Предыстория

Шел первый день отпуска, тратить время отлеживаясь не хотелось совершенно. Было принято решение достать из долгого ящика новостное сообщество, да так, чтобы оно туда больше не попадало.
Человек по своей природе непостоянен, его эмоциональное состояние постоянно меняется: то хочется заниматься одним, то уже другим, то вообще ничем.

Не долго думая, я решил написать для сообщества бота, который был бы надежнее меня и занимался группой постоянно, в любой день.

Сказано — сделано.

За основу был выбран Node.js. Если нужно будет работать с API соц. сетей (на будущее не с одной) и базой данных, то я посчитал, что асинхронность будет кстати. А

набор из множества готовых модулей упростит процесс разработки.

В качестве базы данных выбор пал на SQLite 3, так как в ней есть все необходимое и ничего лишнего.

Писать буду на CoffeeScript. Почему? (Возможно субъективно, но) JavaScript - хороший парень внутри, но снаружи жуткий страшила.

Конечно, в конце-концов все транслируется в JS перед запуском, но нет необходимости работать с ним напрямую. К тому же, я верю в то, что транслятор CS создаст более оптимизированную версию кода JS, чем моя собственная версия.

Набросок архитектуры получился таким:

Подготовка

Создаю папку linews. В ней будет размещаться приложение.

Создаю там package.json с зависимостями:

```
{
  "name": "LinewsBot",
  "version": "1.0",
  "description": "LinewsBot",
  "main": "app.js",
  "dependencies": {
    "config": "^1.19.0",
    "express": "^4.13.4",
    "request": "^2.69.0",
    "sqlite3": "^3.1.1"
  },
  "devDependencies": {
    "gulp": "^3.9.1",
    "gulp-coffee": "^2.3.1",
    "jasmine": "^2.4.1"
  },
  "license": "MIT"
}
```

Про тесты jasmine, кстати, рассказывать не буду. Не в них суть, но в проекте они есть, потому что без BDD сейчас ~~и в туалет не сходить~~ нельзя.

Устанавливаем:

```
npm install
```

Некоторые вещи необходимо будет поставить глобально:

```
npm install gulp -g
npm install jasmine-node -g
```

Создаем gulpfile.js в корне.

Оформляем его:

```
var gulp = require('gulp'),
        coffee = require('gulp-coffee');

gulp.task('coffee', function() {
        gulp.src('./coffee/**/*.coffee')
                .pipe(coffee({bare: false}))
                .pipe(gulp.dest('./'));
});

gulp.task('watch', function() {
        gulp.watch('./coffee/**/*', ['coffee']);
});
```

В корень создаем папки logs и configs (логи и конфиги).

В configs создаем default.json.

```
{
        "database": {
        "statuses_table": "statuses"
        },
        "common": {
                "vk_token": "mytoken",
                "group_id": "mygroup_id"
        },
        "backend": {
                "port": 9999
        }
}
```

Создаем папку coffee в корне проекта.

Внутри coffee создаем tasks и helpers.

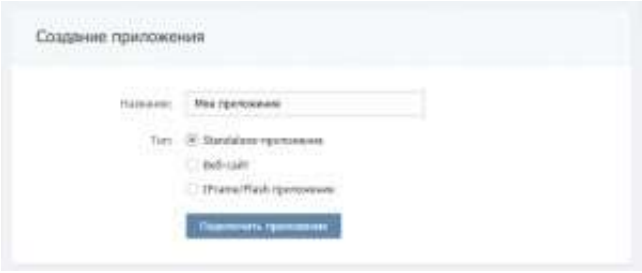
Позже, когда мы напишем скрипты на CoffeeScript и запустим Gulp, он транслирует все в JavaScript, самостоятельно создав необходимые папки в корне.

Про VK api

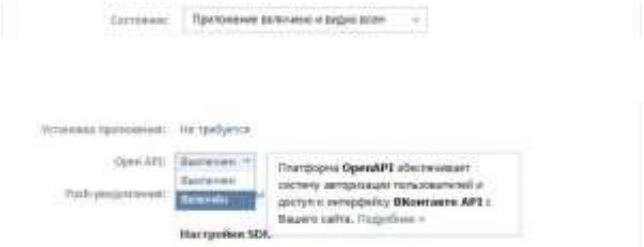
В ходе разбора VK api сбил с ног. Большая часть функционала только для standalone-приложений. Весь функционал, который был необходим мне, требует пользовательской авторизации.

Разбирался неприлично долго, не без помощи со стороны, но решение все же нашел.

Создаем standalone-приложение в VK.



Включаем его.



В настройках, в адресах, указываем адрес бота на дев-машине (для тестов) и на продакшене.

После переходим по ссылке авторизации:

```
https://oauth.vk.com/authorize?client_id=#{app_id}&scope=status,offline&v=5.45
```

Где #{app_id} - id вашего приложения.

Из хэша забираем токен (access_token), время действия — долго (как мне сказали, пока не поменяете пароль аккаунта).

Теперь смотрим как нужно делать запросы к api: https://vk.com/dev/api_requests.

В итоге рисуется такая строка:

```
https://api.vk.com/method/status.set?group_id=#{config.common.group_id}&text=#
```

Меняем все #{переменные} необходимыми параметрами и кидаем в браузер.

Статус поменялся - хорошо. Если нет, то убедитесь, что все правильно сделали. Также убедитесь, что у вас есть права в той группе, где вы собираетесь поменять

статус.

В конце кидаем токен в конфиг (vk_token), вместе с id группы (group_id), пусть лежат.

Пишем app.js

Подключаем express и config.

```
express = require 'express'
app = do express
config = require 'config'
```

Подключаем задачи (саму задачу будем писать ниже):

```
setStatus = require './tasks/setStatus'
```

Прописываем правила в маршрутизаторе (нашу задачу поставим на путь «setstatus»).

```
#init routes
app.get '/setstatus', (req, res) -> setStatus(req, res)

#last route
app.get '*', (req, res) -> res.status(404).send 'error 404'
```

Последним ставлю все пути — так, если маршруты выше не подойдут для пути запроса, мы хоть что-то вернем.

Дальше ставим приложение слушать порт (в моем случае 9999, порт указывается в конфиге).

```
app.listen(
        config.backend.port
        ->
                console.log "Server is listening on port #{config.backend.port
)
```

Полностью app.js должен выглядеть так:

```
express = require 'express'
app = do express
config = require 'config'

#include tasks
setStatus = require './tasks/setStatus'

#init routes
app.get '/setstatus', (req, res) -> setStatus(req, res)

#last route
app.get '*', (req, res) -> res.status(404).send 'error 404'

app.listen(
        config.backend.port
        ->
                console.log "Server is listening on port #{config.backend.port
)
```

Настраиваем SQLite

В папке config создаю файл status.db простым блокнотом.

Захожу в SQLite-менеджер (не обязательно пользоваться им, можно просто отправить запросы на создание таблицы и запись в нее к SQLite, в какой-либо задаче).

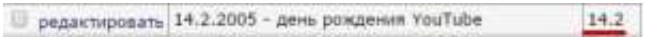
Создаю таблицу statuses с полями status и date, записываю название в конфиг.

Поле status, как ни странно, будет содержать статус.

А поле date будет содержать дату публикации статуса.

Поле date, естественно, текстовое, но на формат данных в нем будут жесткие ограничения.

Необходимо будет записывать дату строго формата «день.месяц», без ведущих нулей.



Это потому что (new Date()).getMouth() в JavaScript возвращает месяц без ведущих нулей (я ленивый и преобразовывать не хотел), да и хранить дополнительные нули в бд зачем?

Пишем первую задачу: set status

Создаем SetStatus.coffee в Coffee/Tasks

Подключаем request.js.

```
request = require "request"
```

Подключаем sqlite3.

```
sqlite3 = do require("sqlite3").verbose
```

Подключаем config.

```
config = require "config"
```

Присваиваем module.exports анонимную функцию.

```
module.exports = (req, res) ->
```

Внутри нам необходимо открыть соединение с БД.

```
db = new sqlite3.Database("#{__dirname}/../config/status.db")

db.serialize  ->
```

Получить текущую дату (код инкапсулирован в helpers/data.coffee).

```
module.exports.toDay = ->
        date  = new Date()
        mouth = do date.getMonth + 1
        day   = do date.getDate

        return "#{day}.#{mouth}"
```

Убедиться SQL-запросом, что у текущей даты есть готовый статус. Заодно и вытащим если есть.

```
db.get(
    "SELECT rowid AS id, status, date FROM #{config.database.statuses_table} W
    (error, row) ->
)
```

Отправить статус в vk API по уже знакомой ссылке через request.js.

```
if row && row.status
        str = encodeURIComponent row.status
        url = "https://api.vk.com/method/status.set?"
        url += "group_id=#{config.common.group_id}"
        url     += "&text=#{str}"
        url     += "&access_token=#{config.common.vk_token}"

        request(
                url
                (err, head, body) ->
                        if err
                                toLog "Request error: #{today} -> #{err}"
                        else
                                toLog "Request body: #{today} -> #{body}"
        )
```

В колбеке записать в лог ошибку или ответ от сервера vk api (код инкапсулирован в helpers/logs).

```
module.exports.writeTo = (file, data) ->
        fs.writeFile(
                (__dirname + '/' + file),
                data,
                (err) -> if(err) then console.log err
        )
```

Полный код задачи:

```
request = require "request"
config  = require "config"
log     = require "../helpers/logs"
date    = require "../helpers/date"
sqlite3 = do require("sqlite3").verbose

toLog   = (data) -> log.writeTo "../logs/status.log", data

module.exports = (req, res) ->
        db = new sqlite3.Database("#{__dirname}/../config/status.db")
        db.serialize( ->
                today = do date.toDay

                db.get(
                        "SELECT rowid AS id, status, date FROM #{config.databa
                        (error, row) ->
                                if error then toLog "SQLite Error: #{today} ->

                                if row && row.status
                                        str = encodeURIComponent row.status
                                        url = "https://api.vk.com/method/statu
                                        url += "group_id=#{config.common.group
```

```
                                        url += "&text=#{str}"
                                        url += "&access_token=#{config.common.

                                        request(
                                                url
                                                (err, head, body) ->
                                                        if err
                                                                toLog "Request
                                                        else
                                                                toLog "Request
                                        )
                )
        )

        do db.close
        do res.end
```

Если все было правильно, то запустив node приложение, и обратившись к порту 9999 по пути /setstatus, отработает наша задача и, в зависимости от текущей даты, в сообществе изменится статус.

Запуск

Далее я установил бота на сервер.

Запустил под модулем forever, чтобы приложение пере-запускалось, если что-то пойдет не так.

```
npm install forever -g
forever start app.js
```

Закрыл порт 9999 извне с помощью iptables и открыл только для 127.0.0.1 .

```
iptables -A INPUT -p tcp -s 127.0.0.1 --dport 9999 -j ACCEPT
iptables -A OUTPUT -p tcp -s 127.0.0.1 --dport 9999 -j ACCEPT
iptables -A INPUT -p tcp --dport 9999 -j DROP
iptables -A OUTPUT -p tcp -dport 9999 -j DROP
```

Установил в планировщик Cron обращение к боту через wget –spider на полночь по

времени сервера.

```
crontab -e
```

```
0 0 * * * wget --spider 127.0.0.1:9999/setstatus
```

И все заботы теперь — обновлять базу статусов по настроению.

Нужно придумать что-то с публикациями...

Бот на GitHub

https://github.com/alexander-shibisty/linews-bot