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

RK 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 _____ В. А. Поздняков

Заведующий кафедрой
к.ф.-м.н. доцент _____ Л.Б. Тяпаев

Саратов 2019

Содержание

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 акутальной 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 отличае от других социальных сетей. Также, данная социальная сеть полностью поддерживает законодательство РФ. В работе будет рассмотрен полный путь от знакомста 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 парамаетрами, 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.

Поскольку API — это всегда некие договоренности между двумя командами разработки (backend и frontend), то появилась необходимость введения некоего стандарта, который бы описывал взаимодействие между клиентом и сервером. Введение такого стандарата позволило бы писать четко структурированный код и упростило бы понимание того, какие методы есть у одной команды для другой.

Обычно, API отвечает на запрос к нему в неком формате. В последнее время это два популярных формата XML и JSON. Ниже можно увидеть два одинаковых по смыслу и количеству информации ответы.

JSON

```
{
  "age" : 12,
  "name" : "Danielle"
}
```

XML

```
<person>
  <age>12</age>
  <name>Danielle</name>
</person>
```

Так как XML еще и выполняет функцию языка разметки, до именно для взаимодействия КЛИЕНТ-СЕРВЕР предпочтительнее использовать JSON

2 Ознакомление с официальной документацией API Вконтакте

Итак, документация данной социальной сети находится по слудующему 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. В целом, Long Polling — это технология, которая позволяет получать данные о новых событиях с помощью «длинных запросов». Иначе говоря, 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. Используя этот подход, Вы можете мгновенно отображать в своем приложении важные события. Чтобы использовать Bots Long Poll API, нужно открыть раздел «Управление сообществом» и на вкладке «Работа с API»?«Long Poll API» выбрать «Включён».

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

Также, необходимо создать сообщество в социальной сети 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 добавлеят бота в беседу. Далее администатор 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 неинфрмативных сообщени (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 предидущим пунктом, защищает пользователей от несвоевременного информирования при отключенных уведомлениях.

Последняя команда выводит информацию о доступных командах на данный момент.

Основываясь на вышеизложенных требованиях, а также на том, что бот будет обращаться с сервером социальной сети через LongPoll API, можно выделить следующие технологии и языки программирования, которые помогут решить поставленную задачу:

— Java SDK

— PHP SDK

— Node.js SDK

Поскольку в боте не будет коммерческой составляющей, а также не потребуется работать с ценами и делать различного рода транзакции, то Java не подойдет из-за её громоздкости. Ведь для простой команды, в стиле "привет, бот в Java потребуется написать кучу классов и придется имплиментировать интерфейсы, даже если мне не будут нужны все остальные фукнции.

Что касается PHP и NodeJS [3] — по своей концепции они оба интерпретируемые. Одинаково просты в освоении, но по скорости работы выигрывает PHP. Так как не предполагается, что ботом будут пользоваться огромное количество народа, то данное примущество не слишком влияет на выбор. А вот неблокирующий IO у NodeJS — очень хорошее приемущество. [4] Ведь это означает, что процесс может продолжить выполение не дожидаясь окончания передачи данных. С помощью этого, можно будет делать запросы к сторонним API внутри бота (например, о погоде), получать данные и отдавать их дальше по цепочке для отправки конечному пользователю социальной сети.

В итоге, для достижения полного механизма работы мной были выбраны следующие технологии:

1. enviroment.
   - Node JS. (серверный интрператор 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 взаимодействием 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 учавствует 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 трбуемого приложения. Полученный файл оказался таким

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

Далее, был создан главный js файл — "konnor.js в котором необходимо с помощью библиотеки "vk-node-sdk"подключиться к серверверу 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 поддежку в дальнейшем и поможет делать важные изменения без затрагивания большого количства кода.

Вот как выглядит данный файл сейчас:

```
const vkGroupFullRight = 'group on vk token fith full rights and
    long-poll';
const groupId = 000;

module.exports = {
    vkGroupFullRight: vkGroupFullRight,
    groupId: groupId,
}
```

В главном файле, импортируем этот файл и инициализируем бота.

```

const DEBUG_MODE = require('./konnor_config');
const util = require('util');
const { Bot } = require('node-vk-bot');

//don't forget to add tokens in file and rename him
const TOKENS = require('./secret_tokens');

const regName = /коннор|connor|конор|андроид/i;


const debugConsole = (variable, depth) => {
    DEBUG_MODE && console.log('debug: ' + util.inspect(variable,
        false, depth = 8));
}

const bot = new Bot({
    token: TOKENS.vkGroupFullRight,
    group_id: TOKENS.groupId,
}).start()

console.log('bot started');


bot.on('poll-error', error => {
    console.error('error occurred on a working with the Long Poll
        server ' +
        `(${util.inspect(error)})`)
})
```

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

Прежде чем писать реакцию на команды, в соотвествии с принципами "Банды четырех" [5] — 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:

```
const имя_команды = {
  callName: 'триггер срабатывания, может быть регулярным
     выражением',
  action: (bot, message, TOKENS) => {
    дейсвие - что она делает
  }
}
```

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 ресуры и облегчает добваление новых команд. [6]

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.

```
bot.get(/./, message => {
  if (message.peer_id > 1000000000) { //message from conversation
      if (!regName.test(message.text)) { //if no name calling - no
          answeer
          return;
      }
  }
  bot.api('messages.setActivity', { type: 'typing', peer_id:
     message.peer_id, group_id: TOKENS.groupId })
      .then(res => console.log(util.inspect(res)));

  message.text = message.text.replace(regName, ''); //delete him
```

```
        name

    for (let skillName in skillList) {
        const regExp = RegExp(skillName, 'i');
        if (regExp.test(message.text)) {
            console.log('matched: ' + skillName);
            skillList[skillName](bot, message, TOKENS);
            return;
        }
    }

})
```

На 7-8 строчках утсанавливаем статус "набирает сообщение— что придаст 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:

```
const mentionAll = {
    callName: 'позови всех',
    action: (bot, message, TOKENS) => {
        bot.api('messages.getConversationMembers', { peer_id:
            message.peer_id, group_id: TOKENS.groupId })
            .then(res => {
                //const usersNamesOrIds = res.profiles.map(profile
                    => profile.screen_name != '' ? profile.
                    screen_name : profile.id );
                //console.log(util.inspect(usersNamesOrIds));
                const mentionIds = res.profiles.map(profile => `@id$
                    {profile.id}`);
                bot.send('я призываю всех ' + `${mentionIds.toString
                    ()}`, message.peer_id).catch(
                    function (e) {
                        console.log(e);
                    }
                );
```

```
            })
            .catch(
                function (e) {
                    console.log(e);
                }
            );
    }
}

module.exports = mentionAll;
```

Для расылки всем объявления:

```
  const sendMessageAll = {
    callName: 'об[ъь]явление',
    action: (bot, message, TOKENS) => {
        const alertMessage = message.text
                .split(/об[ъь]явление/i, 2)[1]
                .trim();
            bot.api('messages.getConversationMembers', { peer_id:
               message.peer_id, group_id: TOKENS.groupId })
                .then(res => {
                    const mentionIds = res.profiles.map(profile => `
                       @id${profile.id}`);
                    bot.send(`сообщение для всех: ${alertMessage} ${
                       mentionIds.toString()}`, message.peer_id).
                       catch(
                         function (e) {
                             console.log(e);
                         }
                    );
                })
                .catch(
                    function (e) {
                        console.log(e);
                    }
                );
    }
}

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 опредения числителя и знаменателя - смотрите в приложении.

4.2 Развертка приложения на сервере

4.2.1 Покупка сервера

Прежде всего необходимо купить сервер.

Для этого, идем на сайт https://vscale.io/panel/scalets/ и жмём кнопку 'создать сервер'.

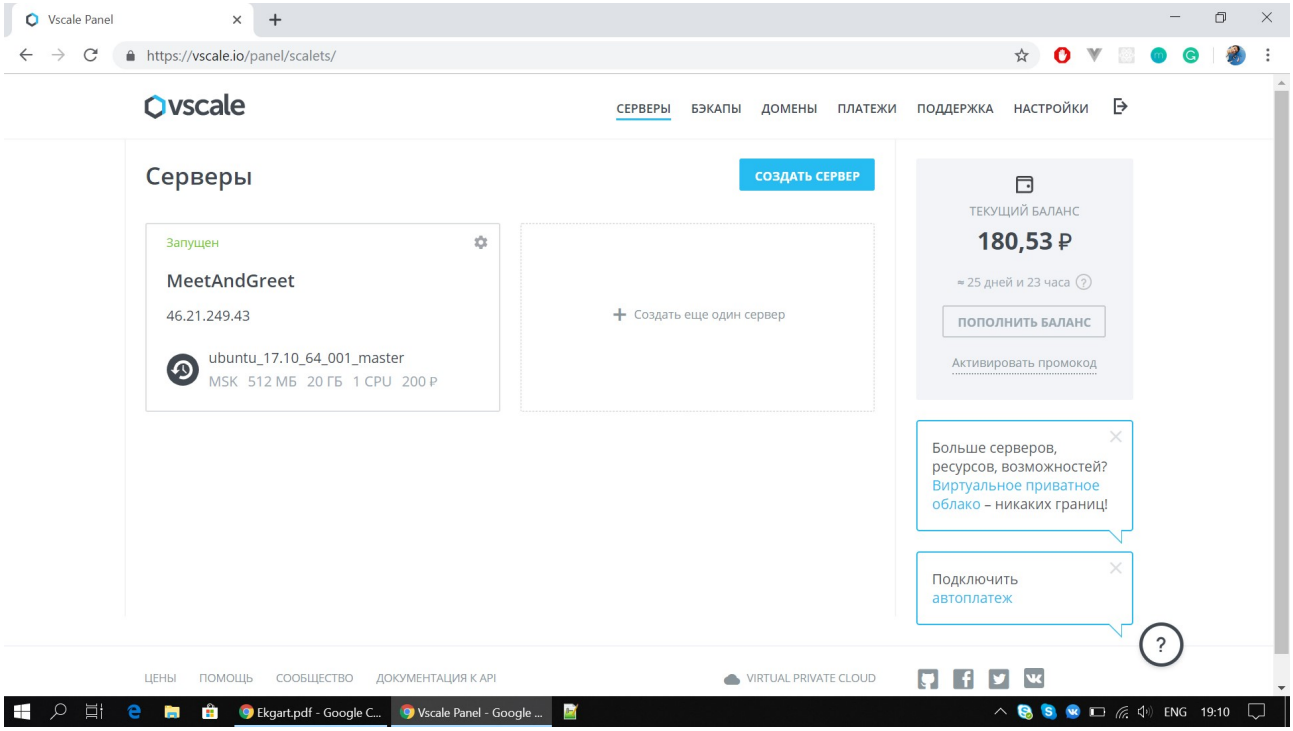

Рисунок 1 – Административная панель управления серверами

После, выбираем необходимую конфигурацию

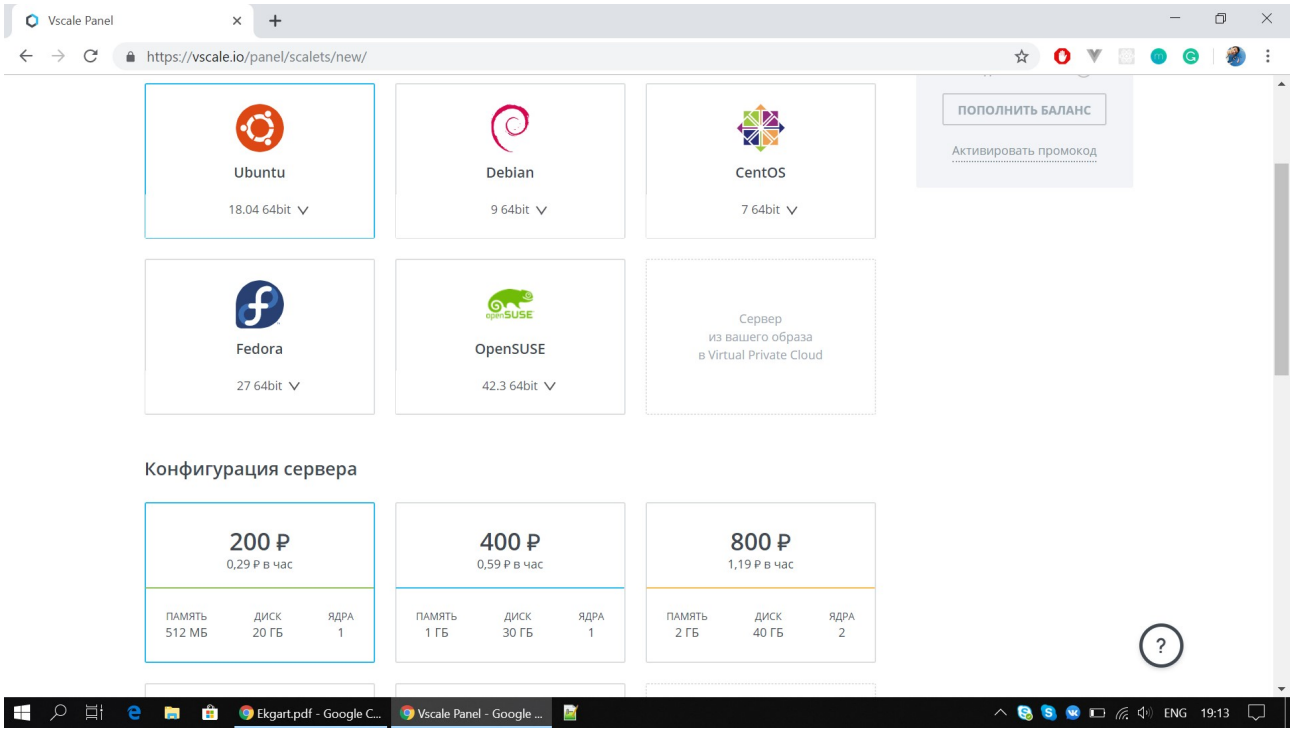

Рисунок 2 – Панель с выбором конфигурации сервера

И выбираем местоположение сервера. Стоит учитывать, что большинство пользователей будут из Саратова, соотвественно наименьший пинг будет до Москвы, так как её географическое положение ближе к середнестатистическому пользователю.

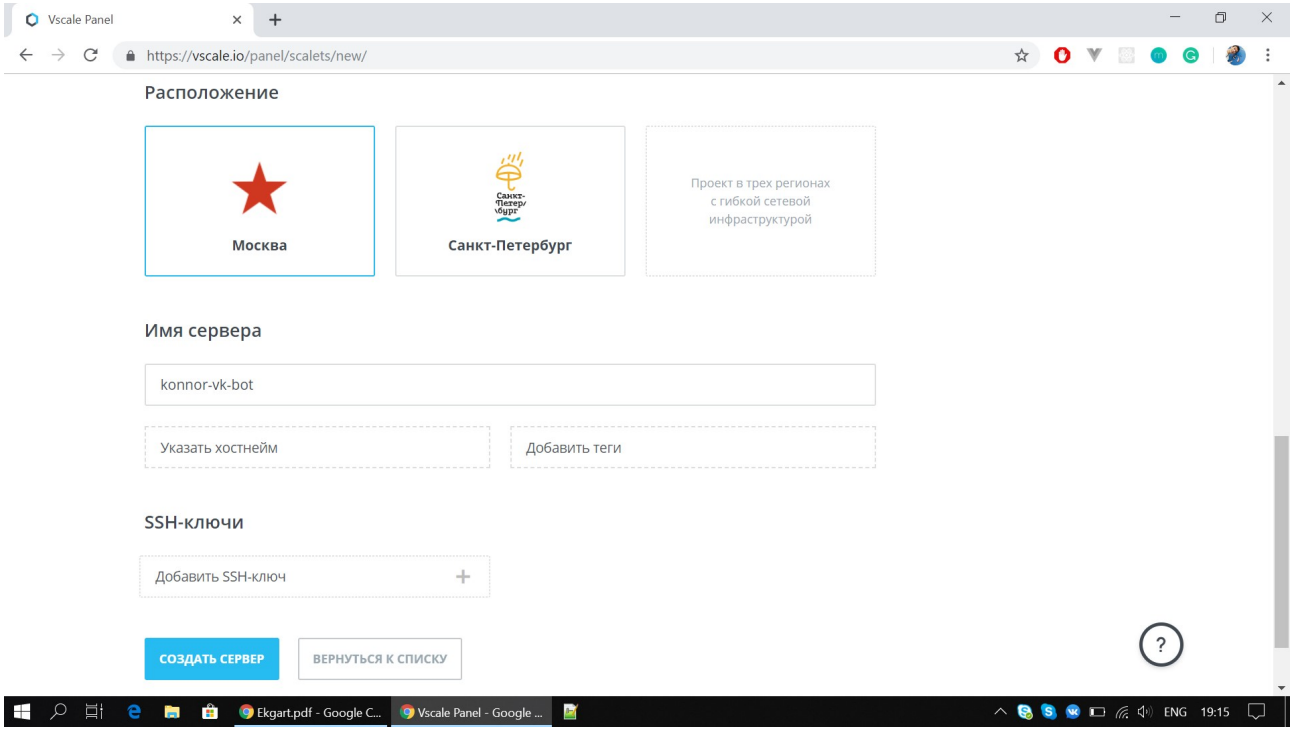

Рисунок 3 – Панель с выбором местоположения сервера.

После создания сервера, на почту, указанную при регистрации, придет письмо с логином и паролем для подлкючения к серверу. Позже, можено будет поменять пароль и завсти отдельную учетную запись без рут прав (в целях безопасности).

4.2.2 Использование сервера

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

IP адрес сервера указан в административной панели управления серверами, а также в письме, которое приходит при создании сервера.

Для подлкючения используем команду

```
1  ssh root@79.143.28.182
```

Где 'root' - имя пользователя, а '79.143.28.182' - адрес сервера. После введения пароля, приступаем к дальнейшей настройке:

Установим библиотеку PM2 глобально с помощью менеджера пакетов NPM:

```
npm i pm2 -g
```

Далее, запустим бота командой:

```
pm2 start konnor.js
```

Теперь, терминал можно зыкрывать - бот запущен и работет.

Для того, чтобы смотреть логи, нужно использовать команду:

```
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. [7]

После тестирования, ошибок не было выявлено.

Ниже на Рисунке 4 и Рисунке 5 приведены скриншоты успешной работы бота.

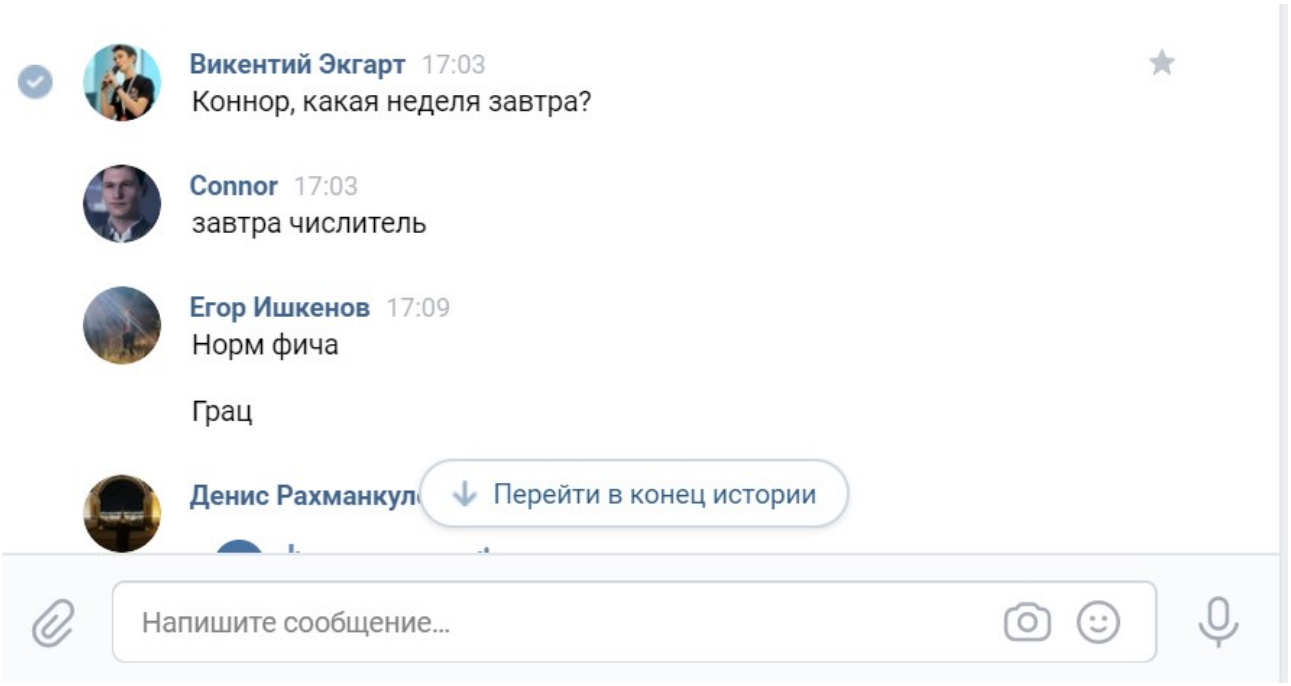

Рисунок 4 – Запрос с целью узнать, какая неделя в расписании будет актуальна завтра.

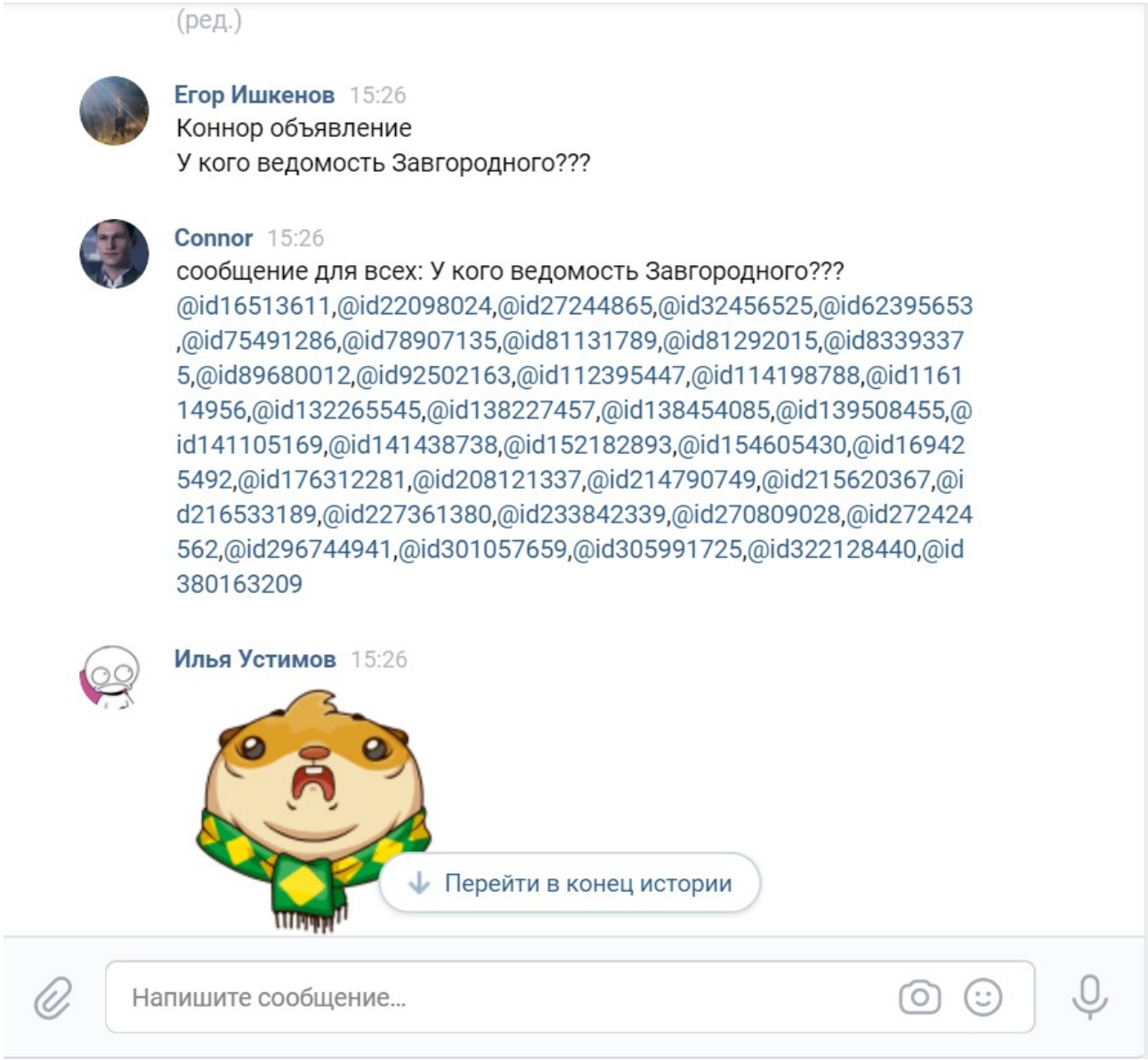

Рисунок 5 – Староста использует возможности бота, чтобы очень быстро обратиться ко всем с важным вопрсом.

5 Приложение

Функция для определения порядка текущей недели - числитель или знаменатель. Используется в модуле 'какая неделя?'

```
const chislOrZnamen = (date) => {
  let now = new Date();
  if (date) {
      now = date;
  }
  //изначально считаем что сейчас числитель
  //true - числитель
  //false - знаменатель

  const dayFirstOfSeptember = (new Date(now.getFullYear(), 8, 1)).
    getDay();
  const dateFirstMondayInSeptember = (dayFirstOfSeptember == 2) ?
    7 : (9 - dayFirstOfSeptember) % 7;

  const dayFirstOfJanuary = (new Date(now.getFullYear(), 0, 1)).
    getDay();
  const dateFirstMondayInJanuary = (dayFirstOfJanuary == 2) ? 7 :
    (9 - dayFirstOfJanuary) % 7;

  const currentDate = now.getDate();
  const currentMonth = now.getMonth();

  const countFullWeeksBeforeNY = Math.floor((now.getTime() - new
    Date(now.getFullYear(), 8, dateFirstMondayInSeptember).
    getTime()) / 86400000 / 7);
  const countFullWeeksAfterNY = Math.floor((now.getTime() - new
    Date(now.getFullYear(), 0, dateFirstMondayInJanuary).getTime
    ()) / 86400000 / 7);

  if (currentMonth > 8 && countFullWeeksBeforeNY % 2 != 0) {
      return 'числитель'
  }
  else if (currentMonth == 8) {
      if (currentDate >= dateFirstMondayInSeptember &&
        countFullWeeksBeforeNY % 2 != 0) {
          return 'числитель'
      }
```

```
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;
```

ЗАКЛЮЧЕНИЕ

Итак, мы имеем бота которого любой пользователь сможет добавить к себе в беседу. И неужели на этом развите бота закончится. Казалось бы - он решает поставленные задачи. Хоть их и не так много. получился такой маленький своеобразный помощник. Но этот бот может решать не только поставленные задачи. Ведь он лишь действует по скриптам и алгоритмам. Да, у меня получился простейший боты, но никто не мешает добавляя множество других модулей сделать его лучше! Например, можно добавить модуль с нейронной сетью, чтобы бот понимал человека или отвечал более человечнее. Ведь для строгого интерфейса взаимодействия у нас есть обычные экраны. А если при общении с ботом стремиться загнать человека в строгие рамки - то толку от такого бота будет мало. Уж проще написать приложение с UI. Бот, который общается с человеком должен стараться быть максимально удобным. Именно поэтому важно тестирование и пользовательский опыт.

Получившийся бот взаимодействует с API социальной сети, помогает старосте (и не только), а также поднимает настроение остальным участникам беседы.

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 PHP vs NodeJS Comparison and Benchmark [Электронный ресурс]: URL: https://thinkmobiles.com/blog/php-vs-nodejs/ (Дата обращения: 19.02.2019) Загл. с экрана. Яз. англ;

4 Synchronous and Asynchronous I/O [Электронный ресурс]: URL: https://docs.microsoft.com/ru-ru/windows/desktop/FileIO/synchronous-and-asynchronous-i-o (Дата обращения: 19.02.2019) Загл. с экрана. Яз. англ;

5 Приемы объектно-ориентированного проектирования. Паттерны проектирования. Влиссидес Джон , Джонсон Р. , Хелм Ричард , Гамма Эрих Яз. англ стр [50-120];

6 JavaScript. Шаблоны . Стоян Стефанов Яз. рус. стр [70-90];

7 Тестирование dot com . Савин Роман Яз. рус. стр 158;