Tutorial: Build a REST API for Your Mobile Apps using Node.js

by Edward Jiang
May 10, 2016

Mobile (https://stormpath.com/blog/category/mobile), Node (https://stormpath.com/blog/category/node)

Behind every great mobile app is a great backend, but building a REST API for your app can be a bit daunting if you haven't done so before. Fear not! This tutorial will show you how to build your first REST API using Node.js, and connect it to an iOS or Android app!

As a mobile app developer, I love to build REST APIs using the Node.js backend for several reasons:

- It's easy to work with JSON in JavaScript, because JSON stands for JavaScript Object Notation!
- Node.js is lightweight and easy to get started with.
- Node.js gives you fine-grained control over your request and responses.

However, when building an API, figuring out how to handle authentication is always a huge challenge. Authentication refers to the practice of understanding exactly who is accessing your data, and securely doing so is not easy. We built Stormpath to help developers easily add secure authentication to their apps, and we'll also show you how to include this in your Node.js-powered REST API.

Node.js and Express for Android & iOS

Today, we'll build the backend powering Stormpath Notes, a simple note taking app that syncs data online. With this knowledge, maybe you'll be able to build a solid competitor to Evernote, OneNote, and other industry giants!

We have a separate tutorial for how to build the iOS (https://stormpath.com/blog/build-note-taking-app-swift-ios/) and Android (https://stormpath.com/blog/build-user-authentication-for-android-app/) apps that can use this backend, so check them out once you've finished this tutorial.

Today we'll be implementing the following endpoints:

`GET /notes` – returns the notes for the authenticated user in the form of a JSON object.

`POST /notes` – takes a JSON object with the notes and saves it for the authenticated user.

The JSON object takes the form of:

```
{"notes": "The notes the user saved"}
```

Stormpath's backend integrations (Express-Stormpath (https://github.com/stormpath/express-stormpath/) being one of them) also expose a common API, including `/register`, `/oauth/token` (for logging in), and other endpoints so we don't have to worry about coding them! You'll learn more about those endpoints later in this tutorial.

We'll also be using the following tools to build and test this backend:

- Node.js (https://nodejs.org/en/) – the runtime for our app.
- Express.js (http://expressjs.com) – a popular, lightweight framework for building Node.js apps
- Stormpath (https://stormpath.com) – a backend service for handling user authentication
- Postman (https://www.getpostman.com) – a HTTP client that allows us to make custom requests to the REST API

Also, the finished code is live and hosted at https://stormpathnotes.herokuapp.com/, so you can play around with the API, and the code is available on GitHub (https://github.com/stormpath/stormpath-express-mobile-notes-example)

Starting Your Node.js Mobile App

To get started, make sure that you:

1. Install a great text editor. I like Sublime (https://www.sublimetext.com), Atom (https://atom.io), or VS Code (https://code.visualstudio.com)
2. Install the Node.js (https://nodejs.org/en/) runtime. You can install the "Current" (v6) version of Node.
3. Sign up for a Stormpath Account (https://api.stormpath.com/register)
4. Install Postman (https://www.getpostman.com)

Setting Up Your App

To get started with your Node.js project, we'll use npm. npm is a package manager for JavaScript projects, and allows you to install JavaScript tools and modules for your project.

It's automatically included in your Node.js install, so all we have to do is start using it in the command line!

To get started, create a folder, navigate to it in the command line, and run:

```
$ npm init
```

npm will ask you a lot of questions — feel free to leave them as the default!

Once you're done, npm will create a `package.json` file in your folder. `package.json` keeps track of your project information and dependencies!

In addition, we need to install express, a popular, minimalist web framework for Node.js. Install it by running this command:

```
$ npm install --save express
```

With this command, npm will install express, and save its version to the package.json file. Now, we can use express in our project.

Code Your First REST API

We're all set with the project, so let's start writing our REST API! Create a file named `index.js`, and type in the following:

```
var express = require('express')

var app = express()

app.get('/notes', function(req, res) {
  res.json({notes: "This is your notebook. Edit this t
})

app.listen(3000)
```

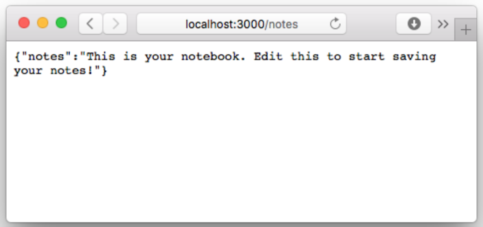
Simple, right? In six lines of code, we:

- Imported the Express module
- Initialized the Express object
- Added a handler for `GET /notes` which responds to the request with a JSON object with sample notes
- Told express to listen to HTTP requests on port 3000

Try running it in the command line:

```
node index.js
```

Congrats, you now have a basic REST API with one endpoint, and it's running on your machine! Try visiting it at http://localhost:3000/notes and see what happens.



```
{"notes":"This is your notebook. Edit this to start saving your notes!"}
```

Adding Authentication to Your Mobile API

Having a `/notes` endpoint is great, but for a note-taking app, we can't have everyone viewing the same set of notes. We need to add authentication to our app. Authentication allows our REST API to know who is accessing the notes endpoint, and only show notes to users that are logged in. For that, we're going to use Stormpath.

Log into the Stormpath Account (https://api.stormpath.com/login) you created, and click on the Node.js quickstart. In the first page, you'll see a section showing you how to set up your API Keys by using environment variables. Environment variables allow you to configure your application without using code, which is great when you have multiple development environments or even servers. Paste the API Keys from the quickstart into your command line:

```
# This is an example; use the values in Stormpath inst
$ export STORMPATH_CLIENT_APIKEY_ID=EXAMPLE
$ export STORMPATH_CLIENT_APIKEY_SECRET=EXAMPLE
$ export STORMPATH_APPLICATION_HREF=https://api.stormp
```

Let's install the `express-stormpath` integration, so we can get a standard API for logging in, and other tasks:

```
$ npm install --save express-stormpath
```

Now, let's add Stormpath to our application in `index.js` :

```
var stormpath = require('express-stormpath')
```

Delete the old code for the `/notes` endpoint, and replace it with:

```
app.use(stormpath.init(app, {
  expand: {
    customData: true,
  },
  web: {
    produces: ['application/json']
  }
}))

app.get('/notes', stormpath.apiAuthenticationRequired,
  res.json({notes: req.user.customData.notes || "This
})
```

In this bit of code, we're initializing `express-stormpath` and attaching it to Express. We're asking it to preload user `customData` when retrieving objects from Stormpath. We need customData to store our user's notes, and this will make them easier to fetch. In addition, we configure it to only produce `application/json` outputs, as otherwise it can also produce HTML views. Express-Stormpath is highly configurable, and supports many different options (https://docs.stormpath.com/nodejs/express/latest/configuration.html) for building your API.

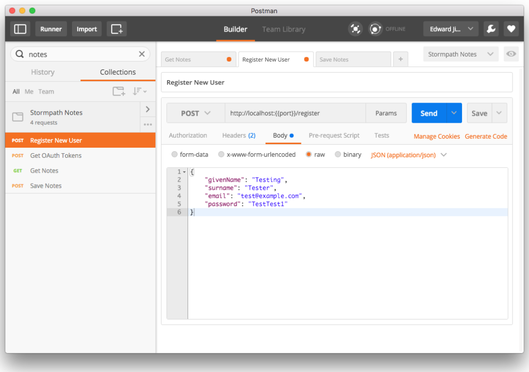
To require authentication, we're attaching the `stormpath.apiAuthenticationRequired` middleware to the `/notes` endpoint. This will inspect the headers on every request, and look for an access token. If there is no access token, it will respond with a `401 Unauthorized` error.

Finally, we respond with the notes in the customData, or, if not present, a default message for your notes.

Let's run this code, and try using our new API. But this time, instead of using a browser, we're going to use Postman.

Testing Your Node.js REST API With Postman

To get started with Postman, let's try registering a new user in our app. To do this, let's first install the Stormpath Notes Postman collection (https://app.getpostman.com/run-collection/6c51d38e029b74a57067#?env%5BStormpath%20Notes%5D=W3sia2V5IjoicG9ydCIsInZhbHVlIjoiMzAwMCIsInR5cGUiOiJ0ZXh0IiwiZW
This will add pre-made requests to your Postman app, which will make it easier for you to test out your API.



http://localhost:{{port}}/register

```
{
    "givenName": "Testing",
    "surname": "Tester",
    "email": "test@example.com",
    "password": "TestTest1"
}
```

Make sure that your server is running, and then click and send the "Register New User" request in Postman. This will load a saved request for the `/register` route. Feel free to click around, and see what Postman is sending to `/register`. If you click "Send", the server will return a JSON object of the new account. Congrats, you've just registered an account on your Stormpath instance! You can double check if the account has been registered in the Stormpath Admin Dashboard (https://api.stormpath.com/).

Getting an OAuth 2 Token

For mobile and web clients, Stormpath uses OAuth 2 access and refresh tokens for authentication. An access token is used for direct authentication with the API, but expires on a periodic basis. When the access token expires, the refresh token can be used to get a new access token.

This makes your API more secure and scalable (https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication#the-benefits-of-tokens).

To get your OAuth 2 tokens, click on and send the “Get OAuth Tokens” request in Postman. You’ll see an access and refresh token pair pop out in JSON:

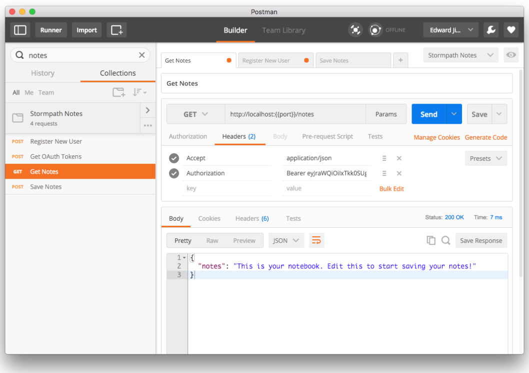
```
{
  "access_token": "eyJraWQiOiIxTkk0SUg3ME9WN1Y3V...",
  "refresh_token": "eyJracy81bWVJUXRuWlBqS0E1Nk5...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "stormpath_access_token_href": "https://api.stormpat
}
```

Getting the Notes

Now that we have our access token, let’s use it to authenticate a request to the API we’ve created. Open up the “Get Notes” request in Postman, and insert your access token into the Authorization header so it says:

`Authorization: Bearer eyJraWQiOiIxTkk0SUg3ME9WN1Y3V...`

Now run the request. You should see the default JSON response!



Get Notes

GET http://localhost:{{port}}/notes

Accept	application/json
Authorization	Bearer eyJraWQiOiIxTkk0SUg
key	value

Status: 200 OK Time: 7 ms

```
{
  "notes": "This is your notebook. Edit this to start saving your notes!"
}
```

Saving the Notes To Your REST API

We’ve gotten and tested out three of our endpoints. It’s time to write the final endpoint, so the user can save the notes back to the server.

In the command line, run:

```
$ npm install --save body-parser
```

In our `index.js` file, add:

```
var bodyParser = require('body-parser')

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: false}))

app.post('/notes', stormpath.apiAuthenticationRequired
  if(!req.body.notes || typeof req.body.notes != "stri
    res.status(400).send("400 Bad Request")
  }

  req.user.customData.notes = req.body.notes
  req.user.customData.save()
  res.status(200).end()
})
```