Using Node.js to Interact with Facebook's Graph API

Mar 12 2017

Here's some context about pairing Node.js with Facebook's Graph API.

The Bad:

- Facebook has a JavaScript SDK for interacting with their Graph API, but it's for client-side JavaScript – there is **no Node.js SDK**.
- Although there are several NPM packages for working with Facebook's Graph API, **no NPM package seems to be as mature as packages for other social media platforms** like Twit for the Twitter API. The best of the few seems to be facebook-node-sdk, but I hesitate to rely on it for a business critical app.
- The Graph API docs have **no examples using vanilla JavaScript** (although there are a few examples using their client-side SDK).
- At the time of this writing (March 12th, 2017), there are **very few blog posts or tutorial on the topic**.

The good:

- If you have some examples, using **Facebook's Graph API with Node.js is fun**.
- This blog post is filled with **useful examples using Node/Express**!

Contents:

What's outlined here is based on my app's needs. For example, most people probably don't need to get a list of pages a user manages. Despite this, the actions outlined below can be used as a model for nearly any other request.

A common need this post doesn't address is working with pagination. However, working with pagination seems straightf

As always, if you have ques · Contact form.

AJAX library

All the examples below use request-promise, so be sure to `npm install --save request request-promise` and

```
const request = require('request-promise');
```

in your code.

Node framework

The examples below you may notice a route or two. As I mentioned above, Express.js is my framework of choice, and it's used in several of the examples below.

1. Authorization & Access Tokens

> *To get started, you need an access token obtained through Facebook authentication.* ***If you have an access token at your disposal, you can skip the rest of this section.*** *If you need to get one, read on.*

Generally, an app acquires a user access token through an authentication procedure. Passport.js is a great authentication package for Node.js apps.

The Passport.js Facebook authentication strategy comes with a very useful demo app that uses Express.js. If you need to acquaint yourself with Passport, this repo is a great place to start.

Getting a token

Unfortunately, it isn't as easy as merely firing up the example linked to above and authenticating. **You must to have an app registered with Facebook.** Instructions for doing so can be found here.

Once you have a registered app, you'll need to use the `app_id` and `app_secret` in the config file of your authentication app. Adjust the code in the demo app so you `console.log()` the returned access token, as you'll need the token for the requests outlined below.

Getting POST permissions

Not all access tokens are created equal. If you intend to post to a user's feed or to a page managed by a user, you need to specifically ask … through a scope object in your authentication.

When you ask for these permissions, a user logging in will see a prompt stating your app is requesting these permissions. If the user clicks the okay button, an access token with the specified permissions is generated.

To request publish permissions include an object with a scope property in the `'/login/facebook'` route, as demonstrated below:

```
app.get('/login/facebook',
  passport.authenticate('facebook', {
    scope: ['publish_actions', 'manage_pages']
  }
));
```

If you don't include an object with the `scope` property, the tokens returned will have `'public_profile'` permissions only.

You may want other permissions, like `'user_about_me'`, `'user_photos'`, or `'user_feed'`, but the two in the code above are the only ones you need for publishing. Take a look at this page for an outline of different permissions you can request during authorization.

Facebook's App Review Process

Without submitting your app to an official FB review process, the above code will only give you `'public_profile'` access. To request additional permissions you have to prove that you're a well-intentioned developer with a well-intentioned app.

For the right to obtain access tokens with additional permissions, you'll have to fill out the 'App Review' section of your app's settings page in the Facebook Developers website.

The review process is quite an endeavor, and requires you to describe your app and submit a video walk-through of how your app is used.

Remember, the procedure for getting an access token is not related to interacting with the Graph API. The only reason I mention it is so you can obtain an access token (or an access token with permissions to post), as code examples below require one.

Graph API Search

11 min

If nave an access token, you

In this section, we'll look at querying users and pages through Facebook's `https://graph.facebook.com/v2.8/search` url. Other objects should be similar to these, so you can use the examples here as a guide. Here's a list of the object you can search:

- `user`
- `page`
- `event`
- `group`
- `place`
- `placetopic`

The search route below is a POST endpoint that requires a payload like the one below. The payload object has two properties: a search term (`queryTerm`) property, and a search type (`searchType`) property.

Although the Node/Express route is a POST route, the request we are sending to the Graph API is a GET.

For the app we are building, the value assigned to `searchType` could be `'page'` or `'user'` , and each type will return different properties. In reality, the `searchType` could be any of the objects listed above.

Check out the docs for the User object, and Page object to see the properties that belong to each object, and can be requested.

```
const payload = {
  queryTerm: 'Fiat',
  searchType: 'page'
}
```

Below is the route:

```
const request = require('request-promise');

module.exports = (app) => {

  // you'll need to have requested 'user_about_me' permissions
  // in order to get 'quotes' and 'about' fields from search
  const userFieldSet = 'name, link, is_verified, picture';
  const pageFieldSet = 'name, category, link, picture, is_verified';
```

11 min

```
app.post('/facebook-search', (req, res) => {
  const { queryTerm
  const options = {
    method: 'GET',
    uri: 'https://graph.facebook.com/search',
    qs: {
      access_token: config.user_access_token,
      q: queryTerm,
      type: searchType,
      fields: searchType === 'page' ? pageFieldSet : userFieldSet
    }
  };

  request(options)
    .then(fbRes => {
// Search results are in the data property of the response.
// There is another property that allows for pagination of results.
// Pagination will not be covered in this post,
// so we only need the data property of the parsed response.
      const parsedRes = JSON.parse(fbRes).data;
      res.json(parsedRes);
    })
});
}
```

Notice the the `fields` property and the `userFieldSet` and `pageFieldSet` constants. When searching **pages**, the code above requests the `'name, category, link, picture, is_verified'` fields to be returned. These are all part of the page public profile and do not require special permissions.

The `fields` requested when searching **users** are part of the user public profile. Although these fields are all publicly available, this data will not be returned unless specified in the `fields` property of the request.

The Graph API's search end point only returns publicly available information (*I think*), and there's a good chance that you'll want more than this; you'll probably want the additional information your user access token permits you.

In my case, my employer has clients that manage quite a few Facebook pages. One of these clients will want to search pages, chose one that they manage, and get more info. This requires a second API call – a request for a specific object (in this scenario, a `page` object) from the Graph API.

3. Querying a Single Graph Object

This section covers what you'll need to do to get information about a single object. The section starts with querying a single user, and ends with querying a single photo.

When querying the search route (see above), the id is not needed as part of the object type and fields specified. You need this id [illegible] 'photo', 'post', etc.).

Below is an example of querying a single user by id.

Permissions: If you have a Graph object's id, you can request detailed data as long as the access token you're using has permissions for the fields you are requesting.

The access token used in the request below has the following permissions:

- 'user_relationships'
- 'user_about_me'
- 'user_location'
- 'user_website'
- 'user_photos'
- 'user_posts
- 'email'

Given all the above permissions, the code below will return detailed user information.

Other Graph objects can be requested the same way. The only difference would be the fields specified.

If you don't have a *user* access token from authorization, you can use your *app's* access token. In the latter case, you'll only be able to request publicly available profile data.

```
app.get('/facebook-search/:id', (req, res) => {

  // you need permission for most of these fields
  const userFieldSet = 'id, name, about, email, accounts, link, is_verified, significant_other, relationship_s

  const options = {
    method: 'GET',
    uri: `https://graph.facebook.com/v2.8/${req.params.id}`,
    qs: {
      access_token: user_access_token,
      fields: userFieldSet
    }
  };
  request(options)
    .then(fbRes => {
      res.json(fbRes);
    })
})
```

11 min

Below is the JSON respons cess token all the permissions listed above).


```
{
  "id": "1458722400807259",
  "name": "Loren Stewart",
  "email": "youwish!",
  "location": {
    "id": "109434625742337",
    "name": "West Hollywood, California"
  },
  "accounts": {
    "data": [
      {
        "access_token": "youwish!",
        "category": "Internet Company",
        "name": "Codemore",
        "id": "1775471846051866",
        "perms": [
          "ADMINISTER",
          "EDIT_PROFILE",
          "CREATE_CONTENT",
          "MODERATE_CONTENT",
          "CREATE_ADS",
          "BASIC_ADMIN"
        ]
      }
    ],
    "paging": {
      "cursors": {
        "before": "MTc3NTQ3MTg0NjA1MTg2NgZDZD",
        "after": "MTc3NTQ3MTg0NjA1MTg2NgZDZD"
      }
    }
  },
  "link": "https://www.facebook.com/app_scoped_user_id/1458722400807259/",
  "is_verified": false,
  "significant_other": {
    "name": "Alex Galeczka",
    "id": "10210777928898847"
  },
  "relationship_status": "In a relationship",
  "website": "http://lorenstewart.me/",
  "picture": {
    "data": {
      "is_silhouette": false,
      "url": "https://scontent.xx.fbcdn.net/v/t1.0-1/p50x50/14463080_1313697801976387_8357030504322451215_n.jp
    }
  },
  "photos": {
    "data": [
      {
        "created_time": "2015-05-11T12:23:08+0000",
        "id": "994251753920995"
```

11 min


```
        "created_time": "2014-06-07T01:55:50+0000",
        "name": "Worker bee",
        "id": "8104522
      },
// the list goes on
    ],
    "paging": {
      "cursors": {
        "before": "T1RrME1qVXhOelV6T1RJd09UazFPakUwTXpFek5Ea3dORGs2TXprME1EZAzVOalF3TmpRM09ETTIZD",
        "after": "TXpReE1USXlOREkxT1RBNE5UZAzVPakV6TWpjNU1EQTFOakE2TXprME1EZAzVOalF3TmpRM09ETTIZD"
      }
    }
  },
  "feed": {
    "data": [
      {
        "message": "Hey all. Don't just delete Uber's app. Delete your account.",
        "created_time": "2017-01-30T02:20:41+0000",
        "id": "1458722400807259_1445514005461432"
      },
      {
        "message": "I'm teaching a React workshop at the end of February 🎉  \nPlease share with your develop
        "story": "Loren Stewart shared an event.",
        "created_time": "2017-01-29T21:00:17+0000",
        "id": "1458722400807259_1445248875487945"
      },
      {
        "story": "Loren Stewart updated his cover photo.",
        "created_time": "2017-01-04T02:56:35+0000",
        "id": "1458722400807259_1418730148139818"
      },
      {
        "story": "Loren Stewart at Artists Palette, Death Valley.",
        "created_time": "2016-12-31T04:19:52+0000",
        "id": "1458722400807259_1415002355179264"
      }
// the list goes on
    ],
    "paging": {
      "previous": "https://graph.facebook.com/v2.8/1458722400807259/feed?since=1485742841&access_token=youwish
      "next": "https://graph.facebook.com/v2.8/1458722400807259/feed?access_token=youwish!!"
    }
  }
}
```


Although a lot of information is returned, much important information is left out. For example, no urls are provided for the photos in my photo library. To get a photo's url, you have to ask for the `link` field of the photo. `photos` is a field of a `User` node; `link` is field of `photo`. To get the link of each photo in our response, we need to request a field (`link`) of a field (`photo`).

The syntax for requesting the field of a field is `field{nestedField}`. To request more than one nested field, separate them by commas like so: `field{nestedField1, nestedField2, nestedField3}`. While we're on the

If I only want two photos returned with my initial data I can specify `photos.limit(2)`. And if I want to include the link to the photo, the syntax would be `photos.limit(2){link, tags, comments.limit(3)}`.

A request for a user's last two photos, along with the link and the last two comments, looks like this:

```
const getMediaOptions = {
  method: 'GET',
  uri: `https://graph.facebook.com/v2.8/${user.facebook_id}`,
  qs: {
    access_token: user.access_token,
    type: 'user',
    fields: 'photos.limit(2).order(reverse_chronological){link, comments.limit(2).order(reverse_chronological)}'
  }
};
```

Notice that I specify the order in which I want the photos back. By specifying `.order(reverse_chronological)` I get the latest two photos.


```
{
  "photos":{
    "data":[
      {
        "link":"https://www.facebook.com/photo.php?fbid=994251753920995&set=a.393129324033244.98295.1000000853
        "comments":{
          "data":[
            {
              "created_time":"2015-05-11T19:29:01+0000",
              "from":{
                "name":"Danny Human",
                "id":"1245359685745239"
              },
              "message":".......   https://youtu.be/sGiZoE5eqZc",
              "id":"994251753920995_994327993913389"
            },
            {
              "created_time":"2015-05-11T15:24:23+0000",
              "from":{
                "name":"Jose Sapien",
                "id":"1245359685745235"
              },
              "message":"Wish I were there!",
              "id":"994251753920995_994404017239188"
            }
          ]
          "id":"994251753920995"
      },
      {
        "link":"https://www.facebook.com/photo.php?fbid=810452288967610&set=a.260137377332440.73629.1000000853
        "comments":{
          "data":[
```

```
            "created_time":"2015-04-19T04:17:01+0000",
            "from":{
              "name"
              "id":"
            },
            "message
            "id":"994251753920995_994327993913389"
          },
          {
            "created_time":"2015-04-19T01:56:13+0000",
            "from":{
              "name":"Jose Sapien",
              "id":"1245359685745235"
            },
            "message":"Is that real!",
            "id":"994251753920995_994404017239188"
          }
        ],

        "id":"810452288967610"
      }
    ]
}
```

Many thanks to Ian Williams for helping me out with the Graph API's unusual query syntax.

4. Querying for the Pages a User Manages

This request is probably not one you need to use very frequently, but if your app needs to manage the pages for which the user has admin permissions, this example will be useful.

In the example below, notice **the `fields` property is not included** because the data you're requesting is implicit in the url.

The user query in section #3 above also returned this information because the user access token that is used has permissions for this information, and `'accounts'` is specified in the `fields` property of the request.

If you have `accounts` permissions, and only want a list of accounts the user manages (rather than all user data), then this is the query you want to perform.

```
const options = {
// let's assume id below is 1458722400807259,
// which is my user id
  method: 'GET',
  uri: `https://graph.facebook.com/v2.8/${req.params.id}/accounts`,
  qs: {
      access_token: user_access_token
```

```
};
request(options)
  .then(fbRes => {
    res.json(fbRes);
  })
```


Here's the response:

```
{
  "data": [
      {
          "access_token": "youwish!",
          "category": "Internet Company",
          "name": "Codemore",
          "id": "1775471846051866",
          "perms": [
              "ADMINISTER",
              "EDIT_PROFILE",
              "CREATE_CONTENT",
              "MODERATE_CONTENT",
              "CREATE_ADS",
              "BASIC_ADMIN"
          ]
      }
  ],
  "paging": {
      "cursors": {
          "before": "MTc3NTQ3MTg0NjA1MTg2NgZDZD",
          "after": "MTc3NTQ3MTg0NjA1MTg2NgZDZD"
      }
  }
}
```

The request above returns the Facebook pages I manage, and since I only manage one page, the list is short!

Similar to other responses, there's no url to the Facebook page; that has to be requested separately (see section #5 below).

The user permission types with regard to each page/account are also listed in the response, and may be useful to your project.

So many access tokens

When you query for user accounts (i.e. pages they manage), each account returns an access token. If you want to perform an action on that page, this is the token to use.

For example, if I wanted to post content to my Facebook page, I would need to use the page access token above (not my user access token) in order to get the page token, the page token has my permissions with regard to that page. For some pages I may be able to post content, other pages I may only be a

5. Posting Content

Below are examples for posting text, images, and videos (the latter have to be under 1GB in size) to a page or a user's feed. We'll start with posting text to a feed.

Posting text

```
const id = 'page or user id goes here';
const access_token = 'for page if posting to a page, for user if posting to a user\'s feed';

const postTextOptions = {
  method: 'POST',
  uri: `https://graph.facebook.com/v2.8/${id}/feed`,
  qs: {
    access_token: access_token,
    message: 'Hello world!'
  }
};
request(postTextOptions);
```

Posting an image

```
const id = 'page or user id goes here';
const access_token = 'for page if posting to a page, for user if posting to a user\'s feed';

const postImageOptions = {
  method: 'POST',
  uri: `https://graph.facebook.com/v2.8/${id}/photos`,
  qs: {
    access_token: access_token,
    caption: 'Caption goes here',
    url: 'Image url goes here'
  }
};
request(postImageOptions)
```

Posting a video

Videos over 1GB in size need to be chunk uploaded, and I won't go over that here (because I don't have a good solution yet!). Fortunately, uploading a video (under 1GB) is no more difficult than an image or text.

```
const postVideoOptions = {
  method: 'POST',
  uri: `https://graph.
  qs: {
    access_token: acce
    description: 'Caption goes here ,
    file_url: 'Video url goes here'
  }
};
request(postVideoOptions);
```

Uploading media without inclusion in feed

It's possible to add an image or video to the library of the user/page without having it show up in their feed. To do this, in the `qs` property of the request, add the property `no_story` and set it to `true`. If `no_story` isn't specified, Facebook defaults to `no_story: false` and the upload is visible in the user's/page's feed.

Getting the url of your post

After the POST you can use the returned `id` or `post_id` to make another request for the post's url. Uploading an image returns an object with `id` and `post_id` properties, while uploading a video or posting text returns only an `id` property.

Here are some example responses:

```
// example video/text post response
{
  id: '323262661405199'
}

// example image post response
{
  id: '323262661405199',
  post_id: '309953479402795_323262661405172'
}
```

To obtain a url, the field you need to request is `'permalink_url'`.

```
const fb_id = 'the returned post_id or id goes here';
const type = '`video` or `post`, see comment below'

const getUrlOptions = {
  method: 'GET',
  uri: `https://graph.facebook.com/v2.8/${fb_id}`,
    qs: {
      access_token: user_access_token,
```

```
10        // and should be 'video' for a video url
          type: type,
          fields: 'permali
13      }
14  };
15  return request(getUrl(
16    .then(fbUrlRes => {
17      const permalink = JSON.parse(fbUrlRes).permalink_url;
18        if(video) {
19          permalink = `https://www.facebook.com${permalink}`;
20        }
21        return {postUrl: permalink};
22    })
```


Image and text posts return a full url. However, the video response returns only the end of the url so you'll need to construct the full url yourself (see line 19 above). In order to construct a video url, you have to prepend `'https://www.facebook.com'` to the returned value.

If you'd like to be notified when I publish new content, sign up for my mailing list in the navbar.

WRITTEN BY

Loren Stewart

Writing code with the wonderful devs at Speaker, a data-driven influencer marketing company. Also an educator for Codemore.

Los Angeles, CA http://codemore.io

TAGS

node.js api express.js JavaScript

A Functional Approach to Data Validation

Get updated with our new content

email address

✓

validating data usually st... pp grows, validation logic can quickly become ...

11 min

04 JUN 2017

PREVIOUS POST

Twitter API: Uploading Videos using Node.js

I work for an influencer marketing and data company, Speakr, and we work with social media APIs all the…

03 FEB 2017