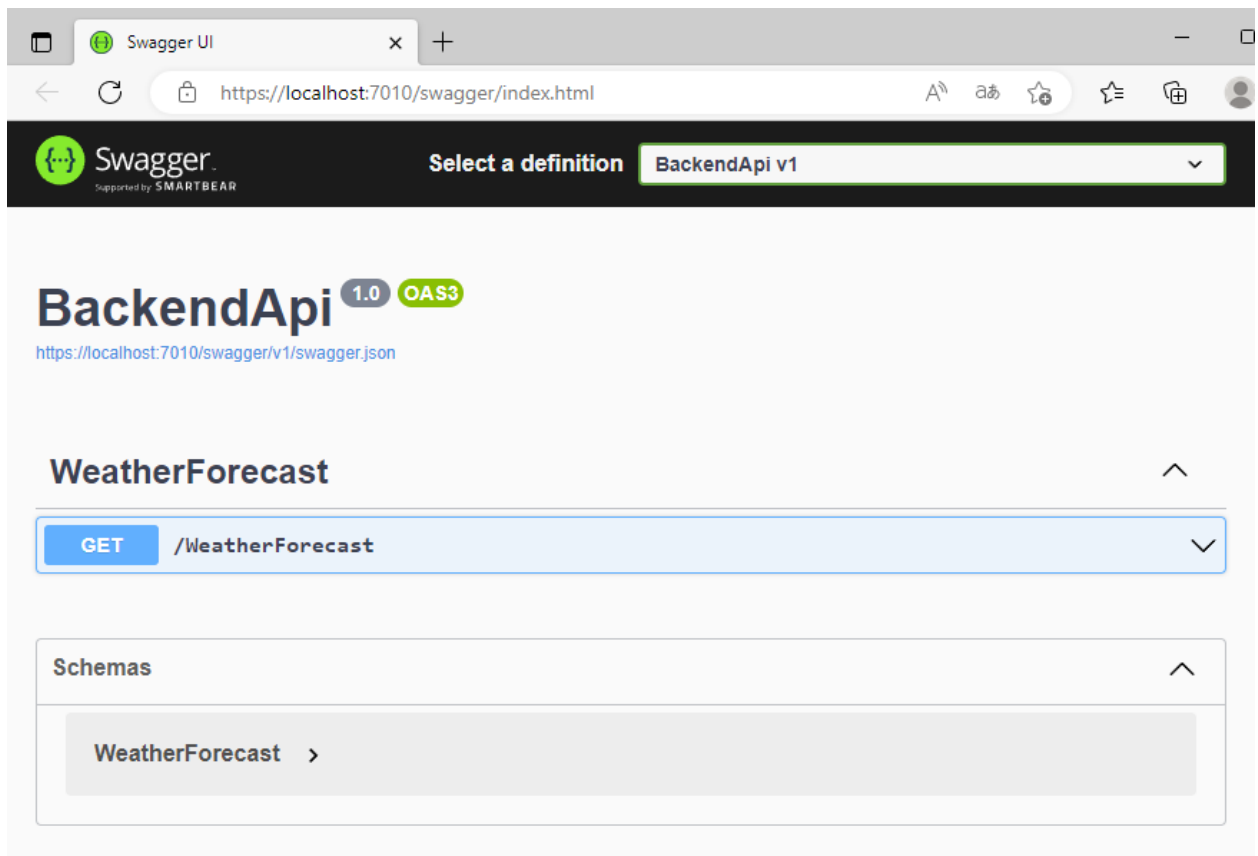
Затем укажите ему название (Например BackendApi) и в разделе “Дополнительные сведения” убедитесь, что установлены следующие галочки.



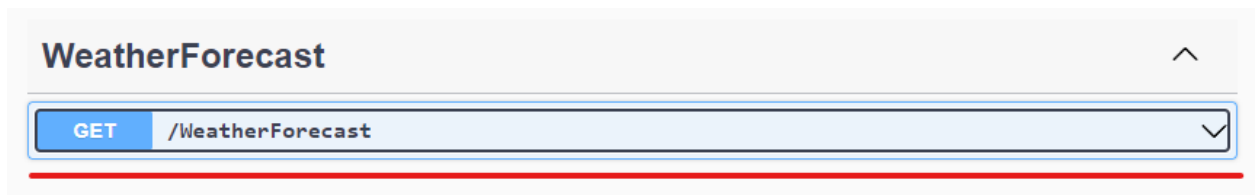
После создания проекта, в обозревателе решений откройте файл WeatherForecastController.cs в папке Controllers



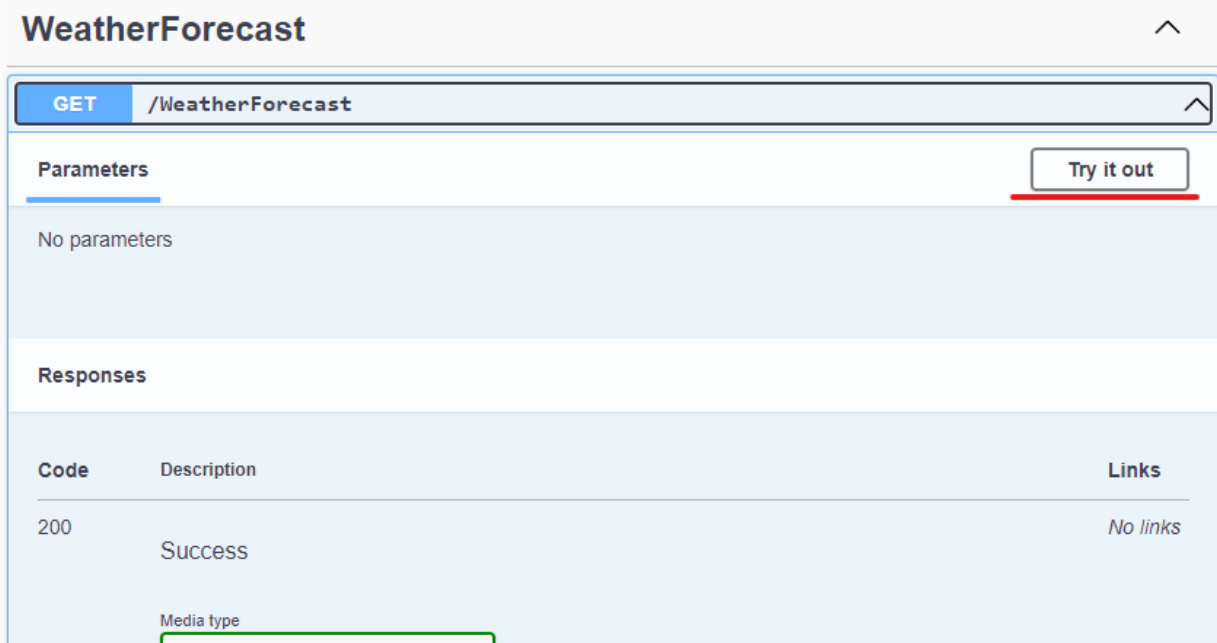
Запустите проект и убедитесь, что браузер при открытии отображает следующую страницу.



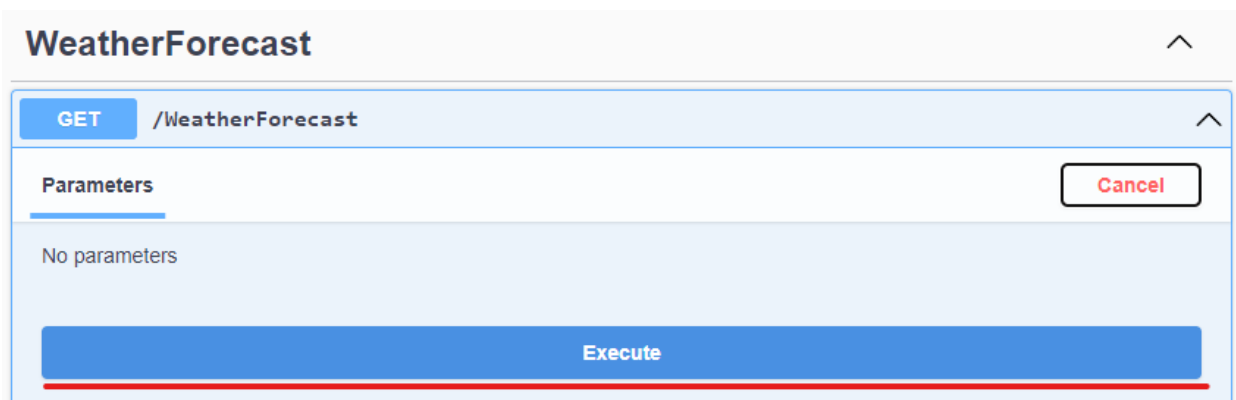
Раскройте раздел **GET /WeatherForecast**



Внутри него нажмите на кнопку **Try it out**



И затем на кнопку **Execute**



Изучите содержимое ответа. Именно в таком формате происходит обмен данными между клиентом и сервером.

Code

Details

200

Response body

```
[
  {
    "date": "2023-02-01T12:29:54.4239511+03:00",
    "temperatureC": 35,
    "temperatureF": 94,
    "summary": "Warm"
  },
  {
    "date": "2023-02-02T12:29:54.4244791+03:00",
    "temperatureC": 43,
    "temperatureF": 109,
    "summary": "Warm"
  },
  {
    "date": "2023-02-03T12:29:54.4244807+03:00",
    "temperatureC": 7,
    "temperatureF": 44,
    "summary": "Chilly"
  },
  {
    "date": "2023-02-04T12:29:54.4244808+03:00",
    "temperatureC": -10,
    "temperatureF": 15,
    "summary": "Bracing"
  },
  {
    "date": "2023-02-05T12:29:54.424481+03:00",
    "temperatureC": 32,
```



Download

Изменение контроллера API

Вернитесь к файлу WeatherForecastController.cs

```
WeatherForecastController.cs BackendApi: 0630p
BackendApi BackendApi.Controllers.WeatherForecastController
1 using Microsoft.AspNetCore.Mvc;
2
3 namespace BackendApi.Controllers
4 {
5     [ApiController]
6     [Route("[controller]")]
7     public class WeatherForecastController : ControllerBase
8     {
9         private static readonly string[] Summaries = new[]
10         {
11             "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
12         };
13
14         private readonly ILogger<WeatherForecastController> _logger;
15
16         public WeatherForecastController(ILogger<WeatherForecastController> logger)
17         {
18             _logger = logger;
19         }
20
21         [HttpGet(Name = "GetWeatherForecast")]
22         public IEnumerable<WeatherForecast> Get()
23         {
24             return Enumerable.Range(1, 5).Select(index => new WeatherForecast
25             {
26                 Date = DateTime.Now.AddDays(index),
27                 TemperatureC = Random.Shared.Next(-20, 55),
28                 Summary = Summaries[Random.Shared.Next(Summaries.Length)]
29             })
30             .ToArray();
31         }
32     }
33 }
```

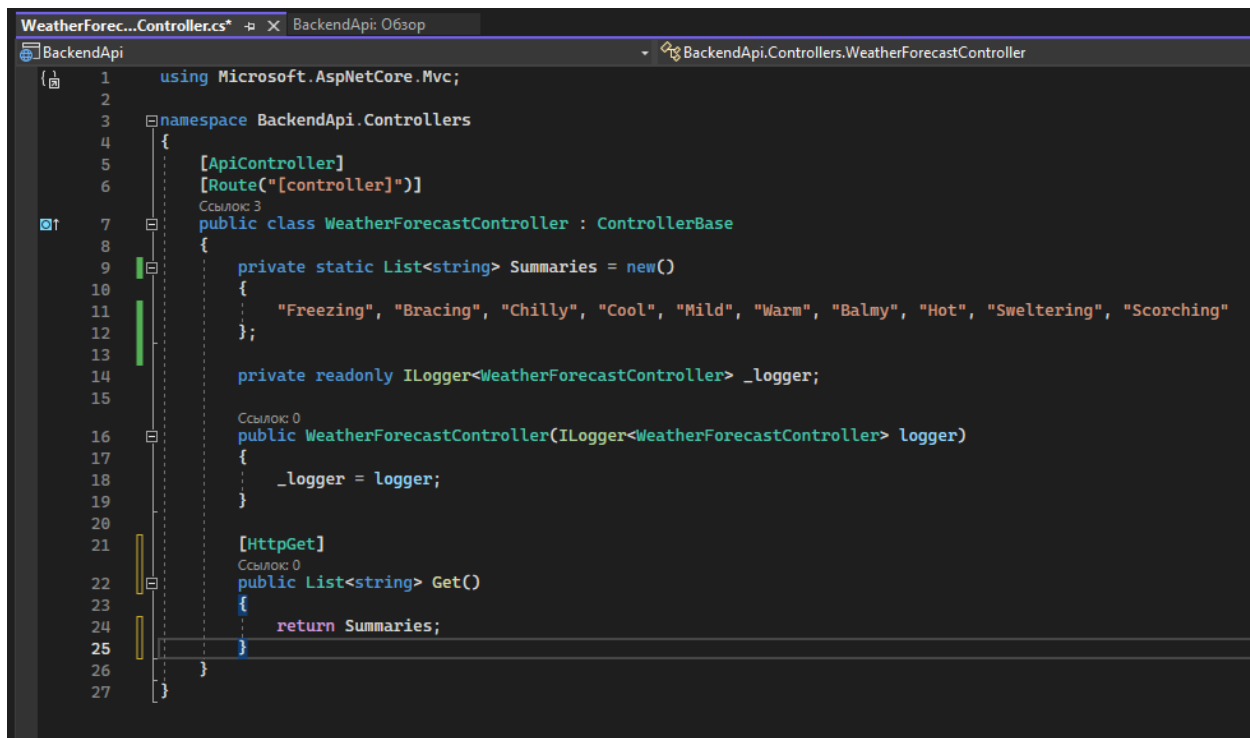
Измените тип переменной

```
[ApiController]
[Route("[controller]")]
Ссылка: 3
public class WeatherForecastController : ControllerBase
{
    private static List<string> Summaries = new()
    {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
    };
};
```

Переделайте метод Get в следующий вид.

```
[HttpGet]
Ссылка: 0
public List<string> Get()
{
    return Summaries;
}
```

В результате у вас должен получиться следующий код:



```
1 using Microsoft.AspNetCore.Mvc;
2
3 namespace BackendApi.Controllers
4 {
5     [ApiController]
6     [Route("[controller]")]
7     public class WeatherForecastController : ControllerBase
8     {
9         private static List<string> Summaries = new()
10         {
11             "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
12         };
13
14         private readonly ILogger<WeatherForecastController> _logger;
15
16         public WeatherForecastController(ILogger<WeatherForecastController> logger)
17         {
18             _logger = logger;
19         }
20
21         [HttpGet]
22         public List<string> Get()
23         {
24             return Summaries;
25         }
26     }
27 }
```

Запустите проект и проверьте метод Get

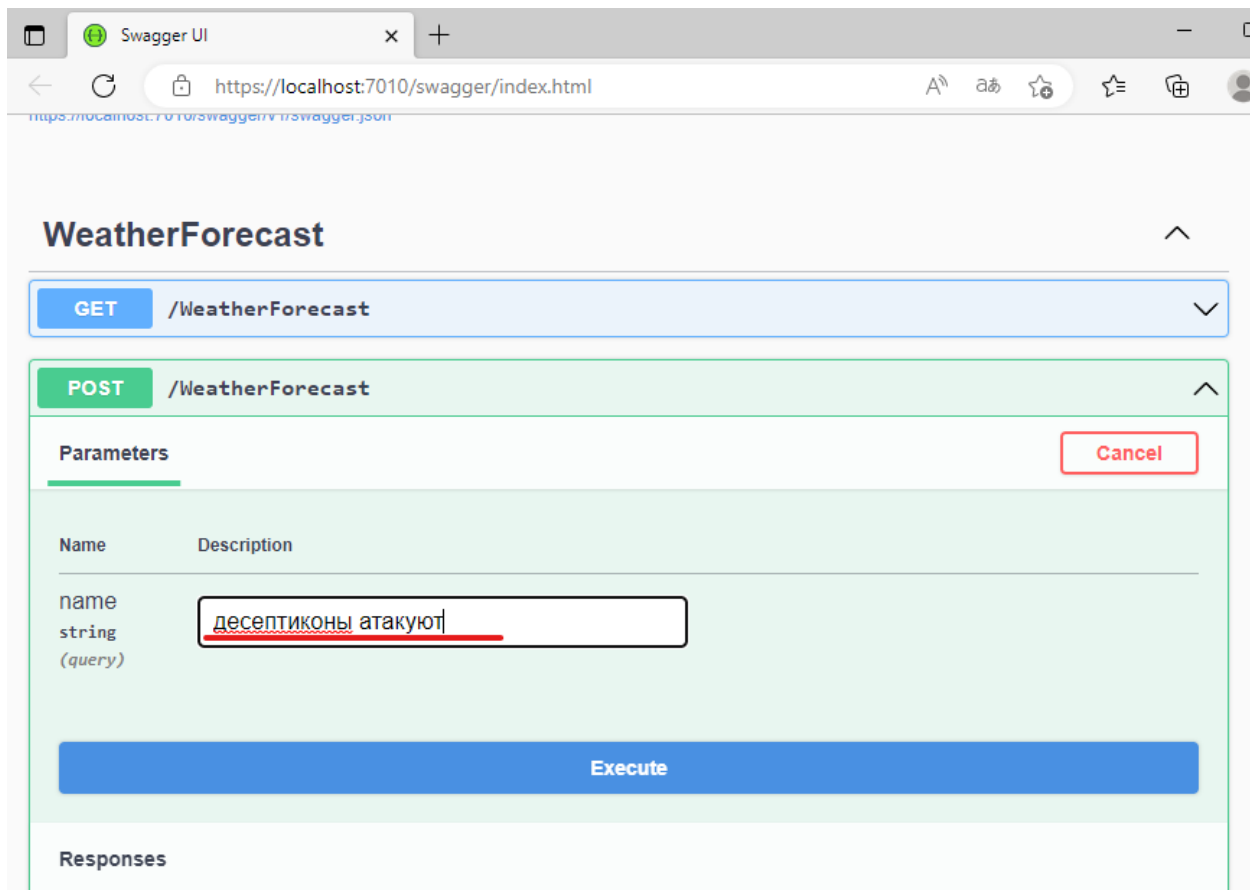
Внедрение функции добавления наименования в API

Добавьте под функцией Get следующий код.

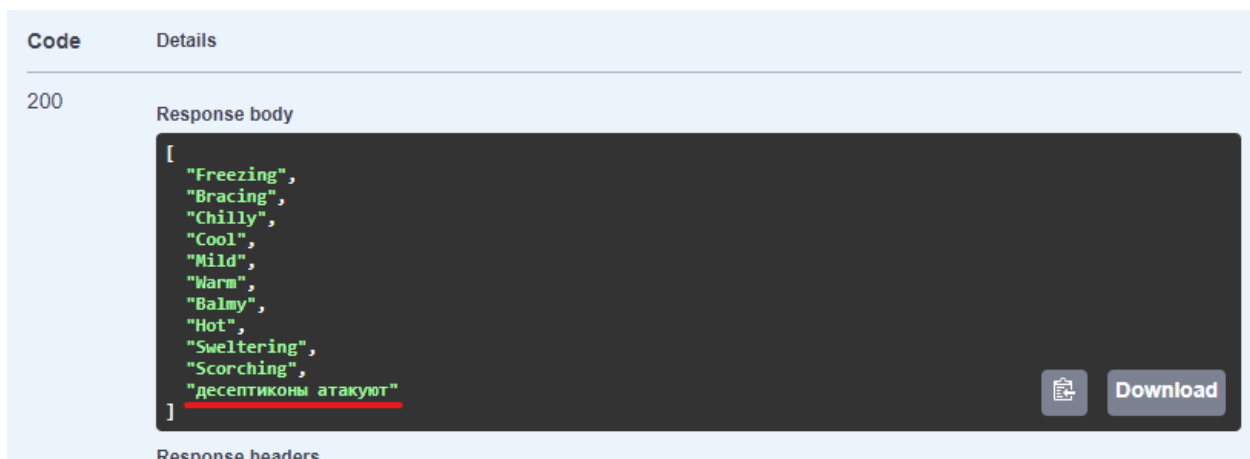
```
20
21 [HttpGet]
    Ссылка: 0
22 public List<string> Get()
23 {
24     return Summaries;
25 }
26
27 [HttpPost]
    Ссылка: 0
28 public IActionResult Add(string name)
29 {
30     Summaries.Add(name);
31     return Ok();
32 }
33
34 }
```

Запустите проект и проверьте метод добавления имени.

Раскройте метод POST и напишите в поле **name** любое значение и нажмите на кнопку **Execute**.

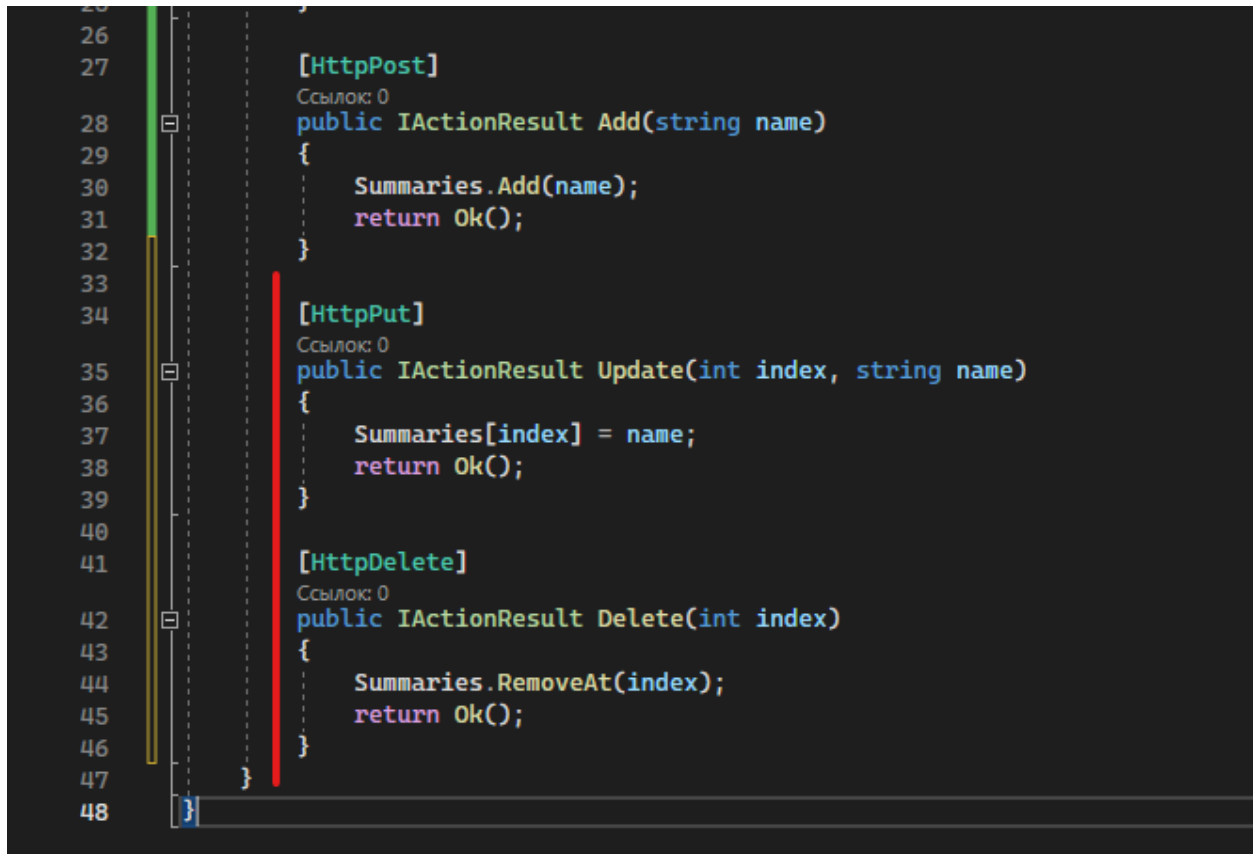


Раскройте метод GET, нажмите на кнопку **Execute** и убедитесь, что ваше значение добавилось.



Внедрение функций изменения и удаления наименования в API

Добавьте под функцией Add следующий код.



```
26
27 [HttpPost]
    Ссылка: 0
28 public IActionResult Add(string name)
29 {
30     Summaries.Add(name);
31     return Ok();
32 }
33
34 [HttpPut]
    Ссылка: 0
35 public IActionResult Update(int index, string name)
36 {
37     Summaries[index] = name;
38     return Ok();
39 }
40
41 [HttpDelete]
    Ссылка: 0
42 public IActionResult Delete(int index)
43 {
44     Summaries.RemoveAt(index);
45     return Ok();
46 }
47 }
48 }
```

Запустите проект и проверьте работу обновления и удаления записи.

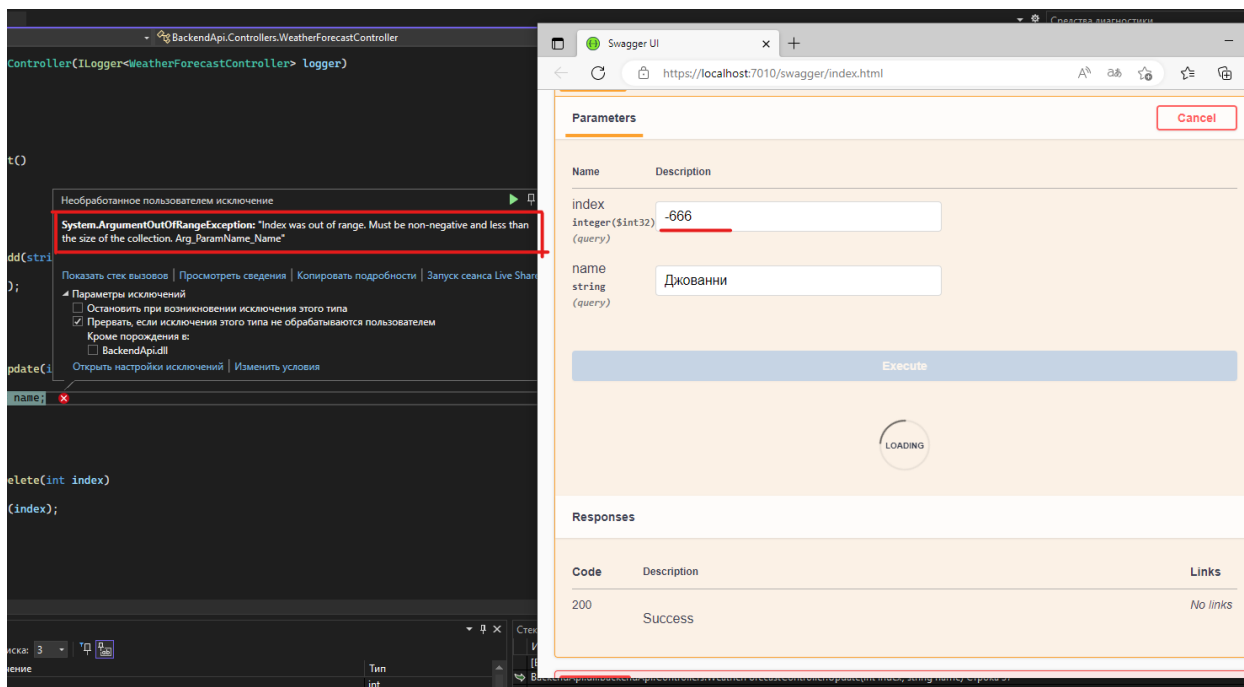
Указываемый **индекс** обозначает номер элемента по счёту - счёт начинается с 0

Проверка ошибок при вводе данных

В каждой функции должна присутствовать проверка на ошибки - иначе при вводе таких данных может произойти критическая ошибка, которая приведет к выведению из строя разработанное приложение.

Добавим проверку входных значений для функции Update.

В Update у нас указываются параметры index и name, если указать неверный индекс произойдет следующая ошибка.



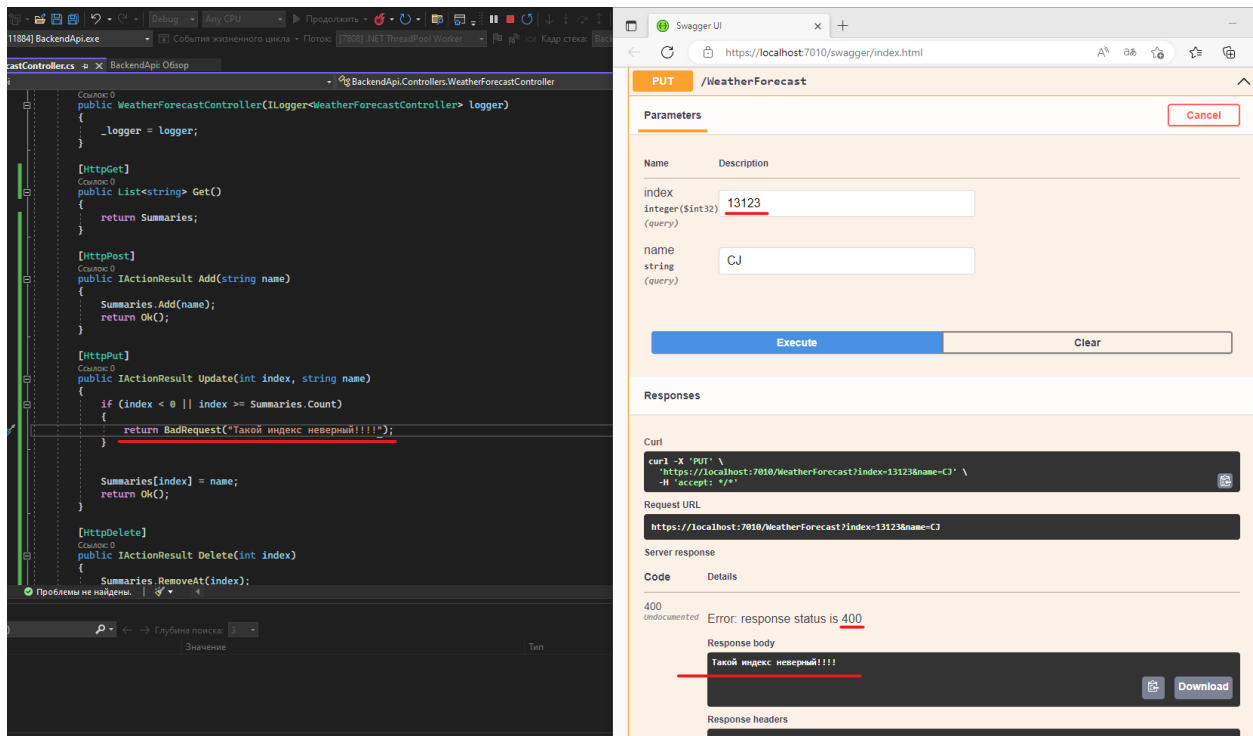
Для того, чтобы приложение не “падало”, необходимы проверки вводимых значений.

Добавим для этого метода следующее условие. (Если индекс меньше нуля ИЛИ больше количества наименований, то выбросить сообщение об ошибке)

```
[HttpPut]
Ссылка: 0
public IActionResult Update(int index, string name)
{
    if (index < 0 || index >= Summaries.Count)
    {
        return BadRequest("Такой индекс неверный!!!!");
    }

    Summaries[index] = name;
    return Ok();
}
```

Теперь приложение работает стабильно



Задания:

1. Добавьте проверку для всех методов, у которых указывается параметр (Например не допускать указание отрицательного индекса)
2. Добавьте метод для вывода одного наименования по указанному индексу. В атрибуте `HttpGet` укажите следующий параметр.

```
[HttpGet("{index}")]
```

3. Добавьте метод для получения количества записей по указываемому имени. Для данного метода укажите следующий параметр.

```
[HttpGet("find-by-name")]
```

4. Добавьте для метода получения всех записей (`GetAll`) необязательный параметр, который вернёт отсортированный список. Для данного метода включите необязательный параметр `strategy`. Работа метода должна быть следующей:

- а. Если значение параметра `null` - верните список таким, какой он есть.

- b. Если значение параметра 1 - верните отсортированный список по возрастанию.
- c. Если значение параметра -1 - верните отсортированный список по убыванию.
- d. При всех остальных значениях вернуть ошибку (BadRequest) с сообщением "Некорректное значение параметра sortStrategy"

Выглядеть метод GetAll должен следующим образом:

```
[HttpGet]  
Ссылка: 0  
public IActionResult GetAll(int? sortStrategy)
```