Федеральное Государственное Бюджетное Образовательное

Учреждение Высшего Образования

Вятский Государственный Университет

Факультет автоматики и вычислительной техники

Кафедра систем автоматизации управления

Отчет по практической работе

Тема: **«Разработка телеграм-бота»**

Дисциплина: «Научно исследовательская деятельность»

|  |  |
| --- | --- |
| Выполнила студентка гр. ПИпб-3301 | Краева Т.А. |
|
|
|
|  |  |
| Проверил преподаватель кафедры САУ | Земцов М.А. |

Киров 2018 г.

1. Постановка проблемы. Описание ее актуальности

Я столкнулась с тем, что в Кирове неудобная система предоставления информации о расписании городского транспорта. Заходить на сайт каждый раз неудобно, а уличные мониторы не везде установлены, да и не всегда исправны.

Перед началом работы был проведен опрос студентов Вятгу (рисунок 1), в ходе которого выяснилось, что 71,22% людей он был бы полезен. Отсюда был сделан вывод о том, что данная проблема является актуальной.



Рисунок 1 «Опрос»

1. Методы решения проблемы
2. Создание более удобного сайта, чем имеющийся

Данный метод был отклонен, т.к. он не очень удобен в плане мобильности.

1. Создание мобильного приложения

Данный метод был также отклонен, т.к. уже существует аналог, и перспектива открывать отдельное приложение не совсем интересна.

1. Создание telegram bot

Для решения этой проблемы я решила реализовать telegram bot, благодаря которому можно намного быстрее узнать информацию по автобусам и троллейбусам, не выходя из messenger-а.

1. Путь решения, который был выбран и реализован в течение этого семестра

Этапы реализации:

* Проведение опроса

В самом начале был проведен опрос, который показал, что данный бот интересен пользователям.

* Выбор средств реализации

Создавать бота я решила я среде “Microsoft Visual Studio” , на языке программирования c#, с использование базы данных Ado.Entities и дополнительных библиотек AngleSharp и Telegram.Bot.

* Выбор библиотеки для парсинга

Мною были рассмотрены 2 библиотеки: HtmlAgilityPack и AngleSharp, в своем проекте я решила использовать вторую, т.к. она мне показалась наиболее понятной и удобной.

* Создание консольного приложения

Первым делом было реализовано консольное приложение, которое включало в себя следующие классы (рисунок 2):

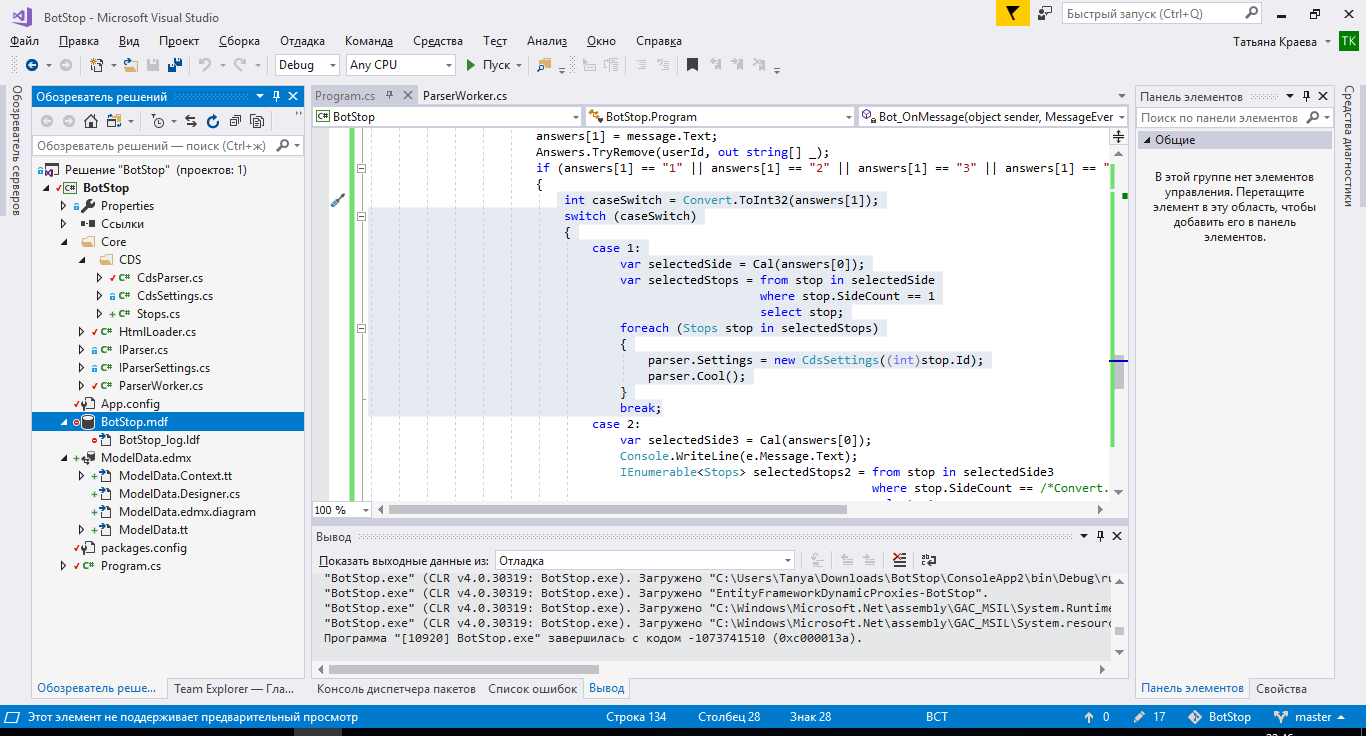


Рисунок 2 «Классы»

* Создание базы данных

Для более удобного метода хранения данных была создана база данных, куда были занесены номера остановок и их названия.

Код Program.cs представлен в листинге 1:

Листинг 1:

class Program

{

static internal ITelegramBotClient botClient;

static int a;

static void Main(string[] args)

{

botClient = new TelegramBotClient("784742481:AAFrYubmGTfNAWSjZKK99uLmoazLUQrGWoY");

var me = botClient.GetMeAsync().Result;

Console.WriteLine(

$"Hello, World! I am user {me.Id} and my name is {me.FirstName}."

);

//botClient.StartReceiving();

botClient.OnMessage += Bot\_OnMessage;

botClient.StartReceiving();

Thread.Sleep(int.MaxValue);

}

static string LoadPage(string url)

{

var result = "";

var request = (HttpWebRequest)WebRequest.Create(url);

var response = (HttpWebResponse)request.GetResponse();

if (response.StatusCode == HttpStatusCode.OK)

{

var receiveStream = response.GetResponseStream();

if (receiveStream != null)

{

StreamReader readStream;

if (response.CharacterSet == null)

readStream = new StreamReader(receiveStream);

else

readStream = new StreamReader(receiveStream, Encoding.GetEncoding(response.CharacterSet));

result = readStream.ReadToEnd();

readStream.Close();

}

response.Close();

}

return result;

}

static IEnumerable<Stops> Cal(string r)

{

BotStopEntities1 db = new BotStopEntities1(); //бд

// список остановок

List<Stops> stops = new List<Stops>();

foreach (Stop s in db.Stop)

{

stops.Add(new Stops((int)s.Id, s.StopStart.Split(new char[] { ',' }), s.StopEnd, s.SideCount));

}

var selectedSide = from stop in stops

from name in stop.StopStart

where name == r.ToLower()

select stop;

return selectedSide;

}

const long TargetChannelId = 784742481;

static ConcurrentDictionary<int, string[]> Answers = new ConcurrentDictionary<int, string[]>();

static async void Bot\_OnMessage(object sender, MessageEventArgs e)

{

//бд

BotStopEntities1 db = new BotStopEntities1(); //бд

ParserWorker<string> parser; //парсинг

parser = new ParserWorker<string>(

new CdsParser()

);

// список остановок

List<Stops> stops = new List<Stops>();

foreach (Stop s in db.Stop)

{

stops.Add(new Stops((int)s.Id, s.StopStart.Split(new char[] { ',' }), s.StopEnd, s.SideCount));

}

Message message = e.Message;

int userId = message.From.Id;

if (message.Type == MessageType.Text)

{

if (Answers.TryGetValue(userId, out string[] answers))

{

int count = 0;

if (answers[0] == null)

{

answers[0] = message.Text;

await botClient.SendTextMessageAsync(message.Chat, "Выберите остановку:");

var selectedSide2 = Cal(e.Message.Text);

//выводит стороны, какие есть

foreach (Stops stop in selectedSide2)

{

count++;

string s = $"{count}- {stop.StopEnd}";

await botClient.SendTextMessageAsync(

text: s,

chatId: e.Message.Chat

);

}

}

else if (answers[1] == null)

{

answers[1] = message.Text;

Answers.TryRemove(userId, out string[] \_);

if (answers[1] == "1" || answers[1] == "2" || answers[1] == "3" || answers[1] == "4")

{

int caseSwitch = Convert.ToInt32(answers[1]);

switch (caseSwitch)

{

case 1:

var selectedSide = Cal(answers[0]);

var selectedStops = from stop in selectedSide

where stop.SideCount == 1

select stop;

foreach (Stops stop in selectedStops)

{

parser.Settings = new CdsSettings((int)stop.Id);

parser.Cool();

}

break;

case 2:

var selectedSide3 = Cal(answers[0]);

Console.WriteLine(e.Message.Text);

IEnumerable<Stops> selectedStops2 = from stop in selectedSide3

where stop.SideCount == /\*Convert.ToInt32(e.Message.Text)\*/ 2

select stop;

foreach (Stops stop in selectedStops2)

{

parser.Settings = new CdsSettings((int)stop.Id);

Console.WriteLine(stop.Id);

parser.Cool2();

}

break;

case 3:

var selectedSide4 = Cal(answers[0]);

Console.WriteLine("Case 3");

Console.WriteLine(e.Message.Text);

selectedStops = from stop in selectedSide4

where stop.SideCount == Convert.ToInt32(e.Message.Text)

select stop;

foreach (Stops stop in selectedStops)

{

parser.Settings = new CdsSettings((int)stop.Id);

Console.WriteLine(stop.Id);

parser.Cool3();

}

break;

}

}

}

}

else

{

Answers.TryAdd(userId, new string[2]);

await botClient.SendTextMessageAsync(message.Chat, "Введите название остановки!");

}

}

}

}

}

* Реализация телеграм бота

Для реализации бота я использовала библиотеку Telegram.Bot для языка c#. Перед использованием я изучила документацию, разработанную к этой библиотеке, предоставленной на сайте github.com.

Первым делом я добавила в телеграме бота BotFather и создала своего бота (рисунок 3):

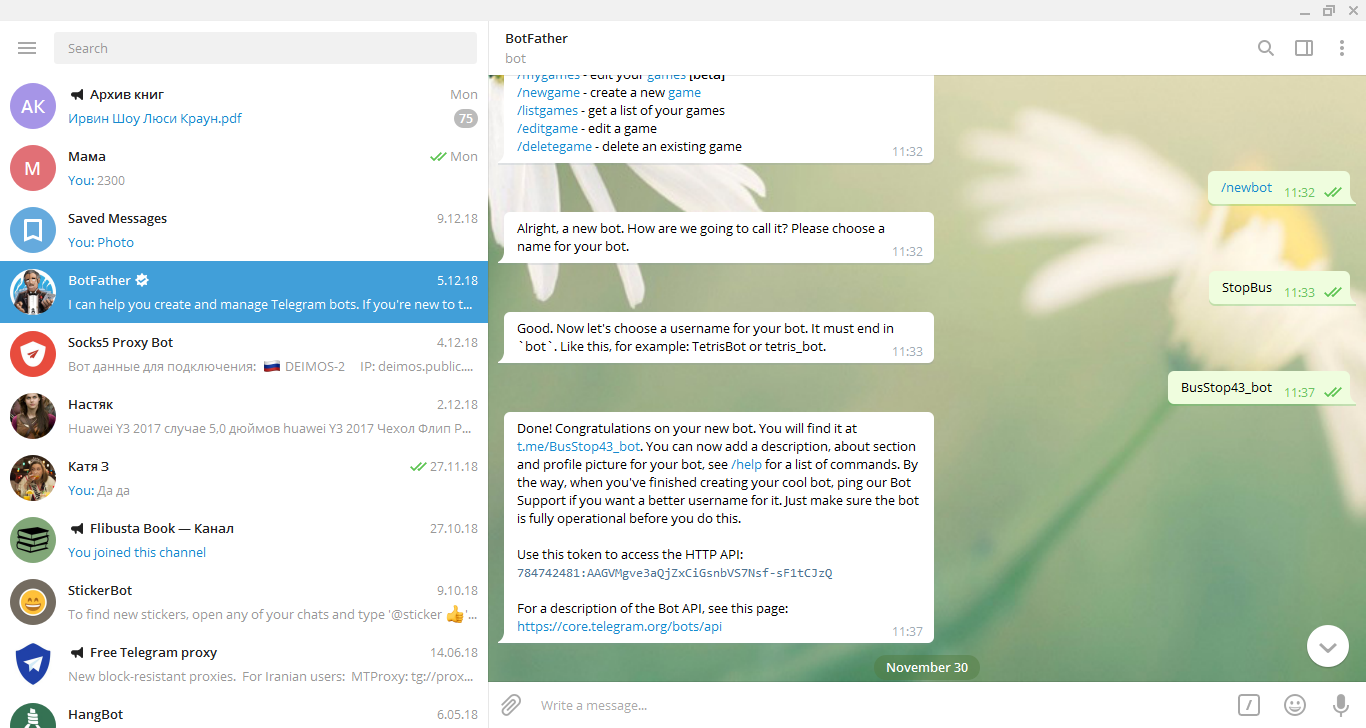


Рисунок 3 “Получение токена”

Затем, я приступила к написанию кода, представленного в листинг 1.

1. Итог

В результате был реализован бот, который показывает расписание автобусов и троллейбусов на необходимых остановках (рисунок 4). К сожалению, бот работает не совсем корректно, а именно не всегда с первого раза отвечает на запрос пользователя, пытаясь решить эту проблему, я выяснила что он почему-то не всегда обращается к необходимому методу, почему так происходит – неизвестно. Реализуя этот проект, я узнала много нового как о самом языке c#, парсинге сайтов, так и о реализации непосредственно самого бота.

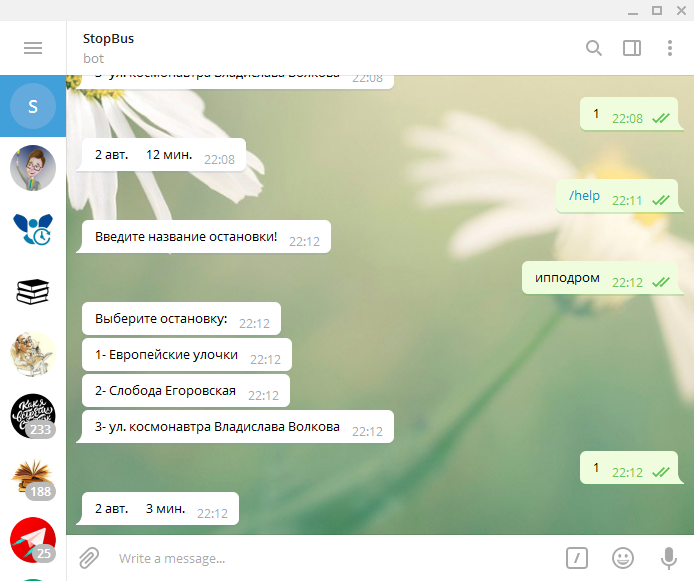


Рисунок 4 «Работа бота»

(На рисунке один автобус, т.к. запрос делался в 22:12)