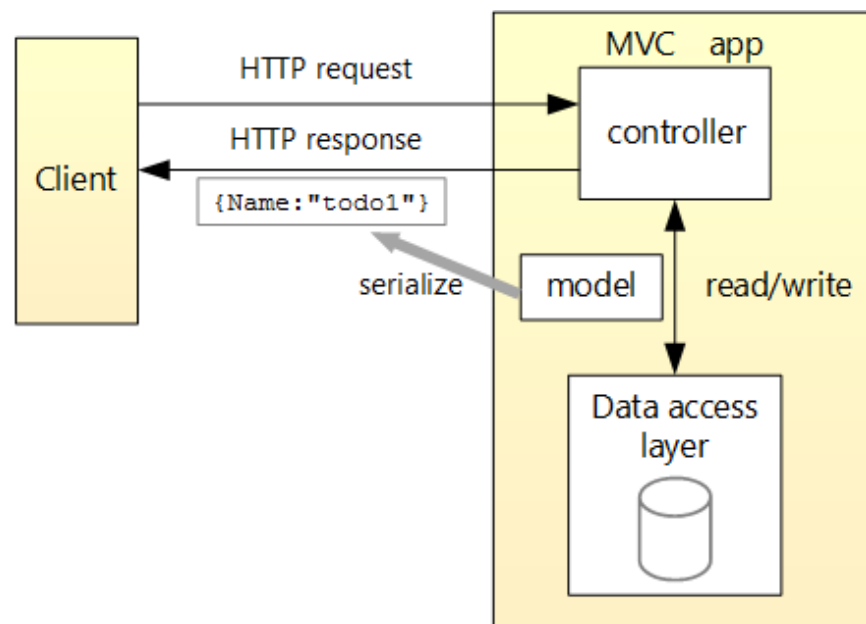


Взаимодействие с API в качестве клиентского приложения. Создание телеграм-бота для интернет-магазина

В данной практической работе попробуем поработать с API в качестве клиента. В качестве примера клиента нашего API будет создан телеграм-бот.

Overview

Давайте вспомним схему работы API и Клиента.



На упрощенной схеме вы можете увидеть, что имеются 2 жёлтых блока

1. Client (Клиентское приложение), который может быть кем-угодно [Мобильное или настольное приложение, Веб-сайт, Микроволновка или другой API]

2. MVC app (Или же просто сервер), который принимает или возвращает какие-либо данные, чтобы потом передать/получить данные базы данных

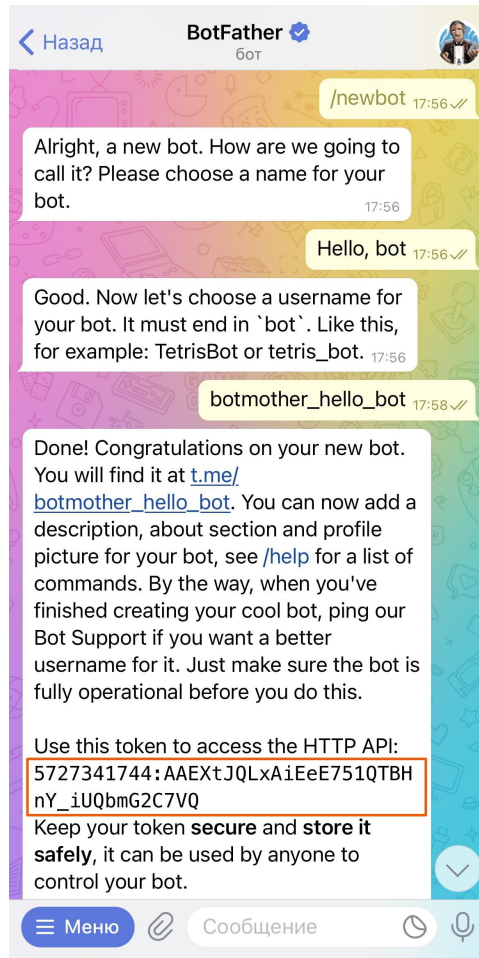
Создание телеграм-бота через BotFather

В Telegram чат-боты создаются с помощью специального бота [@BotFather](#). Через него можно управлять данными бота, добавить описание, аватар и т.д.

Создадим бота и придумаем ему название:

1. Откройте в Telegram бота [@BotFather](#).
2. Напишите ему /newbot.
3. Придумайте и напишите название бота. Оно будет отображаться в контактах и чатах. Например, «Hello, bot».
4. Придумайте и напишите юзернейм. Он используется для упоминания бота и в ссылках. Юзернейм должен быть на латинице и обязательно заканчиваться на «bot». Например, «botmother_hello_bot».

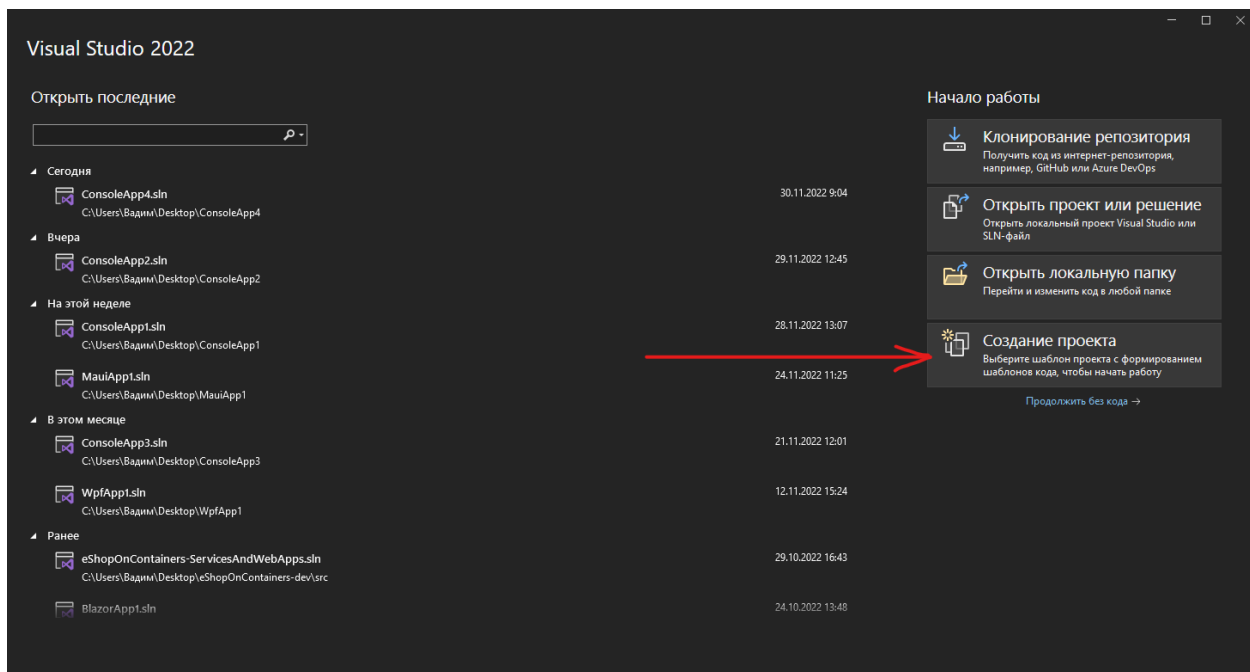
Если бот будет создан успешно - полученный токен вы должны сохранить. Им придется воспользоваться немножечко позже.



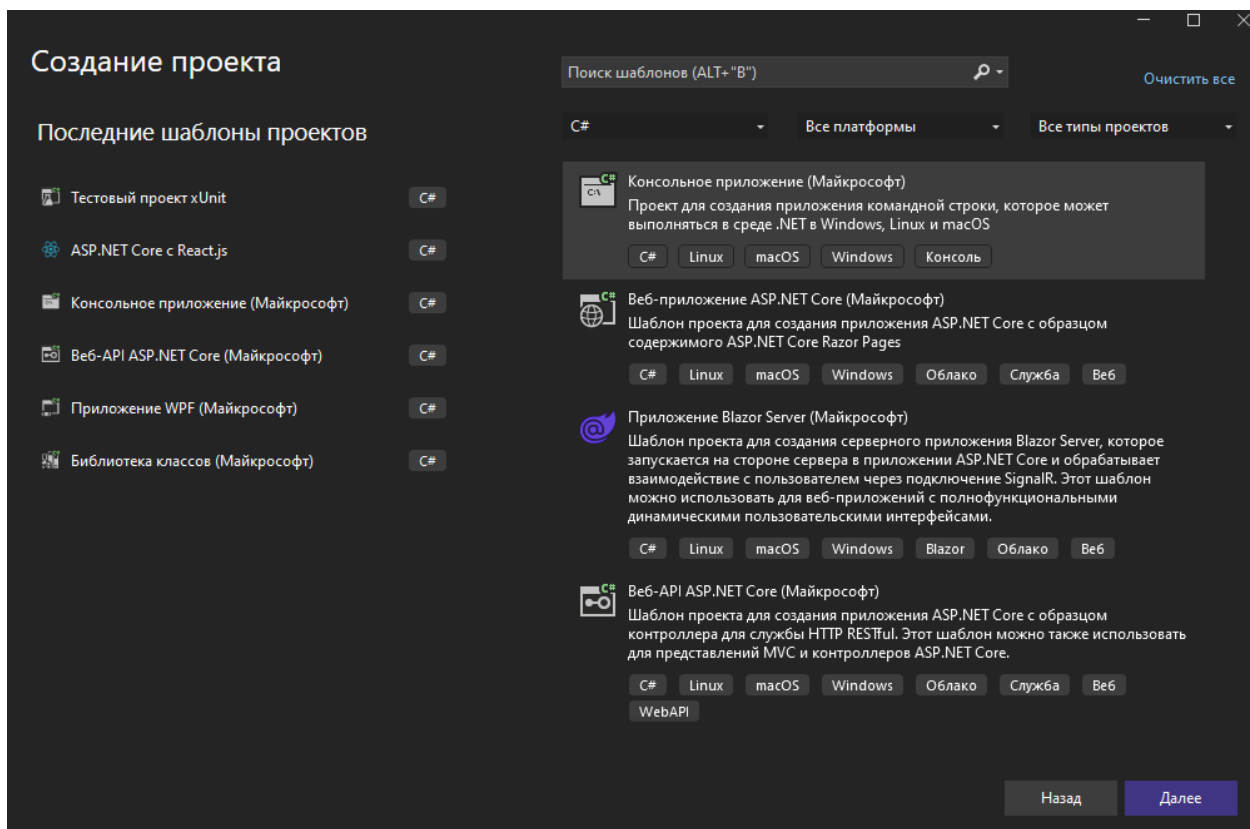
Подготовка приложения для написания основной логики бота

Перед выполнением следующих шагов, необходимо создать какой-нибудь проект. Так как бот может функционировать без графического приложения - создадим консольное приложение.

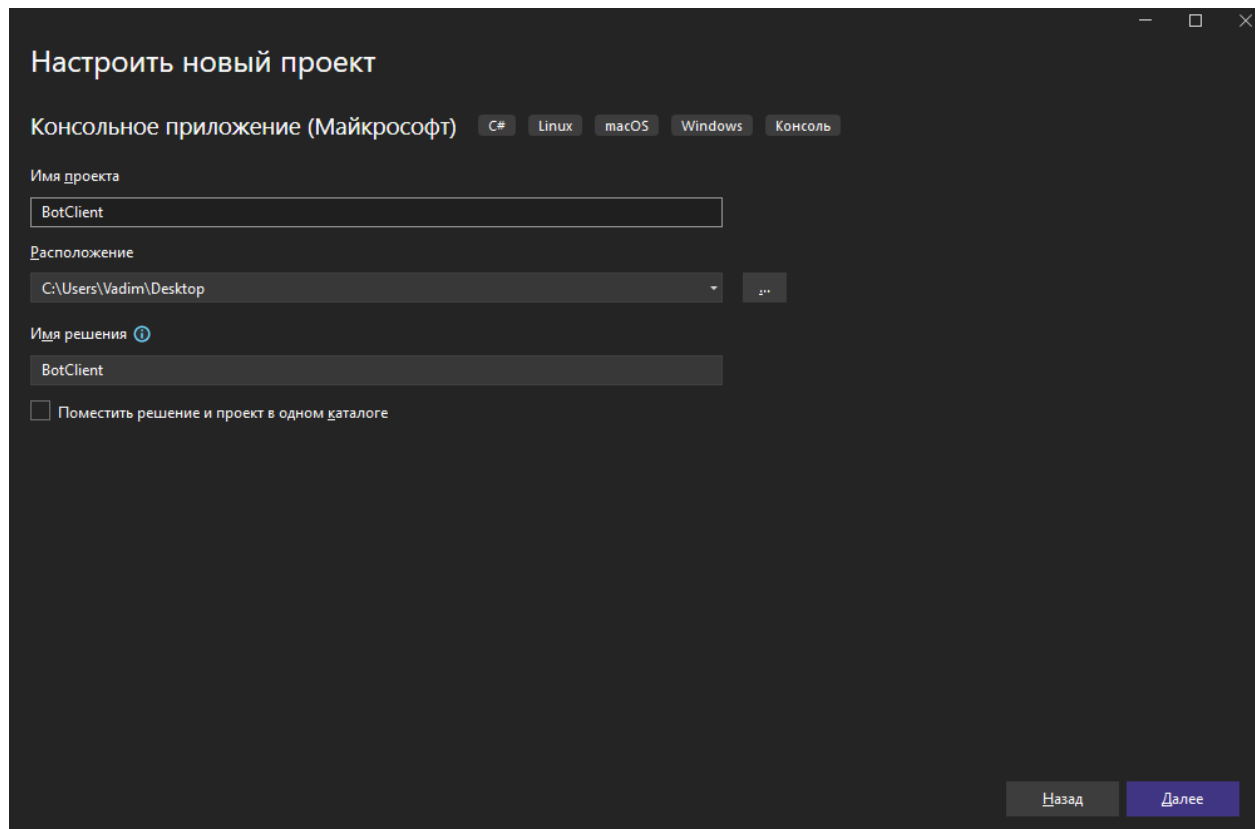
- 1) Откройте Visual Studio и перейдите во вкладку "Создание проекта"



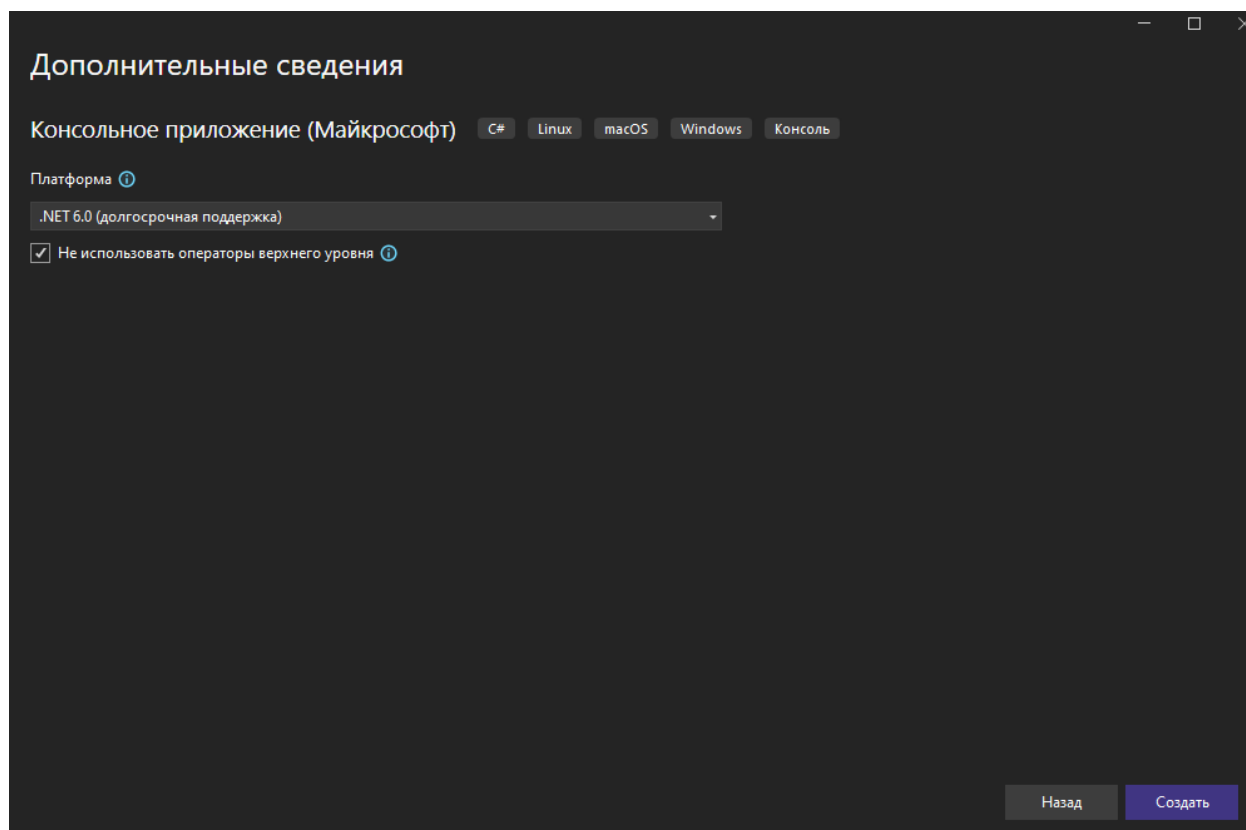
2) Выберите тип приложения “Консольное приложение (Майкрософт)”



3) Желательно проект назвать не как ConsoleApp1 ;)



5) В разделе “Дополнительные сведения” обязательно поставьте галочку “Не использовать операторы верхнего уровня”, чтобы Program.cs выглядел привычным образом.



Подключение библиотек в проект для взаимодействия работы с ботом

Если мы попытаемся найти функционал в данной платформе для работы с Telegram API, то попытка окажется провальной. Потому что данного функционала у платформы нет.

Есть два варианта решения данной проблемы:

1. Написать самому и потратить много времени
2. Использовать готовые решения

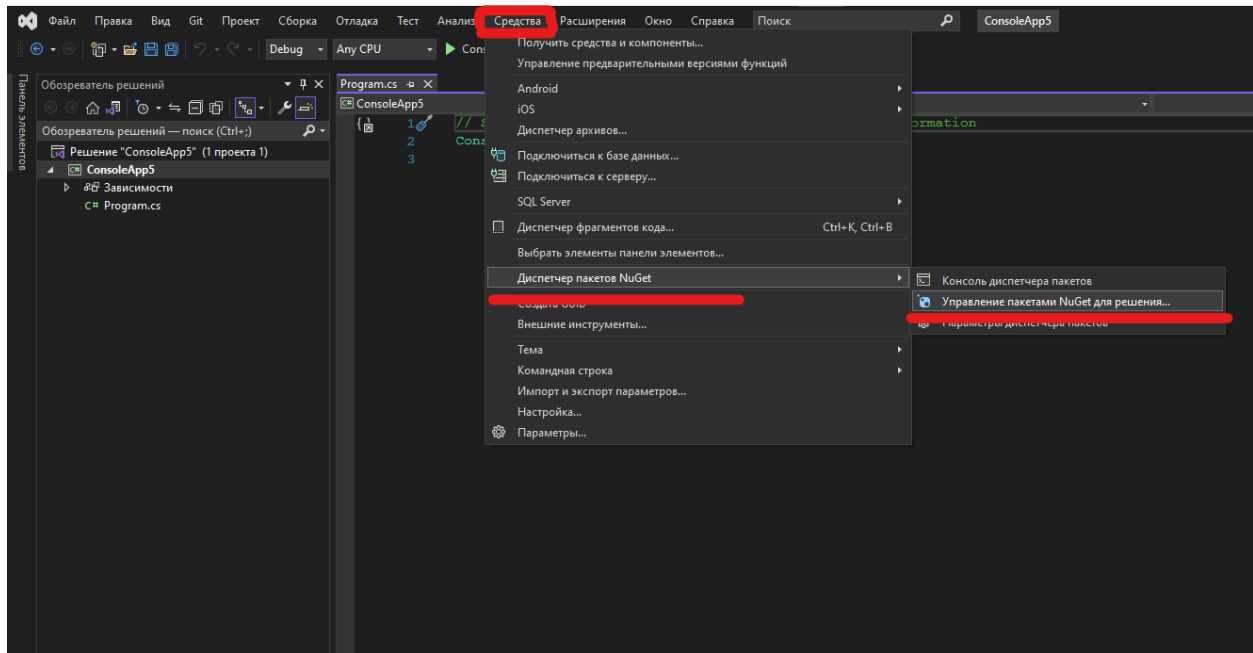
Воспользуемся вторым вариантом и задействуем библиотеку под названием Telegram.Bot.

Подключение пакета Telegram.Bot

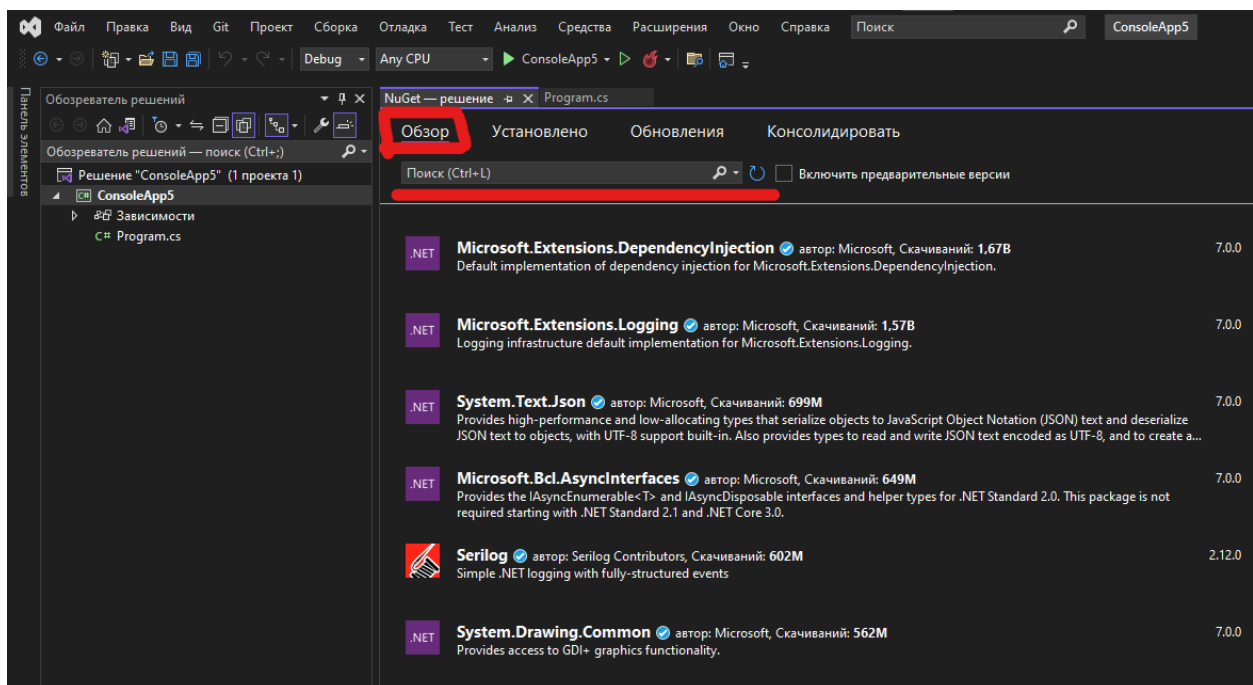
Для этого потребуется открыть менеджер пакетов Nuget (см. скриншот ниже)

1. Нажать на кнопку “Средства”

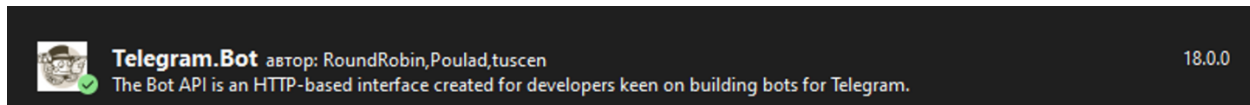
2. В выпадающем окне навести на раздел “Диспетчер пакетов NuGet”
3. В выпадающем окне нажать на “Управление пакетами NuGet для решения...”



В открывшемся окошке выбираете раздел “Обзор” и в поле поиска указываете название данной библиотеки: в данном случае это **Telegram.Bot**

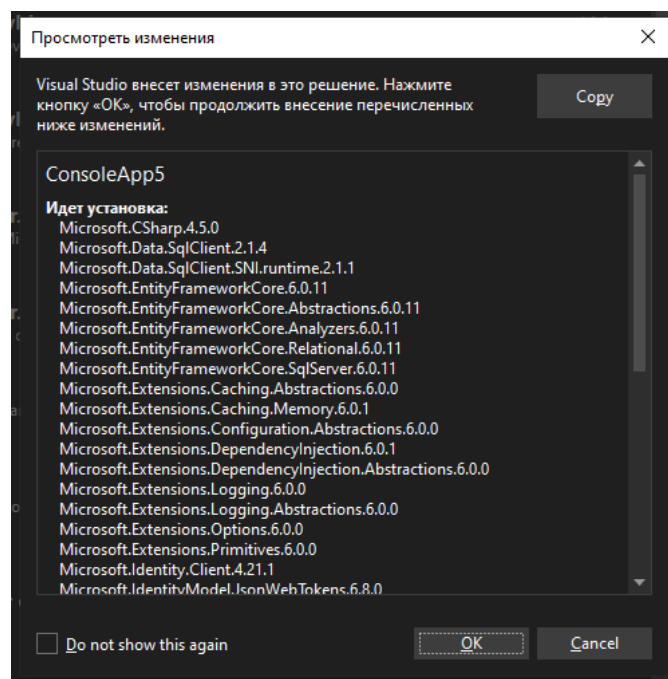


В результате поиска, выбираете библиотеку **Telegram.Bot** и устанавливаете его.



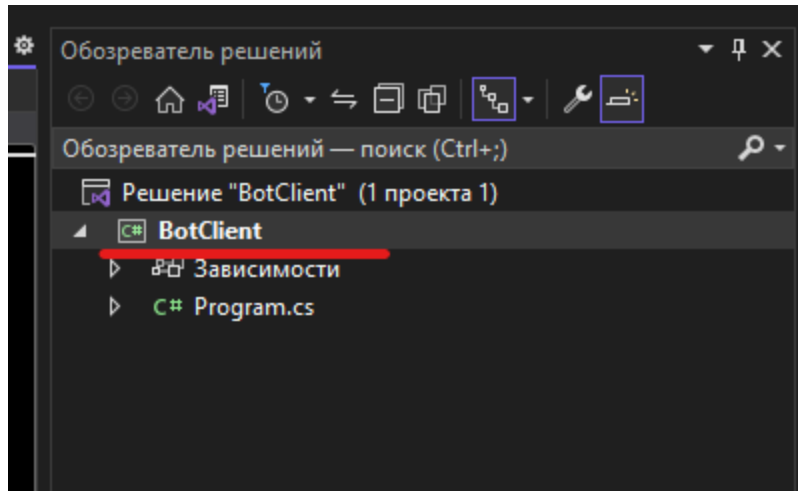
В правой области устанавливаете галочку у проекта, куда будет ставиться данный пакет и нажимаете на кнопку “Установить”.

Как только вы нажимаете на кнопку “Установить”, начинается процесс установки пакета - во всех проявляющихся окнах нажимаете кнопку “Ок” или “I Assent”

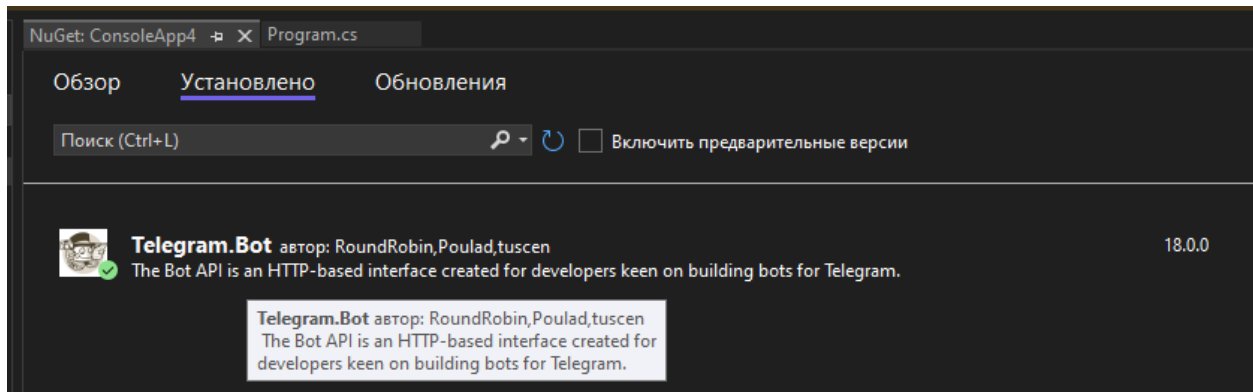


Пример окна, с которым вы столкнетесь

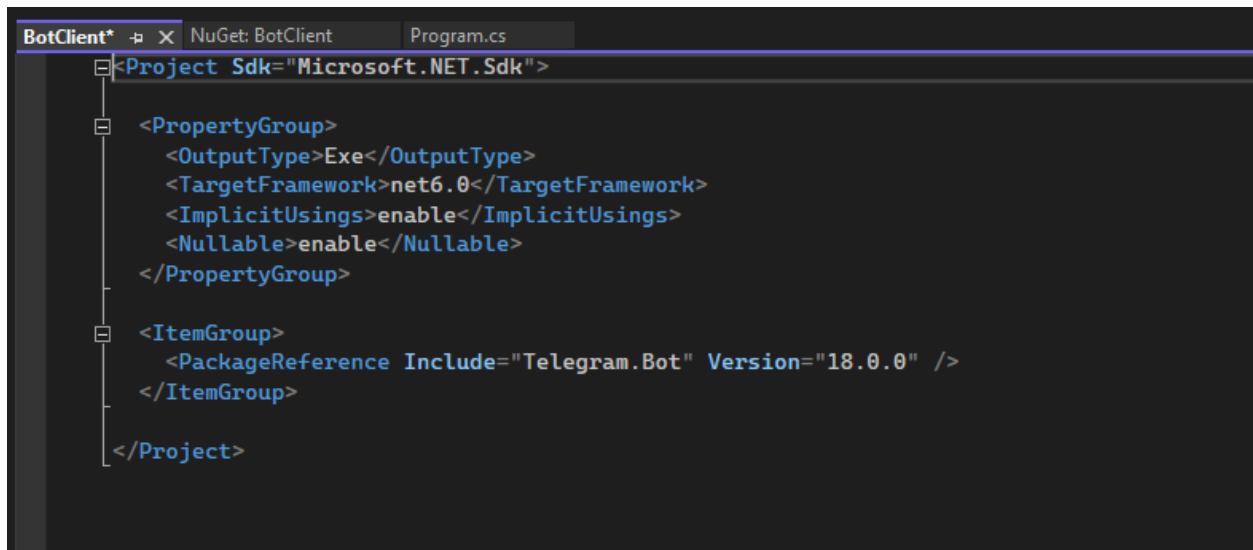
Переходим в окно проекта, чтобы убедиться, что все пакеты установлены. Для этого нужно двойным щелчком мыши нажать на проект.



Таким образом в разделе “Установлено” должен лежать данный пакет.

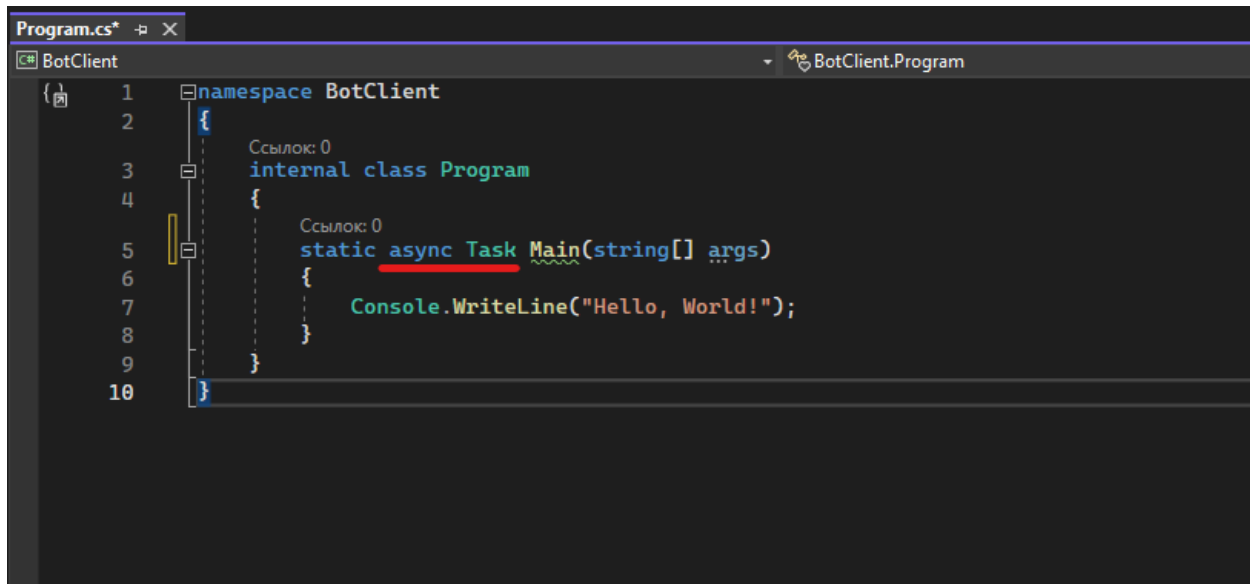


В результате конфигурация проекта должна иметь следующий вид



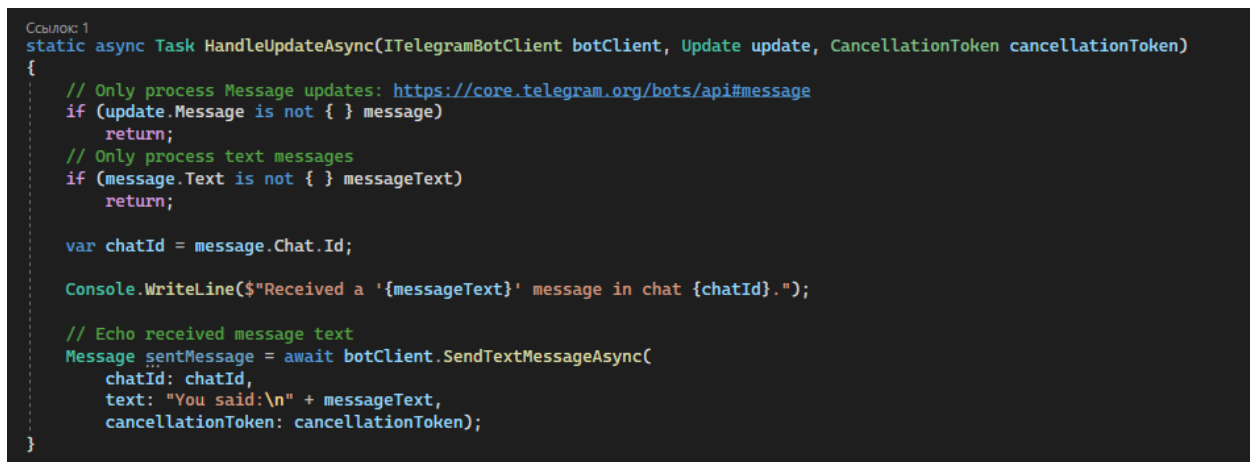
Внедрение логики для обработки сообщений пользователя

Для начала изменим стартовый метод Main и сделаем его асинхронным.



```
Program.cs*  X
BotClient
1 namespace BotClient
2 {
3     Ссылка: 0
4     internal class Program
5     {
6         Ссылка: 0
7         static async Task Main(string[] args)
8         {
9             Console.WriteLine("Hello, World!");
10        }
11    }
12 }
```

Создадим метод **HandleUpdateAsync** в котором будет описана логика обработки различных событий. В данном случае ограничим этот метод и будем использовать его только в контексте получаемых сообщений от пользователя



```
Ссылка: 1
static async Task HandleUpdateAsync(ITelegramBotClient botClient, Update update, CancellationToken cancellationToken)
{
    // Only process Message updates: https://core.telegram.org/bots/api#message
    if (update.Message is not { } message)
        return;
    // Only process text messages
    if (message.Text is not { } messageText)
        return;

    var chatId = message.Chat.Id;

    Console.WriteLine($"Received a '{messageText}' message in chat {chatId}.");

    // Echo received message text
    Message sentMessage = await botClient.SendTextMessageAsync(
        chatId: chatId,
        text: "You said:\n" + messageText,
        cancellationToken: cancellationToken);
}
```

Для того, чтобы бот реагировал на слова - напишем следующий фрагмент кода

```

Ссылка 1
static async Task HandleUpdateAsync(ITelegramBotClient botClient, Update update, CancellationToken cancellationToken)
{
    // Only process Message updates: https://core.telegram.org/bots/api#message
    if (update.Message is not { } message)
        return;
    // Only process text messages
    if (message.Text is not { } messageText)
        return;

    var chatId = message.Chat.Id;

    Console.WriteLine($"Received a '{messageText}' message in chat {chatId}.");

    // Echo received message text
    Message sentMessage = await botClient.SendTextMessageAsync(
        chatId: chatId,
        text: "You said:\n" + messageText,
        cancellationToken: cancellationToken);

    if (message.Text == "Проверка")
    {
        await botClient.SendTextMessageAsync(
            chatId: chatId,
            text: "Проверка: ОК!",
            cancellationToken: cancellationToken);
    }
}

```

Теперь при отправке сообщения “Проверка” - бот будет выводить сообщение “Проверка: ОК!”

Обработка исключений

Так как что-то может пойти не так при работе нашего бота - добавим обработку исключений. Для этого создадим метод `HandlePollingErrorAsync`, который будет отображать информацию в консоли в случае различных неполадок.

```

Ссылка 1
static Task HandlePollingErrorAsync(ITelegramBotClient botClient, Exception exception, CancellationToken cancellationToken)
{
    var ErrorMessage = exception switch
    {
        ApiRequestException apiRequestException
            => $"Telegram API Error:\n[{apiRequestException.ErrorCode}]\n{apiRequestException.Message}",
        _ => exception.ToString()
    };

    Console.WriteLine(ErrorMessage);
    return Task.CompletedTask;
}

```

Инициализация бота в методе Main

Для того, чтобы бот запустился и работал - опишите метод `Main` следующим образом:

```

Ссылка: 0
static async Task Main(string[] args)
{
    Console.WriteLine("Hello, World!");

    var botClient = new TelegramBotClient("{YOUR_ACCESS_TOKEN_HERE}");

    using CancellationTokenSource cts = new();

    // StartReceiving does not block the caller thread. Receiving is done on the ThreadPool.
    ReceiverOptions receiverOptions = new()
    {
        AllowedUpdates = Array.Empty<UpdateType>() // receive all update types
    };

    botClient.StartReceiving(
        updateHandler: HandleUpdateAsync,
        pollingErrorHandler: HandlePollingErrorAsync,
        receiverOptions: receiverOptions,
        cancellation_token: cts.Token
    );

    var me = await botClient.GetMeAsync();

    Console.WriteLine($"Start listening for @{me.Username}");
    Console.ReadLine();

    // Send cancellation request to stop bot
    cts.Cancel();
}

```



Параметр {YOUR_ACCESS_TOKEN_HERE} необходимо заменить на токен, который выдал непосредственно BotFather

Use this token to access the HTTP API:

5727341744:AAEXtJQLxAiEeE751QTBH
nY_iUQbmG2C7VQ

Keep your token secure and store it

В случае, если в консоли будет отображено сообщение “Start listening for ...” - значит бот успешно инициализировался.

Использование HttpClient для работы с API интернет-магазина.

В основном для общения с Backend-приложениями используют протокол HTTP, поэтому, чтобы общаться с нашим API интернет-магазина, воспользуемся классом HttpClient.



Используйте документацию по работе с HttpClient, например:
<https://zetcode.com/csharp/httpclient/>

Давайте попробуем получить следующие данные из API

Интернет Магазин API v1 OAS3

<https://localhost:7010/swagger/v1/swagger.json>

Описание ASP.NET Core Web API

[Пример контакта - Website](#)

[Пример лицензии](#)

Goods

GET /api/Goods Получение списка всех пользователей БД

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7010/api/Goods' \
  -H 'accept: */*'
```

Request URL

```
https://localhost:7010/api/Goods
```

Server response

Code Details

200

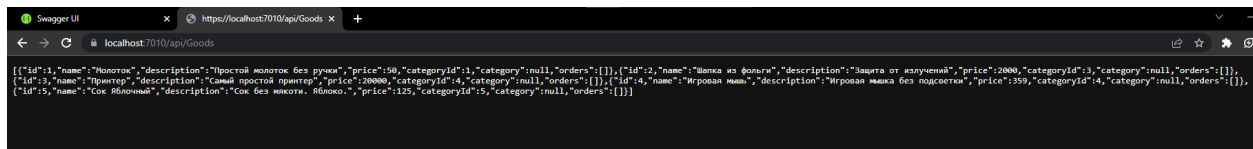
Response body

```
{
  "id": 1,
  "name": "Молоток",
  "description": "Простой молоток без ручки",
  "price": 50,
  "categoryID": 1,
  "category": null,
  "orders": []
},
{
  "id": 2,
  "name": "Шапка из фольги",
  "description": "Защита от излучений",
  "price": 2000,
  "categoryID": 2,
  "category": null,
  "orders": []
},
{
  "id": 3,
```

Перейдите по следующей ссылке, чтобы посмотреть в каком виде будут получены данные из API



Так как это GET-запрос - данные в браузере будут отображены. Проверить методы, которые не являются GET-методом не получится.



HttpClient на примере получения товаров

Для того, чтобы обратиться к API - создайте объект класса HttpClient.

Чтобы вызвать GET-метод из API - используйте метод GetAsync. Внутри метода укажите ссылку, по которой будет вызываться GET-запрос.

Для вызова POST, PUT, DELETE - используйте соответствующие методы.

```
HttpClient client = new HttpClient();  
var result = await client.GetAsync("https://localhost:7010/api/Goods");
```

Посмотрим на ответ API из метода result

```
HttpClient client = new HttpClient();  
var result = await client.GetAsync("https://localhost:7010/api/Goods");  
  
Console.WriteLine(result);
```

В консоли будет отображен следующий ответ:

```
Консоль отладки Microsoft Visual Studio
Hello, World!
StatusCode: 200, ReasonPhrase: 'OK', Version: 1.1, Content: System.Net.Http.HttpConnectionResponseContent, Headers:
{
  Date: Fri, 21 Apr 2023 08:01:57 GMT
  Server: Kestrel
  Transfer-Encoding: chunked
  Content-Type: application/json; charset=utf-8
}

C:\Users\Vadim\Desktop\BotClient\BotClient\bin\Debug\net6.0\BotClient.exe (процесс 29672) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" -> "Параметры" -> "Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно: _
```

Чтобы получить именно данные - для переменной result сделаем следующие действия:

```
HttpClient client = new HttpClient();
var result = await client.GetAsync("https://localhost:7010/api/Goods");

Console.WriteLine(result);

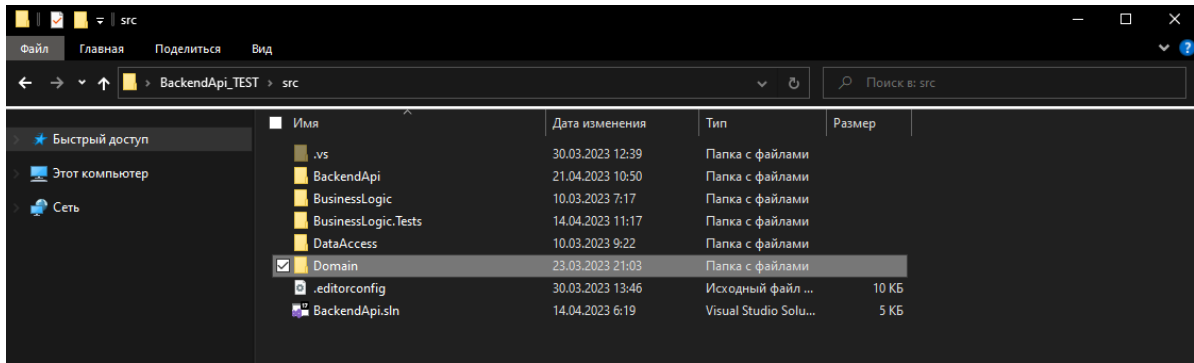
var test = await result.Content.ReadAsStringAsync();
Console.WriteLine(test);
```

Теперь в консоли можно увидеть следующий ответ:

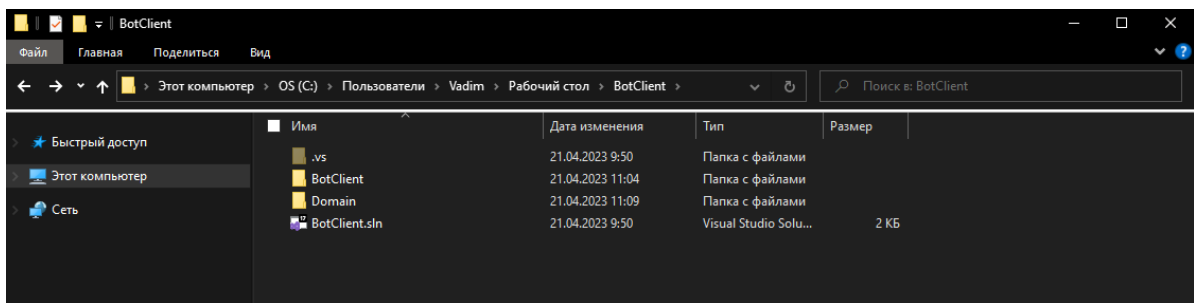
```
Консоль отладки Microsoft Visual Studio
Hello, World!
StatusCode: 200, ReasonPhrase: 'OK', Version: 1.1, Content: System.Net.Http.HttpConnectionResponseContent, Headers:
{
  Date: Fri, 21 Apr 2023 08:04:41 GMT
  Server: Kestrel
  Transfer-Encoding: chunked
  Content-Type: application/json; charset=utf-8
}
[{"id":1,"name":"Молоток","description":"Простой молоток без ручки","price":50,"categoryId":1,"category":null,"orders":[]}, {"id":2,"name":"Шапка из фольги","description":"Защита от излучений","price":2000,"categoryId":3,"category":null,"orders":[]}, {"id":3,"name":"Принтер","description":"Самый простой принтер","price":20000,"categoryId":4,"category":null,"orders":[]}, {"id":4,"name":"Игровая мышь","description":"Игровая мышка без подсветки","price":359,"categoryId":4,"category":null,"orders":[]}, {"id":5,"name":"Сок Яблочный","description":"Сок без мякоти. Яблоко.","price":125,"categoryId":5,"category":null,"orders":[]}]
```

Так как данные представлены в виде текста - отфильтровать их не получится. Для этого сделаем следующее:

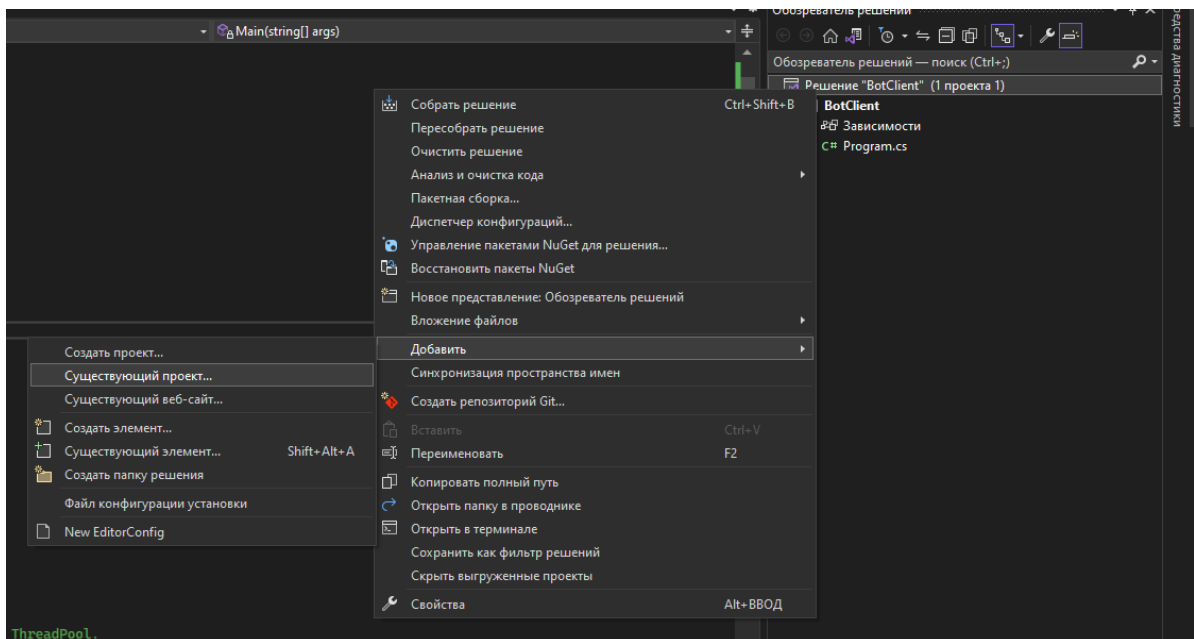
1. Скопируем папку Domain из нашего проекта API



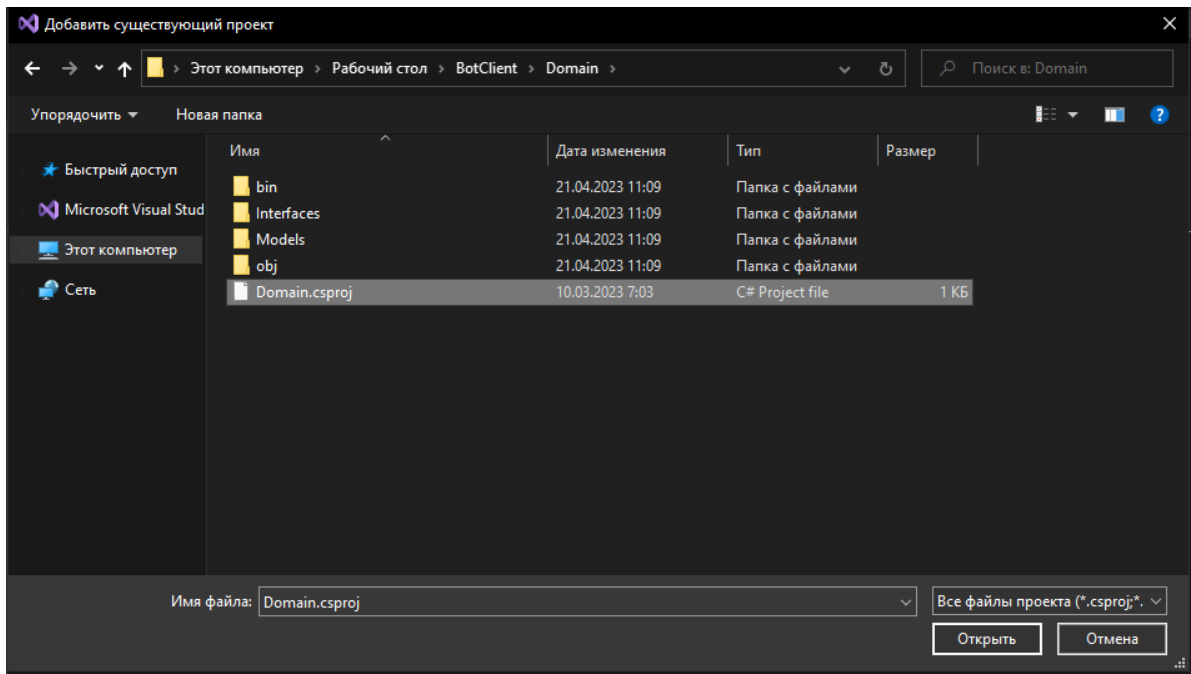
2. И разместим в проекте телеграм бота



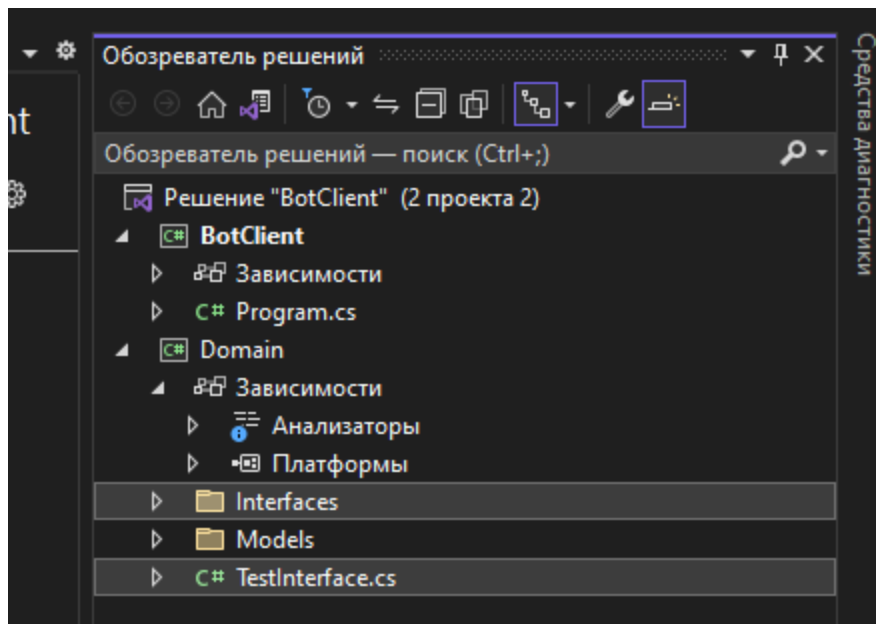
3. Вызовем окно для подключения существующего проекта



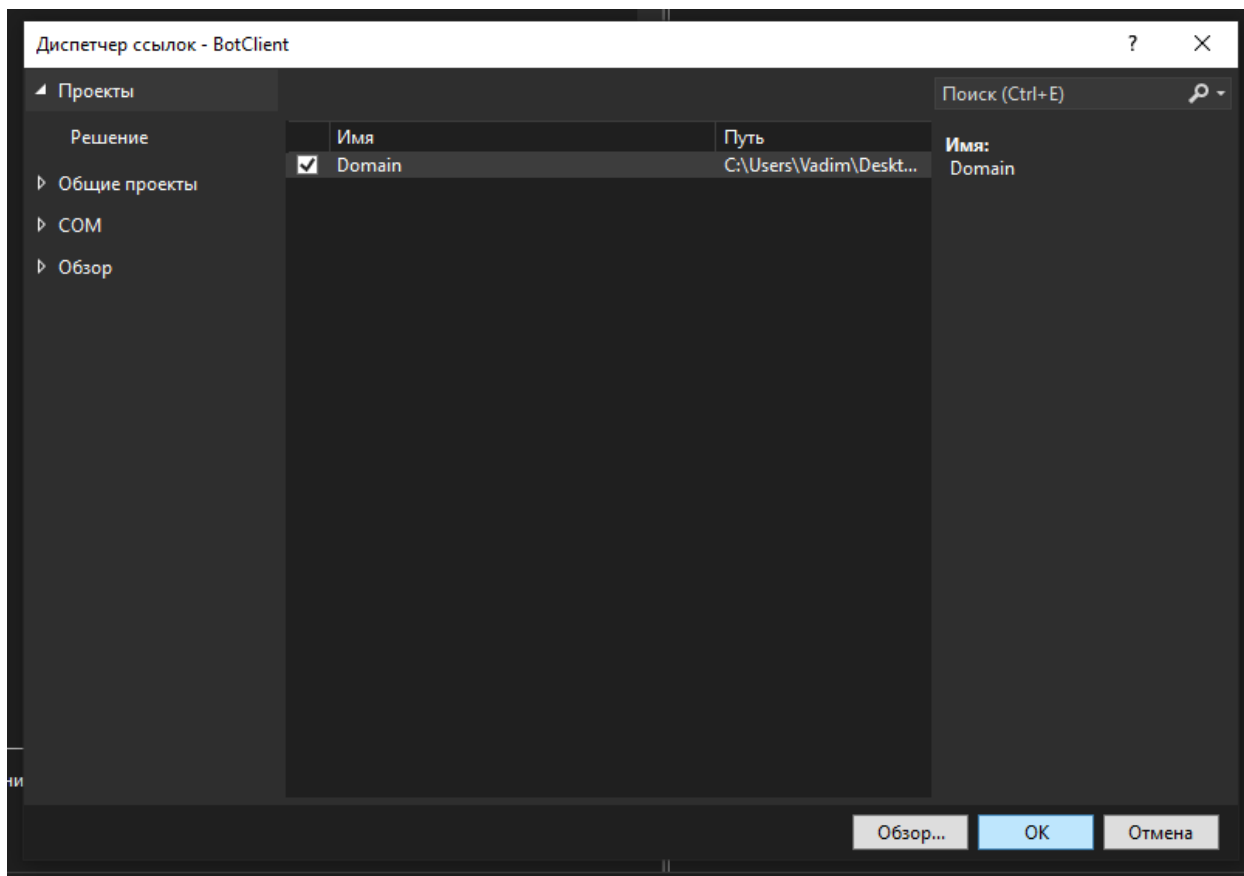
4. В окне выбора выберите файл Domain.cproj



5. Удалите лишние папки - для клиентского приложения они не понадобятся



6. Добавьте ссылку на проект



Теперь, можно быстро преобразовывать полученные данные. Для этого воспользуемся библиотекой Newtonsoft.Json.

```
var test = await result.Content.ReadAsStringAsync();
Console.WriteLine(test);

Good[] goods = JsonConvert.DeserializeObject<Good[]>(test);
```

Таким образом данные формата Json можно быстро преобразовать в необходимый формат. В данном случае в массив продуктов (Good)

```
Good[] goods = JsonConvert.DeserializeObject<Good[]>(test);

foreach(var good in goods)
{
    Console.WriteLine(good.Id + " " + good.Name + " " + good.Price);
}
```

Таким образом данные можно с легкостью фильтровать:

```
Консоль отладки Microsoft Visual Studio
Hello, World!
StatusCode: 200, ReasonPhrase: 'OK', Version: 1.1, Content: System.Net.Http.HttpConnectionResponseContent, Headers:
{
  Date: Fri, 21 Apr 2023 09:40:24 GMT
  Server: Kestrel
  Transfer-Encoding: chunked
  Content-Type: application/json; charset=utf-8
}
[{"id":1,"name":"Молоток","description":"Простой молоток без ручки","price":50,"categoryId":1,"category":null,"orders":[
]],{"id":2,"name":"Шапка из фольги","description":"Защита от излучений","price":2000,"categoryId":3,"category":null,"ord
ers":[]},{id":3,"name":"Принтер","description":"Самый простой принтер","price":20000,"categoryId":4,"category":null,"or
ders":[]},{id":4,"name":"Игровая мышь","description":"Игровая мышка без подсветки","price":359,"categoryId":4,"category
":null,"orders":[]},{id":5,"name":"Сок Яблочный","description":"Сок без мякоти. Яблоко.","price":125,"categoryId":5,"ca
tegory":null,"orders":[]}]
1 Молоток 50
2 Шапка из фольги 2000
3 Принтер 20000
4 Игровая мышь 359
5 Сок Яблочный 125

C:\Users\Vadim\Desktop\BotClient\BotClient\bin\Debug\net6.0\BotClient.exe (процесс 16356) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

Работа с документацией

Ваша задача ознакомиться с документацией для Telegram.Bot и находить примеры кода, которые позволят выполнить задания.

Документация пакета расположена по данной ссылке:

<https://telegrambots.github.io/book/1/example-bot.html>

Выполнение заданий

1) На основе документации сформируйте следующие ответы:

1. Текстовый ответ (Например на ваше сообщение “Привет” он бы ответил:
“Здравствуй, Олег”)
2. Ответ в виде картинки (Например на сообщение “Картинка” он бы вам прислал картинку в ответ)
3. Ответ в виде видео
4. Ответ в виде стикера
5. Ответ в котором бы содержались ещё и кнопки

2) Реализуйте функционал (При помощи кнопок) для работы с товарами, такие как: Отображение категорий товаров, Просмотр товаров по категории. Например:

