4

A Quote Slack Bot

If you haven’t been living isolated in a galaxy very far away, eons from Earth with no internet connection during the last year or so, I am sure you already heard about [Slack](https://slack.com/), the famous real time messaging app and collaboration suite for teams.

Slack has been built from the ground up to be easy and fun to use. It offers a broad set of APIs that allows developers to extend its capabilities to make it even more useful.

One of the features I most enjoy from Slack is the Slackbot. It is friendly Bot available in every Slack team to guide users to create their profiles and to explain them how Slack works.

If you think SlackBot sounds cool, what you’ll really love even more is the possibility to build your own custom Bots, which can act as automated users that can respond to specific events and help your team do useful things.

Throughout this chapter we’ll explore how we can use the Slack Real Time messaging API in order to create our own custom SlackBot. We’ll walk through the whole process so you can get a really good idea of what is possible.

By the end of this chapter, you should feel right at home with creating your own SlackBot and know a bit more about Slack and how it could help your team and you.

Overall the process should be a lot of fun and easy to follow, so let’s not wait any longer and get started!

## Getting Started

We are going to be building a Bot that gives quotes as responses to the general channel. The idea is to have a Bot that inspires your team in their daily activities and quotes are definitely a great way to get inspired.

The first thing we need to do is to setup our Bot with Slack and register in order to use the Slack API.

In order to do this, we’ll be using the [Slack Real Time messaging API](https://api.slack.com/rtm), which is a websocket-based API that allows us to receive events in real time and send messages to channels, private groups and users.

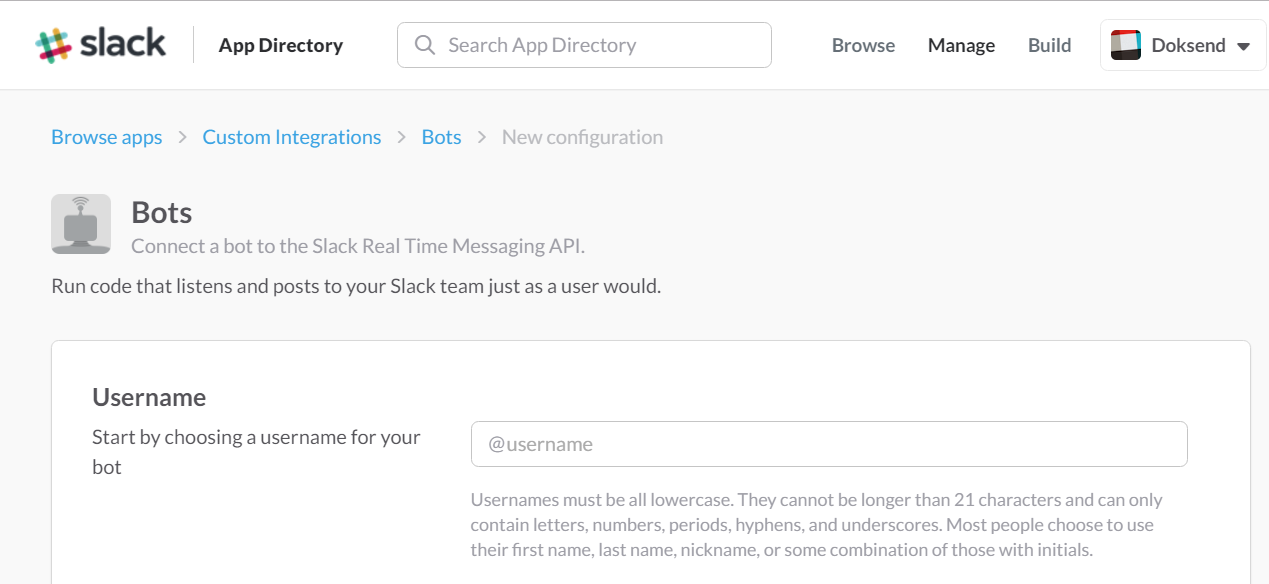
The API is really well constructed and the documentation is easy to follow. We won’t be using websockets directly, but instead a Node.js [module](https://www.npmjs.com/package/slackbots) that makes development much easier, using JavaScript.

We need to configure our channel extensions and create the new Bot. This way we will obtain the API token that is required to authenticate to Slack and get started. So, let’s roll up our sleeves and get moving!

## Registering a Bot on Slack

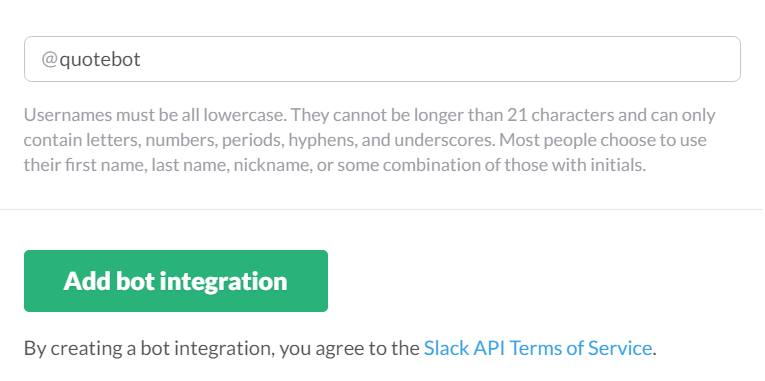
In order to add the Bot into your Slack organization, we’ll need to register it at the following URL: https://yourorganization.slack.com/services/new/bot

Notice that you will need to change “*yourorganization*” with the name of your company or team, which you used when registering your Slack account. Once you open up the URL in your browser, you will be redirected to the following screen.

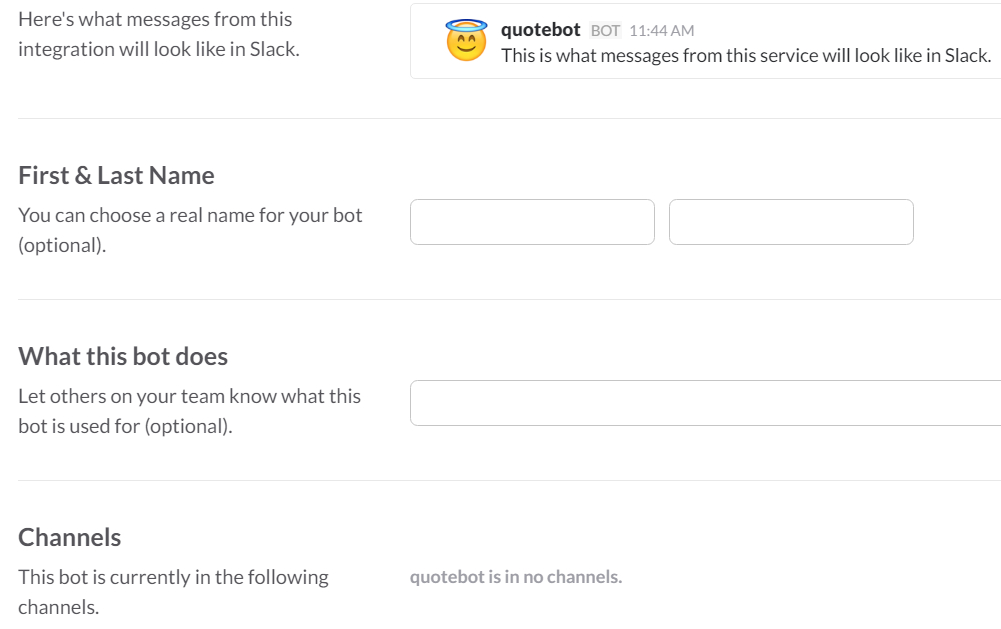


We’ll be calling our Bot, “quotebot”. This is the value we will fill in our username. Notice how Slack requires that all Bot names are all written with lowercase.

Once the name has been entered on the Username field, click on the Add bot integration button.

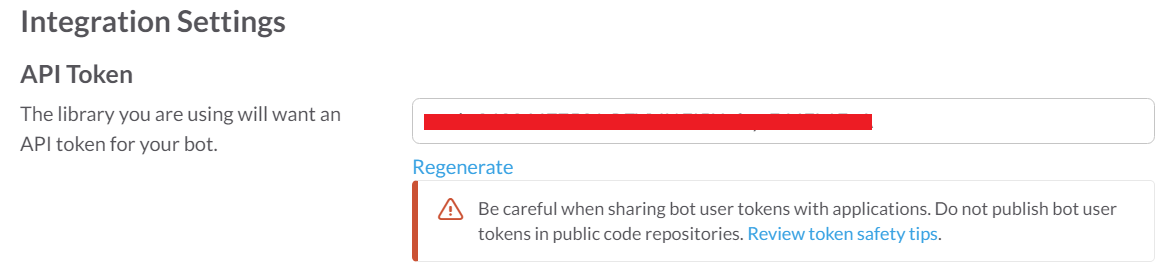


Once that has been done, you will be presented with a screen where the Bot can be further customized and features can be added, such as a picture or emoji. This is what this screen looks like.

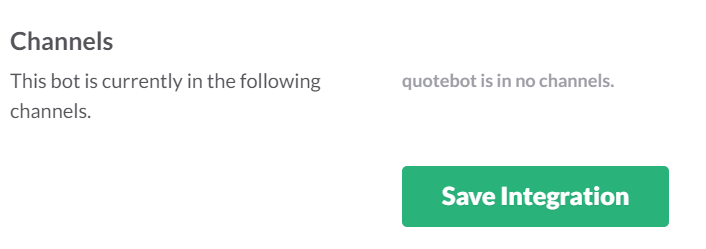


There are further options that are available and can be customized, but they don’t fit in a single screen shot, so you’ll see them once you reach the end of this screen. Nothing too complicated.

This screen also contains the API Token which we will need to reference in our code.



Once you’ve done the necessary configuration adjustments and changes, click on the Save Integration button.



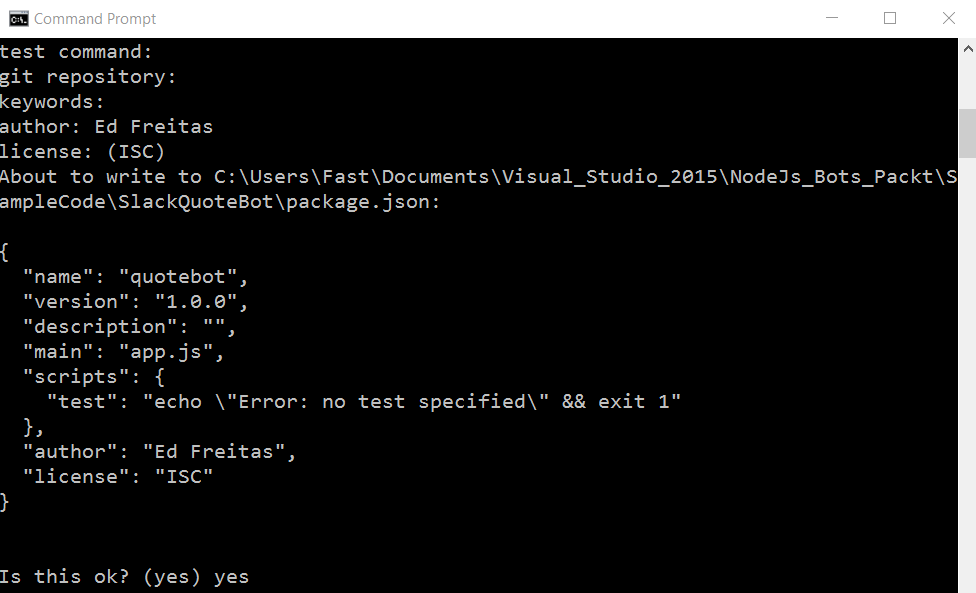
## Setting Up our Node.js App

Now that we’ve registered our Bot on Slack we are ready to setup our Node.js project in order to start coding.

Let’s go ahead and create our package.json file. Open the command prompt and type the following command.

|  |
| --- |
| npm init |

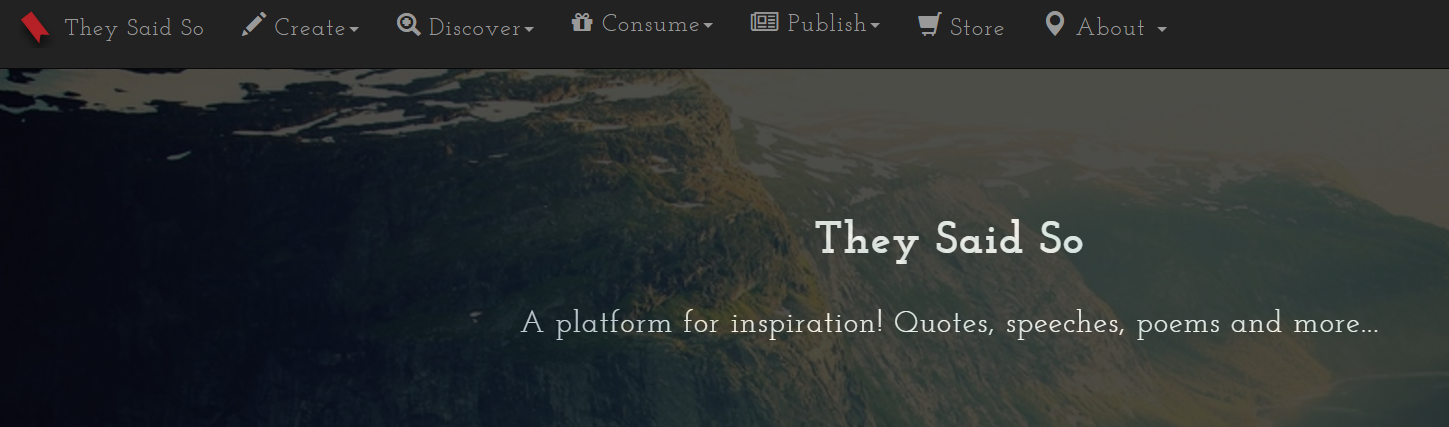
After you have done this, follow the guided configuration procedure which would look similar to this.



Once you have finished with this configuration, it is time to install the dependencies we will need in order to write our bot.

But before we install any dependencies, let’s quickly brainstorm what our Bot should do. In short, our Bot must be able to retrieve a quote and reply back to the general channel.

There’s an awesome site called [They Said So](https://theysaidso.com/) which is a service that provides Quotes-as-a-Service (QAAS). Quotes from multiple authors can be obtained through an easy to use REST API.



As we’ll need to access this service using REST, let’s include a REST client library for Node.js in our app. Just like we did in the previous chapter, we’ll be using this [one](https://www.npmjs.com/package/request).

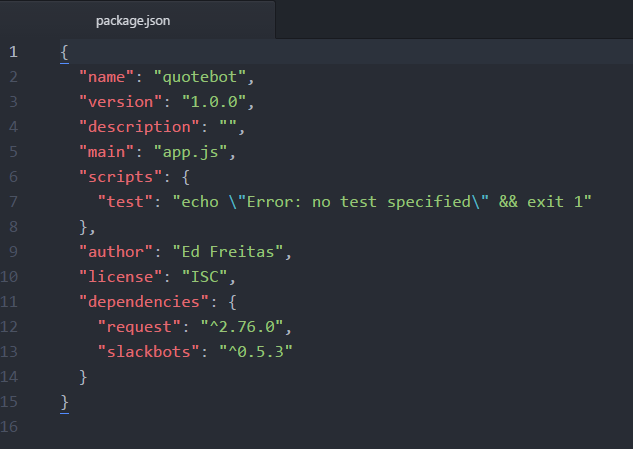
In order to get this library installed, execute this instruction from the command line.

|  |
| --- |
| npm install request --save |

Now that this has been done, the next step is to install a library called [SlackBots](https://www.npmjs.com/package/slackbots) that will act as an abstraction layer to deal with the Slack Real Time Messaging API.

|  |
| --- |
| npm install slackbots --save |

After doing this our package.json file will be updated as follows.



With our Node.js Bot settings all wired up, we are now ready to start writing some code.

## SlackBots Library Basics

As mentioned earlier, in order to interact with the Slack Real Time Messaging API, we’ll be using a Node.js library (NPM package) called [SlackBots](https://www.npmjs.com/package/slackbots).

Before we write any code, let’s have a look at the main functions offered by this module by looking at the following short example.

|  |
| --- |
| var Bot = require('slackbots');  var settings = {  token: 'API TOKEN',  name: 'quotebot'  };  var bot = new Bot(settings);  bot.on('start', function() {  bot.postMessageToChannel('channel-name', 'Hi channel.');  bot.postMessageToUser('a-username', 'Hi user.');  bot.postMessageToGroup('a-private-group', 'Hi private group.');  }); |

## Before you run this code, please substitute the strings channel-name, a-username and a-private-group with your own values, taken from your Slack organization.

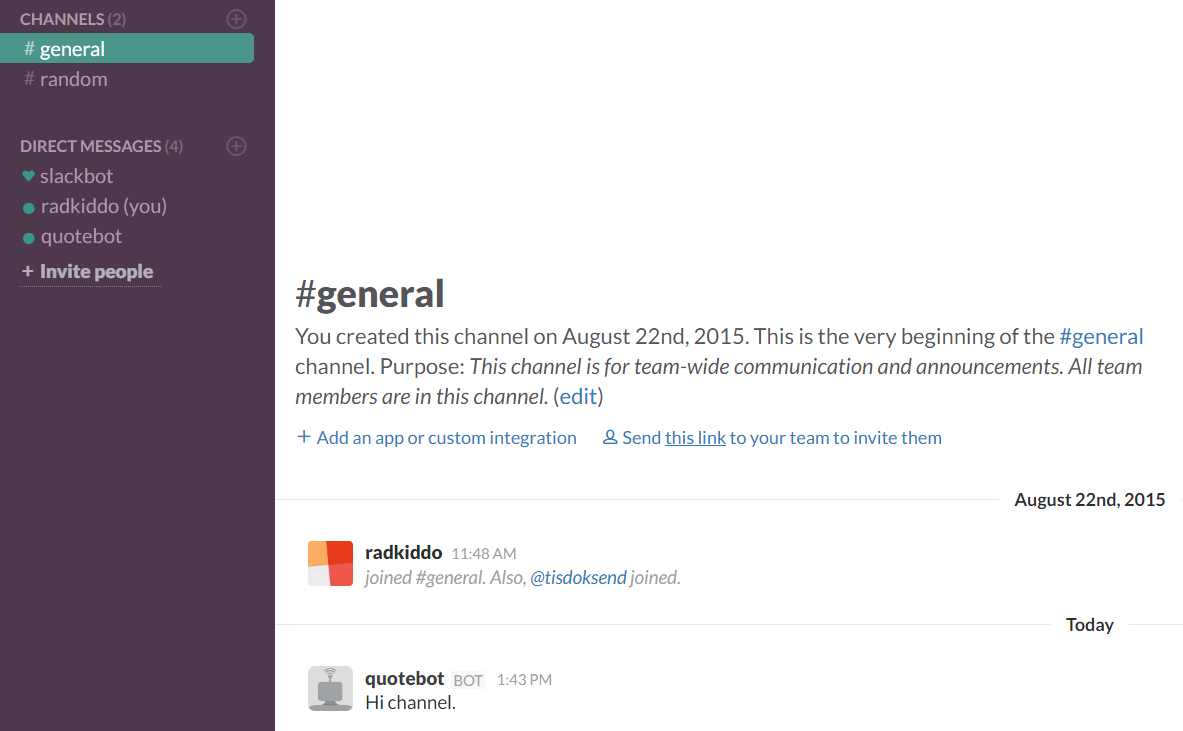
## You’ll also need to replace the string API TOKEN with the quotebot token you were given when the Bot was created. The code should now look similar to this.

|  |
| --- |
| var Bot = require('slackbots');  var settings = {  token: 'xoxb-………-R7VVJ1FI5Hzfcyt………',  name: 'quotebot'  };  var bot = new Bot(settings);  bot.on('start', function() {  bot.postMessageToChannel('general', 'Hi channel.');  bot.postMessageToUser('radkiddo', 'Hi user.');  bot.postMessageToGroup('tisdoksend', 'Hi private group.');  }); |

## Once you have replaced those value, you can run the app from the command line as follows.

|  |
| --- |
| Node app.js |

## If you login to Slack and open your team’s page, you should be able to see this when you browse to the **#general** channel.



## Awesome, our quotebot just came to life with its first ever message! Now let’s break the code down into pieces in order to understand it a bit better.

|  |
| --- |
| var Bot = require('slackbots'); |

## As you can see from the code above, the first thing we need to do is to require the SlackBot constructor. From there we can instantiate a new Bot object and add callbacks to specific events.

## On this code we use the event start that is triggered when the Bit connects successfully to the Slack server.

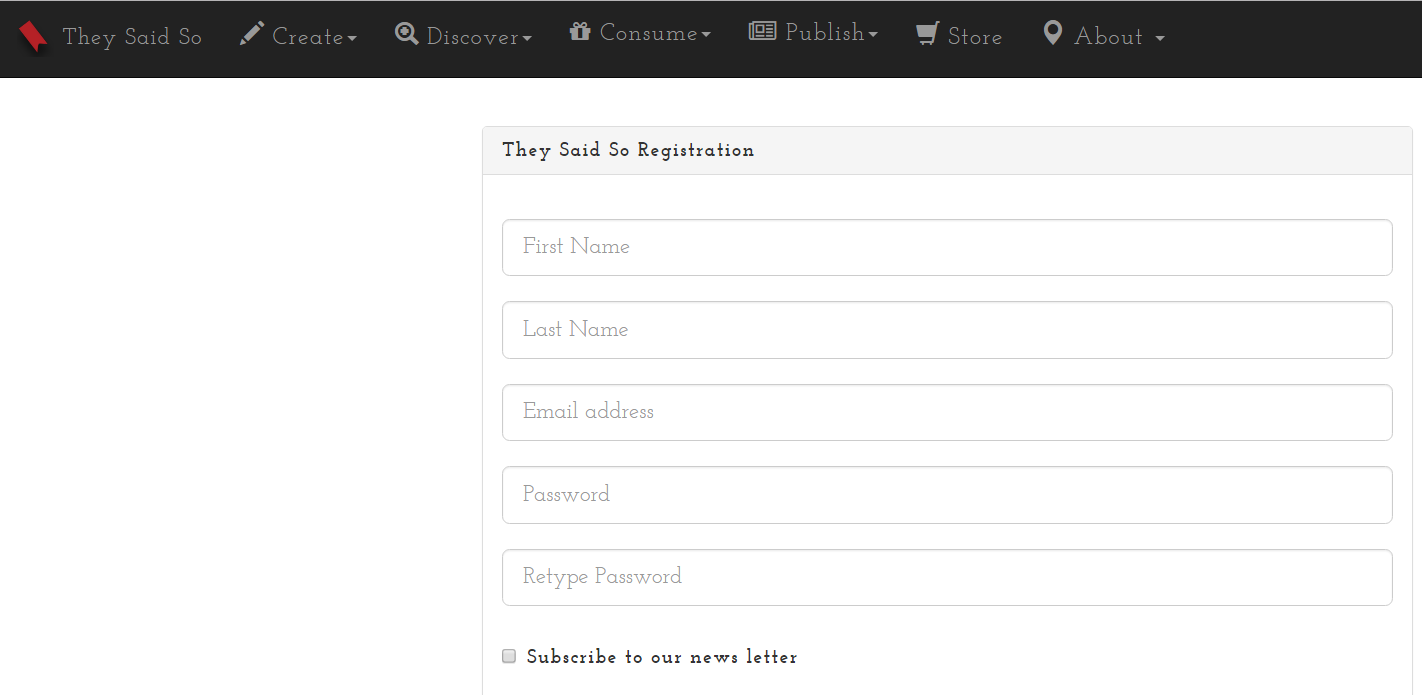
|  |
| --- |
| bot.on('start', function() {  bot.postMessageToChannel('general', 'Hi channel.');  bot.postMessageToUser('radkiddo', 'Hi user.');  bot.postMessageToGroup('tisdoksend', 'Hi private group.');  }); |

## Then we can use the methods offered by the library to post a message in a channel using the method postMessageToChannel, to a user as private message using postMessageToUser or in a private group conversation by calling postMessageToGroup.

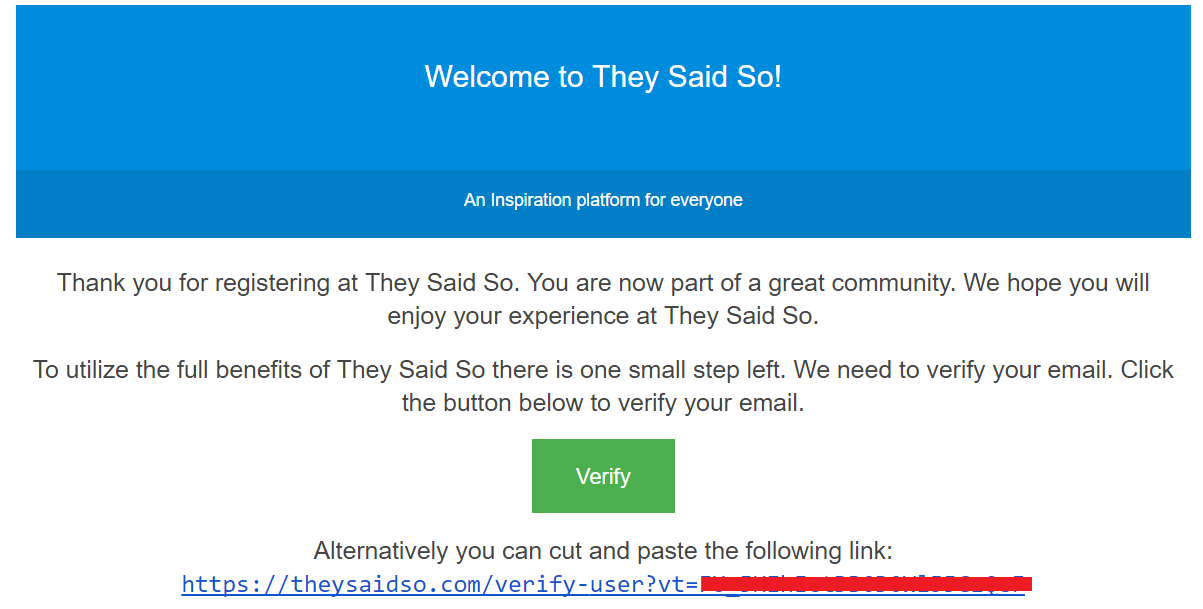
## With these fundamentals covered, we can move on to explore the [They Said So](https://theysaidso.com/) API, which is necessary in order to for us to build our Bot.

## The Said So API

The[They Said So](https://theysaidso.com/)service has a huge collection of quotes in their database and the Quotes API is a great and convenient way to access this data. In order to consume the Quotes API, you’ll first need to sign up for the service at this [URL](https://theysaidso.com/register).

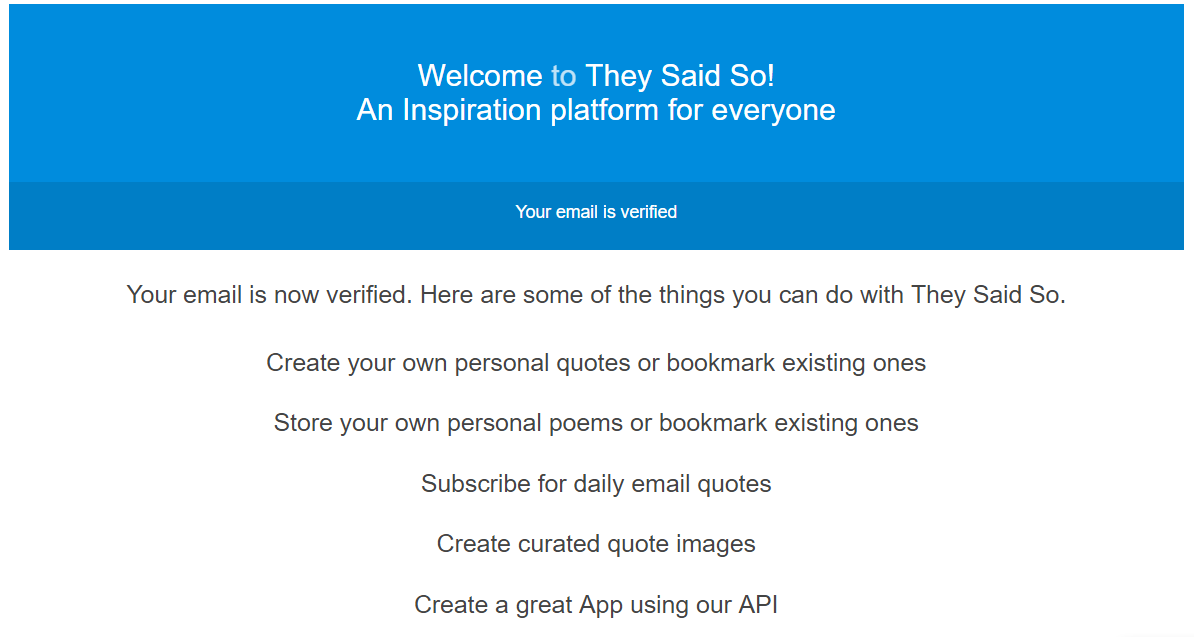


Once you have entered your details and register you will receive an automated verification email which will look like this.



When you receive this, simply click on the Verify button in order to validate your newly registered account and start enjoying the service.

Once you’ve done that, you will shortly receive this email with next steps.



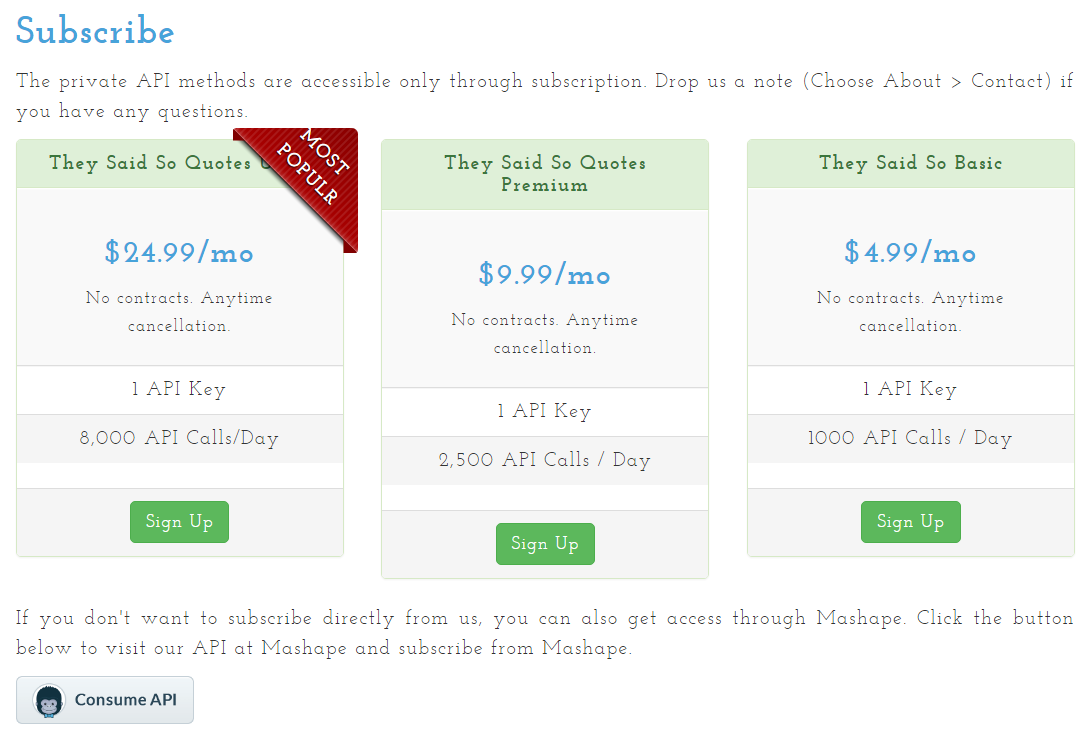
The next step is to subscribe to the Quotes API in order to start consuming it. This can be done by visiting the following [URL](https://theysaidso.com/api/#subscribe) or alternatively click in the Create a great App using our API link from the email received.

When you open this, scroll to the very bottom of the page and you will see the following API subscription plans on the screen.

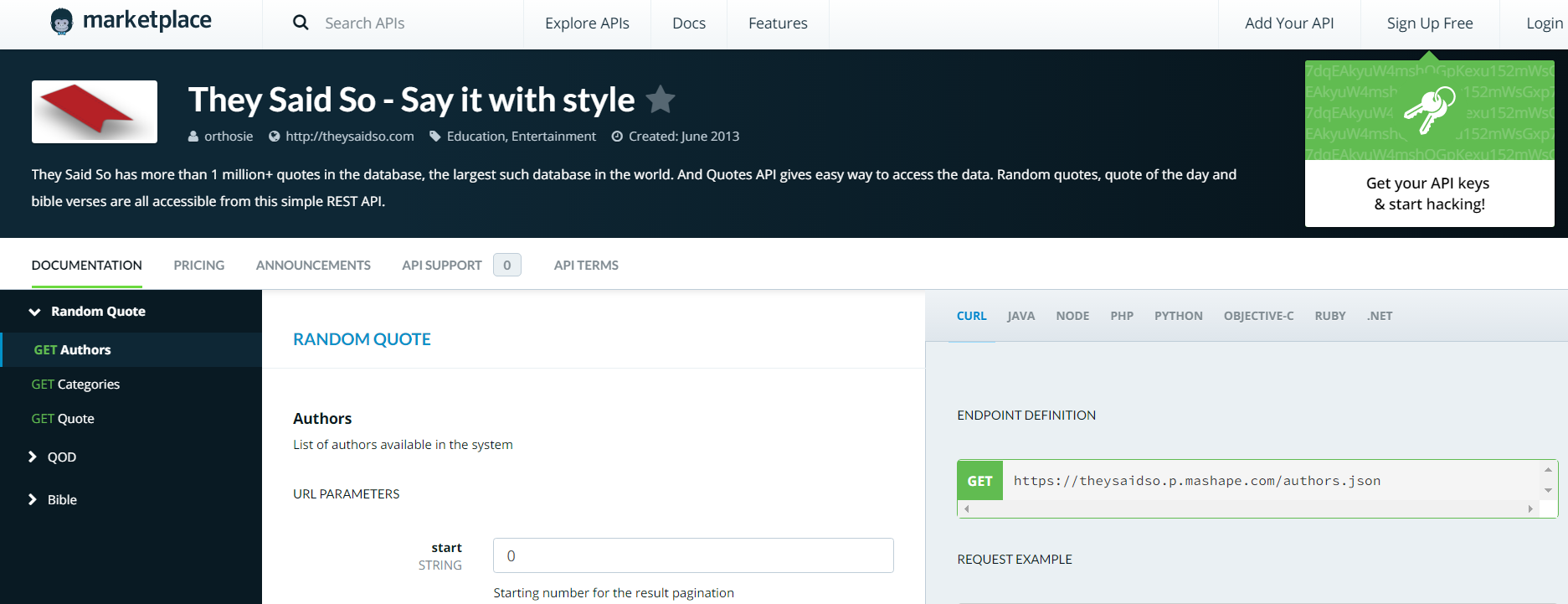
In our case, because we are building a demo application, we won’t sign up for any specific paid plan but instead we’ll consume the API through [Mashape](https://market.mashape.com/orthosie/they-said-so-say-it-with-style/).

However, if you wish to get a paid plan, you may Sign Up for any of the paid API options that the service offers. The advantage of doing this, is that you won’t need to sign on to Mashape in order to consume the API.

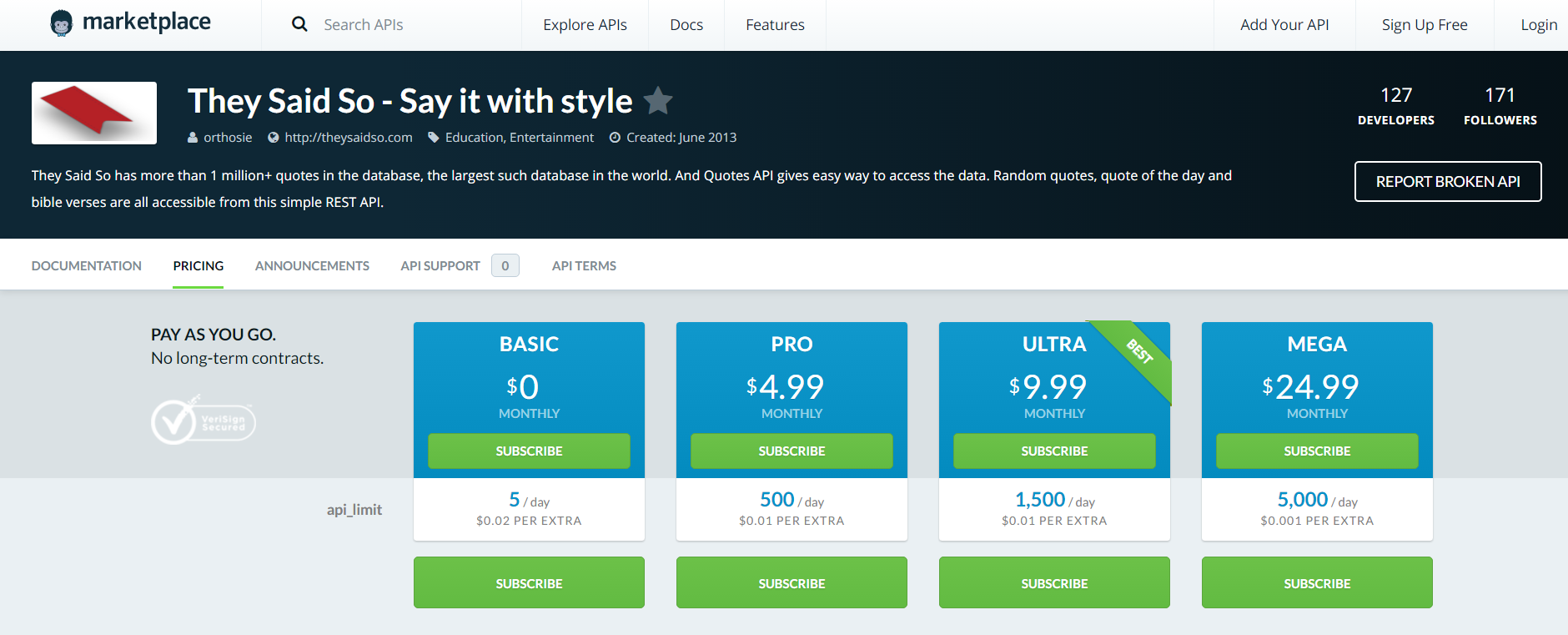
So in use the API through Mashape we can click on the Consume API button at the bottom of the screen.



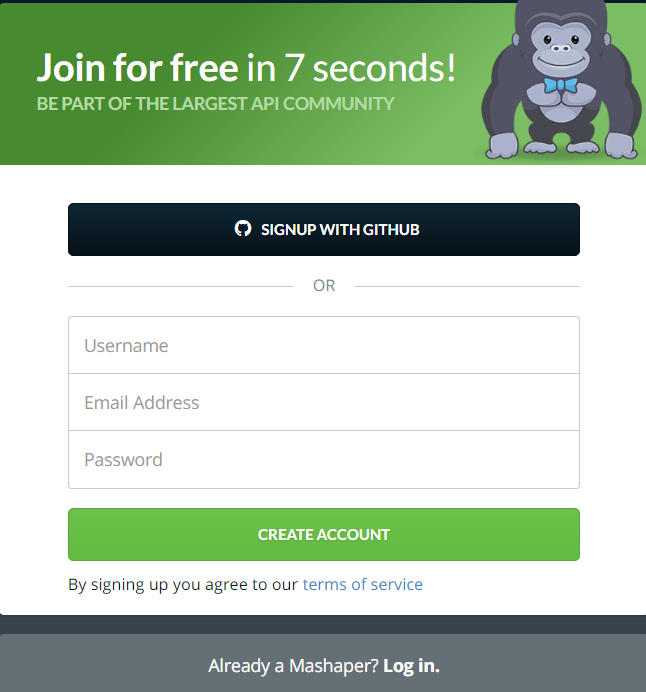
When we click on that button, the next thing we’ll see is this screen. Once there, click on the PRICING tab.



On the PRICING tab, under the BASIC plan, click on the SUBSCRIBE button.

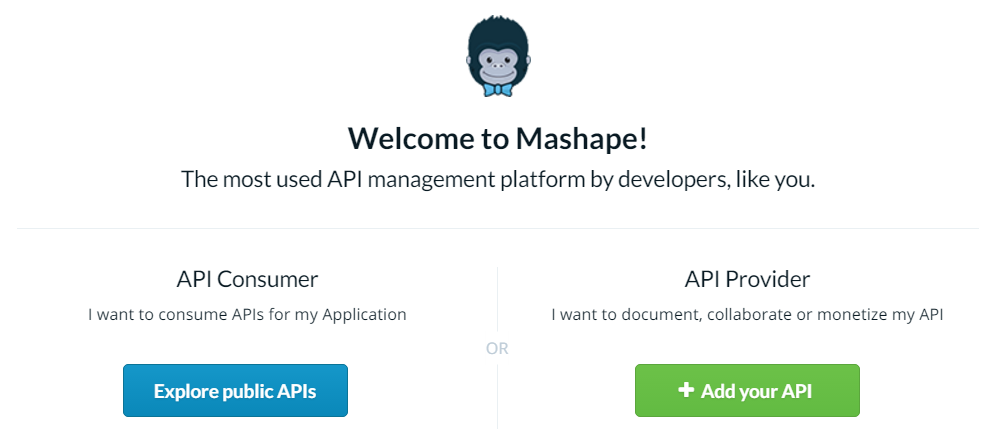


When you click on the SUBSCRIBE button, you’ll see the following pop-up screen.



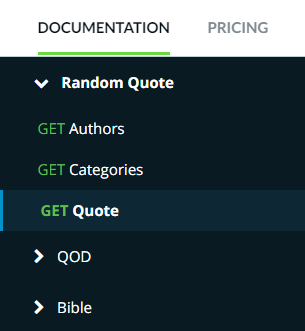
If you have a Github or Mashape account, you can simply subscribe to the API by logging into the service with any of those accounts. Otherwise you will have to create an account on Mashape.

The process anyway is very easy and straightforward. Once that’s been done, we are ready to start exploring the API and consuming it. You’ll see this on your screen.

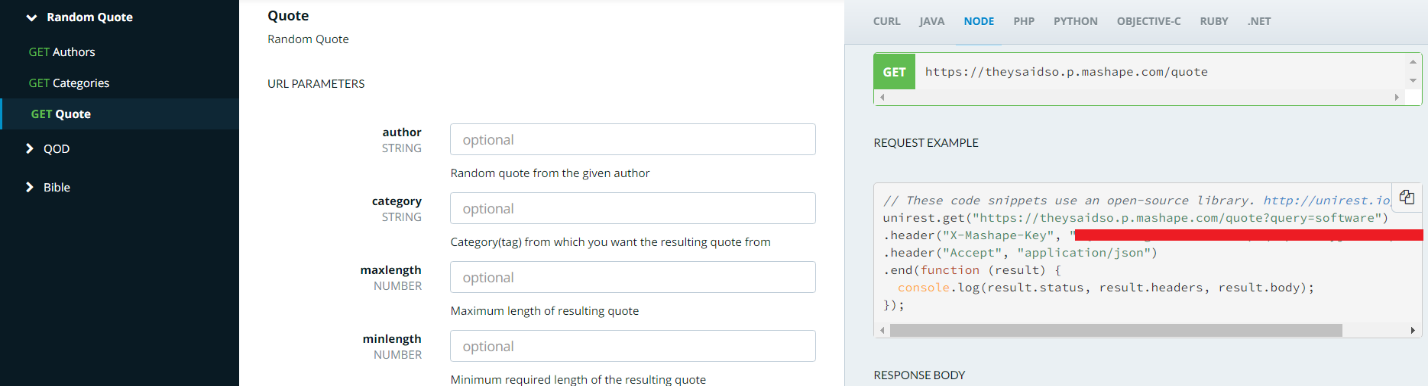


Simply click on the Explore public APIs button and then hit the back button on your browser or navigate to this [URL](https://market.mashape.com/orthosie/they-said-so-say-it-with-style/) to start exploring the API. So let’s do that.

From the API, we are interested in the quotes section. We can look at this by clicking on the GET Quote link on the left side of the screen.



This will take us to the following page, where we can see how to construct an API call in order to get a quote.



There are also multiple examples in various programming languages, including Node.js which uses the [Unirest](http://unirest.io/) library to make HTTP requests. In our application, we’ll be using the [Request](https://www.npmjs.com/package/request) library instead.

Notice how on all the sample code, including the Node.js one mentioned on the GET Quote documentation page, the API Token key is passed on the header of the HTTP request as the value of the X-Mashape-Key parameter.

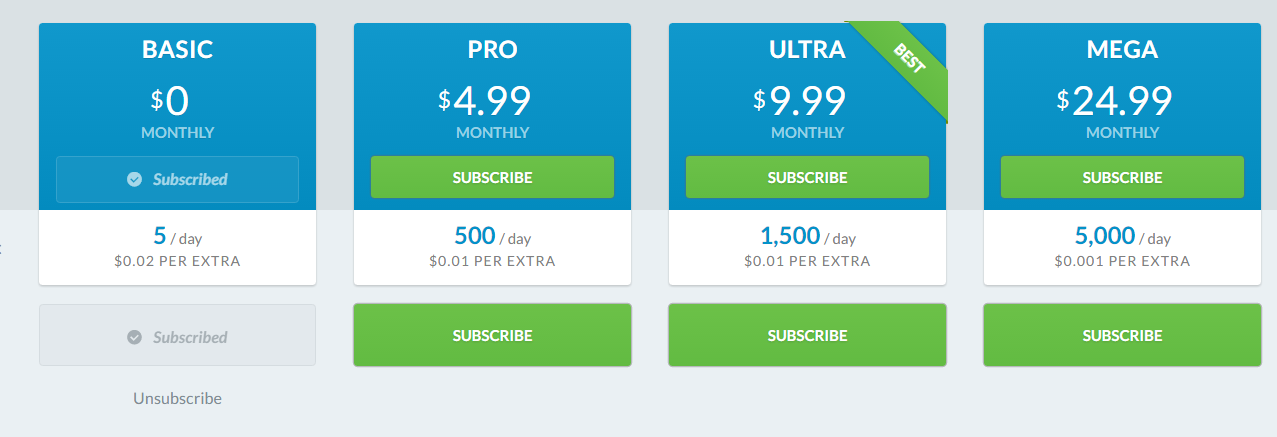
So let’s see how we would be able to write a small example on how to retrieve a quote using the Request library.

Let’s create a new file called TestRequest.js so we don’t mix up this test code with the main quotebot code that we have started to write using App.js.

|  |
| --- |
| var rq = require('request');  var token = 'Your They Said So API Key';  GetQuote = function() {  var options = {  url: 'https://theysaidso.p.mashape.com/quote?query=software',  headers: {  'User-Agent': 'request',  'X-Mashape-Key': token  }  };  rq(options, function (error, response, body) {  if (!error && response.statusCode == 200) {  console.log(body);  }  })  };  GetQuote(); |

Before you can run this, make sure you are subscribed to the Basic plan which includes 5 calls per day. You will still have to enter a credit card number, which will be billed if you go over 5 requests per day.

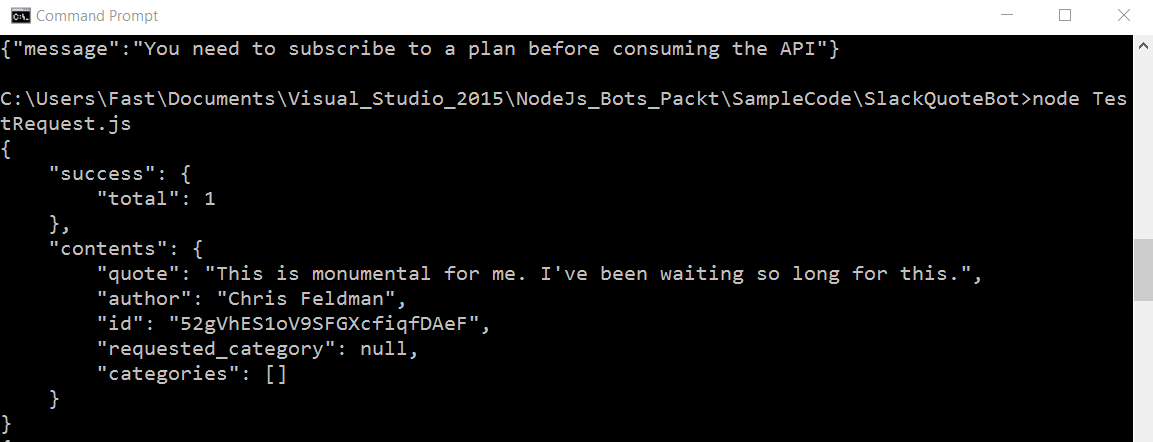
You can always unsubscribe on this [URL](https://market.mashape.com/orthosie/they-said-so-say-it-with-style/pricing) by clicking on the Unsubscribe link under the BASIC plan.



## Now you may run this script from the command line as follows.

|  |
| --- |
| Node TestRequest.js |

This produces the following result.



Now that we know how to interact both APIs, we can expand the basic code we initially wrote in order to create a full blown quotebot.

In our previous code snippet we used the start event. Going forward, we’ll also need the message event, which will be used to intercept an incoming message and based on that reply back.

We need to have a function that will intercept every real time API message that is readable by our Bot, pretty much every chat message in any channel where the Bot has been installed, but also private messages directed to the Bot or other real time notifications such as a user typing in a channel, edited or deleted messages, users joining or leaving the channel and so on.

Real time API messages are not just chat messages, but any kind of event that occurs within our Slack organization. This is important to keep in mind.

Ideally we would like the Bot to filter all these events to detect public messages in channels that mentions “getquote” or the name of the Bot, then we want to react to this message by replying with a random quote, fetched from the API we have subscribed using Mashape.

Ideally we want to divide all these checks in a list of operations this is exactly what we need to do. These are:

1. Verify if the event represents a chat message.
2. Verify if the message comes from a user that is different from quotebot (to circular references and loops).
3. Verify if the message mentions getquote.

## The code would look as follows.

|  |
| --- |
| onMessage = function (msg) {  if (isChatMsg(msg) &&  !isFromQuoteBot(msg) &&  isMentioningQuote(msg)) {  replyWithRandomQuote(bot, msg);  }  }; |

The onMessage function receives a msg object as parameter. The msg contains all the information that describes the real time event received through the Slack Real Time API.

Now let’s look at each helper function, one by one.

|  |
| --- |
| isChatMsg = function (msg) {  return msg.type === 'message';  }; |

This function verifies if a real time event corresponds to a msg sent by a user. With our first helper function in place, let’s have a look at the second one.

|  |
| --- |
| isFromQuoteBot = function (msg) {  return msg.username === 'quotebot';  }; |

This helper function allows us to see if the msg comes from a user who is not the quotebot itself.

Last but not least, our final helper function checks if messages contain the string “getquote”. Without this verification we could potentially end up with an infinite loop of quotes.

|  |
| --- |
| isMentioningQuote = function (msg) {  return msg.text.toLowerCase().indexOf('getquote') > -1;  }; |

With all the helper verification functions done, our random quote reply back method would look like this.

|  |
| --- |
| replyWithRandomQuote = function (bot, oMsg) {  var options = {  url: 'https://theysaidso.p.mashape.com/quote?query=software',  headers: {  'User-Agent': 'request',  'X-Mashape-Key': token  }  };    rq(options, function (error, response, body) {    if (!error && response.statusCode == 200) {  bot.postMessageToChannel(bot.channels[0].name, body);  }  })  }; |

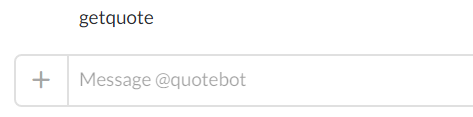
Finally we tie it all together by passing the onMessage callback to the listening event as follows.

|  |
| --- |
| bot.on('message', onMessage); |

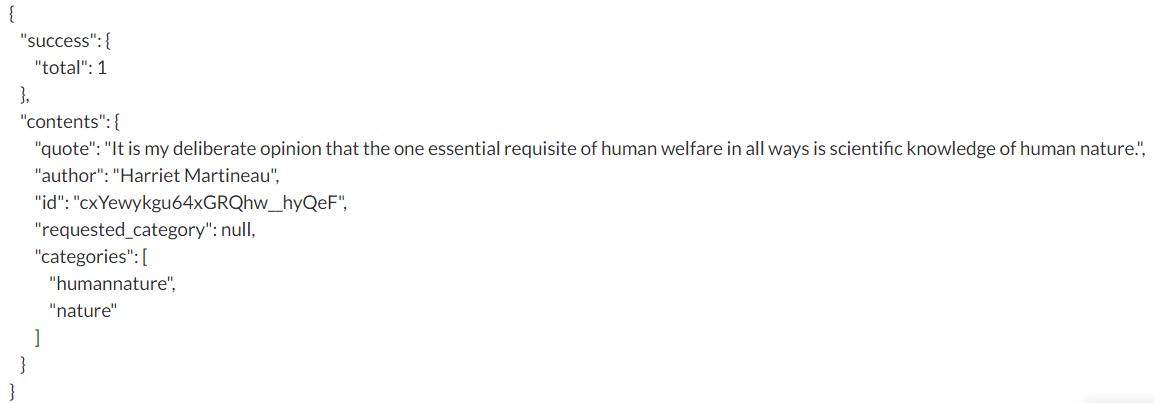
## The full code looks like this.

|  |
| --- |
| var Bot = require('slackbots');  var rq = require('request');  var token = ' YOUR MASHAPE API TOKEN ';  var settings = {  token: 'YOUR SLACK API TOKEN',  name: 'quotebot'  };  var bot = new Bot(settings);  isChatMsg = function (msg) {  return msg.type === 'message';  };  isFromQuoteBot = function (msg) {  return msg.username === 'quotebot';  };  isMentioningQuote = function (msg) {  return msg.text.toLowerCase().indexOf('getquote') > -1;  };  replyWithRandomQuote = function (bot, oMsg) {  var options = {  url: 'https://theysaidso.p.mashape.com/quote?query=software',  headers: {  'User-Agent': 'request',  'X-Mashape-Key': token  }  };    rq(options, function (error, response, body) {    if (!error && response.statusCode == 200) {  bot.postMessageToChannel(bot.channels[0].name, body);  }  })  };  bot.on('message', function (msg) {  if (isChatMsg(msg) &&  !isFromQuoteBot(msg) &&  isMentioningQuote(msg)) {  replyWithRandomQuote(bot, msg);  }  }); |

In order to see this in action simply message quotebot with the text getquote on Slack.



Once you do that, you will receive the following feedback on the ‘general’ channel.



That’s awesome! Notice however that we have returned the full body response.

This could be further optimized and you could eventually parse the body response and just output the quote and the author, without any of the other details.

This is totally up to you and a nice exercise in order to improve this code. Further to this, you could also add additional code in order to process natural language, interpret more commands and also to respond to different channels.

The possibilities are frankly endless and all that is needed is time, imagination and dedication. So thus, we leave the challenge open for you to further expand and explore.

## Summary

We’ve seen briefly how Slack is a great collaboration platform and also how incredibly easy it is to interact with its real time API.

In a matter of minutes you can have a small demo Bot up and running.

In the following chapters we’ll explore other interesting platforms that are also quite popular nowadays and this should be also a lot of fun to play around with.

Hope you have enjoyed following this example and the next chapters will touch other fascinating topics. Keep having fun!