

Cosmic JS Blog

Stay tuned for feature roll-outs, community news,
and updates from the Cosmic JS team.

Create Free Account → (/signup)

How to Build a Facebook Bot App Using Node.js

June 6, 2017 Noah Vaillancourt

Share

Here are the following commands for use.

'create': add a new reminder

'config': lists your current bucket config

'setup': add your bucket info such as slug and write key

create

What would you like your reminder to be? etc 'I have an appointment tomorrow from 10 to 11 AM' the information will be added automatically

I have an appointment tomorrow from 10 to 11 AM

reminder added correctly 😊

This website uses cookies to ensure you get the best experience on our website. Learn more

Type a message...

(/cookie-policy)



Got it!



TL;DR

View the source code on GitHub (<https://github.com/cosmicjs/facebook-bot>).


The process of building your messenger bot is fairly simple the hardest part is setting up your machine to talk to Facebook. That's why today I'm going to walk you through that real quick. Once it is all done you can get right on the way to creating your own bot.

Installing The Project

Go to folder where you would like the project to be and run.

```
git clone https://github.com/cosmicjs/facebook-bot
cd facebook-bot
yarn install
```

One more thing! Inside of your index.js file remove everything after line 38. We'll go back in and add that later. It should look like this.

This website uses cookies to ensure you get the best experience on our website. [Learn more](#) 
(/cookie-policy)

Got it!

```

const express = require('express')
const bodyParser = require('body-parser')
const request = require('request')
const app = express()
const Cosmic = require('cosmicjs')
const BootBot = require('bootbot')
require('dotenv').config()
const chrono = require('chrono-node')
var schedule = require('node-schedule')
const EventEmitter = require('events').EventEmitter

var config = {}

const reminders = []

const eventEmitter = new EventEmitter()

app.set('port', (process.env.PORT || 5000))
app.use(bodyParser.urlencoded({extended: false}))
app.use(bodyParser.json())

app.get('/', function(req, res) {
  res.send("hey there boi")
})



app.get('/webhook/', function(req, res) {
  if (req.query['hub.verify_token'] === process.env.VERIFY_TOKEN){
    return res.send(req.query['hub.challenge'])
  }
  res.send('wrong token')
})

app.listen(app.get('port'), function(){
  console.log('Started on port', app.get('port'))
})

```

This will only start the express server and prevent us from getting errors before we get a chance to start. See the line of code that says 'process.env.APP_SECRET' this is an environmental variable. There are a few ways you can set these up but for now we'll use my favorite method the 'env' package.

Create a file to store all of your variables.

This website uses cookies to ensure you get the best experience on our website. [Learn more](#) 
 You can view our cookie policy [here](#)  (/cookie-policy)

This allows us to store our variables in plain text and clean up the start command. For example mine looks like this. I removed all the sensitive code.

```
APP_SECRET='some secret string of numbers and letters'  
ACCESS_TOKEN='some secret string of numbers'  
VERIFY_TOKEN='DogLover49'
```

We will fill in APP_SECRET and ACCESS_TOKEN later but for now make sure that you have something in the VERIFY_TOKEN field. It can be anything at all. For the sake of this example I used DogLover49.

Starting Your Application

Now your going to need to start your application for the next part to work. Open two seperate terminal windows inside of your application root folder.

```
npm run start
```

Run this first and keep it running. Now in the next terminal window we will install and setup local tunnel.

Local Tunnel is a nice little utility that takes a port you specify and routes it to the outside world by giving it a secure web address. We'll just have to install it real quick.

```
npm install -g localtunnel
```

All you need to do now is run.



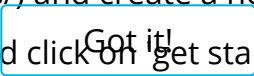
```
lt --port 3000 --subdomain <domainpick>
```

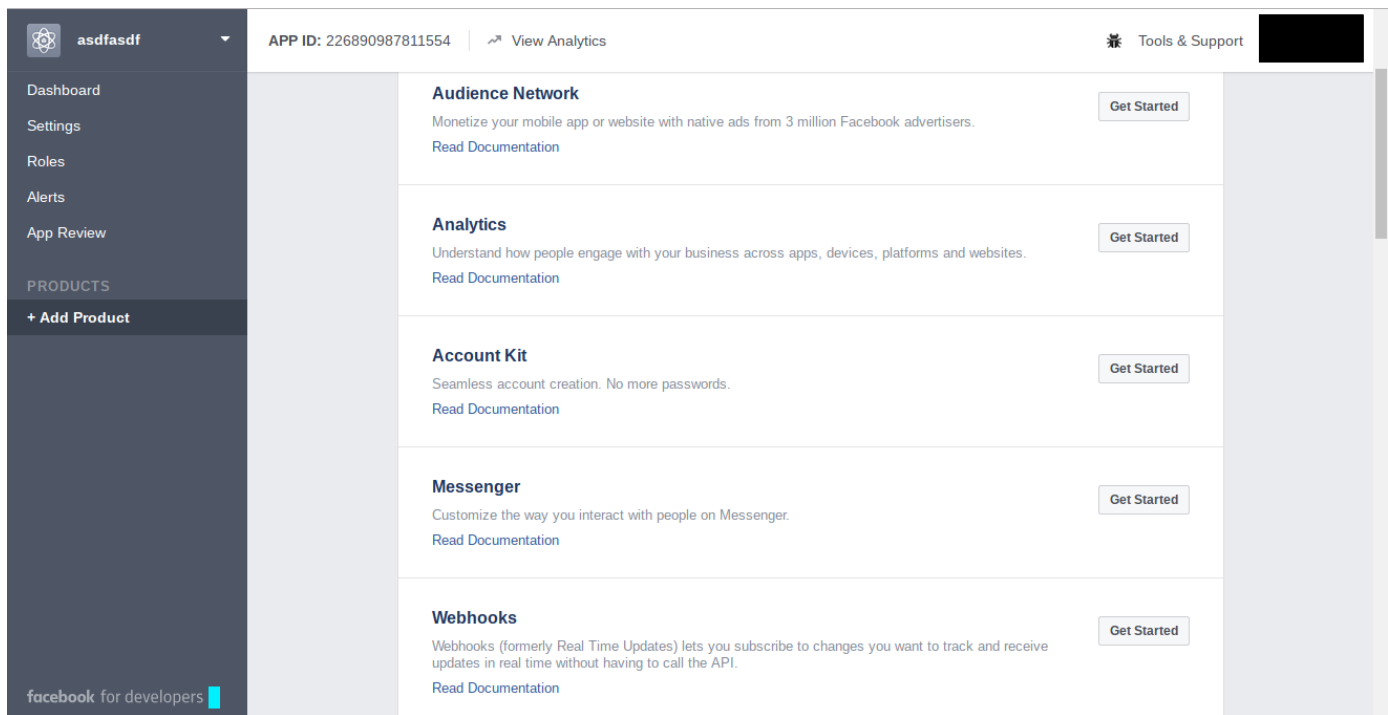
It should return this.

```
your url is: https://<domainpick>.localtunnel.me
```

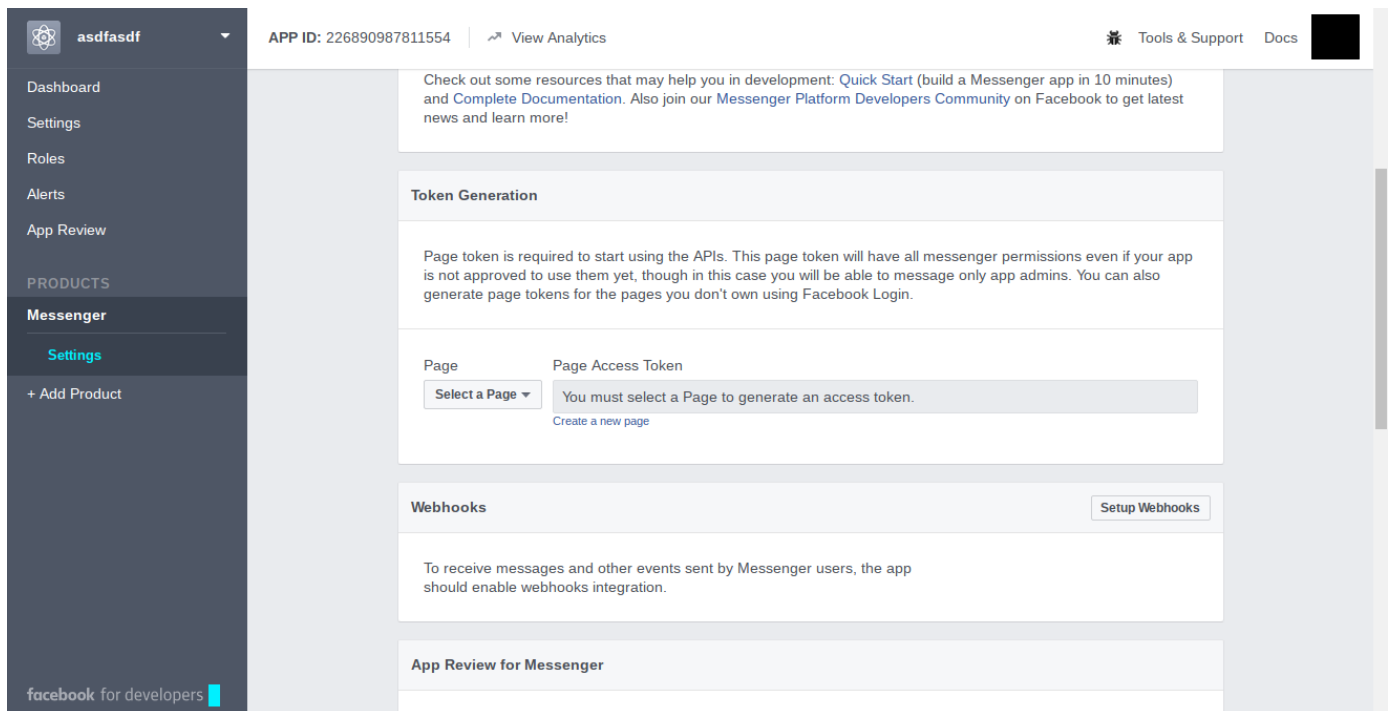
This starts a tunnel that gives your machine a url for Facebook to look for. Whenever you start the application make sure that you start it before you initiate local tunnel or else it won't find the port and return errors.

Facebook Dev Panel Time

This website uses cookies to ensure you get the best experience on our website. Learn more 
First thing you need to do is create a new application. Go to the Facebook Developer Panel 
(<https://developers.facebook.com/apps/>) and create a new app. Once your inside under 'Add Product' find the Messenger section and click on  'get started'.



After that you'll need to click on the settings tab which will pop up right after. Scroll down until you see this.



There are two important sections to note here 'Token Generation' and 'Webhooks'. The first one we are going to touch on is token generation. Select a Facebook page your going to use this will be the account you send a message to.

This website uses cookies to ensure you get the best experience on our website. [Learn more \(/cookie-policy\)](#)

Got it!

The screenshot shows the Facebook Developer Portal interface. On the left is a dark sidebar with navigation links: Dashboard, Settings, Roles, Alerts, App Review, PRODUCTS, Messenger (highlighted), and Settings (highlighted in blue). Below these is a '+ Add Product' button and the 'facebook for developers' logo. The main content area has a top bar with 'APP ID: 226890987811554' and a 'View Analytics' link. Below this is a 'Tools & Support' link and a 'Docs' link. The main content area contains several sections: a 'Token Generation' section with a 'Page' dropdown (showing a blacked-out page name) and a 'Page Access Token' field containing the token 'EAADOWyy7MulBAOI4ZBGBGtz1k5NErpvEzGZAciWyTRWU2C1xb5ZBs2RYwf'. Below the token is a 'Create a new page' link. There is also a 'Webhooks' section with a 'Setup Webhooks' button and an 'App Review for Messenger' section.

After you picked a page from the drop down it will generate a page access token. You can now go back and add this to your .env file. Once that is done we are going to change our focus to the webhooks.

Once you click on the button that says 'setup webhooks' you will see this. It may seem scary at first but don't worry.

The screenshot shows the 'New Page Subscription' dialog box. It has a title bar with a close button. The form contains the following fields and sections:

- Callback URL:** A text input field with placeholder text 'Webhooks updates for this topic will be sent to this URL.'
- Verify Token:** A text input field with placeholder text 'Token that Facebook will echo back to you as part of callback URL verification.'
- Subscription Fields:** A section with a grid of checkboxes for various permissions:
 - messages
 - message_deliveries
 - messaging_pre_checkouts
 - messaging_referrals
 - messaging_postbacks
 - message_reads
 - messaging_checkout_updates
 - message_echoes
 - messaging_optins
 - messaging_payments
 - messaging_account_linking

 At the bottom of the dialog, there is a 'Learn more' link, a 'This website uses cookies to ensure you get the best experience on our website. Learn more (/cookie-policy)' notice, and two buttons: 'Cancel' and 'Verify and Save'. A red box highlights the 'Got it!' button at the very bottom of the page, below the dialog box.

The only subscription fields you need to select are messages, messaging_postbacks, messaging_optins, and message_deliveries. Now for the verification token field. Do you remember the variable you created in your .env file under VERIFY_TOKEN? That is exactly what you want to put inside the box in this modal. For the callback url use the url from local tunnel and add '/webhook' to the end for me that would be `https://noahsmessengerbog.localtunnel.me/webhook`.
(`https://noahsmessengerbog.localtunnel.me/webhook`.) If you filled in the fields correctly it should look something like this.

New Page Subscription

Callback URL

https://yourlocaltunnel.localtunnel.com/webhook

Verify Token

a_string_you_provide

Subscription Fields

☒ messages

☒ messaging_postbacks

☒ messaging_optins

☒ message_deliveries

☐ message_reads

☐ messaging_payments

☐ messaging_pre_checkouts

☐ messaging_checkout_updates

☐ messaging_account_linking

☐ messaging_referrals

☐ message_echoes

[Learn more](#)

Cancel

Verify and Save

Now go back to your application and make sure that it is running and you are also forwarding the url. Double check the url and make sure it's the same thing as shown in the callback url field.


Once it is all running successfully you can click 'verify and save' this will have Facebook send a request to the callback url and expect you to return the correct item from a json object.

This website uses cookies to ensure you get the best experience on our website. [Learn more](#) (/cookie-policy)
At this point you should have almost all of the items you need inside of the .env file except for the App Secret. This can be easily found by going to your dashboard.

Got it!

The Bot Comes Next

Remember how earlier we removed everything after line 38? Now you can go and paste it back in. It should match this file exactly. When that is all done we can go over the code and learn just how it works. It should look like this.

This website uses cookies to ensure you get the best experience on our website. [Learn more](#) 
(/cookie-policy)

Got it!


```
const express = require('express')
const bodyParser = require('body-parser')
const request = require('request')
const app = express()
const Cosmic = require('cosmicjs')
const BootBot = require('bootbot')
require('dotenv').config()
const chrono = require('chrono-node')
var schedule = require('node-schedule')
const EventEmitter = require('events').EventEmitter

var config = {}

const reminders = []

const eventEmitter = new EventEmitter()

app.set('port', (process.env.PORT || 5000))
app.use(bodyParser.urlencoded({extended: false}))
app.use(bodyParser.json())

app.get('/', function(req, res) {
  res.send("hey there boi")
})

app.get('/webhook/', function(req, res) {
  if (req.query['hub.verify_token'] === process.env.VERIFY_TOKEN){
    return res.send(req.query['hub.challenge'])
  }
  res.send('wrong token')
})

app.listen(app.get('port'), function(){
  console.log('Started on port', app.get('port'))
})

const bot = new BootBot({
  accessToken: process.env.ACCESS_TOKEN,
  verifyToken: process.env.VERIFY_TOKEN,
  appSecret: process.env.APP_SECRET
})

bot.setGreetingText("Hello, I'm here to help you manage your tasks. Be sure to setup your bucket")

bot.setGetStartedButton((payload, chat) => {
  if (config.bucket === undefined){
    chat.say('Hello my name is Note Buddy and I can help you keep track of your thoughts')
    chat.say("It seems like you have not setup your bucket settings yet. That has to be done before we can proceed. Got it!")
  }
})
```

```


    BotUserId = payload.sender.id
  });

  bot.hear('setup', (payload, chat) => {
    const getBucketSlug = (convo) => {
      convo.ask("What's your buckets slug?", (payload, convo) => {
        var slug = payload.message.text;
        convo.set('slug', slug)
        convo.say("setting slug as "+slug).then(() => getBucketReadKey(convo));
      })
    }
    const getBucketReadKey = (convo) => {
      convo.ask("What's your buckets read key?", (payload, convo) => {
        var readkey = payload.message.text;
        convo.set('read_key', readkey)
        convo.say('setting read_key as '+readkey).then(() => getBucketWriteKey(convo))
      })
    }
    const getBucketWriteKey = (convo) => {
      convo.ask("What's your buckets write key?", (payload, convo) => {
        var writekey = payload.message.text
        convo.set('write_key', writekey)
        convo.say('setting write_key as '+writekey).then(() => finishing(convo))
      })
    }
    const finishing = (convo) => {
      var newConfigInfo = {
        slug: convo.get('slug'),
        read_key: convo.get('read_key'),
        write_key: convo.get('write_key')
      }
      config.bucket = newConfigInfo
      convo.say('All set :)')
      convo.end();
    }

    chat.conversation((convo) => {
      getBucketSlug(convo)
    })
  })

  bot.hear(['hello', 'hey', 'sup'], (payload, chat)=>{
    chat.getUserProfile().then((user) => {
      chat.say(`Hey ${user.first_name}, How are you today?`)
    })
  })
}

```

This website uses cookies to ensure you get the best experience on our website. [Learn more](#) 

```

bot.hear('config', (payloadc, hat) => {(/cookie-policy)
  if(JSON.stringify(config.bucket) === undefined){
    chat.say("No config found :/ Be sure to run 'setup' to add your bucket details")
  }
}

```

Got it!

```

    chat.say("A config has been found :) "+ JSON.stringify(config.bucket))
  })

  bot.hear('create', (payload, chat) => {
    chat.conversation((convo) => {
      convo.ask("What would you like your reminder to be? etc 'I have an appointment tomorrow fro
        datetime = chrono.parseDate(payload.message.text)
        var params = {
          write_key: config.bucket.write_key,
          type_slug: 'reminders',
          title: payload.message.text,
          metafields: [
            {
              key: 'date',
              type: 'text',
              value: datetime
            }
          ]
        }
      }
    })
    Cosmic.addObject(config, params, function(error, response){
      if(!error){
        eventEmitter.emit('new', response.object.slug, datetime)
        convo.say("reminder added correctly :)")
        convo.end()
      } else {
        convo.say("there seems to be a problem. . .")
        convo.end()
      }
    })
  })
})

bot.hear('active', (payload, chat) => {
  chat.say('finding all of your ongoing reminders.')
})

eventEmitter.on('new', function(itemSlug, time) {
  schedule.scheduleJob(time, function(){
    Cosmic.getObject(config, {slug: itemSlug}, function(error, response){
      if(response.object.metadata.date == new Date(time).toISOString()){
        bot.say(BotUserId, response.object.title)
        console.log('firing reminder')
      } else {
        eventEmitter.emit('new', response.object.slug, response.object.metafield.date.value)
        console.log('times do not match checking again at '+response.object.metadata.date)
      }
    })
  })
})

```

This website uses cookies to ensure you get the best experience on our website. [Learn more](#)

(/cookie-policy)

Got it!

```
bot.start()
```

This bot comes with a few interactions to get us off the ground. Each one is completely separate from the others so you can remove and modify them as you wish. I left it as bare as possible for this purpose. Let's take a look at them.

```
const bot = new BootBot({  
  accessToken: process.env.ACCESS_TOKEN,  
  verifyToken: process.env.VERIFY_TOKEN,  
  appSecret: process.env.APP_SECRET  
}) // 1
```

```
bot.setGreetingText("Hello, I'm here to help you manage your tasks. Be sure to setup your bucket
```

```
bot.setGetStartedButton((payload, chat) => {  
  if(config.bucket === undefined){  
    chat.say('Hello my name is Note Buddy and I can help you keep track of your thoughts')  
    chat.say("It seems like you have not setup your bucket settings yet. That has to be done be  
  }  
  BotUserId = payload.sender.id  
}); // 3
```

1. This creates an object that talks to the bootbot npm package. This allows us to use webhooks and such things.
2. This shows you a nice little message before you decide to message the Facebook page.
3. Creates a get started button as a barrier to entry before you message the bot. It also checks if you have setup the bucket config information yet. This is done later on by calling a certain command. You can also modify it so It is hardwired with your bucket information.

This website uses cookies to ensure you get the best experience on our website. [Learn more \(/cookie-policy\)](#)

Got it!

```

bot.hear('setup', (payload, chat) => { // 1
  const getBucketSlug = (convo) => { // 2
    convo.ask("What's your buckets slug?", (payload, convo) => {
      var slug = payload.message.text;
      convo.set('slug', slug) // 3
      convo.say("setting slug as "+slug).then(() => getBucketReadKey(convo)); // 3
    })
  }
  const getBucketReadKey = (convo) => {
    convo.ask("What's your buckets read key?", (payload, convo) => {
      var readkey = payload.message.text;
      convo.set('read_key', readkey)
      convo.say('setting read_key as '+readkey).then(() => getBucketWriteKey(convo))
    })
  }
  const getBucketWriteKey = (convo) => {
    convo.ask("What's your buckets write key?", (payload, convo) => {
      var writekey = payload.message.text
      convo.set('write_key', writekey)
      convo.say('setting write_key as '+writekey).then(() => finishing(convo))
    })
  }
  const finishing = (convo) => {
    var newConfigInfo = {
      slug: convo.get('slug'),
      read_key: convo.get('read_key'),
      write_key: convo.get('write_key')
    }
    config.bucket = newConfigInfo // 4
    convo.say('All set :)')
    convo.end();
  }

  chat.conversation((convo) => {
    getBucketSlug(convo) // 5
  })
})

```


1. This initiates a function that listens for specific keywords. Here we are listening for 'setup' but it can be changed to be anything. It can even accept regex statements.
2. Creates a function that can be called later to start the chain.
3. Takes what you send as an answer and sets that to a slug value that can be called in this instance of the conversation. If you started another conversation later in a separate instance this value would not be remembered.
(/cookie-policy)
4. Now we are starting to finish up the setup process with our final touches. First thing we have to do is get all of the information together. Right here we are grabbing all of the
Got it!

info by calling 'convo.get'. Then we add it to the config object declared earlier.

5. This is where everything starts. We start the conversation and start passing the convo value around.

```
bot.hear(['hello', 'hey', 'sup'], (payload, chat)=>{  
  chat.getUserProfile().then((user) => {  
    chat.say(`Hey ${user.first_name}, How are you today?`)  
  })  
})
```

Here we are utilizing the 'bot.hear' method and being friendly to the user. Remember earlier when I said you can use regex to listen for user input? You can also use an array of specific words! When the user says 'hello' we grab there profile information from Facebook and greet them by name. It may seem weird at first but I promise robots knowing your name is normal.

This website uses cookies to ensure you get the best experience on our website. [Learn more](#) 
(/cookie-policy)

Got it!

```

bot.hear('create', (payload, chat) => {
  chat.conversation((convo) => {
    convo.ask("What would you like your reminder to be? etc 'I have an appointment tomorrow fro
    datetime = chrono.parseDate(payload.message.text) // 2
    var params = {
      write_key: config.bucket.write_key,
      type_slug: 'reminders',
      title: payload.message.text,
      metafields: [
        {
          key: 'date',
          type: 'text',
          value: datetime
        }
      ]
    } // 3
    Cosmic.addObject(config, params, function(error, response){ // 4
      if(!error){
        eventEmitter.emit('new', response.object.slug, datetime) //5
        convo.say("reminder added correctly :)")
        convo.end()
      } else {
        convo.say("there seems to be a problem. . .")
        convo.end()
      }
    })
  })
})
})
})

```

1. Inside of the conversation we ask the user a question and wait for the reply.
2. Now we take what the user said and parse it using Chrono, a natural date parsing package (<https://github.com/wanasit/chrono>).
3. We build the params object to be used with the Cosmic JS Object addition.
4. Now we take the Cosmic JS package and insert our new object using the params we created earlier.
5. Here we are sending a nodejs event emitter passing the slug from the return and the datetime we created earlier.

This website uses cookies to ensure you get the best experience on our website. [Learn more](#) (/cookie-policy)

Got it!

```

bot.hear('help', (payload, chat) => {
  chat.say('Here are the following commands for use.')
  chat.say("'create': add a new reminder")
  chat.say("'setup': add your bucket info such as slug and write key")
  chat.say("'config': lists your current bucket config")
})

```

This will return a series of messages telling you what you can and can't do with the bot.

```

eventEmitter.on('new', function(itemSlug, time) { // 1
  schedule.scheduleJob(time, function(){ // 2
    Cosmic.getObject(config, {slug: itemSlug}, function(error, response){ // 3
      if(response.object.metadata.date == new Date(time).toISOString()){ // 4
        bot.say(BotUserId, response.object.title) // 5
        console.log('firing reminder')
      } else {
        eventEmitter.emit('new', response.object.slug, response.object.metafield.date.value) //
        console.log('times do not match checking again at '+response.object.metadata.date)
      }
    })
  })
})

```

Now we are playing with event emitters these are neat functions provided by NodeJS out of the box. They work almost like calling functions but they are way more versatile. 6. We create an event emitter that listens for the new event to be passed. 7. Now we use the schedule package and tell it to wait until the time you passed earlier. 8. Next we call cosmicjs and grab the specific object you just created. 9. Now we take the response and compare it to the time you passed originally if the time changed we then send it back to the event emitter and check again at the changed time. 10. here we call the event emitter again.

If you did everything correctly you should be able to start the bot up by typing in a few commands.

```
npm run start
```

Now in a different terminal window

This website uses cookies to ensure you get the best experience on our website. [Learn more \(/cookie-policy\)](#)

Got it!

This will use local tunnel to let your application talk to Facebook. Now you can modify this as much as you want and it should work just like you want it to. Now that you have your notes saved to the Cosmic JS API, you can retrieve them and send them to any connected application. Thanks for reading, if you have any questions reach out to us on Twitter (https://twitter.com/cosmic_js) and join the Slack community (<https://cosmicjs.com/community>).

← [Back to Blog Posts \(/blog\)](/blog)

Get Started

[Create Free Account \(/signup\)](/signup)

Products

[CMS API \(/cms-api\)](/cms-api)

[Apps \(/apps\)](/apps)

[Extensions \(/extensions/\)](/extensions/)

[Pricing \(/pricing\)](/pricing)

[Enterprise \(/enterprise\)](/enterprise)

Developers

[Documentation \(/docs\)](/docs)

[This website uses cookies](/developers) to ensure you get the best experience on our website. [Learn more \(/cookie-policy\)](/cookie-policy)

[Articles \(/articles\)](/articles)

[Integrations \(/integrations\)](/integrations)

[Got it!](#)

[Knowledge Base \(/knowledge-base\)](/knowledge-base)

[Status \(http://cosmicstatus.com\)](http://cosmicstatus.com)

Resources

[Getting Started \(/getting-started\)](/getting-started)

[Community \(/community\)](/community)

[Case Studies \(/case-studies\)](/case-studies)

[Why Use a CMS API? \(/why-cms-api\)](/why-cms-api)

Company

[Blog \(/blog\)](/blog)

[About \(/about\)](/about)

[Contact Us \(/contact\)](/contact)

[Terms & Privacy \(/terms\)](/terms)

© 2018 Cosmic JS Inc.

 [Slack \(https://cosmicslack.now.sh\)](https://cosmicslack.now.sh)

 [GitHub \(https://github.com/cosmicjs\)](https://github.com/cosmicjs)

 [Twitter \(https://twitter.com/cosmic_js\)](https://twitter.com/cosmic_js)

 [YouTube \(https://www.youtube.com/channel/UCDzCWWJLVWLpoHB5oGH0H-A\)](https://www.youtube.com/channel/UCDzCWWJLVWLpoHB5oGH0H-A)

 [Facebook \(https://facebook.com/cosmicjs\)](https://facebook.com/cosmicjs)

 [LinkedIn \(https://www.linkedin.com/company/cosmic-js\)](https://www.linkedin.com/company/cosmic-js)

 [Instagram \(https://instagram.com/cosmic_js_\)](https://instagram.com/cosmic_js_)

This website uses cookies to ensure you get the best experience on our website. [Learn more ↗ \(/cookie-policy\)](/cookie-policy)

Got it!