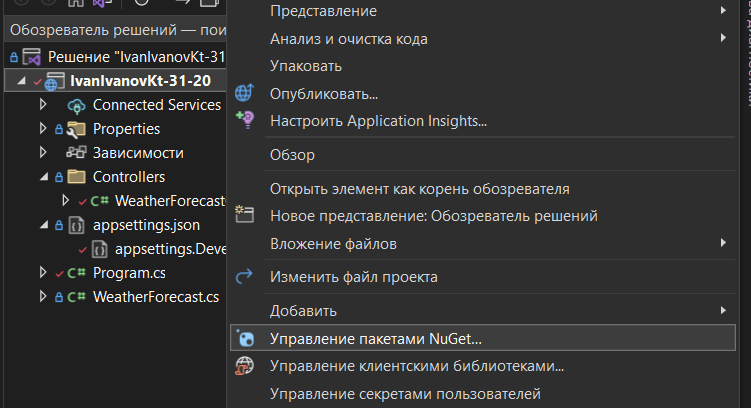
Лабораторная работа 2

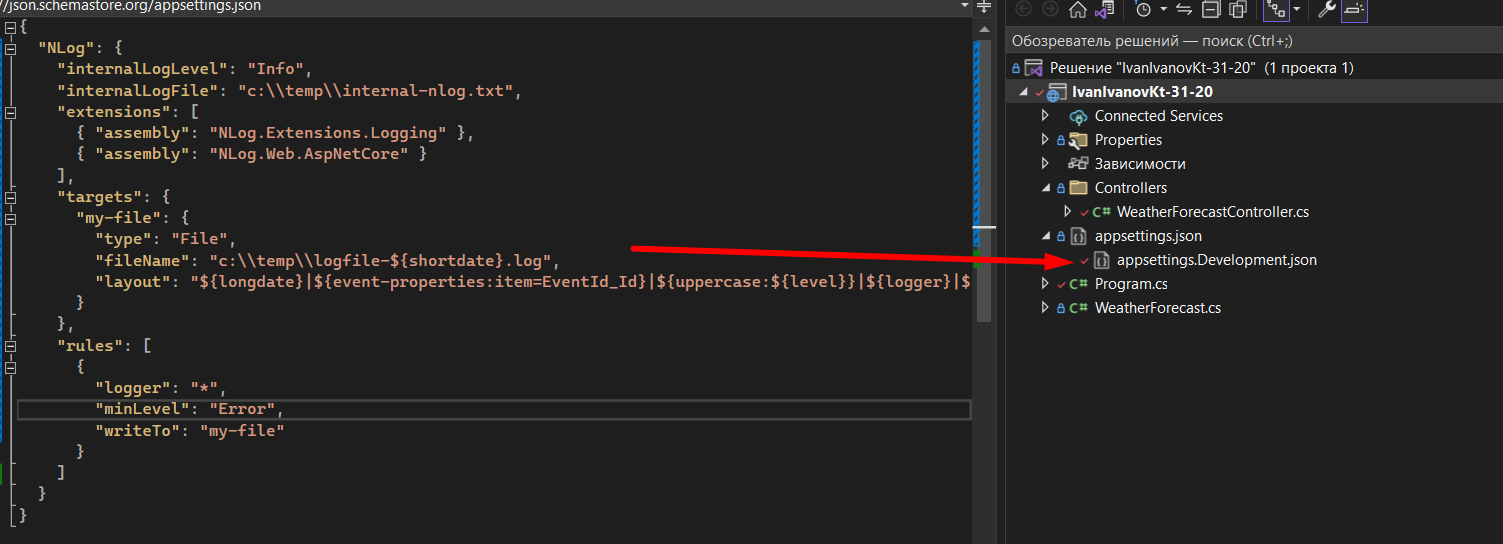
Добавление логирования и методов Rest-запросов

Логирование необходимо любому бэкенд приложению, так как в конечном итоге, рабочее приложение размещается на сервере и при возникновении ошибок, разработчику нужно найти и устранить причину ошибки, логи отличный способ узнать причину ошибки. Логирование устанавливается разработчиками в критических зонах кода, позволяет записывать исключения и прочие действия, происходящие внутри приложения.

1. Добавить в проект Nuget-пакет NLog.Web.AspNetCore

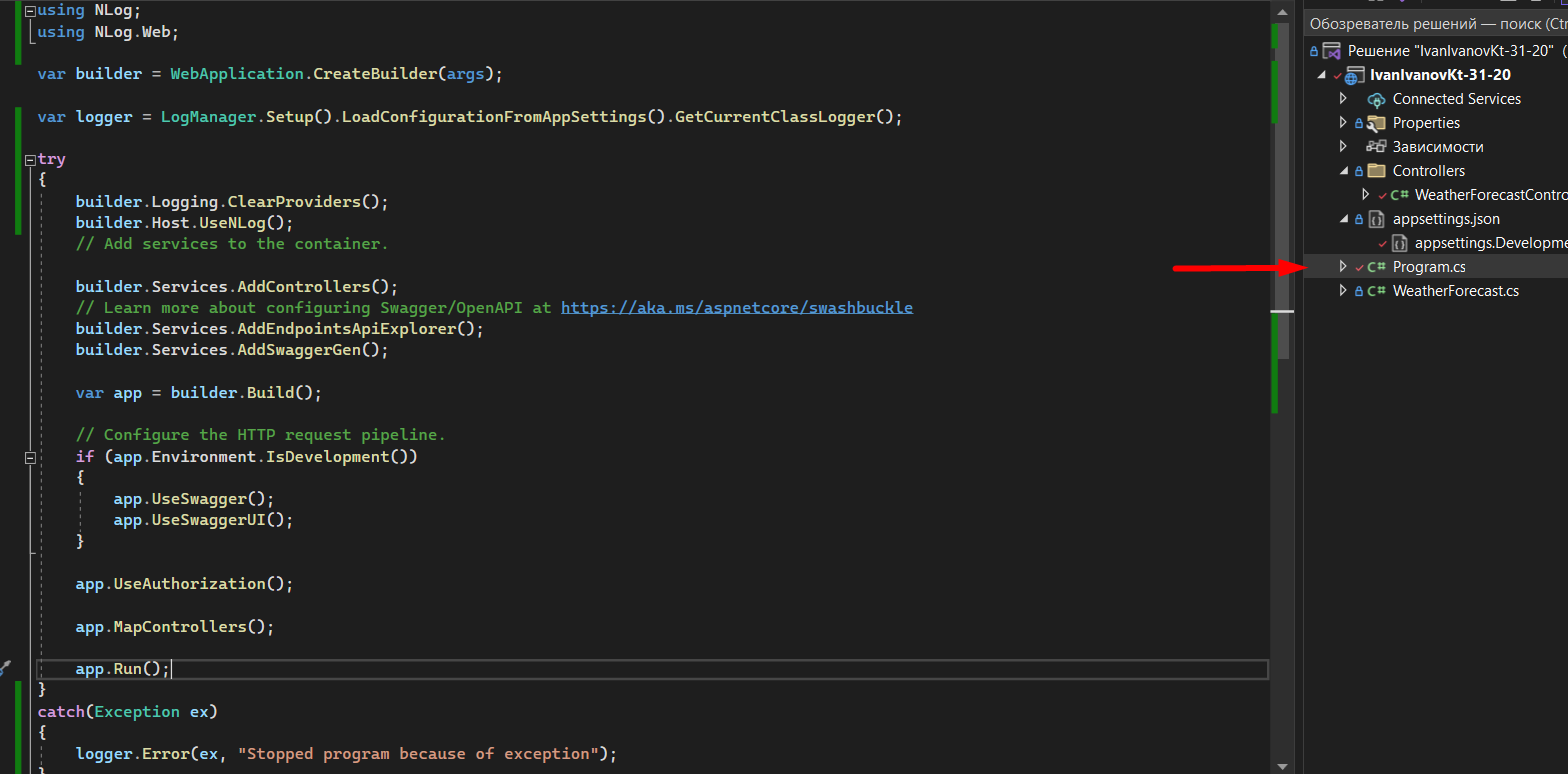
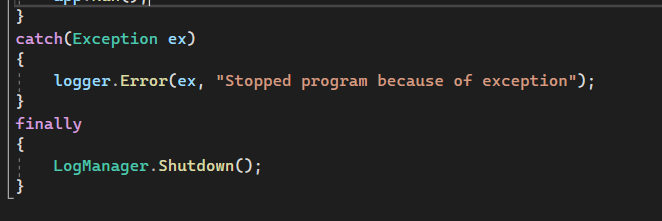


1. Добавить в файл appsettings.Development.json конфигурацию для Nlog (текст конфигурации в конце документа). Путь в internalLogFile и в filename прописываете любой доступный, доступ к диску C может быть ограничен и файлы с логами не появятся.

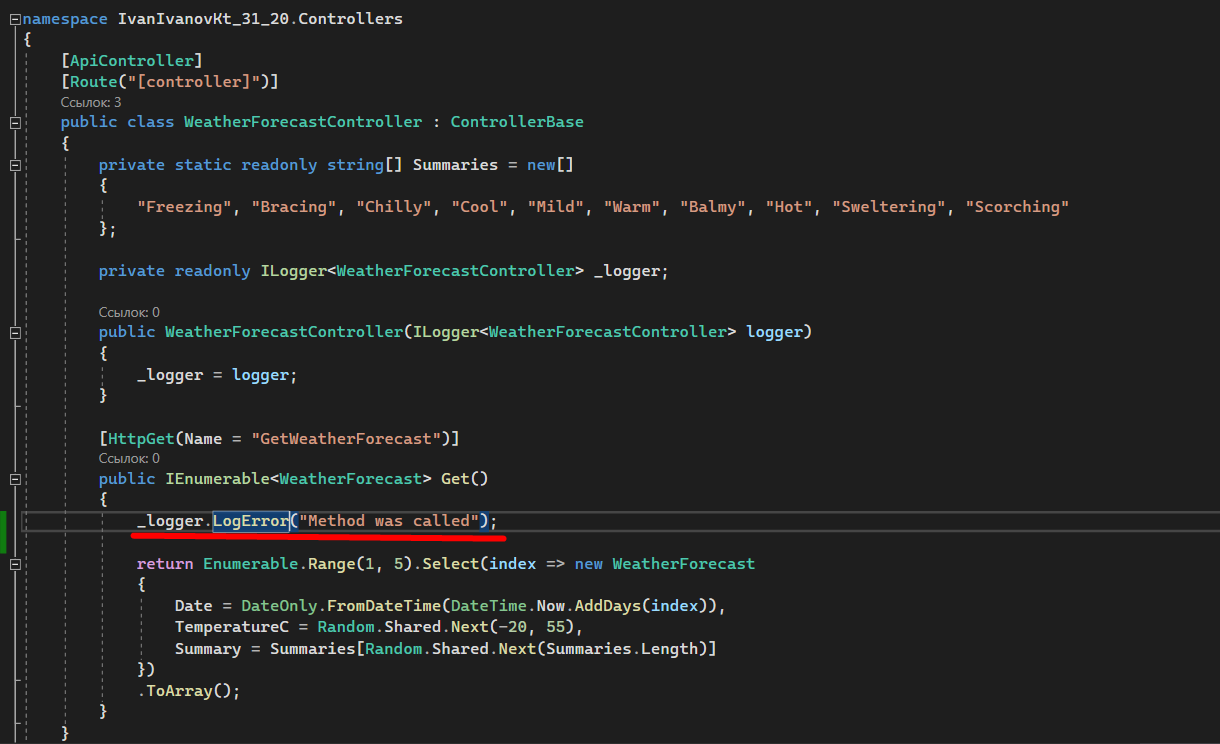


1. Добавить инициализацию логгера, изменив код подобным образом.

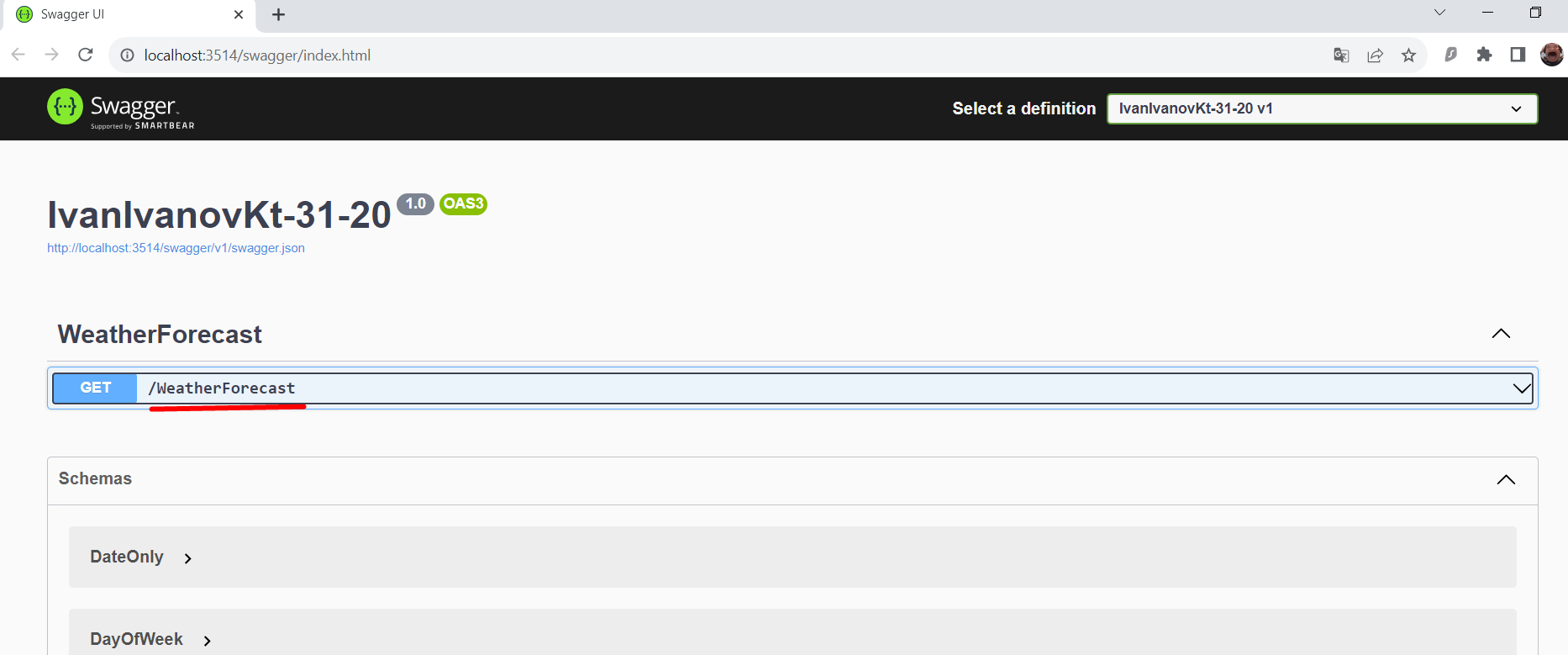
Код оборачивается в блок try/catch/finally, при возникновении необработанной ошибки, логгер в блоке finally будет отключаться, не вызывая утечек памяти.

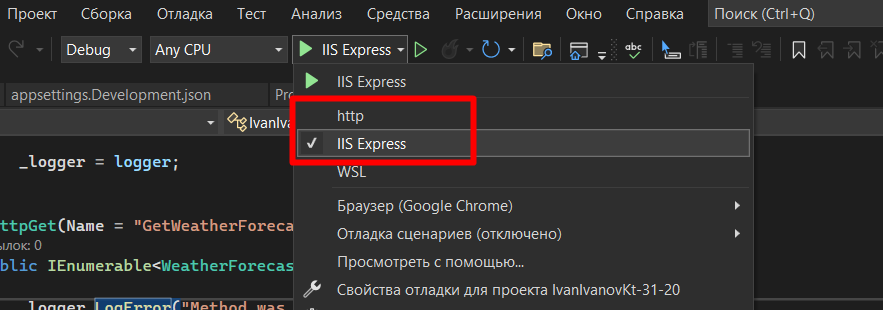
 

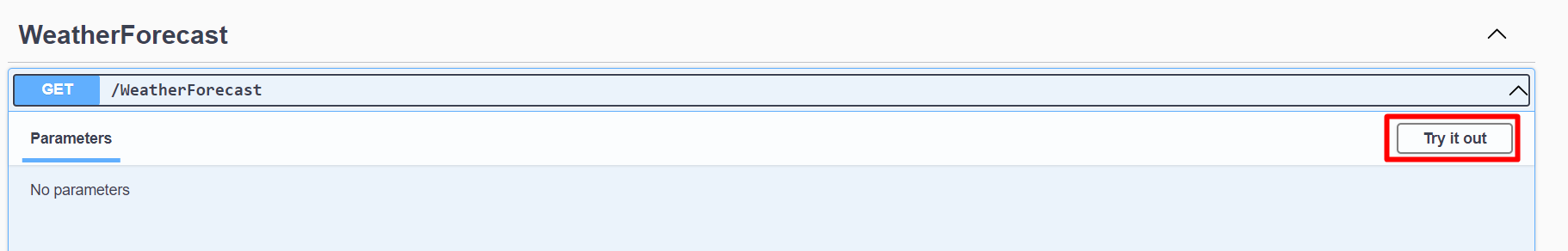
1. В контроллере, который создан шаблоном, добавить тестовое логирование

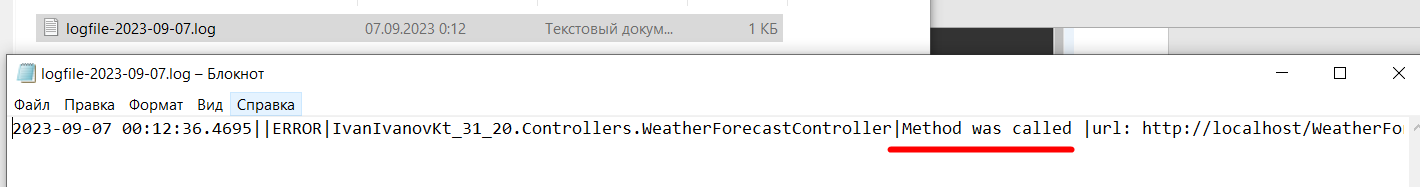


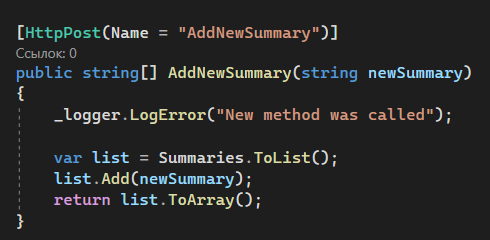
1. Строки кода над public class… Называются атрибуты (таким же образом записываются и фильтры) они задают определенное поведение классу. В нашем случае класс будет выполнять роль api контроллера (написан соответствуюущий атрибут). Путь к нему задается в атрибуте Route. Этот путь используются в адресной строке браузера или в других средствах, когда нужно получить доступ к api. Ключевое слово controller внутри Route означает, что в пути будет использоваться название класса от начала и до слова Controller.

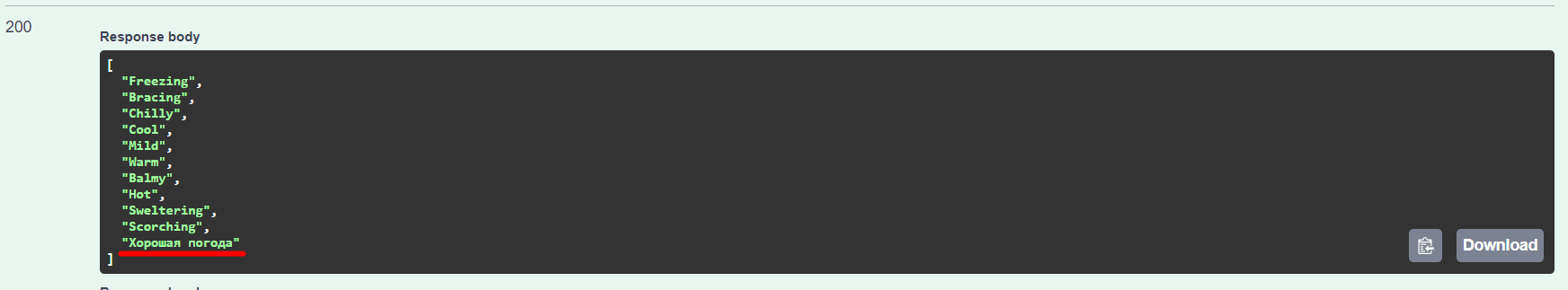


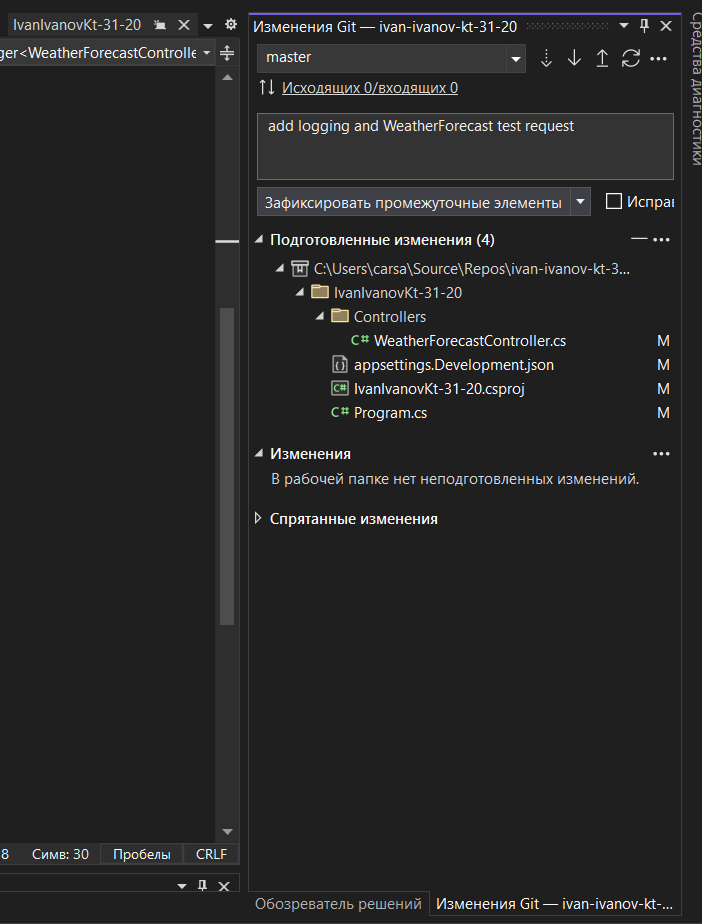
1. Чтобы проверить логирование необходимо запустить приложение и вызвать метод Get у контроллера. Для запуска в параметрах указать либо http (приложение будет запущено вместе с консолью и будет включено, пока не закроется консоль, либо отладка приложения. Необходимо открыть браузер и ввести в адресную строку путь к приложению), либо IIS Express (в таком случае откроется вкладка браузера сразу, но при закрытии вкладки приложение тоже остановится) 
2. Для вызова метода Get у контроллера WeatherForecastController, необходимо раскрыть шеврон и нажать Try it out. Затем Execute. Так как метод не имеет входных параметров, то ничего вводить не нужно.



1. Приложение обработает входящий запрос и выдаст итоговый запрос и ответ. Можно посмотреть сам ответ, а так же заголовки ответа  
    
2. Так как запрос отработан, то и логирование тоже должно было пройти успешно. Посмотреть файл логов можно по пути, который указан в конфигурации appsettings 
3. Добавить новый метод для POST запрос, который добавит в итоговую погоду новое описание в массив итоговой погоды (по факту оно добавляться не будет)



1. После перекомпиляции приложения в swagger появится новый метод Post, внутри можно добавить название, которое в итоге появится в ответе, после выполнения запроса 
2. При успешной работе приложения после внесения всех изменений внести изменения в систему контроля версий (выполнить commit (фиксацию) и push(отправку)



{

"NLog": {

"internalLogLevel": "Info",

"internalLogFile": "c:\\temp\\internal-nlog.txt",

"extensions": [

{ "assembly": "NLog.Extensions.Logging" },

{ "assembly": "NLog.Web.AspNetCore" }

],

"targets": {

"my-file": {

"type": "File",

"fileName": "c:\\temp\\logfile-${shortdate}.log",

"layout": "${longdate}|${event-properties:item=EventId\_Id}|${uppercase:${level}}|${logger}|${message} ${exception:format=tostring}|url: ${aspnet-request-url}|action: ${aspnet-mvc-action}"

}

},

"rules": [

{

"logger": "\*",

"minLevel": "Error",

"writeTo": "my-file"

}

]

}

}