Node.js: how to publish a post on Linkedin via API with ExpressJS

In this article we will see how to publish a post on Linkedin via API with ExpressJS.

The procedure to follow is quite complex. As a first step we must create an application in the developer section of Linkedin. Once the application has been verified, i.e. associated with a company page, we can obtain the two fundamental credentials, namely the client ID and the client secret.

Also in our app, we must set up a callback URL so that users are redirected to our site after consenting to use our application.

We save the configuration in a dedicated file.

```
'use strict';

module.exports = {
    clientId: '',
    clientSecret: '',
    authorizationURL:
'https://www.linkedin.com/oauth/v2/authorization',
    accessTokenURL:
'https://www.linkedin.com/oauth/v2/accessToken',
    redirectURI: 'http://localhost:3000/callback',
    sessionName: '',
    sessionKeys: ['', '']
};
```

So let's create a class to manage the interaction with the Linkedin API.

```
'use strict';

const request = require('request');
const { clientId, clientSecret, authorizationURL,
redirectURI, accessTokenURL } = require('../config');

class API {

}

module.exports = API;
```

The second step is to have users login on Linkedin by creating a specific URL.

```
static getAuthorizationUrl() {
        const state = Buffer.from(Math.round(
Math.random() * Date.now() ).toString()
).toString('hex');
        const scope =
encodeURIComponent('r_liteprofile r_emailaddress
w_member_social');
        const url = `${authorizationURL}?
response_type=code&client_id=${clientId}&redirect_uri
=${encodeURIComponent(redirectURI)}&state=${state}&sc
ope=${scope}`;
        return url;
    }
```

To make it effective, we create a specific route for the redirect.

```
app.get('/auth', (req, res) => {
    res.redirect(API.getAuthorizationUrl());
});
```

When the user authenticates they are redirected to the callback URL that we have specified. The query string of this URL contains the `code` parameter which is essential for moving to the third step, that is, the request for an access token.

In our class we will have this method:

```
static getAccessToken(req) {
        const { code } = req.query;
        const body = {
            grant_type: 'authorization_code',
            code,
            redirect_uri: redirectURI,
            client_id: clientId,
            client_secret: clientSecret
        };
        return new Promise((resolve, reject) => {
            request.post({url: accessTokenURL, form:
body }, (err, response, body) =>
        {
            if(err) {
                reject(err);
            }
            resolve(JSON.parse(body));
        }
        );
        });
    }
```

The callback route will get the access token and save it in the current session.

```
app.get('/callback', async (req, res) => {
    if(!req.query.code) {
        res.redirect('/');
        return;
    }
    try {
        const data = await API.getAccessToken(req);
        if(data.access_token) {
            req.session.token = data.access_token;
            req.session.authorized = true;
        }
        res.redirect('/');
    } catch(err) {
        res.json(err);
    }
});
```

Access token alone is not enough to publish a post on Linkedin. In fact, it is also necessary to obtain the user ID to link the post to a specific profile. In our class we must therefore define a method for finding this ID.

```
static getLinkedinId(req) {
        return new Promise((resolve, reject) => {
            const url =
'https://api.linkedin.com/v2/me';
            const headers = {
                'Authorization': 'Bearer ' +
req.session.token,
                'cache-control': 'no-cache',
                'X-Restli-Protocol-Version': '2.0.0'
            };
```

```
            request.get({ url: url, headers: headers
}, (err, response, body) => {
                if(err) {
                    reject(err);
                }
                resolve(JSON.parse(body).id);
            });
        });
    }
```

The main view will show the login button if the user is not authenticated or the form for creating the post. This form will have as fields the title of the post, the text of the post, the link to be shared and the URL of the image associated with the post. It will have the user ID as a hidden field.

```
app.get('/', async (req, res) => {
    const isAuthorized = (req.session.authorized);
    if(!isAuthorized) {
        res.render('index', { isAuthorized, id: ''
});
    } else {
        try {
            const id = await API.getLinkedinId(req);
            res.render('index', { isAuthorized, id
});
        } catch(err) {
            res.send(err);
        }
    }
});
```

Finally, to publish a post we must combine the access token and the user ID together with the data collected by the form.

```
static publishContent(req, linkedinId, content) {
        const url =
'https://api.linkedin.com/v2/shares';
        const { title, text, shareUrl,
shareThumbnailUrl } = content;
        const body = {
            owner: 'urn:li:person:' + linkedinId,
            subject: title,
            text: {
                text: text
            },
            content: {
                contentEntities: [{
                    entityLocation: shareUrl,
                    thumbnails: [{
                        resolvedUrl:
shareThumbnailUrl
                    }]
                }],
                title: title
            },
            distribution: {
                linkedInDistributionTarget: {}
            }
        };
        const headers = {
            'Authorization': 'Bearer ' +
req.session.token,
            'cache-control': 'no-cache',
            'X-Restli-Protocol-Version': '2.0.0',
            'x-li-format': 'json'
        };

        return new Promise((resolve, reject) => {
```

```
            request.post({ url: url, json: body,
headers: headers}, (err, response, body) => {
                if(err) {
                    reject(err);
                }
                resolve(body);
            });
        });

    }
```

Our route will publish the contents only after the form values pass the validation procedure.

```
app.post('/publish', async (req, res) => {
    const { title, text, url, thumb, id } = req.body;
    const errors = [];

    if(validator.isEmpty(title)) {
        errors.push({ param: 'title', msg: 'Invalid
value.'});
    }
    if(validator.isEmpty(text)) {
        errors.push({ param: 'text', msg: 'Invalid
value.'});
    }
    if(!validator.isURL(url)) {
        errors.push({ param: 'url', msg: 'Invalid
value.'});
    }
    if(!validator.isURL(thumb)) {
        errors.push({ param: 'thumb', msg: 'Invalid
value.'});
```

```
    }

    if(errors.length > 0) {
        res.json({ errors });
    } else {
        const content = {
            title: title,
            text: text,
            shareUrl: url,
            shareThumbnailUrl: thumb
        };

        try {
            const response = await
API.publishContent(req, id, content);
            res.json({ success: 'Post published
successfully.' });
        } catch(err) {
            res.json({ error: 'Unable to publish your
post.' });
        }
    }
});
```

If no errors have occurred, the post will be published on the user's profile.

Source code

GitHub