УДК 004.624

Бот поиска рейсов самолетов на базе Twitter

Кизянов Антон Олегович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс создания бота в twitter, способный искать рейсы полетов самолетов. Для создания потребуется учетная запись в twitter. Созданный бот позволяет найти рейсы самолётов по запросу через хештег в сообщении на twitter.
Ключевые слова: Twitter, бот

Airplane flight search bot based on Twitter

Kizyanov Anton Olegovich
Sholom-Aleichem Priamursky State University
student

Abstract
This article describes the process of creating a twitter bot capable of searching for aircraft flights. You will need a twitter account to create. The created bot allows you to find airplane flights on request through a hashtag in a message on twitter.
Keywords: Twitter, bot

Twitter — это онлайн-служба социальной сети, которая позволяет пользователям отправлять и читать короткие 140-символьные сообщения, называемые твитами, и стала одним из самых популярных способов обмена новостями и информацией во всем мире.

Популярность Twitter резко возросла с момента его создания, и теперь он используется для самых разных целей, таких как обслуживание клиентов, маркетинг, освещение новостей и многие другие. Это один из самых популярных существующих веб-сайтов, который считается SMS-шлюзом в Интернете.

Одно из основных направлений использования Twitter — это передача компаниями информации своим подписчикам. Например, авиакомпании обычно пишут в Твиттере о событиях, связанных с компанией, а также о событиях, которые могут повлиять на пассажиров или их планы.

В этой статье будет создан бот Twitter, который может предоставлять пассажирам информацию о рейсах.

Цель исследования – создать бота по поиску рейсов самолетов через хештег в twitter.

Ранее этим вопросом интересовались Д.Е. Намиот развивал тему «Twitter как транспорт в информационных системах» [1] в которой поднимаются вопросы использования механизмов Twitter для создания информационных систем. В свете перехода мобильных абонентов от SMS к независимым приложениям для обмена сообщениями, как мы можем использовать Twitter для построения информационных систем? Описывается модель, которая позволяет использовать Twitter вместо традиционных в мобильных сервисах систем на базе SMS. М.Г. Хачатрян с темой «Обнаружение ботов в онлайновой социальной сети twitter с помощью алгоритма машинного обучения "случайный лес"» [2], а подробнее про возможность использования алгоритма машинного обучения «Случайный лес» для обнаружения ботов в социальной сети «Twitter». Цель исследования – дать количественную оценку точности обнаружения ботов. Было проведено тестирование алгоритма методом кросс-валидации по десяти блокам на выборке из нескольких тысяч аккаунтов Twitter, состоящей как из настоящих пользователей, так и из различных видов ботов. В результате проведенного тестирования было получено значение F_1-метрики равной 0.982. Данное значение является довольно высоким показателем точности обнаружения ботов по сравнению с большинством предыдущих работ по обнаружению ботов в онлайновой социальной сети Twitter. М.Г. Хачатрян, П.И. Чепик опубликовали статью «Обнаружение ботов в социальных сетях с помощью многослойного перцептрона» [3] описали возможность использования многослойного перцептрона для обнаружения ботов в социальных сетях. Цель исследования — дать количественную оценку качества обнаружения ботов многослойным перцептроном. С помощью алгоритма кросс-валидации определены оптимальные гиперпараметры для многослойного перцептрона и значение качества классификации при найденных гиперпараметрах. Обучение и тестирование многослойного перцептрона выполнено на основе выборки из нескольких тысяч аккаунтов Twitter, состоящей как из настоящих пользователей, так и из ботов двух различных видов. В результате проведенного тестирования на этих данных получено значение метрики = 0,958.

Как и любой другой бот, бот Twitter, по сути, является просто еще одной учетной записью пользователя Twitter. Разница в том, что учетная запись управляется не другим человеком, а автоматизированным процессом, который знает, как отвечать на вводимые данные. Это возможно, потому что Twitter предоставляет API, который позволяет программно взаимодействовать со службой через код.

По сути, все, что можно превратить в сервис, можно превратить в автоматизированный диалог с помощью бота, и Twitter не исключение. Боты могут вести интерактивные беседы практически на любой платформе, в любое время и в любом месте.

Бот для Twitter обычно представляет собой приложение, которое ожидает, что что-то происходит в Twitter, а затем отвечает на это. Бот будет ждать твита с определенным хештегом, а потом отвечать на него. Этот

хэштег будет номером рейса, и бот сможет предоставить обратную связь на его основе.

Первый и самый важный шаг в создании бота Twitter — это собственно создание приложения Twitter. Бот — это просто обозначение, которое будет использовать, но на самом деле это скрытое приложение Twitter, которое может взаимодействовать с API Twitter.

Чтобы иметь возможность взаимодействовать с Twitter API, необходимо иметь зарегистрированную учетную запись Twitter. Нужно перейти на веб-страницу Twitter и зарегистрироваться, если нет учетной записи.

После входа в систему нужно перейти на https://apps.twitter.com/ . Здесь регистрируется приложение Twitter. Чтобы создать бота Twitter, нужно нажать на кнопку «Создать новое приложение» как на рисунке 1.

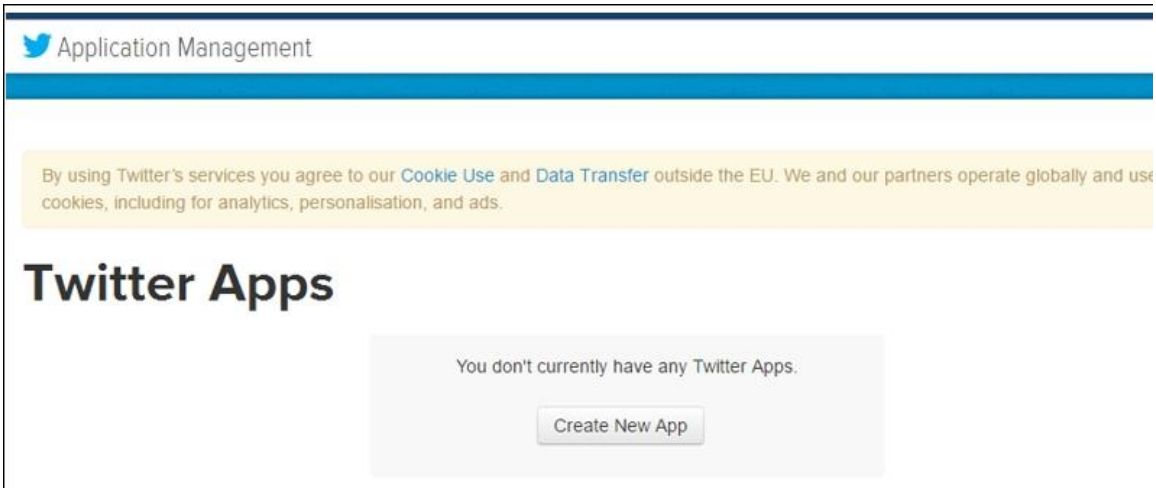


Рис. 1 Первый шаг создания бота Twitter

После появится следующий экран как на рисунке 2.

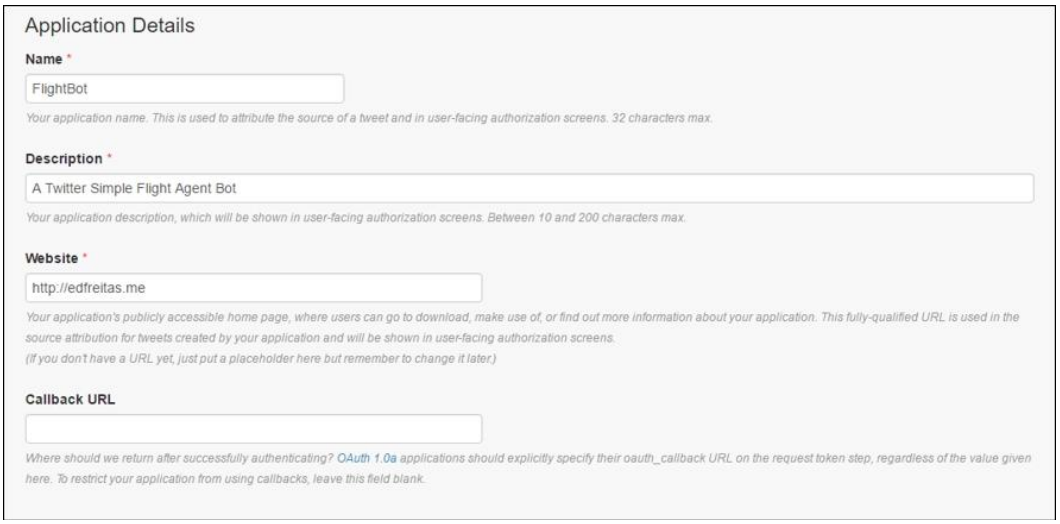


Рис. 2 Форма создания бота Twitter

У бота должны быть имя, описание и ссылка на web-сайт в качестве домашней страницы, последним шагом будет принятие условий разработчика внизу страницы как на рисунке 3.

Рис. 3 Флажок условий разработчика

После ознакомления с соглашение разработчика нужно нажать на кнопке «Create your Twitter application», чтобы продолжить. Это позволит Twitter создать приложение, и после завершения процесса создания будет следующий экран как на рисунке 4.

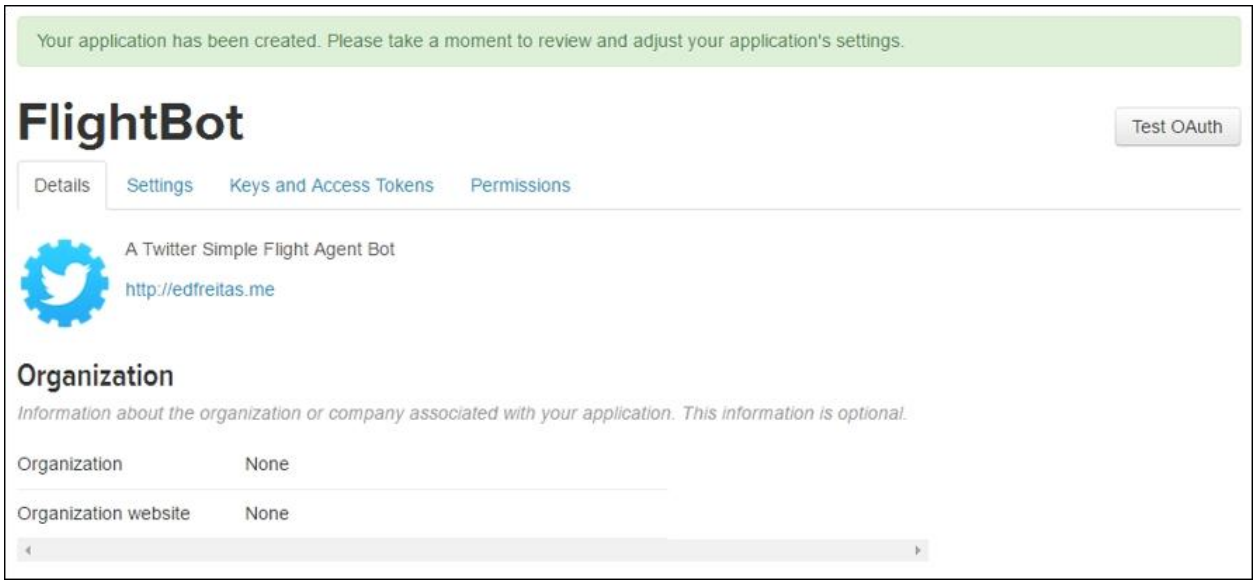


Рис. 4 Настройки бота Twitter

Для дальнейшей работы необходим nodejs. Скачать его можно по адресу https://nodejs.org/. Также нужно создать отдельную папку FlightBot и создать в ней файл package.json с содержимым как на рисунке 5.

package.json app.js

```
{
  "name": "package",
  "version": "1.0.0",
  "description": "FlightBot",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Ed Freitas",
  "license": "ISC"
}
```

Рис. 5 Файл package.json

Теперь нужно установить зависимости, чтобы можно было работать с Twitter API.

```
npm install twitter --save
```

Это установит Twitter Node.js SDK, который будет использоваться для написания бота. Он будет установлен в FlightBot папку, как показано на рисунке 6.

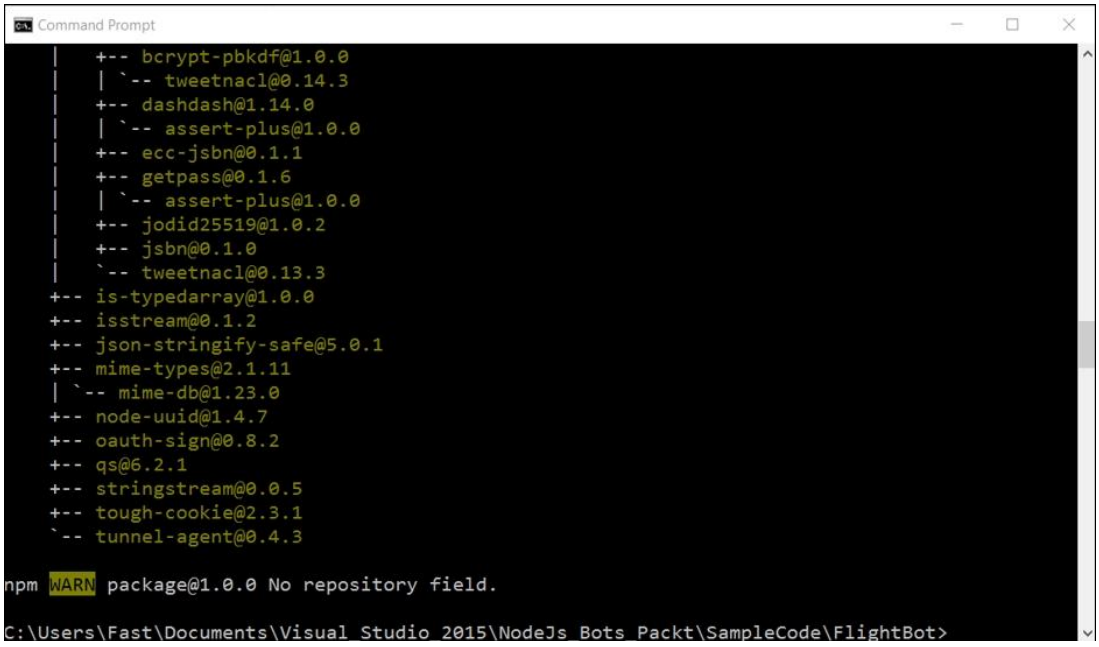


Рис. 6 Процесс установки Twitter SDK

Теперь нужно создать файл app.js и заполнить следующим.

```
var TwitterPackage = require('twitter');
```

Здесь скрипт загружает и импортирует пакет Twitter. Дальше нужно получить ключи доступа к Twitter. На вкладке «Keys and Access Tokens» содержатся ключи, которые нужны для авторизации бота в Twitter, пример страницы показан на рисунке 7.

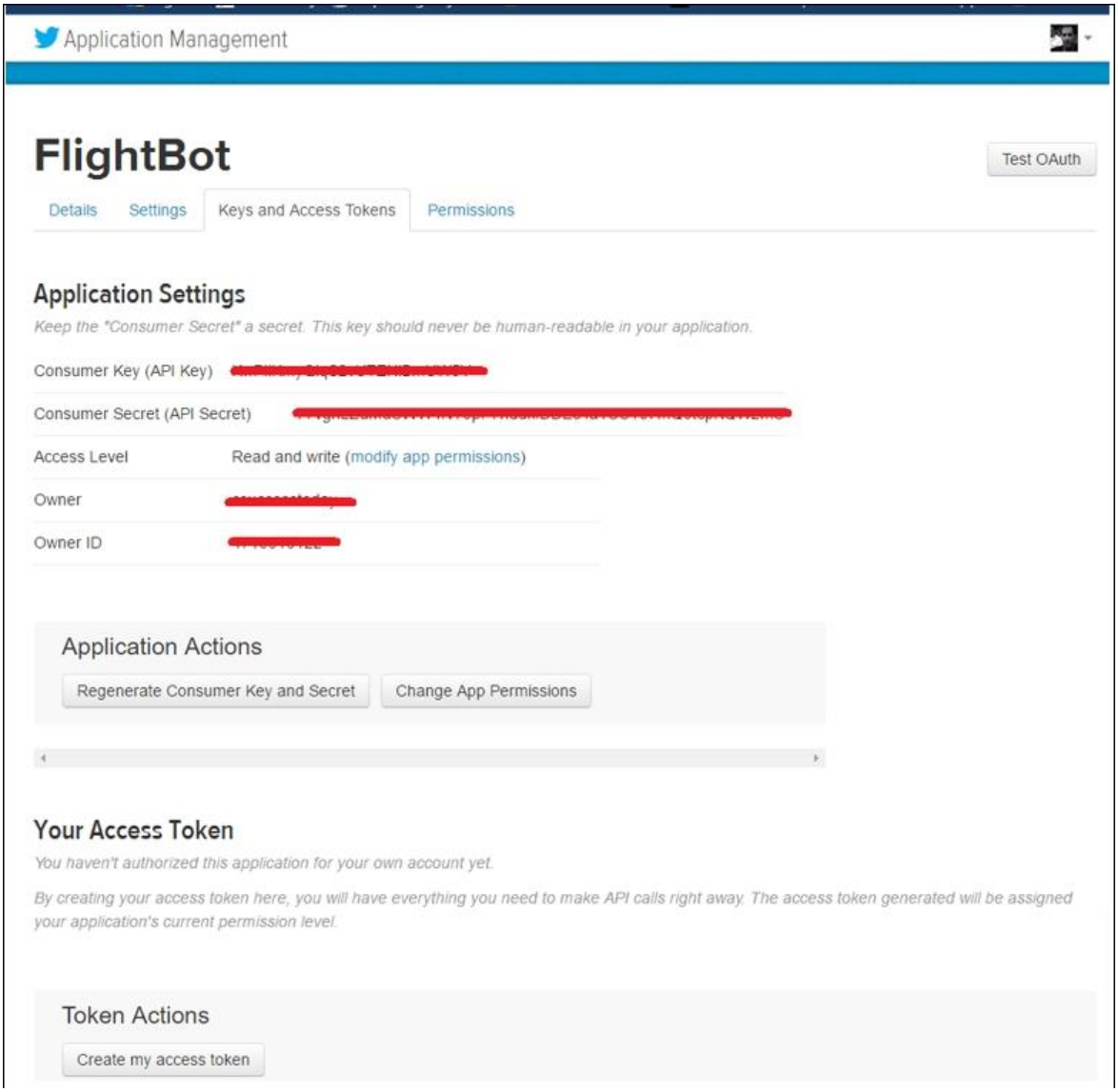


Рис. 7 Страница с ключами

Следует обратить внимание, что Consumer Key и Consumer Secret доступны по умолчанию, но не токены доступа. Чтобы получить токены доступа, нужно нажать кнопку «Создать мой токен доступа» в нижней части экрана. Как только это будет сделано, появится экран как на рисунке 8.

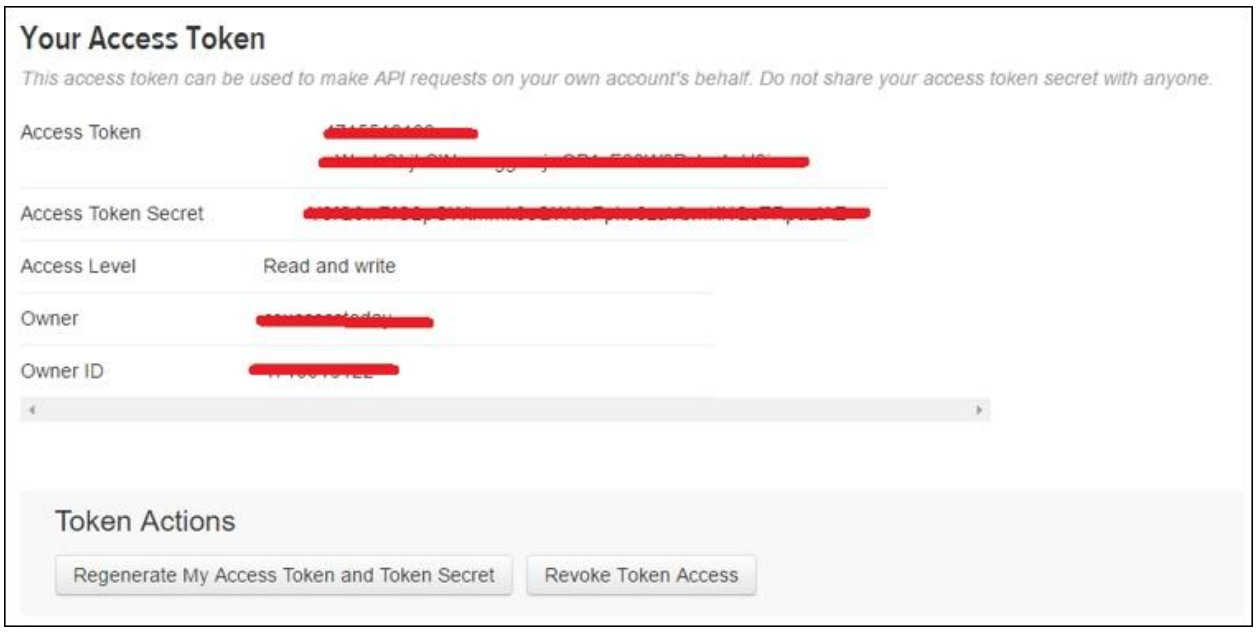


Рис. 8 Страница с токенами доступа

После этого можно добавить токены доступа как переменную в коде. Это можно сделать, определив объектную переменную, которая будет содержать ключ потребителя и секретный объект вместе с токенами доступа, как показано в следующем фрагменте кода:

```
var secret = {
  consumer_key: 'PUT YOURS',
  consumer_secret: 'PUT YOURS',
  access_token_key: 'PUT YOURS',
  access_token_secret: 'PUT YOURS'
}
var Twitter = new TwitterPackage(secret);
```

Позже они будут сохранены в отдельном Json файле, а пока они будут в файле app.js. Они потребуются для аутентификации в Twitter.

Итак, на данный момент приложение в значительной степени создано и имеет очень простую структуру с токенами и кодами доступа, которые используются для аутентификации в службе Twitter. Следующим шагом будет добавление логики в приложение.

Чтобы добавить собственную логику, нужно будет использовать RESTAPI Twitter, который позволит делать несколько вещей. Одна из вещей, которую он может сделать, — это позволить публиковать твиты. Этого можно добиться следующим образом.

```
Twitter.post ('statuses/update', {status: 'This is a simple automated Tweet'},
function (error, twit, response) {
  if (error) {console.log (error);}
```

```
  console.log (twit);
  console.log (response);
});
```

Twitter.post означает, что вызывается post функция в объекте Twitter. Функции передается 'statuses/update', это означает, опубликовать обновление статуса (твит).

{status: 'This is a sample automated Tweet'} — это объект JavaScript, который передается этой функции, где устанавливается статус отправляемого твита.

Хотя он содержит только текст твита, который нужно отправить, существует целый ряд других параметров, которые можно установить в зависимости от того, что нужно опубликовать в Twitter (например, изображения, местоположение и т. д.).

Последнее, что нужно передать, — это функция. В JavaScript можно передавать функции другим функциям; это одна из вещей, которые делают JavaScript функциональным языком программирования.

Twitter.post ожидается, что в этой функции будет передана функция, которая будет выполнена после того, как Twitter попытается опубликовать твит. Это так называемая функция обратного вызова. В этой функции есть три параметра:

1. error: указывает, есть ли ошибка в процессе публикации твита, и в этом случае эта переменная будет содержать объект с информацией о произошедшей ошибке.
2. tweet: Объект, содержащий все данные твита.
3. response: Объект фактического ответа, который Twitter отправляет обратно, когда публикуется твит.

В данном коде просто публикуется твит, а затем печатается в консоли. Теперь нужно удалить 'Hi, this is FlightBot' линию, так как это больше не понадобится, получившийся файл app.js должен получиться так.

```
var TwitterPackage = require('twitter');
var secret = {consumer_key: 'PUT YOURS', consumer_secret: 'PUT YOURS',
  access_token_key: 'PUT YOURS', access_token_secret: 'PUT YOURS'}
var Twitter = new TwitterPackage(secret);
Twitter.post('statuses/update', {status: 'This is a sample automated Tweet'},
function(error, tweet, response){
  if(error) {console.log(error);}
  console.log(tweet); console.log(response);
});
```

Теперь нужно запустить приложение:

```
node app.js
```

Это приведет к следующему результату как на рисунке 9.

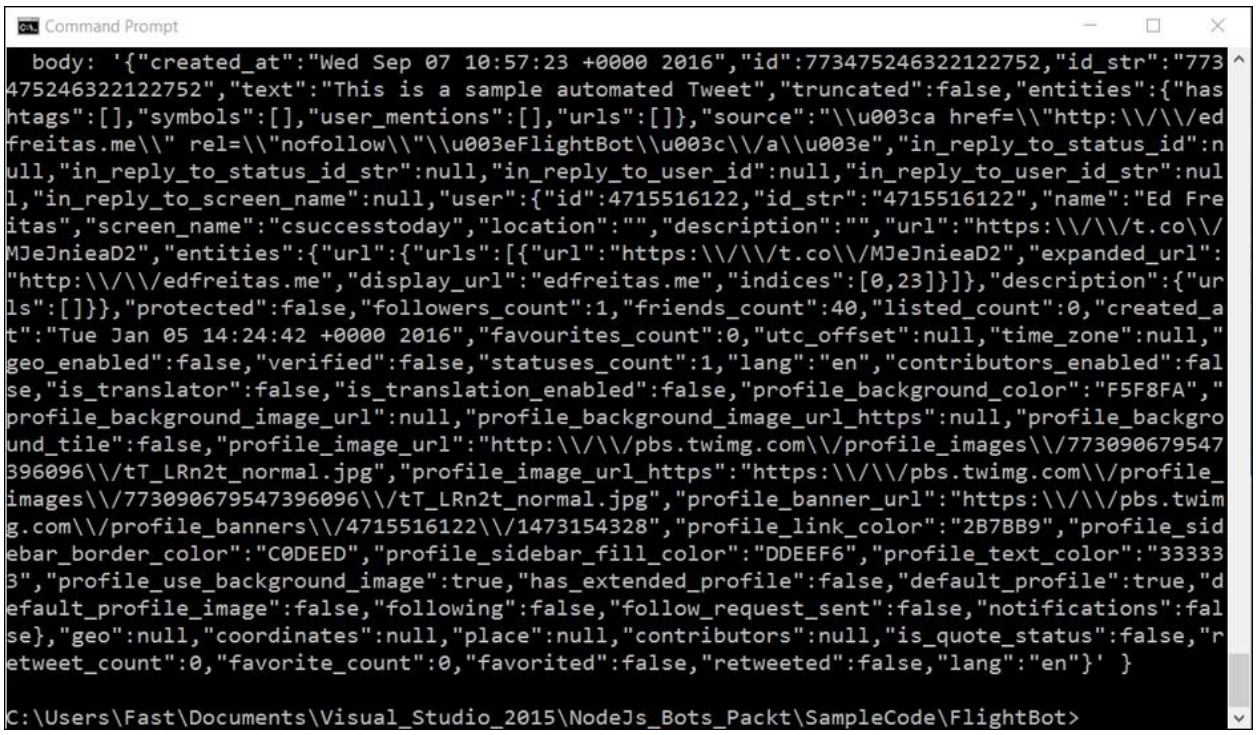


Рис. 9 Вывод в консоль выполнение app.js

После этого следует проверить наличие твита, он должен выглядеть как на рисунке 10.

Рис. 10 Пример первого твита

Чтобы создать функционального бота Twitter, недостаточно просто опубликовать что-то в Twitter. Нужно также прослушивать, что публикуется в Twitter.

У Twitter есть очень полезный API, Streaming который дает информацию о твитах в режиме реального времени. Другими словами, когда кто-то пишет в Твиттере что-то, бот получает все данные об этом твите. Это действительно полезно и действительно здорово.

Такой функционал реализуется следующим образом.

```
Twitter.stream('statuses/filter', {track: '#FlightBot'}, function(stream) {
  stream.on('data', function(tweet) {console.log(tweet.text);});
  stream.on('error', function(error) {console.log(error);});});
```

Twitter.stream Функция принимает три параметра:

Первый параметр — это строка, которая сообщает Twitter, что бот хочет отслеживать статусы с помощью определенного фильтра. В этом случае выполняется фильтрация с помощью хэштега.

Второй параметр — это то место, где определен сам фильтр с объектом. Этот объект содержит свойство «track» которое позволяет определять слово, хэштег или фразу. В данном примере бот будет отслеживать, когда кто-то пишет в Твиттере с хэштегом '#FlightBot'.

Последний параметр — это функция, которая вызывается, когда Twitter завершает настройку потока. Когда он завершит настройку потока, он передает этот объект потока функции. В этой функции можно настроить, что будет происходить при получении твита, а также другие вещи, такие как обработка ошибок и так далее.

Теперь подробнее рассмотрим, что происходит, когда приходят данные.

```
stream.on('data', function(tweet) {
  console.log(tweet.text);
});
```

Итак, используя «stream» объект, он вызывает «on» функцию. Теперь с помощью «on» функции нужно передать строку и функцию. Это означает что, когда происходит твит, бот вызываем эту функцию с этими данными. На данный момент бот просто печатает результат tweet.text, и именно так на данным момент бот получает доступ к фактическому тексту полученного твита, в котором использовался хэштег '#FlightBot'.

Если сохранить app.js файл, а затем вызвать node app.js командную строку, можно заметить, что в командной строке больше не отображается приглашение.

Это потому, что он работает и ожидает поступления каких-то данных из этого потока. Если нужно остановить его, нужно нажать Ctrl + C несколько раз, чтобы вернуться к подсказке.

Чтобы проверить это, нужно отправить любой твит с '#FlightBot', как показано на рисунке 11.

Рис. 11 Пример твита с #FlightBot

Теперь нужно проверить запущенную командную строку. Там должен распечататься твит, как на рисунке 12.

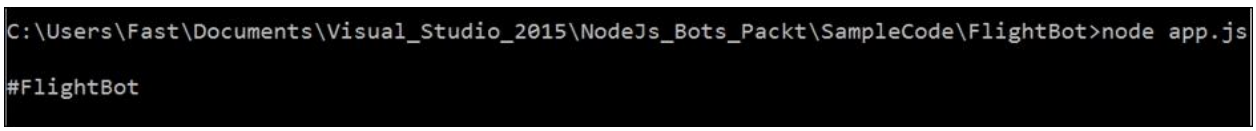


Рис. 12 Пример получения твита

Полная версия кода представлена ниже.

```
var TwitterPackage = require('twitter');
var secret = {
  consumer_key: 'PUT YOURS',
  consumer_secret: 'PUT YOURS',
  access_token_key: 'PUT YOURS',
  access_token_secret: 'PUT YOURS'
}
var Twitter = new TwitterPackage(secret);
Twitter.stream('statuses/filter', {track: '#FlightBot'}, function(stream) {
  stream.on('data', function(tweet) {console.log(tweet.text);});
  stream.on('error', function(error) {console.log(error);});});});
```

На данный момент бот умеет прослушивать поток твитов и находить для себя и создавать нужные твиты. Следующее, что нужно сделать, это объединить обе части вместе в единую базу кода, так как это будет базовый уровень бота. Можно получить доступ к имени пользователя человека, который написал твит с нужным хэштегом, используя tweet.user.screen_name.

Чтобы упомянуть их нужно использовать символ '@' выполнив следующие действия.

```
var mentionString = '@' + tweet.user.screen_name;
```

Затем просто присоединить это к строке, которую нужно опубликовать. Теперь бот отвечает человеку, который написал твит.

Итак, ниже представлен полный код, чтобы получить полную картину:

```
var TwitterPackage = require('twitter');
var secret = {
  consumer_key: 'PUT YOURS',
  consumer_secret: 'PUT YOURS',
  access_token_key: 'PUT YOURS',
  access_token_secret: 'PUT YOURS'
}
var Twitter = new TwitterPackage(secret);
Twitter.stream('statuses/filter', {track: '#FlightBot'}, function(stream) {
  stream.on('data', function(tweet) {
    console.log(tweet.text);
    var statusObj = {status: "Hi @" +
    tweet.user.screen_name + ", Thanks for reaching out. How are you?"}
    Twitter.post('statuses/update', statusObj, function(error,
    tweetReply, response) {
      if(error) {console.log(error);}
      console.log (tweetReply.text);});});});
  stream.on('error', function(error) {console.log(error);});});});
```

Если запустить приложение с помощью node app.js, можно получать любые хэштеги с ключевым словом '#FlightBot'. Как на рисунках 13 и 14.

Рис. 13 Пример вызова бота

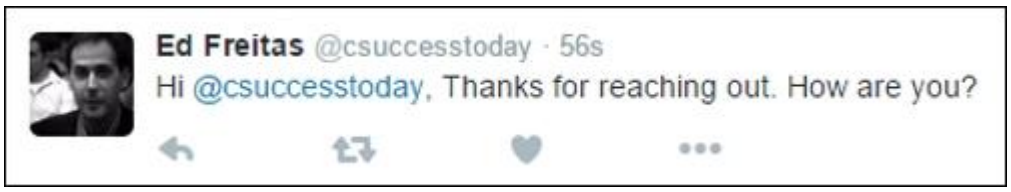


Рис. 14 Пример ответа бота

С помощью всего нескольких строк кода можно создать простого Twitter-бота. Следующим шагом будет добавление логики обработки сведений о рейсе.

Но для начала нужно создать отдельный документ secret.json для хранения кодов доступа и токенов отдельно, а не в общем коде.

Нужно создать новый secret.json файл и сохраните его в той же папке, что и app.js файл. Теперь secret.json файл должен выглядеть так:

```
{
  "consumer_key": "PUT YOURS",
  "consumer_secret": "PUT YOURS",
  "access_token_key": "PUT YOURS",
  "access_token_secret": "PUT YOURS"
}
```

Затем на этот файл ссылается код, и теперь он выглядит следующим образом.

```
var TwitterPackage = require('twitter');
var secret = require("./secret");
var Twitter = new TwitterPackage(secret);
Twitter.stream('statuses/filter', {track: '#FlightBot'}, function(stream) {
  stream.on('data', function(tweet) {
    console.log(tweet.text);
    var statusObj = {status: "Hi @" + tweet.user.screen_name +
    ", Thanks for reaching out. How are you?"}
    Twitter.post('statuses/update', statusObj,
      function (error, tweetReply, response) {
      if(error) {console.log(error);}
      console.log (tweetReply.text);});});});
  stream.on('error', function(error) {console.log(error);});});
```

Теперь, когда есть обновленный код, нужно добавить необходимую логику для отправки информации о статусах рейсов.

Чтобы получить некоторую информацию о рейсах, придется полагаться на API полета. Один из бесплатных и действительно полезных — это продукт Air France-KLM, доступный по адресу https://developer.airfranceklm.com/ .

На сайте нужно зарегистрироваться, Air France-KLM предоставляет несколько API-интерфейсов (их можно найти по адресу https://developer.airfranceklm.com/Our_Apis), таких как бронирование, заказ, предложение полета, статус полета, местоположение, контактная информация, дополнительный магазин и регистрация. Для работы нужно взять API который предоставит актуальную и точную информацию о конкретных номерах рейсов.

API статуса рейса предоставляет информацию о статусе рейса, такую как запланированное и фактическое время прибытия и отправления рейсов.

API поддерживает принятие оперативных решений, таких как уведомление в случае исключительных ситуаций, таких как экстремальные погодные условия.

Самое замечательное в этом API то, что он не требует ключей API и доступен бесплатно. Загвоздка в том, что он возвращает данные только за день, когда выполняется запрос, то есть возвращает только сегодняшние данные и не возвращает никакой информации о статусе прошлых рейсов.

Есть два способа поиска с помощью этого API. Один из них - поиск по номеру рейса, а другой - по маршруту.

При поиске по номеру рейса REST конечная точка предоставит статус рейса для данного рейса, и нужно указать дату; например, рейс KL1699 от 16 сентября 2016 г. Запрос будет выглядеть так:

http://fox.klm.com/fox/json/flightstatuses?flightNumber=KL1699&departureDate=2016-09-16

Это вернет следующий ответ JSON:


```
{"flights": [ {"@type": "OperatingFlight",
        "aircraft": {"registrationCode": "PH-BGT"},
        "carrier": {"code": "KL"},
        "flightNumber": "1699",
        "marketingFlights": [{"carrier": {"code": "KQ"},
         "flightNumber": "1699"},
        {"carrier": {"code": "DL"},
        "flightNumber": "9605"}],
        "operatingFlightLeg": {"arrivesOn": {"@type": "Airport",
            "IATACode": "MAD"},
            "departsFrom": {"@type": "Airport",
            "IATACode": "AMS"},
            "flightStatus": "ARRIVED",
            "legs": [ {"actualArrivalDateTime": "2016-09-16T09:26+02:00",
                  "actualDepartureDateTime": "2016-09-16T06:59+02:00",
                  "arrivesOn": {"@type": "Airport",
                  "IATACode": "MAD"},
                  "departsFrom": {"@type": "Airport",
                  "IATACode": "AMS"},
                  "scheduledArrivalDateTime": "2016-09-16T09:35+02:00",
                  "scheduledDepartureDateTime": "2016-09-16T07:00+02:00",
                  "status": "ARRIVED"}],
            "scheduledArrivalDateTime": "2016-09-16T09:35+02:00",
            "scheduledDepartureDateTime": "2016-09-16T07:00+02:00"
       },"remainingFlyTime": "PT0.000S"}]}
```


Теперь можно исследовать API поиска маршрута.

Конечная точка REST поиска маршрута предоставляет сводку статусов рейсов для всех применимых рейсов в данном маршруте.

Запрос будет выглядеть так:

http://fox.klm.com/fox/json/flightstatuses?originAirportCode=AMS&destinationAirportCode=CDG

Результат JSON будет следующим.


```
{"flights": [{"@type": "OperatingFlight",
```

```
        "_links": {"detailedInfoLink":
"http://fox.klm.com/fox/json/flightstatuses flightNumber=KL1223&departureDate=2016-
09-15&originAirport=AMS&destinationAirport=CDG"},
        "carrier": {"code": "KL"},
        "flightNumber": "1223",
        "operatingFlightLeg": {"arrivesOn": {"@type": "Airport","IATACode": "CDG"},
            "departsFrom": {"@type": "Airport","IATACode": "AMS"},
            "flightStatus": "ARRIVED",
            "scheduledArrivalDateTime": "2016-09-15T08:00+02:00",
            "scheduledDepartureDateTime": "2016-09-15T06:45+02:00"}},
      {"@type": "OperatingFlight",
        "_links": {"detailedInfoLink":
"http://fox.klm.com/fox/json/flightstatuses?flightNumber=KL1227&departureDate=2016-
09-16&originAirport=AMS&destinationAirport=CDG"},
        "carrier": {"code": "KL"},
        "flightNumber": "1227",
        "operatingFlightLeg": {"arrivesOn": {"@type": "Airport",
            "IATACode": "CDG"},
            "departsFrom": {"@type": "Airport",
            "IATACode": "AMS"},
            "flightStatus": "ARRIVED",
            "scheduledArrivalDateTime": "2016-09-16T08:40+02:00",
            "scheduledDepartureDateTime": "2016-09-16T07:15+02:00"}},
      {"@type": "OperatingFlight",
        "_links": {"detailedInfoLink":
"http://fox.klm.com/fox/json/flightstatuses?flightNumber=GA9240&departureDate=2016-
09-16&originAirport=AMS&destinationAirport=CDG"},
        "carrier": {"code": "GA"},
        "flightNumber": "9240",
        "operatingFlightLeg": {"arrivesOn": {"@type": "Airport","IATACode": "CDG"},
            "departsFrom": {"@type": "Airport","IATACode": "AMS"},
            "flightStatus": "ARRIVED",
            "scheduledArrivalDateTime": "2016-09-16T09:25+02:00",
            "scheduledDepartureDateTime": "2016-09-16T07:55+02:00"}}]}
```


Ответ включает в себя следующую информацию - запланированную дату вылета и время, запланированную дату прибытия и время, статус полета, маркетинговые рейсы, оставшееся время полета, информацию о прибытии и отправлении.

Итак, теперь, когда есть API для запросов, можно сделаем бота немного умнее и позволим ему получать статус полета и детали маршрута.

Используя уже имеющийся код, который может ответить человеку, который действительно написал '#FlightBot' хэштег, нужно внести некоторые изменения, чтобы он мог предоставлять сведения о статусе рейсов и маршрутов с использованием API Air France-KLM.

Первое, что нужно сделать для связи с конечными точками Air France-KLM, — это включить REST клиентскую библиотеку для Node.js в наше приложение.

Существуют различные клиентские библиотеки REST для Node.js, в данном коде будет использована библиотеку, которая находится по адресу https://www.npmjs.com/package/request .

Первое, что нужно сделать, это установить ее. Это можно сделать это из командной строки, выполнив эту инструкцию:

```
npm install request --save
```

После этого файл package.json будет обновлен следующим образом как на рисунке 15.

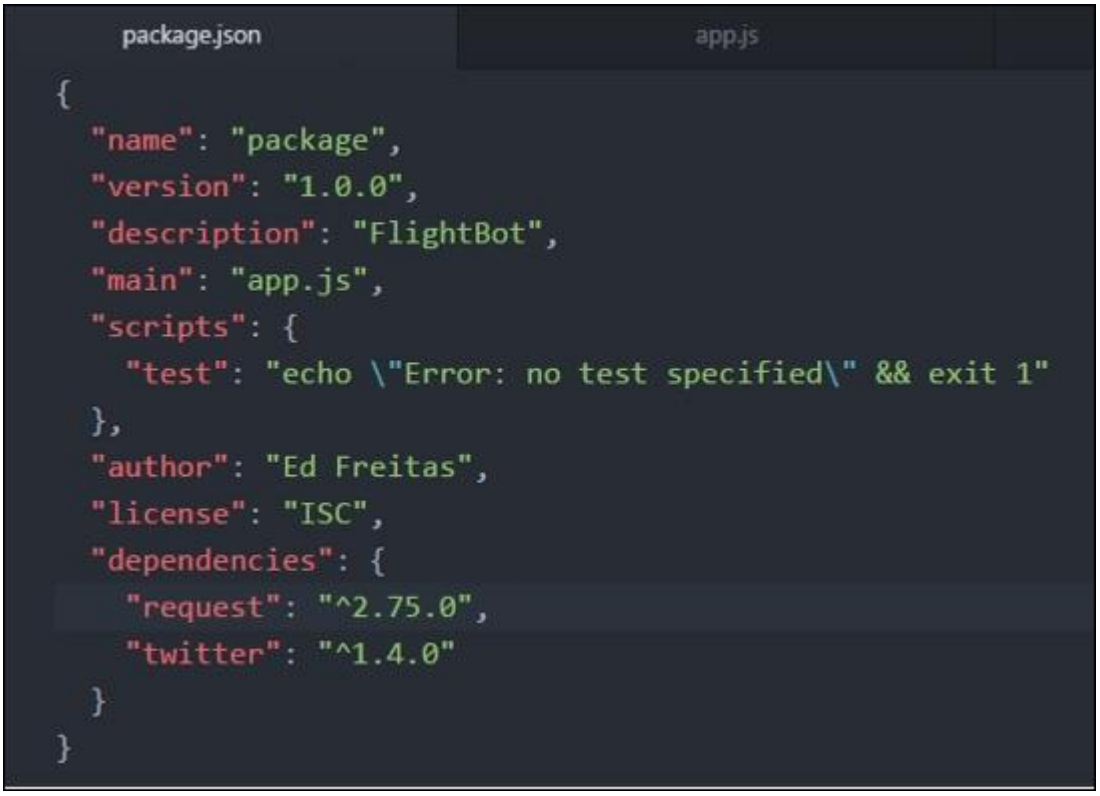


Рис. 15 Пример файла package.json с библиотекой requests

Теперь, когда установки REST клиентской библиотеки, пришло время добавить в приложение некоторую логику для взаимодействия с API Air France-KLM.

Бот должен иметь возможность предоставлять отзывы о рейсах и маршрутах, используя описанные выше конечные точки API. Для этого нужно добавить логику, чтобы бот мог обрабатывать не только хэштег Twitter, но и номер рейса.

Таким образом, автоматизируемый вызов конечной точки REST и связанный с ней ответ, возвращает только некоторые важные биты этих данных, но не все. Нужно автоматизировать эту конечную точку с помощью REST вызова. Также нужно убедиться, что, помимо хэштега, в сообщении также передается номер рейса:

http://fox.klm.com/fox/json/flightstatuses?flightNumber=KL1699&departureDate=2016-09-16

Вот полный обновленный код. Давайте полностью рассмотрим его, а затем разберем его по частям, чтобы понять, какие изменения были внесены:

```
var TwitterPackage = require('twitter');
var secret = require("./secret");
var Twitter = new TwitterPackage(secret);
var request = require('request');
```

```
padLeft = function (str, paddingChar, length) {
  var s = new String(str);
  if ((str.length < length) && (paddingChar.toString().length > 0))
  {for (var i = 0; i < (length - str.length) ; i++)
        s = paddingChar.toString().charAt(0).concat(s);}
  return s;
};
GetDate = function() {
  var dateObj = new Date();
  var month = dateObj.getUTCMonth() + 1;
  var day = dateObj.getUTCDate();
  var year = dateObj.getUTCFullYear();
  return year + '-' + padLeft(month.toString(), '0', 2) + '-' +
    padLeft(day.toString(), '0', 2);
};
FlightNumberOk = function(str) {
  var posi = str.indexOf('KL');
  var fn = str.substring(posi);
  return (posi >= 0 && fn.length === 6) ? fn: '';};
var fd = '';
GetFlightDetails = function(fn) {
  var dt = GetDate();
  var rq = 'http://fox.klm.com/fox/json/flightstatuses?flightNumber=' + fn +
    '&departureDate=' + dt;
  request(rq, function (error, response, body) {
    if (!error && response.statusCode == 200) {fd = body;}})};
Twitter.stream('statuses/filter', {track: '#FlightBot'}, function(stream) {
  stream.on('data', function(tweet) {
    var statusObj = {status: "Hi @" + tweet.user.screen_name +
      ", Thanks for reaching out. We are missing the flight number."};
    var fn = FlightNumberOk(tweet.text);
    if (fn !== '') {
      GetFlightDetails(fn);}
    setTimeout(function() {
      console.log ('fd: ' + fd);
      if (fd !== undefined) {
        var ff = JSON.parse(fd);
        statusObj = {status: "scheduledArrivalDateTime: " +
          ff.flights[0].operatingFlightLeg.scheduledArrivalDateTime};}
      Twitter.post('statuses/update', statusObj,  function(error, tweetReply,
      response) {
        if (error) {console.log(error);}
        console.log(tweetReply.text);
   });}, 1500);});
  stream.on('error', function(error) {
    console.log(error);
});});
```

В код была добавлена ссылка на request библиотеку, которую бот будет использовать для выполнения запросов к REST API:

```
var request = require('request');
```

После этого была добавлена GetDate функция, которая будет возвращать сегодняшнюю дату, чтобы ее можно было передать в REST конечную точку в качестве departureDate параметра:

```
GetDate = function () {
  var dateObj = new Date();
  var month = dateObj.getUTCMonth() + 1;
```

```
  var day = dateObj.getUTCDate();
  var year = dateObj.getUTCFullYear();
  return year + '-' + padLeft(month.toString(), '0', 2) + '-' +
    padLeft(day.toString(), '0', 2);
};
```

GetDate функция использует padLeft функцию, которая отвечает за правильное форматирование каждой части даты, как показано в следующем фрагменте кода:

```
padLeft = function (str, paddingChar, length) {
  var s = new String(str);
  if ((str.length < length) && (paddingChar.toString().length > 0))
  {for (var i = 0; i < (length - str.length) ; i++)
        s = paddingChar.toString().charAt(0).concat(s);}
  return s;
};
```

Эти две функции охватывают departureDate часть конечной точки REST. Итак, теперь перейдем на flightNumber часть.

Для этого была написана функция с именем FlightNumberOk, которая выполняет быструю проверку правильности номера рейса:

```
FlightNumberOk = function(str) {
  var posi = str.indexOf('KL');
  var fn = str.substring(posi);
  return (posi >= 0 && fn.length === 6) ? fn: '';};
```

С правильным номером рейса можно использовать другую функцию, вызываемую GetFlightDetails для фактического выполнения вызова REST конечной точки. Ответ JSON представлен переменной body, которая затем присваивается fd переменной, которая позже используется для отправки ответа пользователю как показано на коде ниже.

```
GetFlightDetails = function(fn) {
  var dt = GetDate();
  var rq = 'http://fox.klm.com/fox/json/flightstatuses?flightNumber=' + fn +
    '&departureDate=' + dt;
  request (rq, function (error, response, body) {
    if (!error && response.statusCode == 200) {fd = body;}})};
```

Итак, функция Twitter.stream теперь выглядит так.

```
Twitter.stream('statuses/filter', {track: '#FlightBot'}, function(stream) {
  stream.on('data', function(tweet) {
    var statusObj = {status: "Hi @" + tweet.user.screen_name + ", Thanks for
      reaching out. We are missing the flight number."};
    var fn = FlightNumberOk(tweet.text);
    if (fn !== '') {
    GetFlightDetails(fn);}
    setTimeout(function() {
      console.log ('fd: ' + fd);
      if (fd !== undefined) {
        var ff = JSON.parse(fd);
        statusObj = {status: "scheduledArrivalDateTime: " +
          ff.flights[0].operatingFlightLeg.scheduledArrivalDateTime};}
      Twitter.post('statuses/update', statusObj,  function(error, tweetReply,
      response) {
```

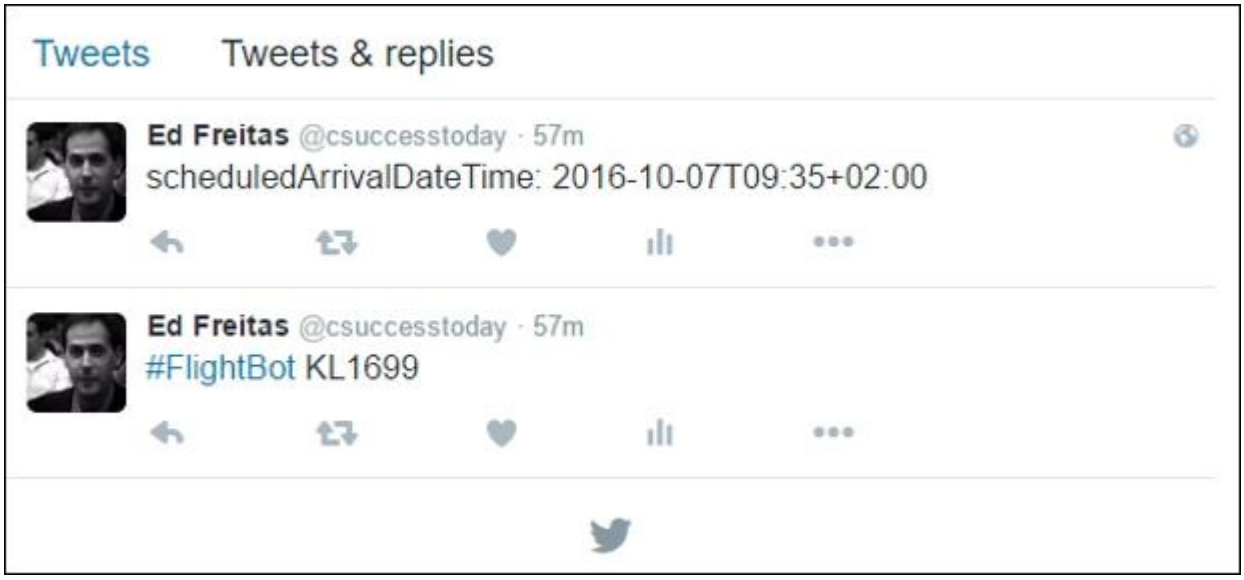
```
        if (error) {console.log(error);}
        console.log (tweetReply.text);
  });}, 1500);});
 stream.on('error', function(error) {
   console.log(error);});});});
```

Стоит обратить внимание, как FlightNumberOk и GetFlightDetails вызываются до того Twitter.Post, как setTimeout функция будет вызвана. Это делается для того, чтобы убедиться, что номер рейса в порядке, и что существует ответ JSON, содержащий детали рейса, перед отправкой твита пользователю.

Ответ на твит в основном отправляется scheduledArrivalDateTime, который получается путем анализа ответа JSON с использованием JSON.parse. Доступ к нему осуществляется следующим образом:

```
ff.flights[0].operatingFlightLeg.scheduledArrivalDateTime
```

Если сейчас запустить программу и напишем в Твиттере #FlightBot KL1699, то получим следующее:



Вывод

В этой статье был разобран процесс взаимодействия с Twitter, а также как запрашивать API Air France-KLM, чтобы получить информацию о рейсе и ответить на твиты. В статье была затронута лишь малая часть работы с API, ведь их возможности безграничны.

Библиографический список

1. Намиот Д.Е. Twitter как транспорт в информационных системах // International Journal of Open Information Technologies. 2014. Т. 2. № 1. С. 42-46. URL: https://elibrary.ru/item.asp?id=21019897 (Дата обращения: 23.01.2021)
2. Хачатрян М.Г. Обнаружение ботов в онлайновой социальной сети twitter с помощью алгоритма машинного обучения "случайный лес" // В сборнике:

Безопасные информационные технологии. Сборник трудов Девятой всероссийской научно-технической конференции. 2018. С. 184-187. URL: https://elibrary.ru/item.asp?id=36796761 (Дата обращения: 23.01.2021)

3. Хачатрян М.Г., Чепик П.И. Обнаружение ботов в социальных сетях с помощью многослойного перцептрона // Политехнический молодежный журнал. 2019. № 4 (33). С. 5. URL: https://elibrary.ru/item.asp?id=38028488 (Дата обращения: 23.01.2021)