

## Лабораторная работа №11

### Разработка Telegram бота

**Цель работы:** получение навыков разработки ботов для мессенджеров на языке C#.

#### Теоретическая часть

Ознакомиться с Telegram API можно по ссылке:

<https://core.telegram.org/api>

Рассмотрим пример разработки простейшего телеграм-бота.

#### Шаг 1

Необходимо создать консольное приложение. В консоль будем выводить все взаимодействия пользователей с ботом.

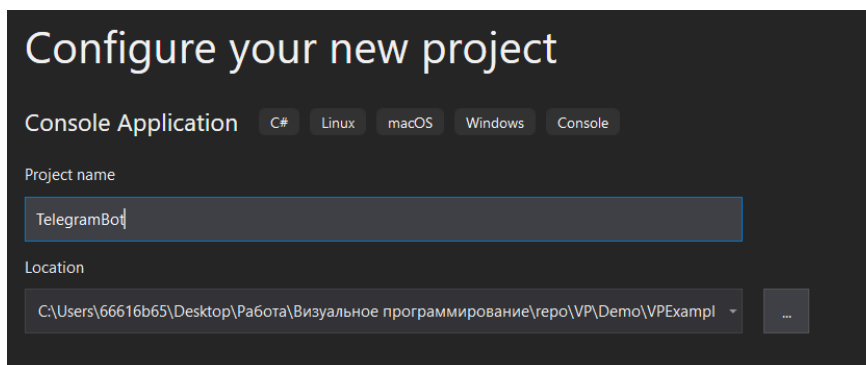


Рисунок 1 – Создание приложения

#### Шаг 2

Через диспетчер пакетов **NuGet** нужно подключить к проекту библиотеки:

- **Telegram.Bot** — для работы с ботом;
- **Newtonsoft.Json** — по желанию, нужна для удобного вывода действий пользователя в консоль.

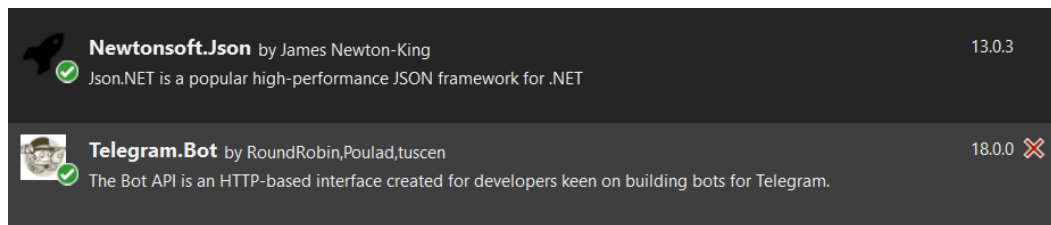


Рисунок 2 – Установленные библиотеки

### Шаг 3

В классе **Program** прописать:

```
using System;  
using System.Threading;  
using System.Threading.Tasks;  
using Telegram.Bot;  
using Telegram.Bot.Polling;  
using Telegram.Bot.Types;  
using Telegram.Bot.Exceptions;  
using System.IO;
```

### Шаг 4

Для создания самого бота нужно открыть Telegram, найти бота BotFather и вызвать команду /newbot. Для бота нужно задать его название и имя пользователя. Тогда бот пришлёт API токен для созданного вами бота.

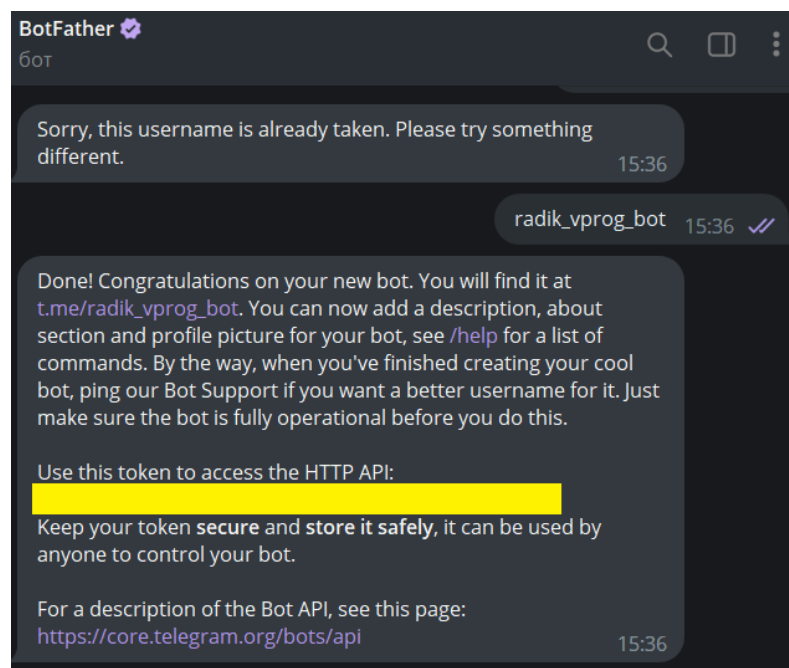


Рисунок 3 – Получение токена

## Шаг 5

В классе Program объявляем переменную, через которую будем работать с ботом:

```
static ITelegramBotClient bot = new TelegramBotClient("сюда вставляем  
токен");
```

## Шаг 6

Для начала разработаем бота, который будет выполнять самое простое действие – отвечать на команду пользователя. Для этого добавим метод, который будет вызываться в ответ на пользовательские действия:

```
public static async Task HandleUpdateAsync(ITelegramBotClient botClient, Update update,
CancellationToken cancellationToken)
{
    // Выводим в консоль действия пользователя
    Console.WriteLine(Newtonsoft.Json.JsonConvert.SerializeObject(update));

    // Проверяем, что пришло сообщение
    if (update.Type == Telegram.Bot.Types.Enums.UpdateType.Message)
    {
        var message = update.Message;
        // Проверяем, что это конкретная команда
        if (message.Text.ToLower() == "/start")
        {
            // Действие бота в ответ на команду
            await botClient.SendTextMessageAsync(message.Chat, "Hello!");
            return;
        }

        // Если это любая другая команда, то тоже что-то ответим
        await botClient.SendTextMessageAsync(message.Chat, "Idk this command");
    }
}
```

Этот метод принимает бота и действие пользователя. Рассмотрим подробнее тело метода.

```
Console.WriteLine(Newtonsoft.Json.JsonConvert.SerializeObject(update));
```

Эта строка нужна для вывода в консоль действия пользователя. Как раз для этого мы и подключили библиотеку **Newton.Json**.

```
if (update.Type == Telegram.Bot.Types.Enums.UpdateType.Message)
```

Здесь мы проверяем тип обновления, так как на данном этапе мы работаем с текстовыми командами.

```

if (message.Text.ToLower() == "/start")
{
    // Действие бота в ответ на команду
    await botClient.SendTextMessageAsync(message.Chat, "Hello!");
    return;
}

```

Проверяем, что пришла именно нужна нам команда, и отвечаем на неё соответствующим образом.

```

await botClient.SendTextMessageAsync(message.Chat, "Idk this command");

```

Дополнительное действие на случай, если пользователь прислал несуществующую команду.

## Шаг 7

Также необходимо предусмотреть метод для обработки ошибок. В нашем случае информация об исключении будет просто выводиться в консоль.

```

public static async Task HandleErrorAsync(ITelegramBotClient botClient, Exception
exception, CancellationToken cancellationToken)
{
    Console.WriteLine(Newtonsoft.Json.JsonConvert.SerializeObject(exception));
}

```

## Шаг 8

Теперь нужно запустить бота. Следующий код будет прописан в методе **Main** класса **Program**.

```

Console.WriteLine("Started bot " + bot.GetMeAsync().Result.FirstName);

var cts = new CancellationTokenSource();
var cancellationToken = cts.Token;
var receiverOptions = new ReceiverOptions
{
    AllowedUpdates = { }, // Принимаем все обновления
};

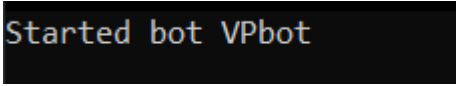
bot.StartReceiving(
    HandleUpdateAsync,
    HandleErrorAsync,
    receiverOptions,
    cancellationToken
);

Console.ReadLine();

```

## Шаг 9

Попробуем запустить бота. В консоли выводится информация о том, что бот запущен.



```
Started bot VPbot
```

Рисунок 4 – Запуск бота

Попробуем написать боту команду.

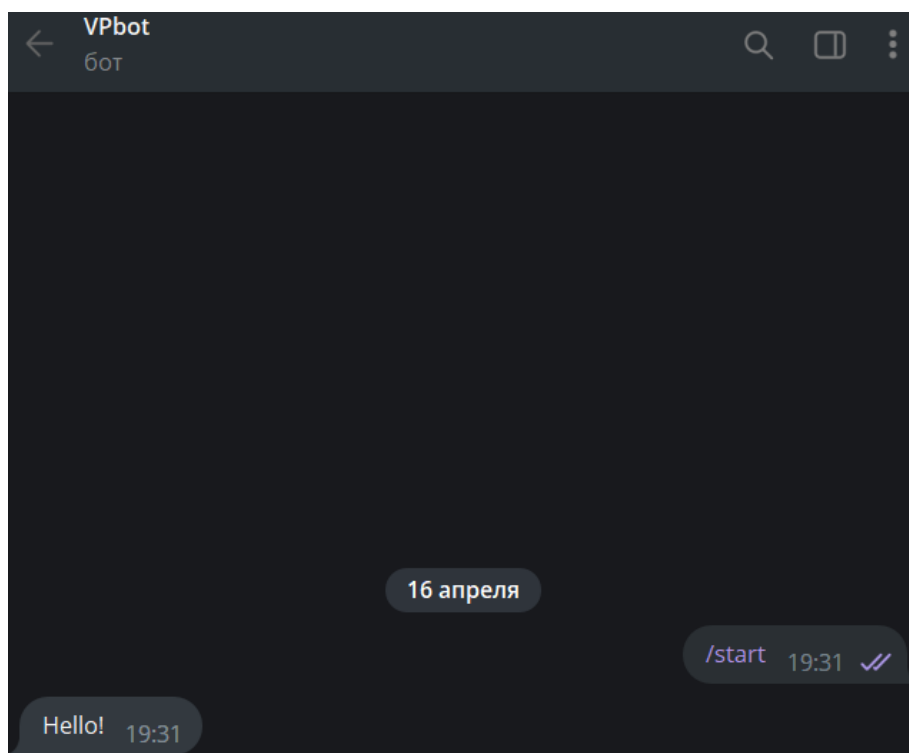
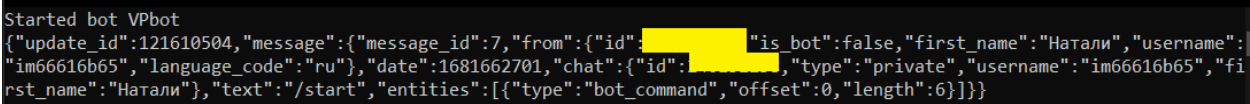


Рисунок 5 – Команда

Тем временем в консоли вывелась информация о пользователе и его действии.



```
Started bot VPbot
{"update_id":121610504,"message":{"message_id":7,"from":{"id":,"is_bot":false,"first_name":"Натали","username":"im66616b65","language_code":"ru"},"date":1681662701,"chat":{"id":,"type":"private","username":"im66616b65","first_name":"Натали"},"text":"/start","entities":[{"type":"bot_command","offset":0,"length":6}]}}
```

Рисунок 6 – Информация в консоли

## Шаг 10

Попробуем разнообразить действия бота. Например, по команде /picture он будет присылать случайную картинку из локального хранилища.

Тогда в метод `HandleUpdateAsync` добавим обработку нескольких команд.

```
if (message.Text.ToLower() == "/start")
{
    // Действие бота в ответ на команду
    await botClient.SendTextMessageAsync(message.Chat, "Hello! Use command /picture to get funny pics");
    return;
}

if (message.Text.ToLower() == "/picture")
{
    // Действие бота в ответ на команду
    using (var stream = System.IO.File.Open(GetRandomPicture(), FileMode.Open))
    {
        await botClient.SendPhotoAsync(message.Chat, InputFile.FromStream(stream));
    }
    return;
}

if (message.Text.ToLower() == "/bye")
{
    // Действие бота в ответ на команду
    await botClient.SendTextMessageAsync(message.Chat, "See ya later!");
    return;
}
```

Для получения случайной картинки напишем специальный метод.

```
static string GetRandomPicture()
{
    string path = @"D:/pictures";

    var pictures = Directory.GetFiles(path);

    Random random = new Random();
    int index = random.Next(pictures.Length);

    return pictures[index];
}
```

## Шаг 11

Запустим обновленный бот.

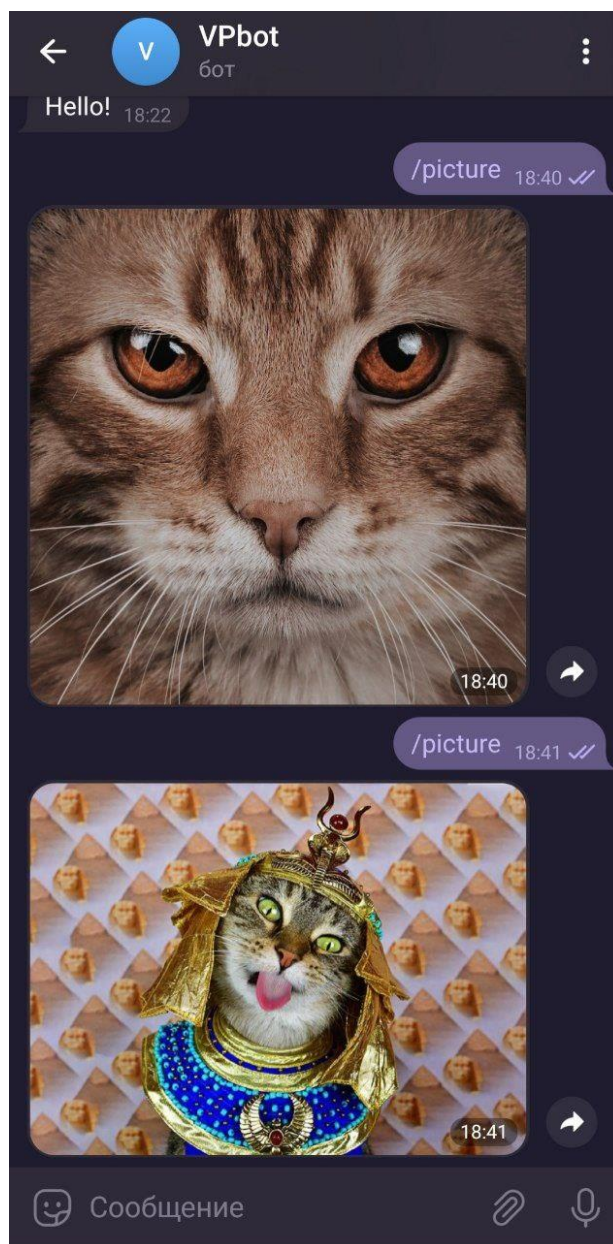


Рисунок 7 – Бот присылает картинки

## Практическая часть

Расширить функционал бота за счёт выполнения следующих действий:

- 1) Бот принимает строку типа **число операция число** и возвращает результат операции. В качестве операции могут быть сложение,

вычитание и умножение. Числа целые. Предусмотреть удаление лишних пробелов.

- 2) Бот принимает изображение и возвращает его перевернутым на 180 градусов. Для работы с изображением можно использовать класс `System.Drawing.Bitmap`.

### **Содержание отчета**

1. Титульный лист
2. Цель работы
3. Задание
4. Код программы
5. Результат выполнения