



Upgrade

Open in app



Alex · Follow

Apr 15, 2020 · 10 min read · Listen



How to send emails using NodeMailer, gmail and OAuth2

Hi there, I thought that since sending emails was one of the first features of my app I am building I thought it would be cool to do a tutorial on it!



The email will be sent from an API end point and will use OAuth2 credentials to send them, this is much better from a security point of view as don't have to store actual email credentials but instead store and use access tokens and these can only be used for certain purposes for a short amount of time.

Since we will be sending the email from an API endpoint we will first need to create a NodeJS project with the command below in the terminal

```
npm init myamazingproject
```

You can call your project whatever you like, now after you have ran this statement you will need to open the project in your favourite code editor I always use VS Code. Now create a file in the root of the project named "server" and make sure it is a JavaScript file, we will now need to install multiple packages including express, body-parser, nodemailer and google apis. You can do this using the command below





Upgrade

Open in app

The package nodemailer is basically a package to allow us to send emails through NodeJS.

We will now need to require all 4 packages in our server.js file so we can use them. We have created an extra variable called “OAuth2” which is equal specifically to the OAuth2 section of the google apis module.

```
const express = require('express');

const bodyParser = require('body-parser');

const nodemailer = require('nodemailer');
const { google } = require("googleapis");
const OAuth2 = google.auth.OAuth2;
```

We then also create a “PORT” const which is set to either the environment variable of the system it is deployed to i.e. Heroku for example. Otherwise it is set to port 3000. This matters a lot more if you are deploying the API to an online hosting service, if you were to just set the variable to 3000 and deployed it to Heroku or Amazon then an error would occur as both those services automatically configure the environment variable port for you.

We then also need to create a variable which is an instance of the express module, i have called it “app” for my project.

```
const PORT = process.env.PORT || 3000;

const app = express();
```

One additional line of code which we will need for this tutorial is

```
app.use(bodyParser.json());
```

This uses the instance of the “express” object and also uses the variable we created to be equal to the “body-parser” module. It is basically allowing us to parse application/json data in the body of our requests.

I have also created a simple default endpoint which simply returns a message in the response.

```
app.get('/', function(req, res){

  res.send({

    message: 'Default route in email tutorial project'

  })

});
```

Furthermore, to start the server you will need to add a “start” script in the package.json file so the server.js file gets ran when the command “npm start” is ran. Below is an example of this and how the “scripts” section should look.

```
"scripts": {

  "test": "echo \"Error: no test specified\" && exit 1",
```





Upgrade

Open in app

Finally, I have added a piece of code that runs the server and listens on a certain port which takes in the “PORT” variable as the first parameter and a callback as the second. This is always at the bottom of the file

```
app.listen(PORT, function (req, res) {  
  console.log(`Listening on port ${PORT}`);  
})
```

Onto the meat and potatoes of this tutorial, with nodemailer you can use different services to send the email from, for this tutorial I will be using gmail, one way you can send emails in nodemailer is directly entering your email and password you want the email to be sent from.


However if you deploy this you might have issues with gmail blocking the email being sent if the server you deployed the API to is in a different location, meanwhile, for this tutorial I will be showing you how to incorporate OAuth2 access tokens into your project so all you need is your email to send from and a bunch of tokens that will be generated for you further into the tutorial.

The first thing you will need to do is create a new project in google cloud console

Google Cloud Platform
Google Cloud Platform lets you build, deploy, and scale applications, websites, and services on the same infrastructure...
console.cloud.google.com


This will enable you to create a project that will allow you to create credentials to send emails via nodemailer. Once you have gone onto this website and logged in you will see a button labelled “select a project”, click on this and then click on the button that says “new project”. The screen will look like the below

New Project

 You have 12 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)
[MANAGE QUOTAS](#)

Project name *
 ?

Project ID: It cannot be changed later. [EDIT](#)

Location *
 [BROWSE](#)

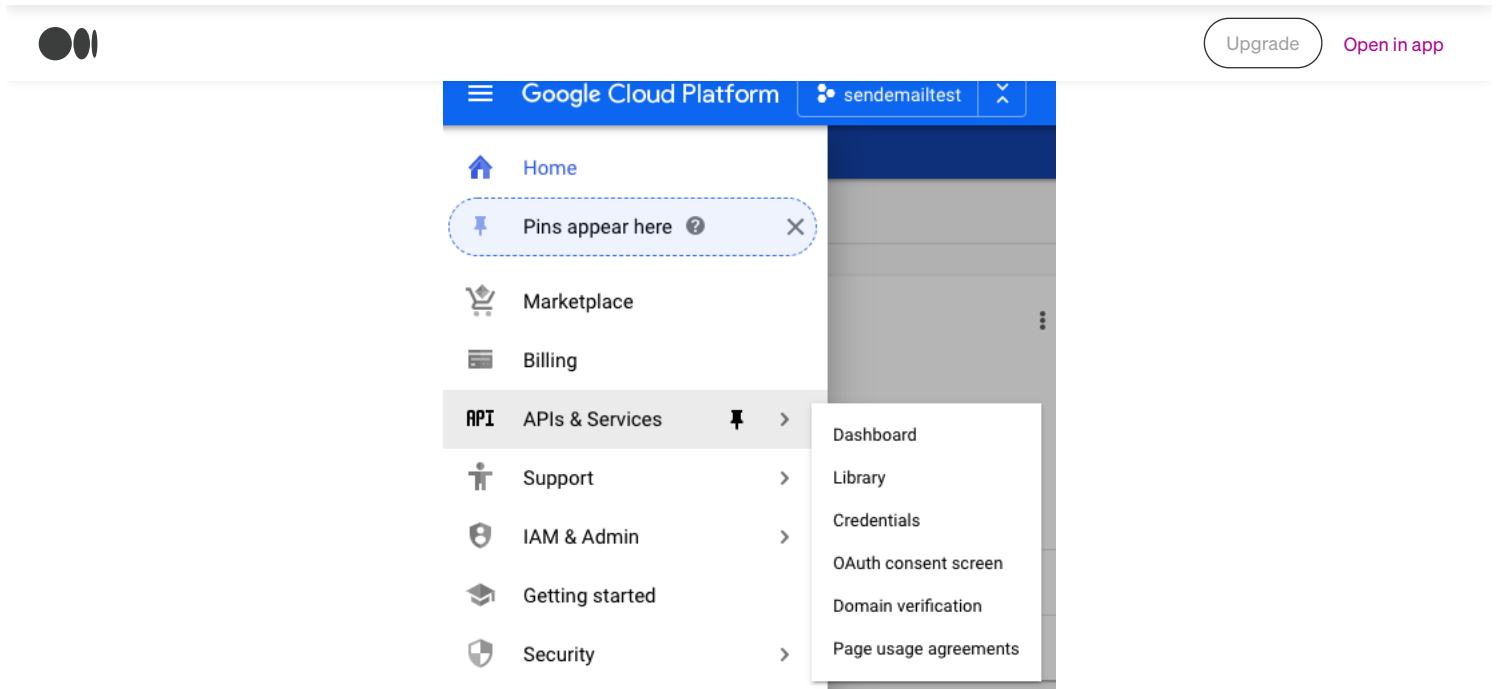
Parent organisation or folder

[CREATE](#) [CANCEL](#)

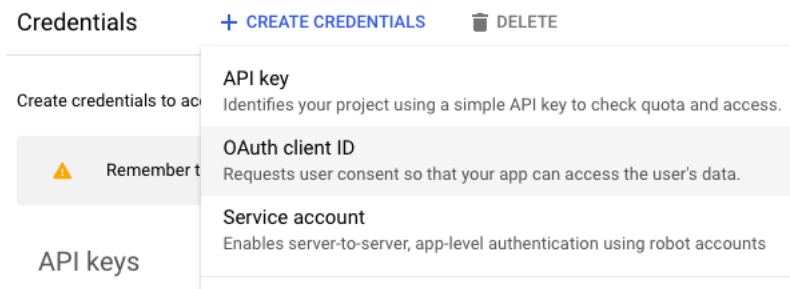
Click “create”, the process of creating the project will then begin, after it has finished click on the notifications bell and click on the latest notification, this will take you directly to the projects dashboard.

Next step is to click on the hamburger menu on the top left, hover over “APIs & services” and then click on “credentials”.





In addition, click on the “create credentials” button and then click on “OAuth client ID”



You will now be confronted with a screen saying you need to configure your consent screen, click on this button on the top right of the page.

On the next page set the user type as “external”

Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.

User Type

☐ Internal ?

Only available to users within your organisation. You will not need to submit your app for verification.

☐ External ?

Available to any user with a Google Account.

CREATE

In the next screen set a name for the application and then click save at the bottom, now click on the “credentials” tab on the left.



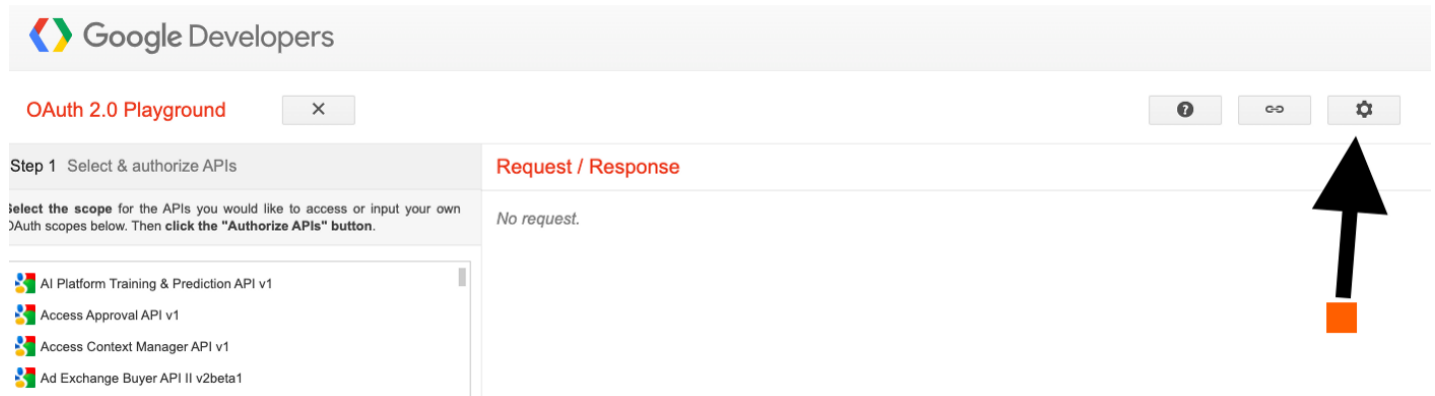
Upgrade

Open in app

We will be using the above link to obtain our unique client and access tokens.

Now click create and after a few seconds a popup should come up showing your **Client ID** and **Client secret** (Keep these safe somewhere!)

Our next step is to navigate to a google oauth playground where we can put our client ID and secret to use. Go to this link <https://developers.google.com/oauthplayground> and after you have done this click on the “cog” icon on the top right of the page.



After clicking on this a menu will open, click on the checkbox labelled “Use your own OAuth credentials”, 2 new textboxes will show now within that settings menu labelled client ID and client secret, enter the credentials (client id and client secret) you have just received from the google project (the ones you should have kept safe ;)) into the textboxes.

Once you have done this on the left hand side of the page there should be a textbox with a placeholder of “Input your own scopes”, paste the text <https://mail.google.com/> into that box and then click “Authorize APIs”.

If you are logged in to an account, the below screen should show (If you are not logged into an account log in to the one that you used to create the google cloud console project earlier)



This app isn't verified

This app hasn't been verified by Google yet. Only proceed if you know and trust the developer.

If you're the developer, submit a verification request to remove this screen. [Learn more](#)

[Advanced](#)

BACK TO SAFETY

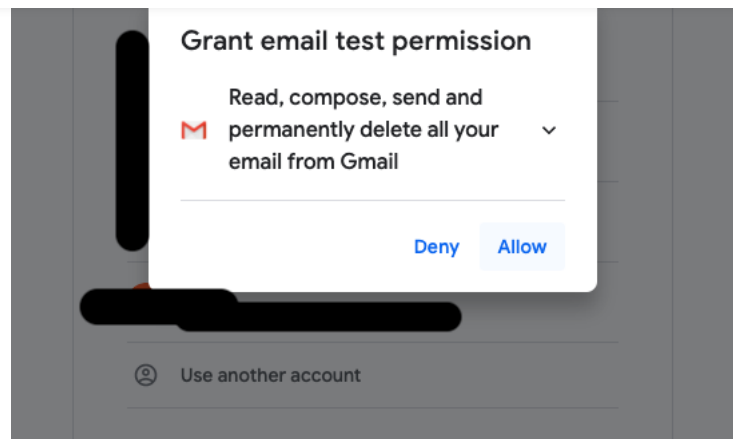
Clicked on the word “Advanced”, then click on “go to yourprojectname” this should then show the below





Upgrade

Open in app



Once you click “Allow” another screen should pop asking you to confirm that you are allowing the app access. Click on “Allow” again, you will then be redirected to the OAuth playground website with the “Authorization code” textbox already filled in with a value, now click on the button labelled “Exchange authorization code for tokens”.

This will create a refresh and access token (Once again keep these safe!)

Step 1 Select & authorize APIs

Step 2 Exchange authorization code for tokens

Once you got the Authorization Code from Step 1 click the **Exchange authorization code for tokens** button, you will get a refresh and an access token which is required to access OAuth protected resources.

Authorization code:

[Exchange authorization code for tokens](#)

Refresh token:

Access token: [Refresh access token](#)

☐ Auto-refresh the token before it expires.

The access token will expire in **3464** seconds.

Note: The OAuth Playground will automatically revoke refresh tokens after 24h. You can avoid this by specifying your own application OAuth credentials using the Configuration panel.

We now need to go back to our project and in the “server.js” code and create a variable equal to an instance of an “OAuth2” object, create this just underneath the “app.use(bodyParser.json());” statement. The first value entered inside the parameter is the client id, the second value is the client secret and the third value is the redirect url.

```
const myOAuth2Client = new OAuth2(
  "client ID goes here",
  "client secret goes here",
  )
```

We next need to use a built in method named “setCredentials” to set the value of the refresh token.





Upgrade

Open in app

```
refresh_token:"refresh token from oauth playground goes here"

});
```

Since our access token will expire within a specific amount of time we need to write a line of code that will retrieve a new access token when it expires.

```
const myAccessToken = myOAuth2Client.getAccessToken()
```

We now need to create a transport object from the nodemailer instance that will contain the service we will use which will be gmail as well as necessary credentials. Write the below in your “server.js” file

```
const transport = nodemailer.createTransport({
  service: "gmail",
  auth: {
    type: "OAuth2",
    user: "your email here", //your gmail account you used to set the project up in google cloud
console"
    clientId: " Client ID Here",
    clientSecret: "Client Secret Here",
    refreshToken: "Refresh Token Here",
    accessToken: myAccessToken //access token variable we defined earlier
  }
});
```

We now need to create another end point and this time it will be a “POST” request named “sendemail” it looks like the below in the server.js file.

```
app.post('/sendemail',function(req,res){

})
```

In addition, a JavaScript object will need to be created inside of this new endpoint that will contain a few options which are self-explanatory, i have called this object “mailOptions” and will look like the below. (If you want more html in your email you could create a variable prior to the “mailOptions” object and write it there, as it will look neater, the “to” value is set the email value found within the body of the request which we will be setting in postman.)

```
const mailOptions = {
  from: 'youremailaddresshere@email.com', // sender
  to: req.body.email, // receiver
  subject: 'My tutorial brought me here', // Subject
  html: '<p>You have received this email using nodemailer, you are welcome ;)</p>'// html body
}
```

To actually send an email we will need to use the “transport” variable that we created earlier and use the sendMail method of it and pass in the “mailOptions” object as the first parameter, the second parameter will be a callback function that either returns a successful result or an error. We will do basic error checking and if there is no error when we test this bad boy in postman we will send a legitimate response





Upgrade

Open in app

```
const mailOptions = {

  from: 'youremailaddresshere@email.com', // sender
  to: req.body.email, // receiver
  subject: 'My tutorial brought me here', // Subject
  html: '<p>You have received this email using nodemailer, you are welcome ;)</p>'// html body

}

transport.sendMail(mailOptions,function(err,result){

  if(err){
    res.send({
      message:err
    })
  }

  }else{
    transport.close();

    res.send({
      message:'Email has been sent: check your inbox!'
    })
  }
})
})
```

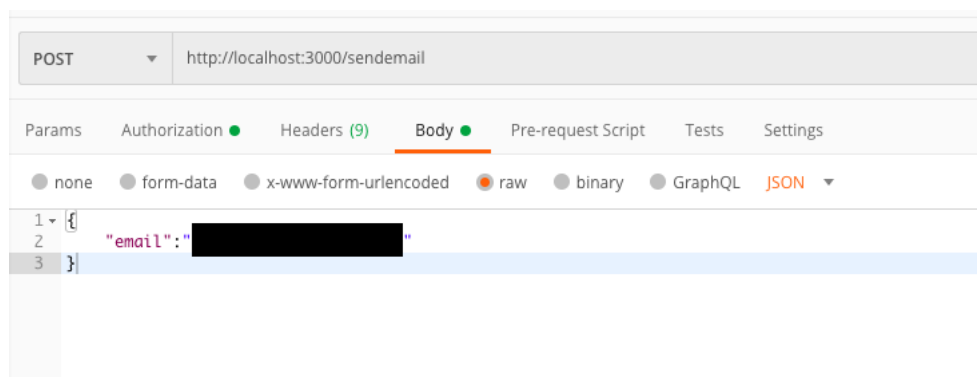
Postman is a really great tool for testing local or deployed systems request and response status's. You can download it from the link below

Download Postman App

Be the first to experience new Postman features! Can't wait to see what Postman has in store for you? Be the first to...

www.postman.com

Run your server using the command “npm start” in the terminal of your chosen code editor. Now open Postman and make sure the method is set to “POST”, the text is set to “raw” and “JSON” is chosen in the data type drop down, finally, in the body tab ensure the email key and value are in between double quotes within curly brackets like below, set the email value equal to the email address you want to send to!



Then click send (which is to the right of the url) and watch your email you set up get sent to your inbox ;)





Upgrade

Open in app

You have received this email using nodemailer, you are welcome ;)

There you have it, a very simple message sent. However, you can get more complex with nodemailer in terms of multiple recipients, more complicated html and even bulk mailing!

I hope you enjoyed this article and remember to give a clap or share if you learnt something. Anything I can do to be better or improve this article feel free to tell me.

Feel free to look at my own blog too — <https://cowboycode.co.uk/>

Stay safe!

Get an email whenever Alex publishes.



Subscribe

Emails will be sent to matias@everythink.ai.

[Not you?](#)

