

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра дискретной математики и
информационных технологий

РК СЕРИИ "КОННОР" ДЛЯ ДЛЯ ПОМОЩИ В РАССЛЕДОВАНИИ
ДЕЛ, СВЯЗАННЫХ С ДЕВИАНТНЫМИ АНДРОИДАМИ

КУРСОВАЯ РАБОТА

Студента 3 курса 321 группы
направления 09.03.01 — Информатика и вычислительная техника
факультета КНиИТ
Экгарт Викентия Александровича

Научный руководитель
ассистент

П.П. Поздняков

Заведующий кафедрой
к.ф.-м.н. доцент

Л.Б. Тяпаев

Саратов 2019

Содержание

ВВЕДЕНИЕ	3
1 Понятие «API»	4
2 Ознакомление с официальной документацией API Вконтакте	5
3 Выбор технологии для решения поставленной задачи	6
4 Разработка чат бота	8
4.1 чат-бот.....	8
4.2 Развертка приложения на сервере	15
4.3 Ручное тестирование бота.....	15
ЗАКЛЮЧЕНИЕ	18
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	19

ВВЕДЕНИЕ

Целью данной курсовой работы является разработка чат-бота, работающего с API социальной сети "ВКонтакте" для помощи старосте в оповещении студентов о различных мероприятиях и прочих объявлениях, а также для мгновенного получения актуальной информации о порядке текущей недели в расписании, без необходимости ручного высчитывания или обращения к сторонним ресурсам. Данная социальная сеть была выбрана, поскольку количество пользователей (в том числе и студентов ВУЗа) в ней максимально, в отличие от других социальных сетей. Также, данная социальная сеть полностью поддерживает законодательство РФ. В работе будет рассмотрен полный путь от знакомства с API социальной сети и, до конечного запуска бота на сервере.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Дать определение понятию «API».
2. Ознакомиться с официальной документацией API от выбранной социальной сети.
3. Выбрать технологии для решения поставленной задачи.
4. Разработать чат-бота.

1 Понятие «API»

API — application programming interface, по русски - программный интерфейс приложения, Т.е. это описание способов (набор определенных "открытых" методов с параметрами, констант для изменения) которыми одно компьютерное программное обеспечение может взаимодействовать с другим компьютерным ПО. При этом, как правило, при хорошем проектировании API, не имеет значения на каком языке будет написана программа, которая будет обращаться к этому API. Так, например, программа написанная на C++ может работать с API программы, написанной на Java.

2 Ознакомление с официальной документацией API Вконтакте

Итак, документация данной социальной сети находится по следующему адресу <https://vk.com/dev/manuals>.

Для решения поставленной задачи, заходим в раздел с чат ботами, и видим, что для взаимодействия бота и сервера социальной сети предусмотрено два способа общения "клиент-сервер":

- Callback API.
- Long Poll API.

Callback API — согласно официальной документации, работает следующим образом - как только в сообществе происходит нужное событие - от сервера Вконтакте приходит уведомление на наш сервер. При этом, событие может быть каким угодно: комментарий к фотографии, новая запись на стене, вступление в сообщество, отправка сообщения, и многое другое. [2].

Второй способ получения обновлений — это подключение к Long Poll серверу. В отличие от Callback API, Long Poll сервер будет присылать на наш сервер только те обновления, которые связаны с сообщениями. Никаких других событий из сообщества в нём нет. То есть, узнать о том, присоединился ли пользователь к сообществу - нельзя.

Исходя из поставленной задачи, а также из-за некоторых преимуществ в скорости Long Poll API над обычным [1], принято решение использовать Long Poll API.

Теперь необходимо создать сообщество в Вконтакте, от имени которого будет писать чат бот.

3 Выбор технологии для решения поставленной задачи

Выбранный мною функционал чат-бота предполагает следующий сценарий использования : Создатель беседы (беседа - чат между несколькими пользователями) или любой другой её участник добавляет бота в беседу. Далее администратор беседы открывает боту доступ ко всем сообщениям в ней и делает бота администратором (необходимо для работы функции "позови всех")

После этого доступны следующие команды боту :

- какая неделя [сейчас, завтра, следующая]
- позови всех
- объявление
- инфо, помощь, что ты умеешь

Первая команда должна вернуть информацию о том, какая сейчас (или иная, зависит от запроса пользователя) неделя. Например, бот может ответить "сейчас числитель что означает что в данный момент актуальны пары из верхних ячеек расписания.

Вторая команда "позови всех" отправляет каждому пользователю уведомление, даже если уведомления у этого пользователя отключены. Полезно, так как многие пользователи в виду частых и неинформативных сообщений (флуд) отключают уведомления из беседы и могут пропустить что-то важное.

Третья команда "объявление" — отправляет каждому пользователю уведомление с текстом, идущем после этой команды. Работет даже если уведомления у этого пользователя отключены. Как в случае и с предыдущим пунктом, защищает пользователей от несвоевременного информирования при отключенных уведомлениях.

Последняя команда выводит информацию о доступных командах на данный момент.

Для достижения такого механизма работы мной были выбраны следующие технологии:

1. enviroment.

- Node JS. (серверный интерпретатор JS)
- PM2. (перезапуск процесса в случае ошибки, а также более продвинутое логирование)

2. bot.

- NPM (пакетный менеджер для работы с зависимостями).
- vk-node-sdk (библиотека для взаимодействия с LongPoll API).
- util (библиотека для логирования ошибок)
- JavaScript (язык программирования).

3. Сервер.

- Ubuntu 17.10 64bit (512 МБ RAM 20 ГБ SSD 1 CPU).

Посльку не требуется хранить какие-либо данные, полученные от пользователя, база данных не участвует в проекте.

В качестве IDE была использована VS Code от ©Microsoft. Данная IDE имеет встроенную поддержку JavaScript, системы контроля версий, а также распространяется бесплатно.

4 Разработка чат бота

Разработка велась в три этапа — программирование и создание серверной части - самого бота, загрузка проекта на сервер и ручное тестирование функционала.

4.1 чат-бот

С помощью CLI (command line interface) была создана директория для разработки и далее была введена команда ‘npm init’, которая создает конфигурационный файл требуемого приложения. Полученный файл оказался таким

```
1 {
2   "name": "bot_vk",
3   "version": "1.0.0",
4   "description": "konnor testing",
5   "main": "konnor.js",
6   "dependencies": {
7     "node-vk-bot": "1.2.1",
8     "util": "^0.11.0",
9     "vk-node-sdk": "^0.1.8",
10  },
11  "scripts": {
12    "test": "echo \"Error: no test specified\" && exit 1"
13  },
14  "author": "",
15  "license": "ISC"
16 }
17
```

Далее, был создан главный js файл — "konnor.js" в котором необходимо с помощью библиотеки "vk-node-sdk" подключиться к серверу Вконтакте для приёма уведомлений.

Для этого, создал файл с константами, где будут указаны токены для работы с ботом, а также ID сообщества, от чьего имени будет отвечать бот. Разнесение архитектуры приложения на несколько файлов упростит его поддержку в дальнейшем и поможет делать важные изменения без затрагивания большого количества кода.

Вот как выглядит данный файл сейчас:


```

1  const vkGroupFullRight = 'group on vk token fith full rights and
    long-poll';
2  const groupId = 000;
3
4  module.exports = {
5      vkGroupFullRight: vkGroupFullRight,
6      groupId: groupId,
7  }

```

В главном файле, импортируем этот файл и инициализируем бота.

```

1
2  const DEBUG_MODE = require('./konnor_config');
3  const util = require('util');
4  const { Bot } = require('node-vk-bot');
5
6  //don't forget to add tokens in file and rename him
7  const TOKENS = require('./secret_tokens');
8
9  const regName = /коннор|connor|копор|андроид/i;
10
11
12  const debugConsole = (variable, depth) => {
13      DEBUG_MODE && console.log('debug: ' + util.inspect(variable,
14          false, depth = 8));
15  }
16
17  const bot = new Bot({
18      token: TOKENS.vkGroupFullRight,
19      group_id: TOKENS.groupId,
20  }).start()
21
22  console.log('bot started');
23
24  bot.on('poll-error', error => {
25      console.error('error occurred on a working with the Long Poll
26          server ' +
27          '({util.inspect(error)})')
28  })

```

Для запуска бота введем команду "node ./konnor.js". Бот запущен!

Прежде чем писать реакцию на команды, в соответствии с принципами "Банды четырех" [3] — выносим каждую команду в отдельный модуль и делаем "горячее подключение" модулей. То есть не требуется переписывать общую логику бота, при добавлении новой команды в будущем.

Каждая команда - это объект, вида:

```
1  const имя_команды = {  
2    callName: 'триггер срабатывания, может быть регулярным  
        выражением',  
3    action: (bot, message, TOKENS) => {  
4      действие — что она делает  
5    }  
6  }
```

Также, подключение команд реализуем в отдельном файле, дабы не трогать основной.

Сама же логика бота - на какую команду он среагирует будет осуществляться следующим образом — бот получает список всех доступных команд и входящее сообщение. Далее простым циклом проходясь по списку команд, он сверяет "триггер"(регулярное выражение) каждой команды с входным сообщением. И если регулярное выражение проходит (результат true), то бот исполняет нужную команду и передаёт свой ответ на отправку.

Данный паттерн называется "Стратегия" и очень сильно экономит ресурсы и облегчает добавление новых команд. [4]

Теперь реализуем действие на событие "новое сообщение для этого напишем в функции "on.message" указанный сверху цикл. И добавим fallback - сообщение, которое отправится, если никакая команда не сработает.

```
1  bot.get(/./, message => {  
2    if (message.peer_id > 1000000000) { //message from conversation  
3      if (!regName.test(message.text)) { //if no name calling — no  
        answer  
4      return;  
5    }  
6  }  
7  bot.api('messages.setActivity', { type: 'typing', peer_id:  
    message.peer_id, group_id: TOKENS.groupId })  
8    .then(res => console.log(util.inspect(res)));  
9  
10 message.text = message.text.replace(regName, ''); //delete him
```

```

11     name
12   for (let skillName in skillList) {
13     const regExp = RegExp(skillName, 'i');
14     if (regExp.test(message.text)) {
15       console.log('matched: ' + skillName);
16       skillList[skillName](bot, message, TOKENS);
17       return;
18     }
19   }
20
21 })

```

На 7-8 строчках утсанавливаем статус "набирает сообщение— что придет боту больше человечности и будет понятно, что он получил сообщение и готовиться на него ответить.

Со 2 по 6 строки, проверяем, обращались ли к нему по имени, потому что данная функция будет вызываться при любом сообщении из беседы, даже тогда, когда от бота ничего не требуется.

Ниже приведены три необходимых "навыка"бота для достижения цели этой работы.

Для призыва всех людей в беседу:

```

1  const mentionAll = {
2    callName: 'позови всех',
3    action: (bot, message, TOKENS) => {
4      bot.api('messages.getConversationMembers', { peer_id:
5        message.peer_id, group_id: TOKENS.groupId })
6        .then(res => {
7          //const usersNamesOrIds = res.profiles.map(profile
8            => profile.screen_name != '' ? profile.
9              screen_name : profile.id );
10         //console.log(util.inspect(usersNamesOrIds));
11         const mentionIds = res.profiles.map(profile => '@id$
12           {profile.id}');
13         bot.send('я призываю всех ' + `${mentionIds.toString
14           ()}` , message.peer_id).catch(
15           function (e) {
16             console.log(e);
17           }
18         );
19       });
20     }
21   };

```

```

14         })
15         .catch(
16             function (e) {
17                 console.log(e);
18             }
19         );
20     }
21 }
22
23 module.exports = mentionAll;

```

Для рассылки всем объявления:

```

1  const sendMessageAll = {
2      callName: 'об[ъ]явление',
3      action: (bot, message, TOKENS) => {
4          const alertMessage = message.text
5              .split(/об[ъ]явление/i, 2)[1]
6              .trim();
7          bot.api('messages.getConversationMembers', { peer_id:
8              message.peer_id, group_id: TOKENS.groupId })
9              .then(res => {
10                  const mentionIds = res.profiles.map(profile => '
11                      @id${profile.id}');
12                  bot.send('сообщение для всех: ${alertMessage} ${
13                      mentionIds.toString()} ', message.peer_id).
14                      catch(
15                          function (e) {
16                              console.log(e);
17                          }
18                      );
19              })
20              .catch(
21                  function (e) {
22                      console.log(e);
23                  }
24              );
25      }
26  }
27
28 module.exports = sendMessageAll;

```

Для определения числитель или знаменатель:

```

1
2 const chislOrZnam = require( './logic' );
3
4 const weekPair = {
5     callName: 'какая неделя|неделя какая',
6     action: (bot, message) => {
7         let d = new Date();
8         d.setHours(d.getHours() + 1); //for GMT+4
9
10        let parity = 'сейчас ' + chislOrZnam(d);
11
12        if (/завтра/i.test(message.text)) {
13            d.setDate(d.getDate() + 1);
14            parity = 'завтра ' + chislOrZnam(d);
15        }
16        if (/следующая/i.test(message.text)) {
17            d.setDate(d.getDate() + 7);
18            parity = 'следующая неделя ' + chislOrZnam(d);
19        }
20        bot.send(parity, message.peer_id).catch(
21            function (e) {
22                console.log(e);
23            }
24        );
25    }
26 }
27
28 module.exports = weekPair;

```

Логика определения числителя и знаменателя:

```

1 const chislOrZnamen = (date) => {
2     let now = new Date();
3     if (date) {
4         now = date;
5     }
6     //изначально считаем что сейчас числитель
7     //true — числитель
8     //false — знаменатель
9
10    const dayFirstOfSeptember = (new Date(now.getFullYear(), 8, 1)).
    getDay();

```

```

11     const dateFirstMondayInSeptember = (dayFirstOfSeptember === 2) ?
        7 : (9 - dayFirstOfSeptember) % 7;
12
13     const dayFirstOfJanuary = (new Date(now.getFullYear(), 0, 1)).
        getDay();
14     const dateFirstMondayInJanuary = (dayFirstOfJanuary === 2) ? 7 :
        (9 - dayFirstOfJanuary) % 7;
15
16     const currentDate = now.getDate();
17     const currentMonth = now.getMonth();
18
19     const countFullWeeksBeforeNY = Math.floor((now.getTime() - new
        Date(now.getFullYear(), 8, dateFirstMondayInSeptember).
        getTime()) / 86400000 / 7);
20     const countFullWeeksAfterNY = Math.floor((now.getTime() - new
        Date(now.getFullYear(), 0, dateFirstMondayInJanuary).getTime
        ()) / 86400000 / 7);
21
22     if (currentMonth > 8 && countFullWeeksBeforeNY % 2 !== 0) {
23         return 'числитель'
24     }
25     else if (currentMonth === 8) {
26         if (currentDate >= dateFirstMondayInSeptember &&
            countFullWeeksBeforeNY % 2 !== 0) {
27             return 'числитель'
28         }
29         else if (currentDate < dateFirstMondayInSeptember) {
30             return 'числитель'
31         }
32     }
33     else if (currentMonth < 8) {
34         if (currentMonth <= 4 && currentMonth >= 0) {
35             if ((currentMonth === 0) && (currentDate <
                dateFirstMondayInJanuary)) {
36                 if (dayFirstOfSeptember === 0 || dayFirstOfSeptember
                    > 4) {
37                     return 'числитель'
38                 }
39             }
40             else if (countFullWeeksAfterNY % 2 === 0) {
41                 return 'знаменатель'

```

```

42         }
43     }
44     else if (currentMonth > 4) {
45         console.log('сейчас лето, ты чего');
46     }
47 }
48 return 'числитель'
49 }
50
51 module.exports = chislOrZnamen;

```

4.2 Развертка приложения на сервере

Далее необходимо подключиться к серверу по SSH и клонировать туда репозиторий проекта, установить все пакеты и запустить.

Установим библиотеку PM2 глобально с помощью менеджера пакетов NPM:

```
1 npm i pm2 -g
```

Далее, запустим бота командой:

```
1 pm2 start konnor.js
```

Теперь, терминал можно зыкрывать - бот запущен и работает.

Для того, чтобы смотреть логи, нужно использовать команду:

```
1 pm2 logs konnor
```

4.3 Ручное тестирование бота

Поскольку предполагается, что чат-ботом будут пользоваться люди, можно ожидать, что кроме доступных команд, они будут пытаться отправить роботу картинки, символы не из юникода и прочее. При этом, бот не должен падать или зависать, а должен лишь строго реагировать на команды. [5]

После тестирования, ошибок не было выявлено.

Ниже на Рисунке 1 и Рисунке 2 приведены скриншоты успешной работы бота.

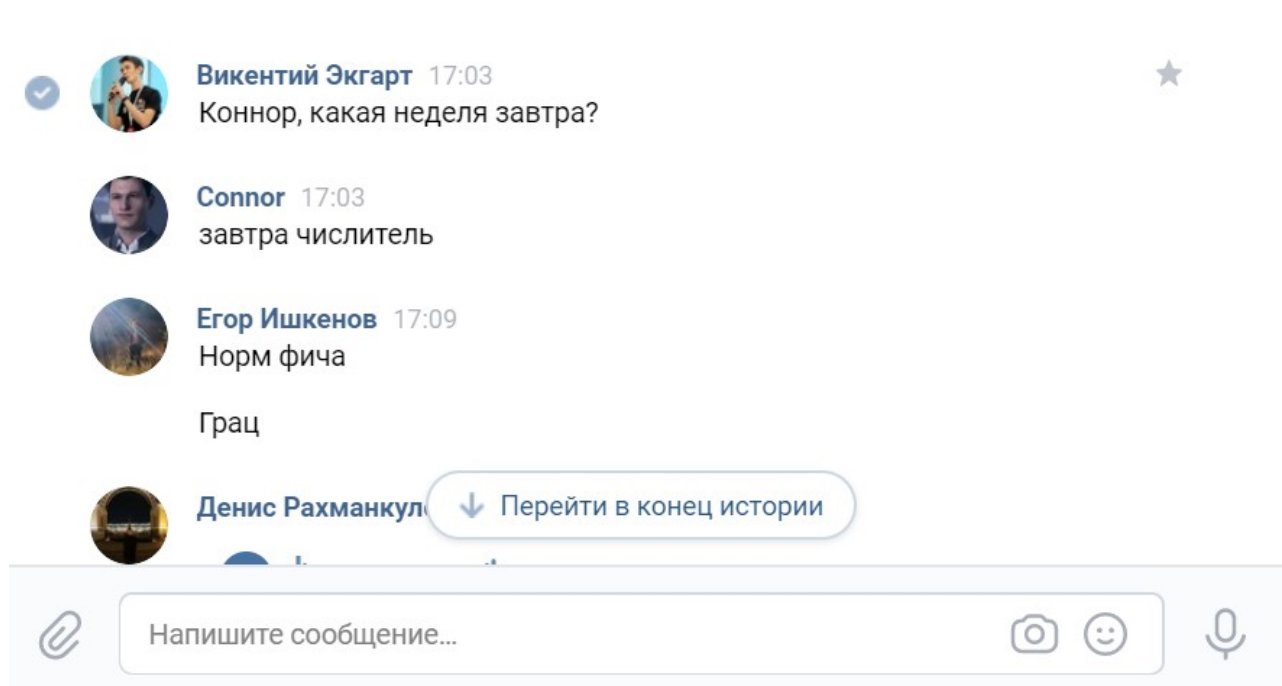


Рисунок 1 – Запрос с целью узнать, какая неделя в расписании будет актуальна завтра.

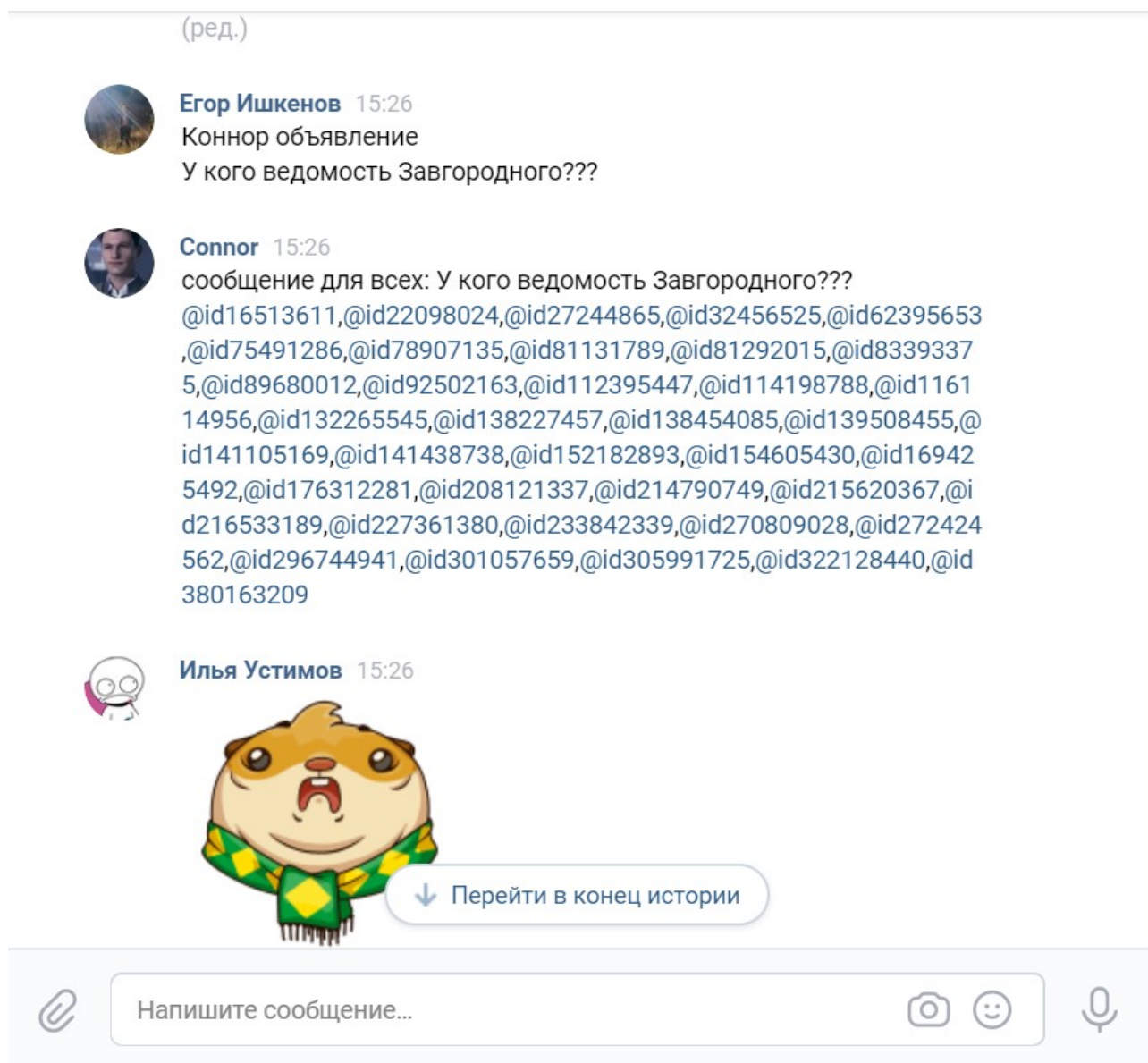


Рисунок 2 – Староста использует возможности бота, чтобы очень быстро обратиться ко всем с важным вопросом.

ЗАКЛЮЧЕНИЕ

При выполнении данной курсовой работы, было проведено ознакомление с понятием API, а также была изучена официальная документация социальной сети для написания ботов и были выполнены поставленные задачи по облегчению управления беседой для старосты.

Исходный код приложения доступен в репозитории GitHub:

<https://github.com/vikegart/konnor>

В дальнейшем планируется внедрить распознавание речи, в целях облегчения пользования чатом для людей, которым прослушивание голосовых сообщений затруднительно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Bots Long Poll API [Электронный ресурс]: URL: https://vk.com/dev/bots_longpoll (Дата обращения: 18.02.2019) Загл. с экрана. Яз. англ;
- 2 API для чат-ботов [Электронный ресурс]: URL: https://vk.com/dev/bots_docs (Дата обращения: 18.02.2019) Загл. с экрана. Яз. англ;
- 3 Приемы объектно-ориентированного проектирования. Паттерны проектирования. Влиссидес Джон , Джонсон Р. , Хелм Ричард , Гамма Эрих Яз. англ стр [50-120];
- 4 JavaScript. Шаблоны . Стоян Стефанов Яз. рус. стр [70-90];
- 5 Тестирование dot com . Савин Роман Яз. рус. стр 158;