СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В наше время все больше набирают популярность различные социальные сети. На данный момент из-за своей популярности и простоты использования в них происходит не только общение, но и торговля, обмен файлами и многое другое. Для облегчения таких процессов люди и начали создавать различных чат-ботов.

Чат-бот – это программа, которая имитирует реальный разговор с пользователем. Чат-боты позволяют общаться с помощью текстовых или аудио сообщений на сайтах, в мессенджерах, мобильных приложениях или по телефону.

Преимущества чат-ботов:

– эффективное взаимодействие с клиентами;
– экономность;
– легкость в эксплуатации;
– охватывают больше клиентов;
– обеспечивают обслуживание 24 на 7.

Объект исследования – социальная сеть.

Предмет исследования – чат-бот.

Цель – разработать чат-бот в социальной сети «Вконтакте».

Задачи работы:

– изучить особенности разработки чат-ботов;
– разработать чат-бота;
– тестирование программного средства.

Курсовой проект содержит 26 рисунков, 3 таблицы, 36 страниц.

В «Введении» указана актуальность, объект, предмет исследования и задачи курсового проекта.

В «Основной части» проведен анализ предмета исследования и объекта исследования. Отображены ключевые этапы проектирования внутреннего функционала программы. Показаны и рассказаны ключевые моменты тестирования программ.

В «Заключении» подведены итоги проделанной работы, выделена актуальность разработки и варианты ее дальнейшего использования.

ОСНОВНАЯ ЧАСТЬ

1 Аналитическая часть

1.1 Описание предметной области

Чат-бот – это программа, работающая внутри мессенджера. Такая программа способна отвечать на вопросы, а также самостоятельно задавать их. Чат-боты используются в разных сферах для решения типовых задач.

Создание чат-бота - это не только следование последним тенденциям. Прежде всего, речь идет о сокращении затрат, увеличении конверсии и улучшении качества обслуживания клиентов. Сотни компаний делают все возможное, чтобы достичь вышеупомянутых целей. И для их достижения необходимы боты как первостепенный инструмент.

Чат-боты могут использоваться в службах поддержки, помогая решить простые вопросы, например, такие как смена пароля.

Чат-боты можно использовать для поиска информации. Например, прогноз погоды, афиша мероприятий. В мессенджере Telegram можно найти десятки тысяч ботов, способных рассказать о погоде или помочь выбрать подарок.

Сфера путешествий стали одними из самых первых использовать чат-боты. Чат-бот может предложить направления/рейсы/рестораны – на основании поисковых запросов и предпочтений пользователя. После покупки программа обеспечивает клиентскую поддержку, предоставляя ответы на часто задаваемые вопросы.

Чат-боты помогают работодателям и соискателям в процессе поиска работы и подбора кадров. Летом 2016 года агентство по поиску работы FirstJob выпустило чат-бота Мия, который не только предлагает подходящие вакансии, но и проводит с кандидатом на вакансию собеседование. В России собеседования проводил чат-бот от компании Superjob для компании «Связной».

Маркетолог Эстер Краудфорд создала чат-бота EtherBot, который рассказывает представителям hr-департаментов о карьере хозяйки, её способностях и хобби.

Чат-боты можно использовать как персональных помощников. Изобретатель хештега Крис Мессина использует бота-помощника, MessinaBot, который отвечает всем тем, кто пишет ему в Facebook.

Чат-боты могут быть относительно простыми программами, основанными на правилах, или даже могут использовать искусственный интеллект (ИИ), который делает их гораздо более изощренными, но пригодными для широкого круга задач.

По типу назначения можно выделить несколько видов:

- боты помощники;
- боты с искусственным интеллектом;
- боты для развлечения;

– боты для бизнеса.

Сейчас наблюдается рост в разработке и использовании чат-ботов по целому ряду причин, в том числе и благодаря последним успехам в области обработки естественного языка (NLP). NLP — это способность компьютера понимать контекст, нюансы, и намерения при общении с человеком.

Как работают чат-боты? Начнем с того, что чат-боты разделяют на два типа: основанные на правилах и те, которые используют искусственный интеллект.

Основанные на правилах чат-боты работают как автоматизированные телефонные системы: «Нажмите 1 для продажи, 2 для счета …». Эти правила встроены в программу, и чат-бот работает строго в рамках этих правил.

Чат-боты, которые используют ИИ имеют гораздо более реалистичные разговоры с пользователями. Это потому, что этот тип чат-ботов учится после каждого взаимодействия с пользователем, и это позволяет ему лучше понимать намерения пользователей и чего они хотят. Другими словами, он в состоянии предвидеть, чего ожидает пользователь, прежде чем тот задаст вопрос.

Один из основных способов использования чат-бота, основанного на правилах, является работа с клиентами и поддержка. Другими словами, они используются, чтобы задавать клиентам ряд вопросов, тем самым приводя к разрешению их проблемы.

Чат-боты, созданные с использованием искусственного интеллекта, предлагают больше возможностей, в том числе еще один дополнительный способ взаимодействия с клиентами. Фактически, они могут использоваться для создания прямых конверсий и действовать как цифровая версия продавца.

Мы не можем сказать относилось ли это к чат-ботам, однако то, что они значительно упрощают процессы и экономят время — это факт.

Есть ограничения, и технология пока новая, но чат-боты предоставляют новые возможности для любой компании, стремящейся привлечь потребителей.

Мы же будем создавать чат-бота помощника, основанного на правилах. Суть нашего бота будет заключаться в автоматических ответах на входящие сообщения от пользователей.

1.2 Анализ существующих разработок и обоснование необходимости

Первым аналогом чат-бота является игровой бот «Lesya» – бесплатный чат-бот основан на правилах. По типу назначения он является развлекательным. Данный бот был создан в 2018 году, но даже спустя 3 года он не потерял свою актуальность из-за систематических обновлений. У бота Lesya имеется своя база данных с информацией о пользователях хоть раз написавших боту. Суть игры чат-бота напоминает некую экономическую стратегию, где мы можем работать, покупать, различную недвижимость и многое другое.

Рисунок 1.2.1 – чат-бот Lesya

Достоинства:

– игра затягивает;
– хороший хостинг;
– игра бесконечна;
– есть база данных;
– большое разнообразие команд;
– имеются кнопки для удобства;
– хорошая поддержка, быстрые ответы;
– систематические конкурсы и раздачи;
– обновления происходят еженедельно.

Недостатки:

– бот существует только в социальной сети «Вконтакте»;
– большие цены на платные услуги.

Вторым аналогом чат-бота является игровой бот «VChat Coin & RoboArena». Это так же бесплатный развлекательный бот, основанный на правилах. В социальной сети «Вконтакте» он с 2019 года. За 3 года он терпел падения и взлеты, но на данный момент он все так же актуален. Суть бота является в улучшении своего профиля, робота, участвовать в боях на робоарене

зарабатывая опыт для дальнейшего повышения уровня, а также для добычи ресурсов для улучшения самого бота (жизни, урон, броня). Данный бот позволяет собирать(майнить) игровую валюту для дальнейшего обмена ее на VKCoin. Чем больше VKCoin, тем лучше у пользователя место в рейтинге.

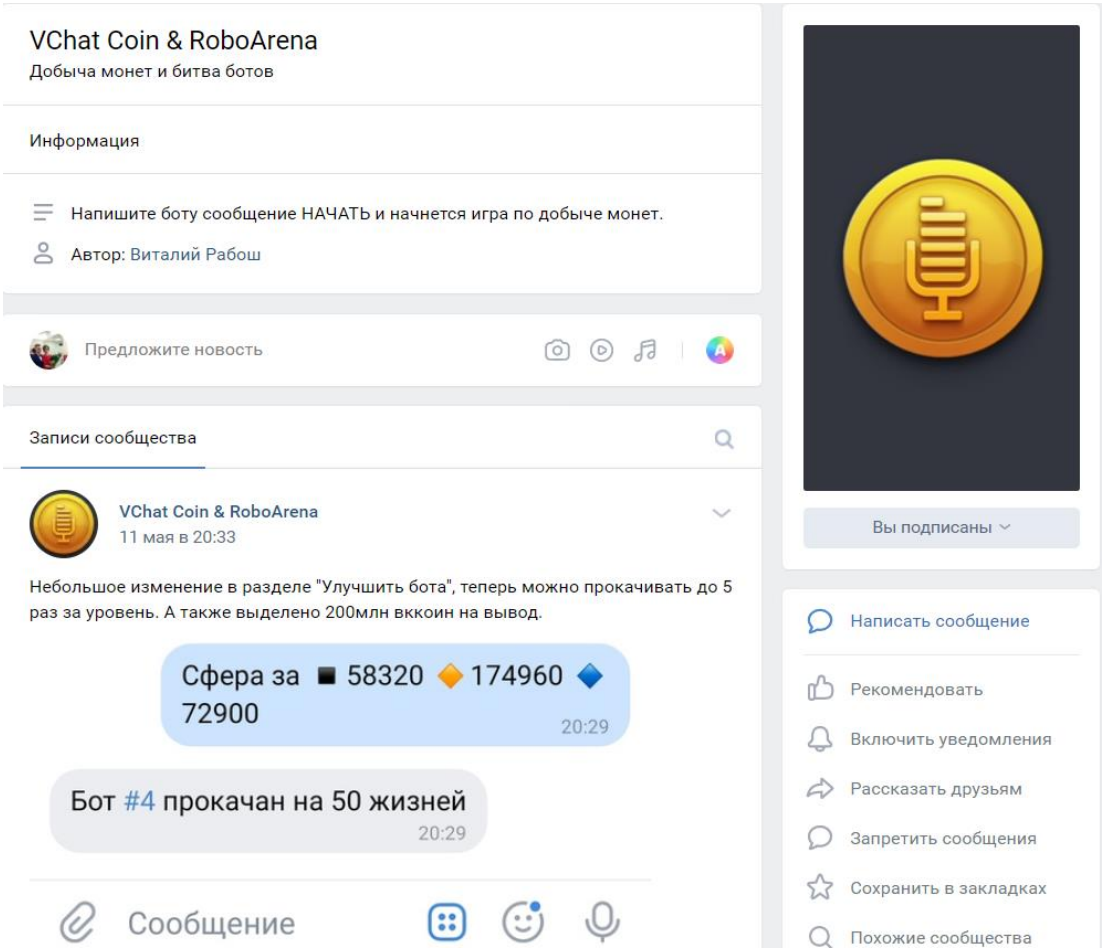


Рисунок 1.2.2 – чат-бот VChat Coin & RoboArena.

Достоинства:

– существует телеграмм версия игры;
– игра не затягивает;
– игра бесконечна;
– хороший хостинг;
– имеются кнопки для удобства;
– плохо разработаны кнопки;

Недостатки:

– малое разнообразие команд;
– плохая поддержка;
– мало обновлений;
– запущенное сообщество в «Вконтакте».

Третьим аналогом чат-бота является бот «Леонардо Дайвинчик» – бесплатный развлекательный бот для знакомств, основанный на правилах. Данный бот был добавлен в социальную сеть «Вконтакте» в 2018 году. На

сегодняшний день он имеет аудиторию в 13+миллионов человек, что довольно внушительные цифры. Суть бота заключается в знакомстве людей. Вы заполняете свою анкету и исходя из вашей анкеты бот ищет пользователя для общения для вас.

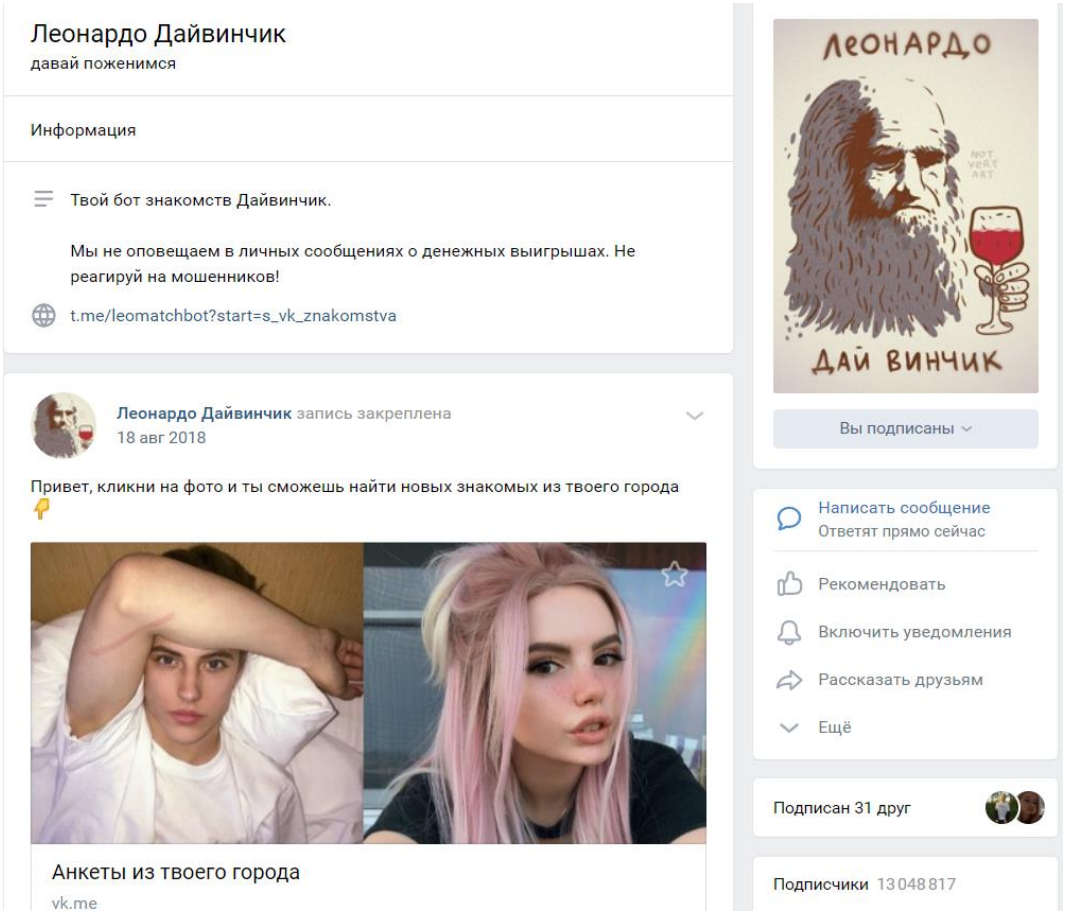


Рисунок 1.2.3 – чат-бот «Леонардо Дайвинчик».

Достоинства:
- большая аудитория;
- хорошая поддержка;
- имеются кнопки для удобства;
- бот существует в телеграмме;
- хороший хостинг.

Недостатки:
- быстро надоедает;
- аудитория от 10 до 16 лет.

2 Проектирование программного средства

2.1 Постановка задачи

В данном курсовом проекте нам необходимо создать программу, которая будет соответствовать следующим задачам:

- приложение должно быть мобильным;
- приложение должно быть быстродействующим;
- приложение должно быть полностью автоматическим;
- приложение должно работать 24/7 без сбоев;
- приложение должно обрабатывать большое количество запросов за малый промежуток времени;
- приложение должно быть грамотно оформленным;
- приложение должно быть удобным и понятным;

Мобильность приложение заключается в наименьшем объеме памяти занимаемой приложением. Все дело в том, что наше приложение в будущем будет обновляться, а это значит, что перезагружать его на различные хост-машины придется систематически, после каждой доработки, но, если приложение будет занимать наименьший объем памяти, оно будет гораздо быстрее загружаться на хостинг, что позволит сократить время тех-работ и пользователи смогут быстрее начать пользоваться приложением.

Быстродействующее приложение подразумевает, что приложение будет быстро обрабатывать события и отвечать на сообщения. Для решения данной задачи нам необходимо выбрать, какой API мы будем использовать, но для начала необходимо разобраться в них получше, но на данный момент мы имеем два претендента:

- Callback API;
- Long Poll API.

Callback API – это инструмент для отслеживания активности пользователей в сообществе ВКонтакте. С его помощью можно реализовать:

- бота для отправки мгновенных ответов на поступающие сообщения;
- систему автоматической модерации контента;
- сервис для сбора и обработки показателей вовлеченности аудитории.

Long Polling — это технология, которая позволяет получать данные о новых событиях с помощью «длинных запросов». Сервер получает запрос, но отправляет ответ на него не сразу, а лишь тогда, когда произойдет какое-либо событие (например, придёт новое сообщение), либо истечет заданное время ожидания.

Но так как нам необходимо только отправлять автоматические ответы на сообщения и у нас нет необходимости создания автомодерации контента и так далее, то мы воспользуемся Long Poll API.

Автоматизация приложения будет реализована путем создания предполагаемых команд приходящих от пользователя и создании ответов на

данные команды. Мы могли бы создать ИИ (искусственный интеллект) для таких целей, в последствии обучив его, но в этом мы так же не нуждаемся, дело в том, что бот у нас нацелен на точные команды, а не на широкий круг различных сообщений.

Приложение должно работать 24/7 для удобства пользователей. Дело в том, что, если бот будет работать не на постоянной основе это будет не удобным и смысла в таком боте не будет. Для такой работы мы должны использовать исключения в коде, чтобы при возникновении какой-либо ошибки программа не останавливала работу, а продолжала работать дальше, ошибки будут в основном из-за отсутствия некоторой входной команды. Не сделай мы блок исключения, смысл в нашем боте не было.

Для того чтобы приложение обрабатывала большое количество запросов за малый промежуток времени нам необходимо хорошее подключение к интернету, а также хорошие характеристики хост-машины, чем эти показатели будут лучше, тем лучше у нас будет скорость ответов. Минимальная скорость интернета должна быть равна 1МБ/с и минимальная оперативная память 4ГБ, а также в дополнении 1ТБ памяти для хранения последних запросов обрабатываемых ботом.

Так как наше приложение заключается в автоматических ответах, что значит бот будет отправлять запрограммированные куски сообщений на различные команды, нам необходимо данные команды грамотно оформить как с программной стороны, так и по правилам русского языка:

- грамотность текста;
- краткость текста;
- правильное и четкое предоставление мысли.

Для удобств отправки сообщений боту мы реализовали конвертирование входящих сообщений в нижний регистр, что позволит игнорировать регистр при отправке текста, улучшить скорость обработки текста и многое другое.

2.2 Описание среды разработки

Для написания чат-бота я использовал редактор кода Visual Studio Community 2019 года (рис 2.2.1).

Microsoft Visual Studio - полнофункциональная интегрированная среда разработки (IDE) с поддержкой популярных языков программирования, среди которых C, C++, VB.NET, C#, F#, JavaScript, Python.

Функциональность Visual Studio охватывает все этапы разработки программного обеспечения, предоставляя современные инструменты для написания кода, проектирования графических интерфейсов, сборки, отладки и тестирования приложений. Возможности Visual Studio могут быть дополнены путем подключения необходимых расширений.

Редактор кода Visual Studio поддерживает подсветку синтаксиса, вставку фрагментов кода, отображение структуры и связанных функций. Существенно

ускорить работу помогает технология IntelliSense - автозавершение кода по мере ввода.

Встроенный отладчик Visual Studio используется для поиска и исправления ошибок в исходном коде, в том числе на низком аппаратном уровне. Инструменты диагностики позволяют оценить качество кода с точки зрения производительности и использования памяти.

Дизайнер форм Visual Studio незаменим при разработке программ с графическим интерфейсом, помогая спроектировать внешний вид будущего приложения и работу каждого элемента интерфейса.

Наконец, Visual Studio предоставляет комплекс инструментов для автоматизации тестирования приложений в части проверки работы интерфейсов, модульного и нагрузочного тестирования.

Для командных проектов Visual Studio предлагает поддержку групповой работы, позволяя выполнять совместное редактирование и отладку любой части кода в реальном времени, а в качестве системы управления версиями использовать Team Foundation или Git.

Основным расширением файла, ассоциированным с Microsoft Visual Studio, является SLN – Visual Studio Solution File (Файл решения Visual Studio), при открытии которого в программу загружаются все данные и проекты, связанные с разрабатываемым программным решением.

Что она умеет? Возможности редактора представлены ниже:

– intelliSense. Технология авто дополнения Microsoft. Дописывает название функции при вводе начальных букв. Кроме прямого назначения, IntelliSense используется для доступа к документации и для устранения неоднозначности в именах переменных, функций и методов, используя рефлексию;

– code Anilizer. Функционал, который помогает найти ошибки в коде. Совмещён с IntelliSense, тем, что все ошибки, уведомления, потенциальные ошибки подсвечиваются. Также вся эта информация отображается в окне "Error List";

– perfomance Analizer. Инструмент, отображающий затраты ресурсов при работе приложения/сервиса в виде статистики и графиков;

– test Manager. Встроенный менеджер тестов. После создания теста можно с помощью специального окна запускать и настраивать тесты;

– extension/Updates Manager. Менеджер плагинов, адаптеров, провайдеров. Позволяет легко найти, установить, обновить любое дополнение.

– nuget. Система управления пакетами для платформ разработки Microsoft, в первую очередь библиотек .NET Framework. Управляется .NET Foundation. Удобная установка библиотек в любой .Net проект;

– git Manager. Встроенный менеджер контроля версий. Изначально работал только с Team Foundation Server. Сейчас можно подключить Team Explorer (Название менеджера) к любому репозиторию. Присутствуют все необходимые функции для работы с git без запросов;

– archivator. Архиватор проектов. После того, как проект готов, нужно собрать исполняемый файл. Для каждой технологии реализован свой архиватор. Не нужно устанавливать отдельный софт, чтобы сделать установочник;

– file Manager. Для добавления нового файла в проект существует встроенный менеджер файлов. Удобное создание любых файлов на основе шаблонов. Реализовано большое количество стандартных шаблонов (Пример: класс). Также можно добавлять свои. При установке новой технологии - добавляются соответствующие шаблоны;

– views. Большое количество различных вкладок для отображения различной полезной информации, вроде списка "GOTO", или отображения данных объекта в Debug режиме;

– customization. Возможность изменить внешний вид Visual Studio под себя. Изменения цветов, темы, шрифтов, отступов и т.д. Расположение окон в удобном вам виде;

– setting. Настройка всего выше перечисленного функционала. Настройка быстрых клавиш, уведомлений, быстрый запуск, стартового окна, вкладок, разметки языков и много другого.

Благодаря огромному количеству настроек, поддерживаемых технологий, быстродействию и удобству Visual Studio считается одной из лучших сред разработки. Из минусов можно выделить огромный вес пакетов технологий.

Рисунок 2.2.1 – Логотип Visual Studio.

2.3 Алгоритмы работы программы

На рисунке 2.3.1 мы видим главный методы «Main» который выполняется в любом случае, в нем мы оповещаем администратора в консоли о запуске бота, а так же запускаем бесконечный цикл на проверку нового события в сообществе. Если новое событие произошло, то вызывается метод «Handler».

Так же на рисунке мы видим команду, которая предназначена для того, чтобы задать цвет текста в консоли «Console.ForegroundColor». После вывода в консоль цветовые настройки сбрасываются командой «Console.ResetColor();».

Так же мы видим цикл «while» который выполняется пока условие равно истинно.

В цикле мы видим простейший список однотипных объектов «List».

Так же в цикле расположен оператор цикла «foreach», который предназначен для перебора элементов нашего списка.

```
private static void Main(string[] args)
{
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("Бот запущен!");
    Console.ResetColor();
    while (true)
    {
        List<Api.LongPoll.EventObject> events = LongPoll.Listen();
        foreach (var evt in events)
        {
            Handler(evt);
        }

    }
}
```

Рисунок 2.3.1 – метод «Main»

После того как программа вызвала метод «Handler» наша программа ищет команду в данном методе и если команда существует, то программа отправляет приписанный данной команде ответ пользователю. Пример ответа вы можете увидеть на рисунке 2.3.2. Мы видим, что если первое слово входящего сообщения было равно «помощь», то бот вызывает метод «SendMessage» (рис. 2.3.3) и отправляет в его в качестве параметров ID пользователя, приславшего нам сообщение и текст ответа. Так же для избежания ошибок из-за регистра, мы полученное сообщение переводим в нижний регистр.

```
if (message[0] == "помощь")
{
    SendMessage(evt.PeerId, "Меню бота:\nАбитуриентам\nСтудентам\nКонтакты");
    Console.Write(evt.PeerId + " ");
}
if(message[0] == "абитуриентам")
{
    SendMessage(evt.PeerId, "Отлично, может быть вы хотите узнать получше о нашем колле
    SendMessage(evt.PeerId, "Меню бота:\nСпециальности;\nОбщежития;\nКолледж;
}
```

Рисунок 2.3.2 – пример ответа пользователю

В методе «SendMessage» мы наблюдаем блок исключения «try – catch».

При использовании блока «try–catch» (рис. 2.3.3) в начале выполняются все инструкции в блоке «try». Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок finally. Но так как в нашем коде finally не предусмотрен, то при выполнении блока try метод завершает работу (если он не обнаружил исключительных ситуаций).

Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок catch, который может обработать данное исключение. В нашем случае в блоке catch мы ничего не наблюдаем. Все потому что нам просто нужно чтобы программа после нахождения исключения не завершала свою работу, а продолжила работать.

```
private static void SendMessage(long peerId, string message)
{
    try
    {
        VKApi.CallMethod("messages.send", new Dictionary<string, string>
        {
            {"peer_id", peerId.ToString()},
            {"message", message}
        });
    }
    catch
    {
        // ignored
    }
}
```

Рисунок 2.3.3 – метод «SendMessage»

Так же в нашей программе есть дополнительный класс «Api» который содержит в себе вложенные классы (рис. 2.3.4). На этом же рисунке мы видим подключенные библиотеки, которые используются в нашей программе:

- system;
- system.Collections.Generic;
- system.Net;
- system.IO;
- newtonsoft.Json.Linq;
- system.Text;
- system.Collections.Specialized.

Объявление using System дает возможность ссылаться на классы, которые находятся в пространстве имен System, так что их можно использовать, не добавляя System. Перед именем типа. Пространство имен System содержит много полезных классов. Одним из них является класс Console, который используется при создании консольных приложений.

Большая часть классов коллекций содержится в пространствах имен System.Collections (простые необобщенные классы коллекций), System.Collections.Generic (обобщенные или типизированные классы коллекций)

и System.Collections.Specialized (специальные классы коллекций). Также для обеспечения параллельного выполнения задач и многопоточного доступа применяются классы коллекций из пространства имен System.Collections.Concu rrent.

Пространство имен System.Net довольно обширно и состоит из многих членов. Оно используется для написания кода, связанного с интернетом и содержит более 50 различных классов, окало 10 различных интерфейсов, перечислений и делегатов.

Пространство имен System.IO в .NET – это область библиотек базовых классов, посвященная службам файлового ввода-вывода, а также ввода-вывода из памяти. Подобно любому пространству имен, в System.IO определен набор классов, интерфейсов, перечислений, структур и делегатов, большинство из которых находятся в mscorlib.dll. В дополнение к типам, содержащимся внутри mscorlib.dll, в сборке System.dll определены дополнительные члены пространства имен System.IO. Обратите внимание, что все проекты Visual Studio 2010 автоматически устанавливают ссылки на обе сборки.

Многие типы из пространства имен System.IO сосредоточены на программных манипуляциях физическими каталогами и файлами. Дополнительные типы предоставляют поддержку чтения и записи данных в строковые буферы, а также области памяти. Ниже кратко описаны основные (неабстрактные) классы, которые дают понятие о функциональности System.IO:

LINQ to JSON – это API для работы с объектами JSON. Он был разработан с учетом LINQ, чтобы обеспечить возможность быстрого запроса и создания объектов JSON. LINQ to JSON находится в пространстве имен Newtonsoft.Json.Linq.

Пространство имен System.Text содержит классы, абстрактные базовые классы и вспомогательные классы. Скажем, например, если вы хотите воспользоваться StringBuilder, декодером, кодером и т. д.

System.Collections.Specialized – содержит специализированные и строго типизированные коллекции; например, словарь связанного списка, битовый вектор и коллекции, содержащие только строки.

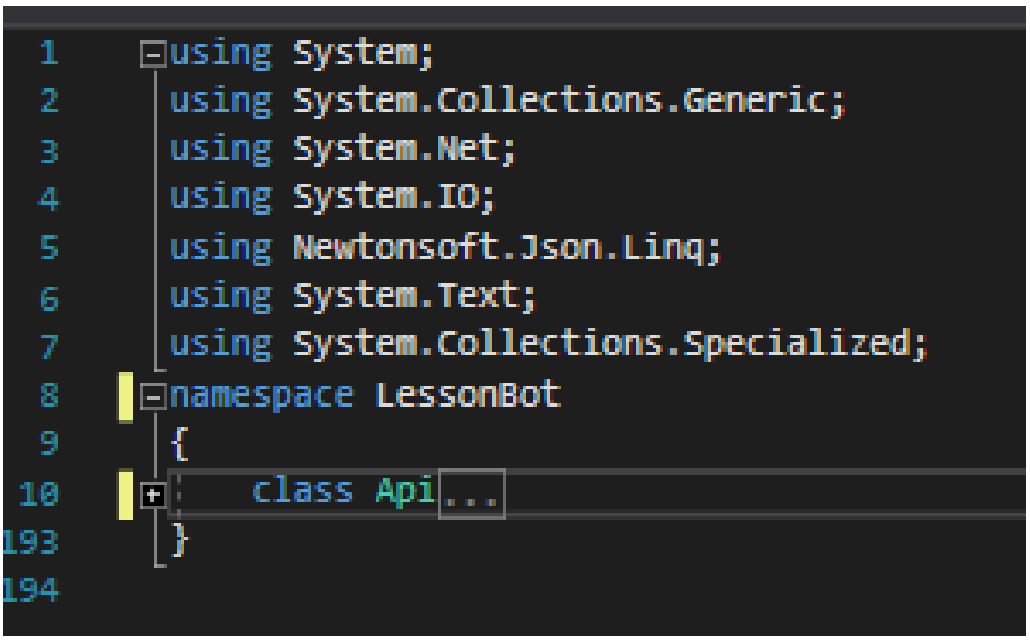


Рисунок 2.3.4 – класс «Api»

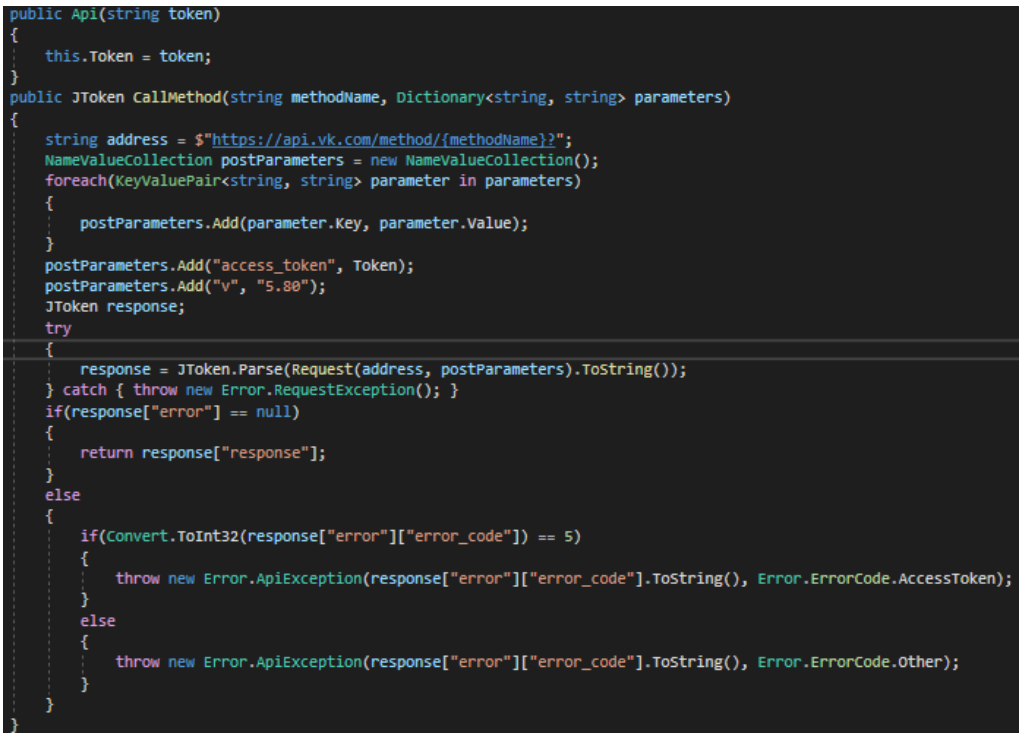
Класс «Api» содержит 2 вложенных класса, которые так же имеют минимум 1 вложенный класс и токен.

Первое что мы видим в нашем классе это конструктор (рис. 2.3.5).

Метод «CallMethod», в нем мы видим базовую ссылку и запрос, отправляемый на сервер, запрос должен выглядеть так: https://api.vk.com/method/{METHOD_NAME}?{PARAMS}&v={API_VERSION}, где {METHOD_NAME} – название метода, {PARAMS} – параметры вызываемого метода, а {API_VERSION} – версия API, которую должен использовать сервер при формировании ответа.

В качестве ответа нам будет возвращен JSON объект с информацией о результатах вызова метода.

Так же в нашем классе есть 2 перегруженных метода «Request». Перегруженный метод – это метод, который имеет похожего по структуре клона, но с разными параметрами, либо разные возвращаемые типы. В нашем случае методы имеют разные параметры.



```
public Api(string token)
{
    this.Token = token;
}
public JToken CallMethod(string methodName, Dictionary<string, string> parameters)
{
    string address = $"https://api.vk.com/method/{methodName}?";
    NameValueCollection postParameters = new NameValueCollection();
    foreach(KeyValuePair<string, string> parameter in parameters)
    {
        postParameters.Add(parameter.Key, parameter.Value);
    }
    postParameters.Add("access_token", Token);
    postParameters.Add("v", "5.80");
    JToken response;
    try
    {
        response = JToken.Parse(Request(address, postParameters).ToString());
    } catch { throw new Error.RequestException(); }
    if(response["error"] == null)
    {
        return response["response"];
    }
    else
    {
        if(Convert.ToInt32(response["error"]["error_code"]) == 5)
        {
            throw new Error.ApiException(response["error"]["error_code"].ToString(), Error.ErrorCode.AccessToken);
        }
        else
        {
            throw new Error.ApiException(response["error"]["error_code"].ToString(), Error.ErrorCode.Other);
        }
    }
}
```

Рисунок 2.3.5 – конструктор и метод «CallMethod»

Первый вложенный класс у нас «LongPoll» (рис 2.3.6) в данном классе получая от вас запрос, сервер ждет, когда произойдет событие, о котором он должен вас уведомить, и, когда оно происходит, Long Poll сервер отправляет ответ на ваш запрос, содержащий информацию о случившемся событии. Мы напишем программу, которая будет уведомлять пользователя о некоторых изменениях в его аккаунте, которые мы будем получать от Long Poll сервера. Чтобы начать получать ответы от сервера, необходимо получить три обязательных параметра, необходимых для работы Long Poll-a: server, key и ts, где:

– key – секретный ключ сессии;

– server – адрес сервера;

– ts – номер последнего события, начиная с которого нужно получать данные.

Поля с этими параметрами мы наблюдаем в нашем классе.

Их не нужно получать каждый раз, вызывая Long Poll, — будет достаточно одного вызова. Из документации следует, что эти значения можно получить, вызвав метод messages.getLongPollServer. Данная операция происходит в методе «GetInfoLongPoll» (рис. 2.3.7).

Для того, чтобы постоянно отправлять запросы к Long Poll, я решил использовать цикл, чтобы не переполнять стек рекурсией. Ниже приведена реализация программы, которая обращается к Long Poll.

Если за 25 секунд произошло какое-либо событие, которое попадает в список обрабатываемых Long Poll сервером, на экран выведется нечто похожее: {'ts': 0000000000, 'updates': [[9, -999999999, 0, 1501588841]]}. Что же значит пришедший ответ и как с ним дальше можно работать?

Во-первых, вы могли заметить, что в ответе содержится параметр 'ts', который мы использовали при отправлении запроса. Он здесь неспроста. Все события, попадающие в список обрабатываемых Long Poll сервера, нумеруются. Когда вы создаете новый аккаунт, первое такое событие имеет номер 1, второе — 2 и так далее. При отправлении запроса на Long Poll, нужно передавать этот параметр для того, чтобы сервер знал, начиная с какого номера нужно присылать обновления. В каждом ответе сервера также приходит этот параметр, чтобы вы использовали его при следующем вызове Long Poll сервера.

Во-вторых, в словаре содержится ключ 'updates' с говорящим названием. Не трудно догадаться, что значение по этому ключу хранит обновления, случившиеся после отправления запроса. Формат обновлений — массив массивов. Каждый массив в массиве — произошедшее обновление, которое нужно обработать. Если в первом массиве массивов больше одного, это значит, что несколько событий произошли одновременно. Параметр 'ts' содержит номер последнего из них. Если массив, доступный по ключу 'updates', пуст, то за время wait не произошло ни одного события. Вы спросите: «А что это за непонятные цифры в массивах?». Ответ довольно прост — это информация о случившемся событии, которую можно преобразовать в более понятный вид. Их обработкой мы займемся позже, а в следующей части напишем программу, которая будет постоянно обращаться к Long Poll серверу.

```
public class LongPoll
{
    public string Ts;
    public string Key;
    public string Server;
    private Api VKApi;
    public LongPoll(Api vkApi)
    {
        VKApi = vkApi;
        GetInfoLongPoll();
    }
    public List<EventObject> Listen()
    {
        while(true)
        {
            JToken response = RequestLongPoll();
            if (Convert.ToInt32(response["failed"]) == 1)
            {
                Ts = response["ts"].ToString();
            }
            else if (Convert.ToInt32(response["failed"]) == 2 || Convert.ToInt32(response["failed"]) == 3)
            {
                GetInfoLongPoll();
            }
            else
            {
                Ts = response["ts"].ToString();
                List<EventObject> updates = new List<EventObject>();
                foreach (var update in response["updates"])
                {
                    updates.Add(new EventObject(update));
                }
                return updates;
            }
        }
    }
```

Рисунок 2.3.6 – класс «Long Poll»

На рисунке 2.3.7 мы видим структуру нашего запроса серверу. Здесь мы, делая запрос отправляем:

– секретный ключ сессии:
– номер последнего события;

– время ожидания обновлений;
– дополнительные опции;
– Long Poll версию.

```
private JToken RequestLongPoll()
{
    NameValueCollection parameters = new NameValueCollection() {
        {"act", "a_check"},
        {"key", Key},
        {"ts", Ts},
        {"wait", "25"},
        {"mode", "2"},
        {"version", "3"}
    };
    return JToken.Parse(VKApi.Request($"https://{Server}", parameters));
}
private void GetInfoLongPoll()
{
    JToken infoLongPoll = VKApi.CallMethod("messages.getLongPollServer", new Dictionary<string, string>(1) {
        {"lp_version", "3"}
    });
    Ts = infoLongPoll["ts"].ToString();
    Key = infoLongPoll["key"].ToString();
    Server = infoLongPoll["server"].ToString();
}
```

Рисунок 2.3.7 – запрос серверу

Так же у нас в программе имеется класс «EventObject» (рис. 2.3.8) данный класс обрабатывает информацию, которую присылает нам сервер в ответ на запрос. Эта информация представлена в виде массивов с информацией, которую нужно обработать. Начать предлагаю с кода 80 — обновление счетчика непрочитанных сообщений, так как, по моему мнению, это событие является наименее сложным. Вот пример событий с кодом 80:

[80, 0, 0]
[80, 1, 0]

Первый параметр, как вы уже знаете, — код события. Остаются 2 элемента: 0 и 0 в первом случае и 1 0 во втором. Последний параметр при коде 80 должен быть всегда равен нулю, поэтому его можно игнорировать; для нас важен лишь второй. Во втором параметре указано, сколько на данный момент у пользователя непрочитанных сообщений. Минимальное значение — 0 (нет новых сообщений), максимальное — не ограничено. Этого достаточно, чтобы обрабатывать все события с кодом 80.

```
public class EventObject
{
    public VKEventType Type = VKEventType.Other;
    public long MessageId = 0;
    public long PeerId = 0;
    public long Time = 0;
    public string Message = "";
    public JToken Extra;
    public JToken Attachments;
    public bool FromUser = false;
    public bool FromChat = false;
    public bool FromGroup = false;
    public long UserId = 0;
    public long ChatId = 0;
    public long GroupId = 0;
    public JToken Row;
    public EventObject(JToken response)
    {
        Row = response;
        if (Convert.ToInt32(response[0]) == 4)
        {
            Type = VKEventType.MessageNew;
            MessageId = response[1].ToObject<long>();
            PeerId = response[3].ToObject<long>();
            Time = response[4].ToObject<long>();
            Message = response[5].ToString();
            Extra = response[6];
            Attachments = response[7];
            if(PeerId < 0)
            {
                FromGroup = true;
                GroupId = -PeerId;
            }
            else if(PeerId > 2000000000)
            {
                FromChat = true;
```

Рисунок 2.3.8 – класс «EventObject»

Последнее что мы видим в нашем классе, это обработка ошибок (рис. 2.3.9). Данная часть кода используется при отправлении и получении запросов серверу.

```
public enum VKEventType
{
    Other,
    MessageNew
}
public class Error
{
    public class RequestException : Exception { public RequestException() : base("Cannot send request.") { } }
    public class ApiException : Exception
    {
        public ErrorCode Code;
        public ApiException(string message, ErrorCode code) : base(message)
        {
            this.Code = code;
        }
    }
    public enum ErrorCode
    {
        Other,
        AccessToken
    }
}
```

Рисунок 2.3.9 – обработка ошибок класс «Error»

Так же мы можем встретить внутри нашего класса 2 вложенных класса. Которые отвечают за обработку ошибок со стороны API и метода «Request».

3 Разработка программного средства

3.1 Построение диаграмм

Unified Modeling Language (UML) – унифицированный язык моделирования. Расшифруем: modeling подразумевает создание модели, описывающей объект. Unified (универсальный, единый) — подходит для широкого класса проектируемых программных систем, различных областей приложений, типов организаций, уровней компетентности, размеров проектов. UML описывает объект в едином заданном синтаксисе, поэтому где бы вы не нарисовали диаграмму, ее правила будут понятны для всех, кто знаком с этим графическим языком — даже в другой стране.

Одна из задач UML — служить средством коммуникации внутри команды и при общении с заказчиком. Давайте рассмотрим возможные варианты использования диаграмм.

Для построения диаграмм использовался сайт «diagrams.net» – удобный сервис для создания блок-схем, UML-диаграмм, моделей бизнес-процессов онлайн. Совместим с большинством популярных инструментов, включая Google Docs, Git, Dropbox, OneDrive и другие.

3.1.1 Диаграмма классов

Диаграмма классов – структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их кооперaций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

В приложении А мы видим диаграмму классов приложения чат-бот.

На диаграмме классов мы видим 2 главных класса и 5 вложенных, а также 2 перечисления. Класс «Api» используется в классе «Program». В класс «Api» вложены 2 класса «Error» и «LongPoll». Класс «Error» в свою очередь имеет еще 2 вложенных класса «RequestException» и «ApiException», а класс «LongPoll» так же имеет один вложенный класс «EventObject».

3.1.2 Диаграмма деятельности

Диаграмма деятельности – UML-диаграмма, на которой показаны действия, состояния которых описаны на диаграмме состояний. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов – вложенных видов деятельности и отдельных

действий, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Диаграмма деятельности представлена в приложении В. Мы видим, что все действия начинаются с того, что пользователь напишет боту, в случае ошибки отправки сообщения пользователь снова пишет боту. Далее у нас происходит отправка запроса на сервер, если событие не обнаружено запрос отправляется снова, если событие обнаружено, происходит обработка запроса, далее сервер формирует и отправляет ответ, что событие произошло и отправляет данные события, после чего наша программа обрабатывает это событие, и если программа не находит ответ на ту команду что прислал пользователь программа игнорирует и пользователь пишет сообщение снова, если ответ обнаружен, то происходит отправка ответного сообщения пользователю. После всех этих действий происходит окончание процесса.

3.1.3 Диаграмма нотации IDEF0

IDEF0 – методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность.

Диаграмма в нотации IDEF0 представлена в приложении С. Здесь мы наблюдаем, что входными данными являются:

- сообщение от пользователя;
- API;
- подключение к интернету.

В управлении у нас находится сервер, а механизм пользователь, так как пользователь начинает процесс, а сервер управляет процессом. На выходе мы получаем ответ пользователю.

3.2 Тестирование приложения

Тестирование программного обеспечения – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.

Тестирование программы производилось тремя разными видами тестирования:

- тестирование производительности;
- юзабилити тестирование;
- функциональное тестирование.

Тестирование производилось на ноутбуке Lenovo с характеристиками, показанными в таблице 3.2.1.

Таблица 3.2.1 – Характеристики ноутбука.

Параметр	Конфигурация
Основные характеристики	
Изготовитель	Lenovo
Модель	81D6
Процессор	
Процессор	AMD A9
Модель	AMD A9-9425
Кол-во. ядер	2
Тактовая частота	3100 МГц
Экран	
Диагональ	15.6”
Разрешение	1920x1080(FullHD)
Соотношение сторон	16:9(широкоформатный)
Оперативная память	
Тип	DDR4
Объем	4
Видеокарта	
Тип	Встроенная
Модель	AMD Radeon R5
Память	
Тип	HDD
Объем	1000ГБ

3.2.1 Тестирование производительности

Тестирование производительности – это комплекс типов тестирования, целью которого является определение работоспособности, стабильности, потребления ресурсов и других атрибутов качества приложения в условиях различных сценариев использования и нагрузок. Тестирование производительности позволяет находить возможные уязвимости и недостатки в системе с целью предотвратить их пагубное влияние на работу программы в условиях использования.

В зависимости от характеристик, которые нам нужно протестировать, тестирование производительности делится на типы:

- нагрузочное тестирование;
- стресс-тестирование;
- тестирование стабильности;
- объемное тестирование.

В нашем проекте мы будем использовать три первых типа тестирования производительности. А для того чтобы их использовать необходимо для начала разобраться для чего они предназначены и как работают.

1) Нагрузочное тестирование – тестирование времени отклика приложения на запросы различных типов, с целью удостовериться, что приложение работает в соответствии с требованиями при обычной пользовательской нагрузке.

Для проведения нагрузочного тестирования нам пришлось включить наше приложение на 4 часа, чтобы пользователи могли общаться с нашим чат-ботом, а мы в свою очередь могли отследить время отклика нашей программы в зависимости от прироста входящих сообщений (рис. 3.2.1.1). На рисунке видно, что время мы указывали в тиках. Для понимания 1 секунда равна 20-и тикам.

Таким маленьким показателям мы обязаны слабой скорости интернета и перепадам нагрузок на наше приложение, так как приложение тестируется на нашем собственном сервере – ноутбуке.

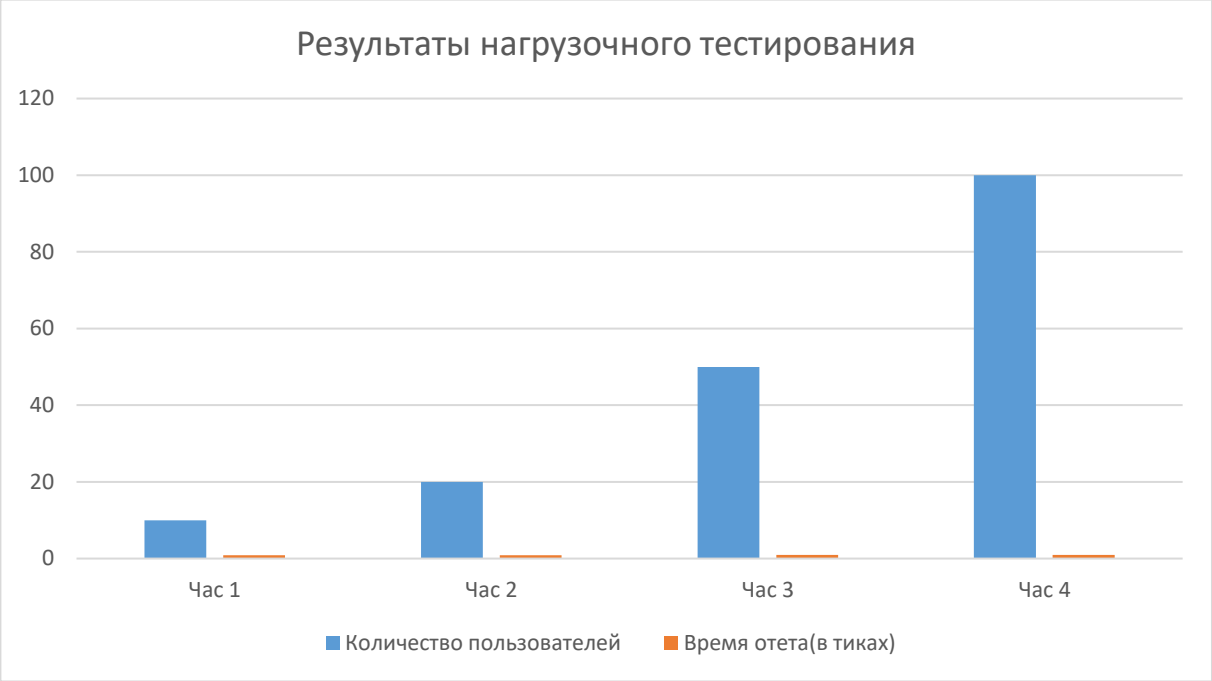


Рисунок 3.2.1.1 – диаграмма о результатах тестирования

2) Стресс-тестирование – тестирование работоспособности приложения при нагрузках, превышающих пользовательские в несколько раз.

Для проведения стресс-теста мы создали бесконечный цикл, который будут «Флудить» пользователям. Каждые 10 пользователей добавлялись через каждые 10 минут. Тестирование производилось до того момента пока нагрузка на процессор на сервере программы не достигла 80%. В качестве сервера выступает персональный компьютер.

Отчет по результатам тестирования мы можете наблюдать на рисунке 3.2.1.2.

Как вы можете видеть после 40 минут тестирования нагрузка на процессор нашего сервера достигла критического состояния (82%). Такие маленькие показатели связаны с тем, что приложение находится не на хост-машине как должно быть, а на нашем собственном сервере – ноутбуке, где 55% нагрузок уже

составляют запущенные процессы. А также на это влияет слабый процессор, тем более что он с 2-я ядрами.

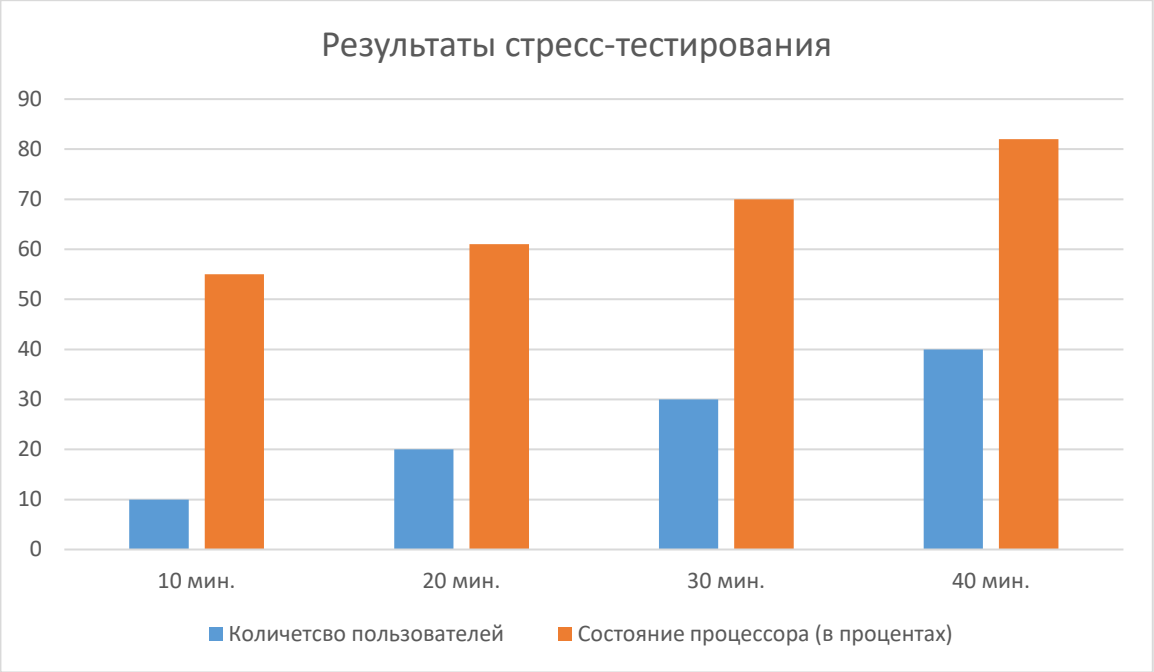


Рисунок 3.2.1.2 – результаты стресс-тестирования

3) Тестирование стабильности исследует работоспособность приложения при длительной работе во времени, при нормальной для программы нагрузке.

Для проведения тестирования стабильности мы включим нашего бота на 7 дней, во время этого наш бот будет общаться с разными пользователями, а мы будем измерять среднее количество пользователей и средний результат нагрузки на сервер.

Как вы можете наблюдать рост пользователей рос на протяжении 6-ти дней, но рост нагрузок на процессор прекратился на 3-й день и держался в среднем на 63-х процентах, что говорит о не плохой стабильности нашей программы.

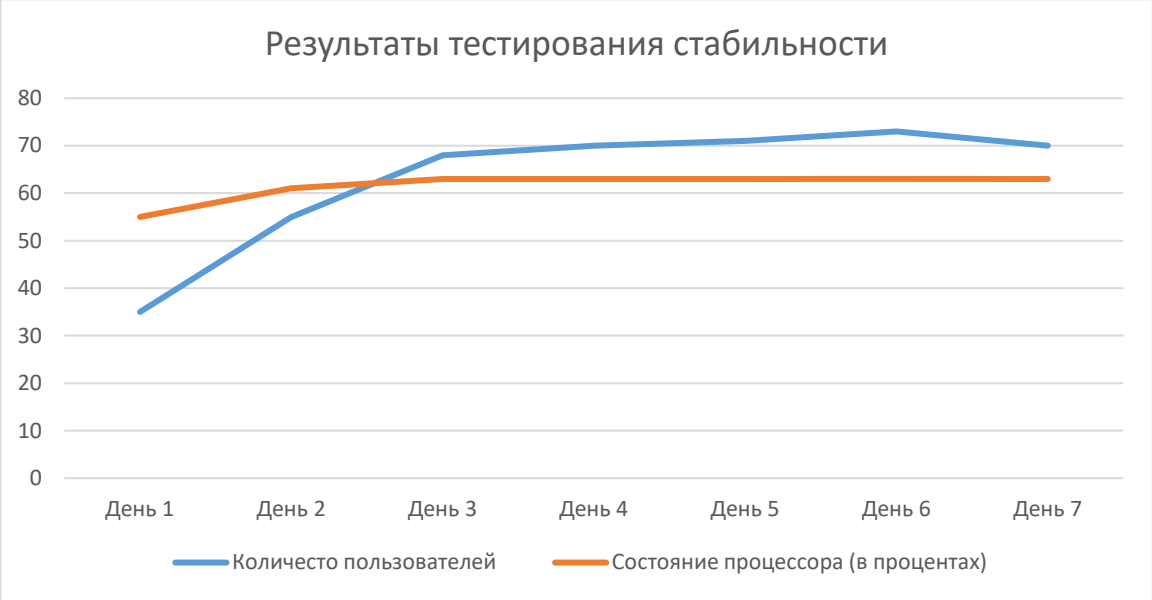


Рисунок 3.2.1.3 – диаграмма результатов тестирования стабильности

3.2.2 Юзабилити-тестирование

Юзабилити-тестирование – это тестирование удобства интерфейса продукта конечными пользователями. Юзабилити-тестирование показывает, насколько продукт соответствует ожиданиям пользователей, выявляет проблемные места в интерфейсе, дает возможность взглянуть на продукт глазами пользователей. В процессе юзабилити-тестирования пользователь выполняет типовые задачи с продуктом в присутствии ведущего тестирования.

Для проведения юзабилити-тестирования были приглашены 3 группы по 5 человек целевой аудитории. В первой группе люди в возрасте от 14 до 18 лет. Во второй группе люди от 18 до 36 лет. В третьей группе от 36 до 48 лет.

Мы предложили каждой из трех групп провести следующие действия с ботом:

– запустить бота;
– подключить бота к группе;
– написать сообщение;
– добавить бота в беседу;
– удалить бота из беседы;
– отключить бота;

После выполнения всех вышеперечисленных действий были выявлены проблемы, описанные в таблице 3.2.2.1:

Таблица 3.2.2.1 – выявленные проблемы

№ проблемы	Проблема	Состояние	Критичность
1	Отсутствие кнопок в диалоге с ботом.	обнаружено	слабая

3.2.3 Функциональное тестирование

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

Для тестирования нашего ПО, мы будем использовать метод «Модульного тестирования» таблица 3.2.3.1.

Модульное тестирование – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения. Его цель заключается в том, чтобы проверить, что каждая единица программного кода работает должным образом. Данный вид тестирование выполняется разработчиками на этапе кодирования приложения. Модульные тесты изолируют часть кода и проверяют его работоспособность. Единицей для измерения может служить отдельная функция, метод, процедура, модуль или объект.

Таблица 3.2.3.1 – отчет по результатам модульного тестирования.

№ теста	Название проверки	Результат	Рекомендации
1	Проверка модуля «Подключения к API»	Положительно	-
2	Проверка модуля «Создание запросов»	Положительно	-
3	Проверка модуля «Систематическая отправка запросов»	Положительно	-
4	Проверка модуля «Получения события»	Положительно	-
5	Проверка модуля «Обработки события»	Положительно	-
6	Проверка модуля «Создание исключений»	Положительно	-
7	Проверка модуля «Отправка сообщения»	Положительно	-
8	Проверка модуля «Получение ID»	Положительно	-
9	Проверка модуля «Получение сообщения»	Положительно	-

В ходе результатов тестирования не было выявлено никаких ошибок в модулях, все модули работают корректно. Следовательно, никаких рекомендаций так же нет.

3.3 Руководство пользователя

Для работы чат-бота необходимо приавязать его к сообществу в социальной сети «Вконтакте» предварительно создав сообщество. Для этого переходим в раздел «Сообщества» и нажимаем создать сообщество(рис. 3.3.1).

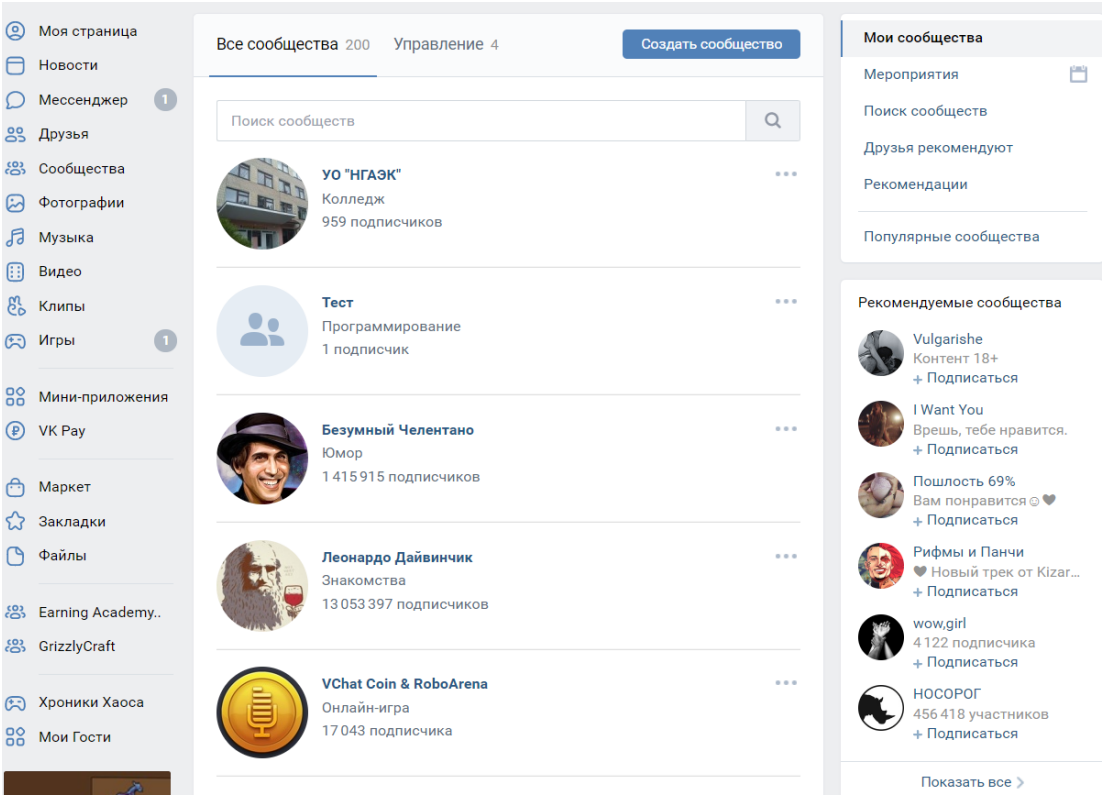


Рисунок 3.3.1 – создание сообщества.

Далее нам необходимо выбрать тип сообщества (рис.3.3.2) тип сообщества никак не повлияет на работу бота, поэтому мы можем выбрать любой из предложенных.

Рисунок 3.3.2 – выбор типа сообщества

В следующем меню нам необходимо заполнить несколько обязательных полей и нажать «Создать сообщество» (рис. 3.3.3).

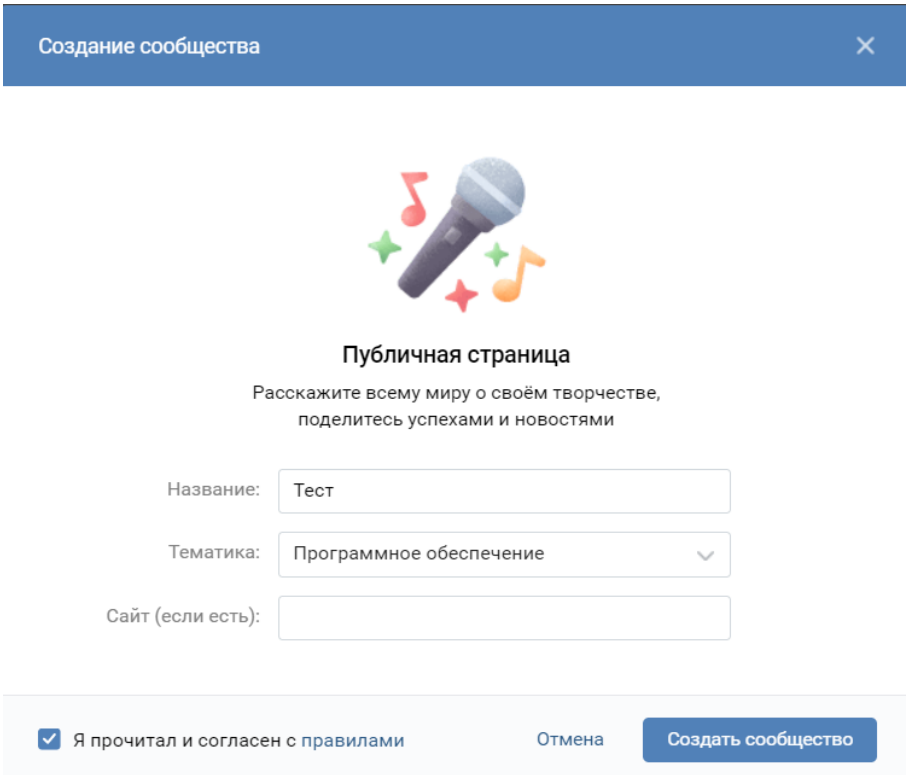


Рисунок 3.3.3 – заполнение полей

Готово! Сообщество создано. Сейчас нам необходимо произвести некоторые настройки для того что бы мы могли подключить нашего чат-бота к нашему сообществу. Для этого переходим в раздел «Управление» (рис. 3.3.4).

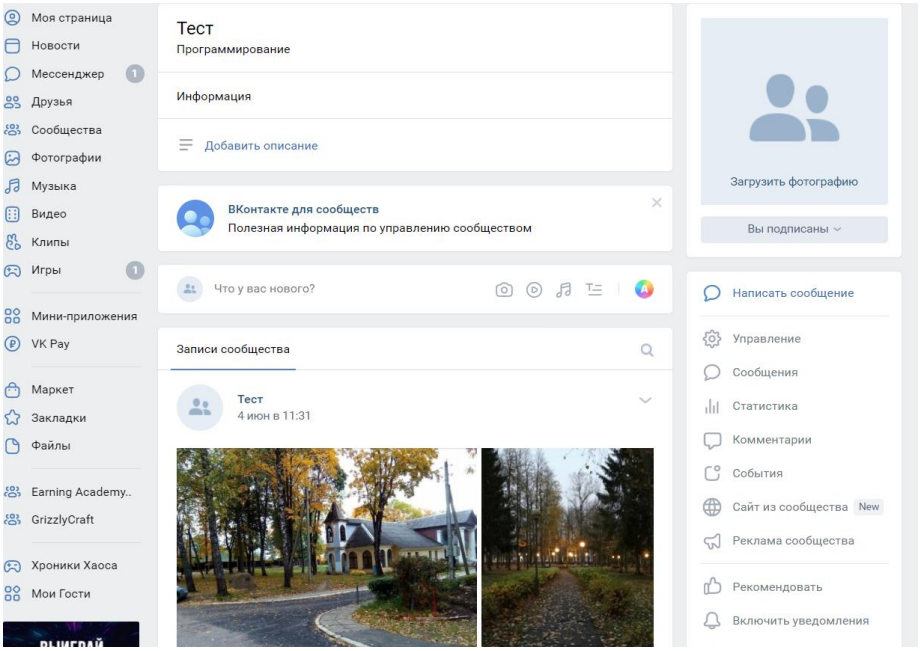


Рисунок 3.3.4 – переход в раздел «Управление»

После того как мы перешли, в меню разделов нам необходимо выбрать раздел «Сообщения» и перейти в подраздел «Настройки для бота» установив там следующие настройки (рис. 3.3.5).

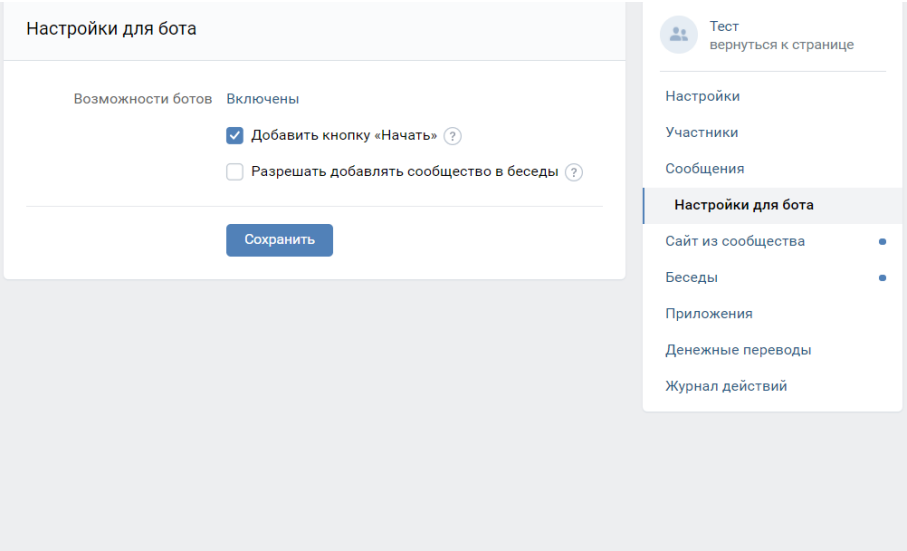


Рисунок 3.3.5 – настройки в разделе «Сообщения»

Далее нам необходимо перейти в раздел «Настройки» и выбрать подраздел «Работа с API». Далее переходим на вкладку «Long Poll API» и устанавливаем настройки как на рисунке 3.3.6.

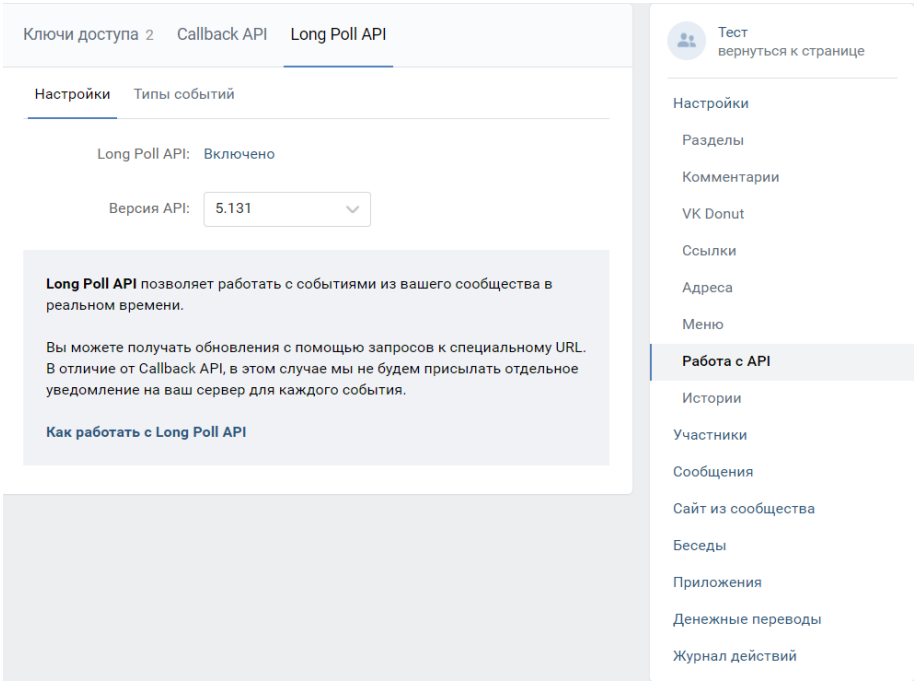


Рисунок 3.3.6 – настройки для подвкладки «Настройки»

Далее переходим в подвкладку «Типы событий» и устанавливаем галочки напротив всех пунктов как показано на рисунке 3.3.7.

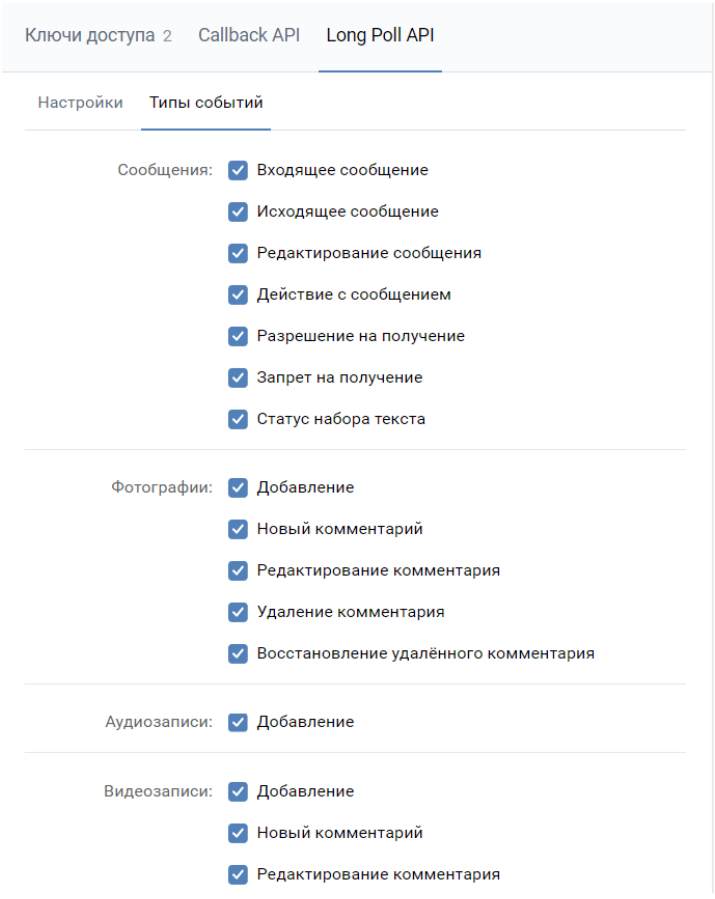


Рисунок 3.3.7 – установка разрешений

Теперь возвращаемся на вкладку «Ключи доступа» и создаем ключ (рис. 3.3.8). Для того чтобы скопировать ключ необходимо будет нажать кнопку показать и подтвердить ваши действия введя полученный код доступа.

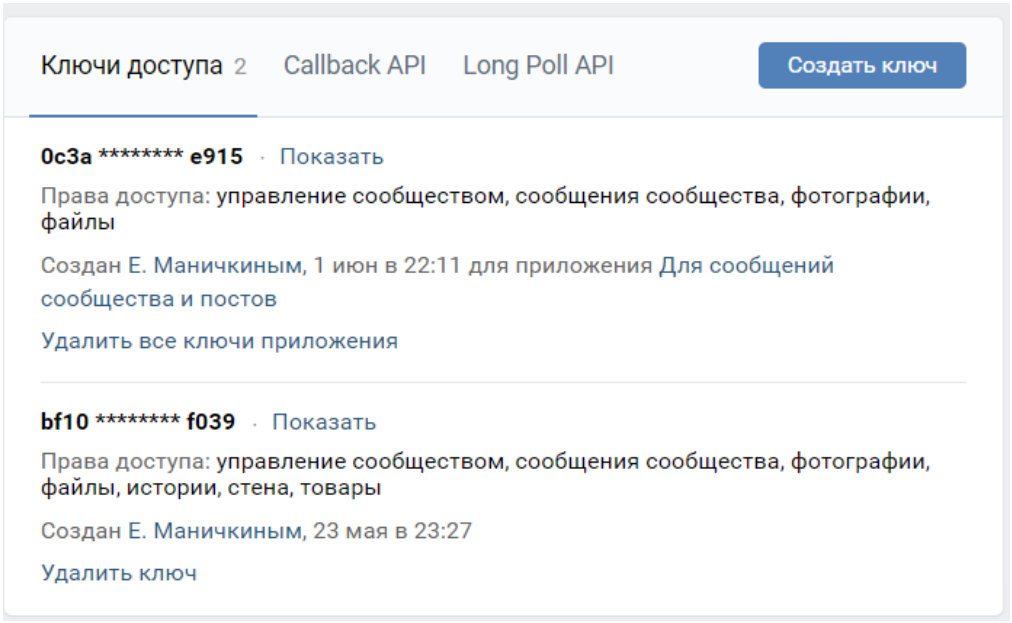
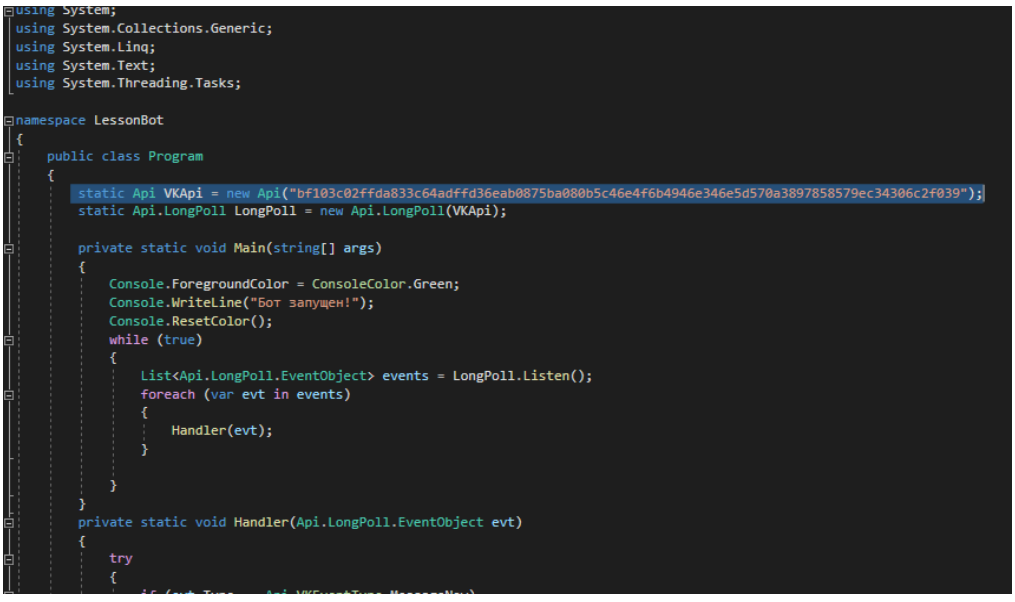


Рисунок 3.3.8 – ключи доступа

После этого настройки сообщества окончены, и мы можем перейти к настройке бота.

Для начала нам нужно вставить наш ключ в переменную «VkApi» (рис. 3.3.9).



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LessonBot
{
    public class Program
    {
        static Api VKApi = new Api("bf103c02ffda833c64adffd36eab0875ba080b5c46e4f6b4946e346e5d570a3897858579ec34306c2f039");
        static Api.LongPoll LongPoll = new Api.LongPoll(VKApi);

        private static void Main(string[] args)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Бот запущен!");
            Console.ResetColor();
            while (true)
            {
                List<Api.LongPoll.EventObject> events = LongPoll.Listen();
                foreach (var evt in events)
                {
                    Handler(evt);
                }

            }
        }
        private static void Handler(Api.LongPoll.EventObject evt)
        {
            try
            {
```

Рисунок 3.3.9 – подключения бота

Запускаем программу, пишем в личные сообщения сообщества «Начать» (рис. 3.3.10) для получения дальнейших инструкций по использованию бота.

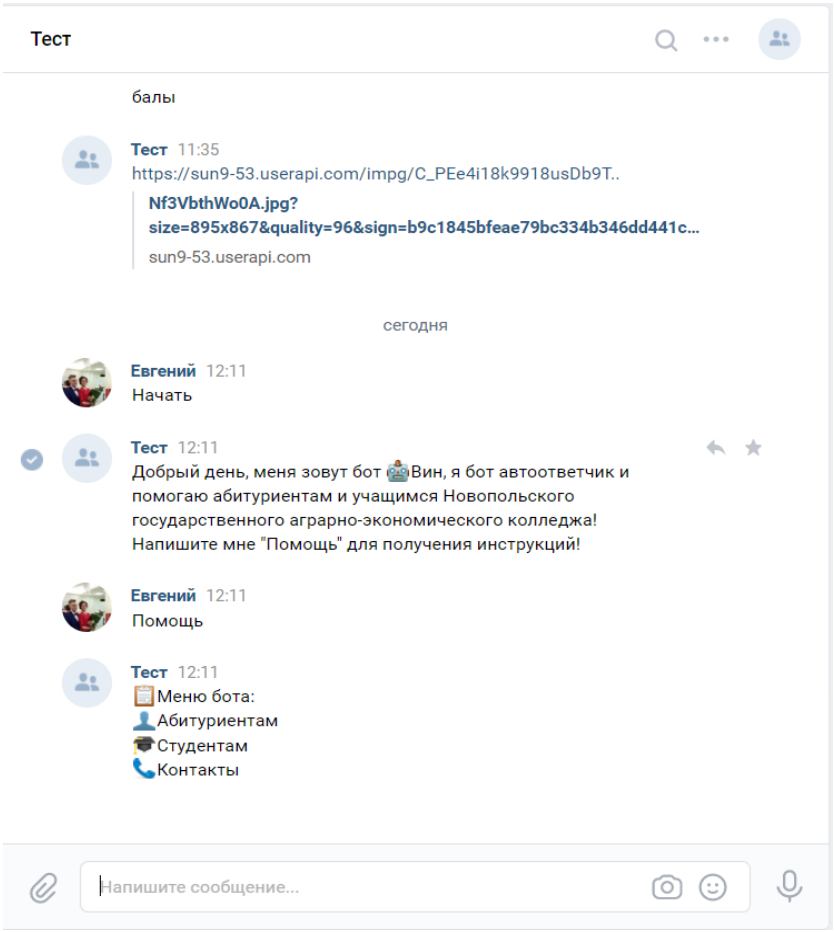


Рисунок 3.3.10 – первые команды для бота

Алгоритм работы бота следующий:

Программа систематически отправляет запрос на сервер чтобы узнать произошли какие-либо события. Отловив событие (в нашем случае входящее сообщение) программа отправляет на нее ответ, если входящего сообщения нет в списке команд, то наш бот ничего не ответит.

Таким образом мы реализовали автоматические ответы на различные команды, получаемые в личных сообщениях бота.

Так же необходимо помнить, что для работы бота необходимо подключение к интернету!

ЗАКЛЮЧЕНИЕ

В процессе разработки чат-бота были закреплены навыки работы с API и объектно-ориентированного программирования.

Бот был разработан для широкого круга пользователей, для использования пользователями социальной сети не требует дополнительных навыков и знаний для пользования ботом. Бот предназначен для ответов на популярные и частые вопросы пользователей.

Достоинства бота состоит в том, что абитуриентам и студентам не требуется искать информацию о колледже в интернете и других информационных системах, достаточно только зайти в группу, написать боту, и узнать все что было необходимо.

Все цели и задачи курсового проекта были достигнуты, а задачи выполнены.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Агуров, Павел C#. Сборник рецептов / Павел Агуров. - М.: "БХВ-Петербург", 2012. - 432 с.

2. Албахари, Джозеф C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. - М.: БХВ-Петербург, 2012. - 944 с.

3. Албахари, Джозеф C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. - М.: БХВ-Петербург, 2013. - 944 с.

4. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 266 с.

5. Бишоп, Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2013. - 472 с.

6. Вагнер, Билл C# Эффективное программирование / Билл Вагнер. - М.: ЛОРИ, 2013. - 320 с.

7. Зиборов, В.В. Visual C# 2012 на примерах / В.В. Зиборов. - М.: БХВ-Петербург, 2013. - 480 с.

8. Зиборов, Виктор Visual C# 2010 на примерах / Виктор Зиборов. - М.: "БХВ-Петербург", 2011. - 432 с.

9. Ишкова, Э. А. Самоучитель C#. Начала программирования / Э.А. Ишкова. - М.: Наука и техника, 2013. - 496 с.

10. Касаткин, А. И. Профессиональное программирование на языке си. Управление ресурсами / А.И. Касаткин. - М.: Высшая школа, 2012. - 432 с.

11. Лотка, Рокфорд C# и CSLA .NET Framework. Разработка бизнес-объектов / Рокфорд Лотка. - М.: Вильямс, 2010. - 816 с.

12. Мак-Дональд, Мэтью Silverlight 5 с примерами на C# для профессионалов / Мэтью Мак-Дональд. - М.: Вильямс, 2013. - 848 с.

13. Марченко, А. Л. Основы программирования на C# 2.0 / А.Л. Марченко. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2011. - 552 с.

14. Подбельский, В. В. Язык C#. Базовый курс / В.В. Подбельский. - М.: Финансы и статистика, Инфра-М, 2011. - 384 с.

15. Прайс, Джейсон Visual C# 2.0. Полное руководство / Джейсон Прайс , Майк Гандэрлой. - М.: Век +, Корона-Век, Энтроп, 2010. - 736 с.