ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

503106

ADVANCED WEB PROGRAMMING

CHAPTER 10: INTEGRATING WITH THIRD-PARTY APIS

LESSON 10 – INTEGRATING WITH THIRD-PARTY APIS

Social Media Plugins and Site Performance

- Most social media integration is a frontend affair. You reference the appropriate Javascript files in your page, and it enables both incoming content (the top three stories from your Facebook page, for example) and outgoing content (the ability to tweet abut the page you’re on, for example). While this often represents the easiest path to social media integration, it comes at a cost: page load times double or even triple thanks to the additional HTTP requests.
- The code that enables a Facebook “Like” button or a “Tweet” button leverages in-browser cookies to post on the user’s behalf. Moving this functionality to the backend would be difficult (and, in some instances, impossible)

Searching for Tweets

- Let’s say that we want to mention the top 10 most recent tweets that contain the hashtag #meadowlarktravel. We could use a frontend component to do this, but it will involve additional HTTP requests.
- The easiest and most portable way to access the Twitter API is to create an app and use it to get access tokens (the same with Facebook). (Link *http://dev.twitter.com*)
- Once you have an application, you’ll see that you now have a *consumer key* and a *consumer secret*, we can communicate with the Twitter REST API.

Searching for Tweets

- We’ll put our Twitter code in a module called *lib/twitter.js*:

```
var https = require('https');

module.exports = function(twitterOptions){

        return {
                search: function(query, count, cb){
                        // TODO
                }
        };
};
```

- Then we can call search:

```
var twitter = require('./lib/twitter')({
        consumerKey: credentials.twitter.consumerKey,
        consumerSecret: credentials.twitter.consumerSecret,
});

twitter.search('#meadowlarktravel', 10, function(result){
        // tweets will be in result.statuses
});
```

Searching for Tweets

- Before we implement the search method, we must provide some functionality to authenticate ourselves to Twitter. The process is simple: we use HTTPS to request an access token based on our consumer key and consumer secret.

```
var https = require('https');
module.exports = function(twitterOptions){
// this variable will be invisible outside of this module
   var accessToken;
// this function will be invisible outside of this module
   function getAccessToken(cb){
      // TODO: get access token
   }
   return {
     search: function(query, count, cb){
        // TODO
     },
   };
};
```

Get access token

```javascript
function getAccessToken(cb){
    if(accessToken) return cb(accessToken);
    var bearerToken = Buffer( encodeURIComponent(twitterOptions.consumerKey) + ':' +
       encodeURIComponent(twitterOptions.consumerSecret)).toString('base64');
    var options = {
        hostname: 'api.twitter.com',
        port: 443,
        method: 'POST',
        path: '/oauth2/token?grant_type=client_credentials',
        headers: {'Authorization': 'Basic ' + bearerToken },
    };
    https.request(options, function(res){
        var data = '';
        res.on('data', function(chunk){ data += chunk;});
        res.on('end', function(){
            var auth = JSON.parse(data);
            if(auth.token_type!=='bearer'){
                console.log('Twitter auth failed.');
                return; }
            accessToken = auth.access_token;
            cb(accessToken);
        });
    }).end();
}
```

Searching

```
search: function(query, count, cb){
    getAccessToken(function(accessToken){
        var options = {
            hostname: 'api.twitter.com', port: 443, method: 'GET',
            path: '/1.1/search/tweets.json?q=' + encodeURIComponent(query) + '&
            count=' + (count || 10),
            headers: {'Authorization': 'Bearer ' + accessToken,},
        };
        https.request(options, function(res){
            var data = '';
            res.on('data', function(chunk){ data += chunk;});
            res.on('end', function(){ cb(JSON.parse(data)); });
        }).end();
});  },
```

Rendering Tweets

- We’ll use the REST API to search for the tweets and the Twitter widget library to display them. Since we don’t want to run up against usage limits (or slow down our server), we’ll cache the tweets and the HTML to display them for 15 minutes.
- We’ll start by modifying our Twitter library to include a method embed, which gets the HTML to display a tweet (make sure you have var querystring = require('query string'); at the top of the file):

Rendering Tweets

- Now we're ready to search for, and cache, tweets. In our main app file, let's create an object to store the cache:

```
var topTweets = { count: 10,
    lastRefreshed: 0,
    refreshInterval: 15 * 60 * 1000,
    tweets: [],
}
```

```
embed: function(statusId, options, cb){
    if(typeof options==='function') {
        cb = options;
        options = {};
    }
    options.id = statusId;
    getAccessToken(function(accessToken){
        var requestOptions = {
            hostname: 'api.twitter.com',
            port: 443,
            method: 'GET',
            path: '/1.1/statuses/oembed.json?' +
                querystring.stringify(options);
            headers: {
                'Authorization': 'Bearer ' + accessToken,
            },
        };
        https.request(requestOptions, function(res){
            var data = '';
            res.on('data', function(chunk){
                data += chunk;
            });
            res.on('end', function(){
                cb(JSON.parse(data));
            });
        }).end();
    });
},
```

Caching Tweets

- Next we'll create a function to get the top tweets. If they're already cached, and the cache hasn't expired, we simply return topTweets.tweets. Otherwise, we perform a search and then make repeated calls to embed to get the embeddable HTML.
- We'll use the Q promises library, so make sure you run npm install --save q and put var Q = require(*q*); at the top of your app file.

Caching Tweets

```
function getTopTweets(cb){
        if(Date.now() < topTweets.lastRefreshed + topTweets.refreshInterval)
                return cb(topTweets.tweets);

        twitter.search('#meadowlarktravel', topTweets.count, function(result){
                var formattedTweets = [];
                var promises = [];
                var embedOpts = { omit_script: 1 };
                result.statuses.forEach(function(status){
                        var deferred = Q.defer();
                        twitter.embed(status.id_str, embedOpts, function(embed){
                                formattedTweets.push(embed.html);
                                deferred.resolve();
                        });
                        promises.push(deferred.promise);
                });
                Q.all(promises).then(function(){
                        topTweets.lastRefreshed = Date.now();
                        cb(topTweets.tweets = formattedTweets);
                });
        });
}
```

Geocoding

- Geocoding refers to the process of taking a street address or place name (Bletchley Park, Sherwood Drive, Bletchley, Milton Keynes MK3 6EB, UK) and converting it to geo- graphic coordinates (latitude 51.9976597, longitude –0.7406863).
- Both Google and Bing offer excellent REST services for Geocoding. We’ll be using Google for our example, but the Bing service is very similar.

Geocoding with Google

- First, let's create a module *lib/geocode.js*:
- See https://developers.google.com/maps/documentation/geocoding

```
var http = require('http');
module.exports = function(query, cb){
    var options = {
        hostname: 'maps.googleapis.com',
        path: '/maps/api/geocode/json?address=' +
        encodeURIComponent(query) + '&sensor=false',
    };
    http.request(options, function(res){
        var data = '';
        res.on('data', function(chunk){ data += chunk; });
        res.on('end', function(){
            data = JSON.parse(data);
            if(data.results.length){
                cb(null, data.results[0].geometry.location);
            }else{  cb("No results found.", null);   }
        });
    }).end();
}
```

Geocoding with Google

- Both Google and Bing have usage limits for their geocoding API to prevent abuse, but they’re very high. At the time of writing, Google’s limit is 2,500 requests per 24-hour period.

Weather Data

- You’ll need to create a free account, which you can do at *http://www.wunderground.com/ weather/api/*. Once you have your account set up, you’ll create an API key (once you get an API key, put it in your *credentials.js* file as WeatherUnderground.ApiKey). Use of the free API is subject to usage restrictions (as I write this, you are allowed no more than 500 requests per day, and no more than 10 per minute).

Weather Data

- *initialize weather cache*

getWeatherData();

```
var getWeatherData = (function(){ // our weather cache
  var c={ refreshed: 0,
    refreshing: false, updateFrequency: 360000, // 1 hour
    locations: [ { name: 'Portland' }, { name: 'Bend' },
      { name: 'Manzanita' },]
  };
  return function() {
    if( !c.refreshing && Date.now() > c.refreshed + c.updateFrequency ){
      c.refreshing = true;
      var promises = [];
      c.locations.forEach(function(loc){
        var deferred = Q.defer();
        var url = 'http://api.wunderground.com/api/' +
            credentials.WeatherUnderground.ApiKey +
          '/conditions/q/OR/' + loc.name + '.json'
        http.get(url, function(res){
          var body = '';
          res.on('data', function(chunk){ body += chunk; });
          res.on('end', function(){
            body = JSON.parse(body);
            loc.forecastUrl = body.current_observation.forecast_url;
            loc.iconUrl = body.current_observation.icon_url;
            loc.weather = body.current_observation.weather;
            loc.temp = body.current_observation.temperature_string;
            deferred.resolve();
          });
        });
        promises.push(deferred);
      });
      Q.all(promises).then(function(){
        c.refreshing = false; c.refreshed = Date.now(); }); }
    return { locations: c.locations };  }
})();
```