Student Digest Twitter Bot

Chris Adams and Mark Eliseo

Abstract

When it comes to Twitter bots, they probably are often thought of as a spam machine or something that is being automated that probably shouldn't be. In our case, we wanted to create something that can be used as a way for students to see a real world application of how node.js can create a Twitter bot that will pull an email from their inbox, format it in a certain way, and tweet the results in a clear and concise manner. To do this, we used the programming language known as node.js, which is an open sourced, JavaScript runtime environment that is extremely flexible and efficient in creating web applications. We used a few different tools to create our Twitter bot, such as Nylas, CronJob, bit.ly, and the Twitter API. Each of these tools interact with one another fluently and allowed for us to create a bot to tweet Siena College's daily Student Digest emails in a timely manner. More specifically, our program will begin pulling the emails from the gmail servers using Nylas, splice up the emails and format our links using bit.ly, tweet the results using the Twitter API starting at 8 am EST everyday using CronJob. In the end, we did achieve the results we were looking for where our Twitter bot starts tweeting at 8 am, and will tweet every hour until there is nothing left to tweet. Once the program finishes executing, it will start back up the next day at 8 am again.

Introduction

Twitter is one of the world's top social media websites and can be used as a way to express what you are thinking at the moment to the variety of followers that you bring in. This can be done using up to two hundred and eighty characters, a photo, or a video and it allows people to have different types of avenues to get their point across in a clear manner. In our case, we have decided to create a twitter bot that will do this for us automatically. More specifically,

our plan was to gain access to the daily Student Digest emails from Siena and format the emails in a way that will be tweeted on a timer throughout the day. The Siena Student Digest is the daily news here at Siena, and each email is sent out at 3:45 am every single day. We have used node.js as well as javascript to create our Twitter bot as we believe these are the two languages that we could do this most efficiently. We also needed a way to gain access to the Twitter API, which was a process on its own.

Applying to become a developer on the Twitter API requires a lot of time and is extremely tedious, Twitter needs to make sure that people are not using their API to do things that they don't agree with. For example, through our application process, we had to re-apply multiple times just to stick to their guidelines. We had to phrase our application in a very specific way so they can ensure that we don't spam the feed or make sure that we aren't automatically liking tweets, as this will throw off statistics. This process took around one to two weeks of going back and forth with Twitter, and they eventually did give us the necessary access to the API. Once we obtained access to the API, we were able to import the four different keys given to us into our program: a consumer key, a consumer secret, an access token, and an access token secret.

How Our Twitter Bot Works and was Made

To begin, our Twitter bot needed a place to look for the emails. In our case, we used Mark's Siena email address, mc12elis@siena.edu, to pull the daily Student Digest email and start parsing through the information. To do this, we used the Nylas email API which allowed for us to connect directly to the gmail servers to obtain the necessary information. This involved creating another developer account to gain access to the Nylas API, and in doing so we received a client ID, a client secret, as well as an access token. To implement the Nylas functionality, we had to import those three keys into our program. We also used a bit.ly module that allowed for us to shorten our links within our tweets, this became useful when needing to save characters to compile a tweet in under two hundred and eighty characters. This was a similar process to using

Nylas and the Twitter API, we had to import an access token into our program when we registered for their site to use the modules. Finally, we used a module known as CronJob which allowed us to schedule specifically when functions execute, allowing us to tweet out the Student Digest email information periodically throughout the day as opposed to one long tweet reiterating the email exactly. We formatted it in a way to start the tweeting at 9 am EST, where it will tweet the headlines and information from the headlines after every hour. For example, if the Student Digest email has five separate headlines with their associated information, we will tweet the first headline at 9 am, the second headline at 10 am and so on.

Node.js is a very interesting language being that it is asynchronous and uses promises. An asynchronous language basically means that the program does not have to wait for one block of code to execute. The program will recognize when a function is asynchronous, and will complete any other necessary tasks prior to executing the asynchronous function. According to the node.js website:

> "The core idea behind promises is that a promise represents the result of an asynchronous operation. A promise is in one of three different states: pending - The initial state of a promise; fulfilled - The state of a promise representing a successful operation; rejected - The state of a promise representing a failed operation. Once a promise is fulfilled or rejected, it is immutable (i.e. it can never change again)" (CITATION).

We had to use this synchronization feature as well as promises when designing our program because we had to utilize the await feature on our promises. Since the promises don't execute sequentially, we had to make the function wait for them to finish before we can execute the program. This was actually one of the first roadblocks that we ran into, and we had a number of options in order to go about fixing it. One way we could have done this was to have all of our promises execute sequentially using the 'then' method. The 'then' method is directly related to promises in that the method is passed a function as one of its parameters and the function is executed only once the initial promise has finished. Another way we could have solved this issue of waiting for the function to return the necessary data was the creation of our own asynchronous functions. This would allow us to utilize the 'await' keyword which forces the program to halt

execution until the asynchronous function is completed. In the end, we decided to use both techniques by first creating an asynchronous function that gathers the necessary data from the email and formats it into a tweet, and lastly using the ‘then’ method to pass the headlines into the CronJob code block that posts tweets on an interval.

Once our asynchronized function begins executing, this is where we start the formatting of the tweet itself and parsing through the information. The first thing that we do is create a variable called message, which uses the Nylas API to obtain the email, looking for the first email within the inbox sent from “student-digest@siena.edu”.

Next, we create a variable that will contain the contents of the email in HTML known as messageBody. From there we can splice the HTML content to only include the block of headlines that we actually care about. In this case, it is the headlines under the section called “Today’s News.” We chose to ignore the “In Case You Missed It” section because we do not want to accidentally tweet headlines repeatedly over and over again.

At this point, we can start looping through the email to find different headlines to format our tweets. We loop to split the new HTML headline block into different partitions. Each of these partitions begins with a header3 that will print exactly what that partition is. Throughout the loop, we are able to get the indexes of each instance, as well as the index of the next partition. We designed our program this way so that once the program recognizes that the index of the next partition is at a -1, the loop will know when to break and carry on with the next part. We continue pushing these partitions onto our array “partitions”, until there is no next partition.

After we have our array of partitions, we can then begin scanning each partition for necessary headline information. To do this, we initialized a “categories” two dimensional array and aliased the partition to a variable called “str”. This is when we start cleaning the partitions by splicing away the unnecessary characters. We then can assign the partition title to the first position in the array by doing the following:

```
categories[i][0] = str.substring(str.indexOf('<b>') + 3, str.indexOf("</b>"));
```

This piece of code will give us the “category” of the tweet (for example, “Lectures/Films/Readings”), it will be the first line of what is tweeted out as it is the main focus of the tweet. As seen in this example, we used the HTML tags to trim our headlines and find the most pertinent information.

We then need a way to get the title of the article, as well as the rest of the information within the email. Using another while loop, we can splice the rest of the email into its necessary variables. First, we initialize a variable for the header that will hold the formatted tweet. We then scan in the raw headline name and concatenate it to the headline variable by taking the index of the HTML tags pertinent to it. This will now give our headline variable the “title” of the tweet (for example, “Physics and Astronomy Colloquium”).

Since we now have what category we are in as well as the title of the article, we now need to get the description of the article from the email. The email typically is formatted to go category, title, author, description, but we decided to swap the description and the author so that the author comes after the description, we believe this is a much more natural way of reading in the information. To obtain the description, we did as follows:

HTML formatting: Science Blogs and Talking Dogs: Reflections on 17 Years in Social Media

Obtaining the information:

```
var desc = str.substring(str.indexOf("</font>") + 7);
headline = headline + desc.substring(desc.indexOf("'black'>") +
8,desc.indexOf("</font>")).trim();
```

Creating the “desc” variable is where we start the splicing. It was necessary to use the second variable, “desc”, to avoid accidentally reacquiring the index of the initial tag. The first line will bring us to the end of , the second line will concatenate what we already have in our headline variable with all of the information between the end of ‘black’> and the start of the second tag and will trim the rest. This will give us the result “Science Blogs and Talking Dogs: Reflections on 17 years in Social Media”. Similarly, we can use the same type of format

as above to obtain the author of the article. The only difference when obtaining the author's name instead of the description is that the "desc" variable is unnecessary as the author text uses differing tags and instead of ".trim()" at the end, we would use a new line variable to format the tweet properly.

Finally, we need to obtain the link itself to the article. In an email, you can disguise the link behind the title, but this is different for a tweet as Twitter does not allow for this. Our workaround to this issue is by getting the link from the email, and using bit.ly as a way to shorten that link down to save as many characters as possible to ensure that we do not go over the limit. As we have been doing, we just scan the hyperlink to the article into a hyperlink variable and format it into bit.ly hyperlink by using substring to get rid of the HTML tags. We can then create a new variable called "biturl" that will use the bit.ly module to automatically shorten the url, which we then add to our headline variable.

Once we complete splicing through the entirety of one section of the email, we can push our results stored into our headline variable onto our categories two dimensional array. After we push our results, we can reassign our str variable to begin formatting on the next HTML block headline to get the next section of the email. This whole process will repeat until there is no other information to be read in from the email, in which our for loop will break and we move onto the next part of the program.

Lastly, we had to figure out the timing of our tweets to make sure that they will start at the correct time and end when there is nothing left. To do this, we had to construct a CronJob object to initiate the email parsing at 8am everyday. We also had to create an interval that allows for us to tweet once per every new hour (3,600,000 milliseconds) once we have completed parsing the digest email. Once we are ready to start posting tweets, we can then concatenate the partition title to the headline. We begin tweeting by posting the last headline element in the two dimensional array. This is due to the fact that we store the partition title in the first position of the array. When working from the end of the array, we can pop the already posted information off the array and know we have reached the end of the category when the array's length is 1. The

entire text body of the tweet is saved into the variable “post”. Once this block of code executes, we can actually post the tweet to Twitter by using the following few lines of code:

```
T.post('statuses/update', {status: post}, function(err, data, response) {
        console.log('Tweet Posted!\n');
});
```

After posting the tweet, we can then go ahead and pop off the most recent headline in our categories array. The program then checks a conditional if there is only a title left in the array, if so we can remove the entire partition from categories. When we are out of partitions, then we can end our interval, therefore stopping the program from automatically posting tweets for the day as there are currently no more partitions to be tweeted.

Conclusion

Twitter bots can be used in a variety of different ways and instances, we chose to create a bot that can automatically format emails from Siena and tweet them on a timely basis. The customization for these bots is essentially endless, there have been a ton of twitter bots that have been made to tweet out friendly reminders for people throughout their days, maybe different photos or videos, or could be used by companies across the world to do something similar to what we made. Node.js made this entire process not nearly as difficult as we first anticipated. Since node.js is open source JavaScript, a lot of the coding can be done server side which allows the code to be executed directly from a computer and/or browser and allows for efficiency and a wide variety of uses. Although our Twitter bot currently only works with the Student Digest email, this could be changed and modified to have a twitter bot that searches for any emails sent out by the school to be tweeted automatically. Our Twitter bot could be expanded in many different ways with that being one of them.

Citations

"API Reference Index - Twitter Developers." *Twitter*, Twitter, https://developer.twitter.com/en/docs/api-reference-index.

Kiessling, Manuel. "The Node Beginner Book." *The Node Beginner Book*, https://www.nodebeginner.org/.

"Npm." *Npm*, https://www.npmjs.com/.

"Promises." *Promises*, www.promisejs.org/.

"Read an Email Inbox With JavaScript / Node.js." *Read an Email Inbox With JavaScript / Node.js*, https://docs.nylas.com/docs/read-an-email-inbox-with-nodejs.

Reinman, Andris. "Mailparser." *Nodemailer*, https://nodemailer.com/extras/mailparser/.