Building a Twitter App Using AngularJS

Preetish Panda(http://www.sitepoint.com/author/ppanda/)

May 27, 2015

In this tutorial we will build a simple Twitter search app from scratch using AngularJS (https://angularjs.org/). The application will let us log in via Twitter and fetch tweets from the user timeline. Besides, the application will feature a search box to let the user searching among the tweets shown. In this tutorial, you will learn about the usage of OAuth for the Twitter authentication, various functions to retrieve tweets, load more tweets through the use of a “Load more” button, and search tweets with the help of AngularJS two-way data binding.

Application Flow

Once the app is loaded, the UI shows a “Connect Twitter” button to the users. After clicking on it, a popup is opened and the user is shown a Twitter login screen to authorize the application to access his/her timeline.

OAuth is used to provide secure authorized access to the Twitter API. OAuth doesn't require users to provide their password to third-party applications and this increases the account security. After logging in, twenty tweets from the user timeline will be loaded but using the "Load more" button, the user is able to load more tweets. The "Sign out" button will let the user to log out.

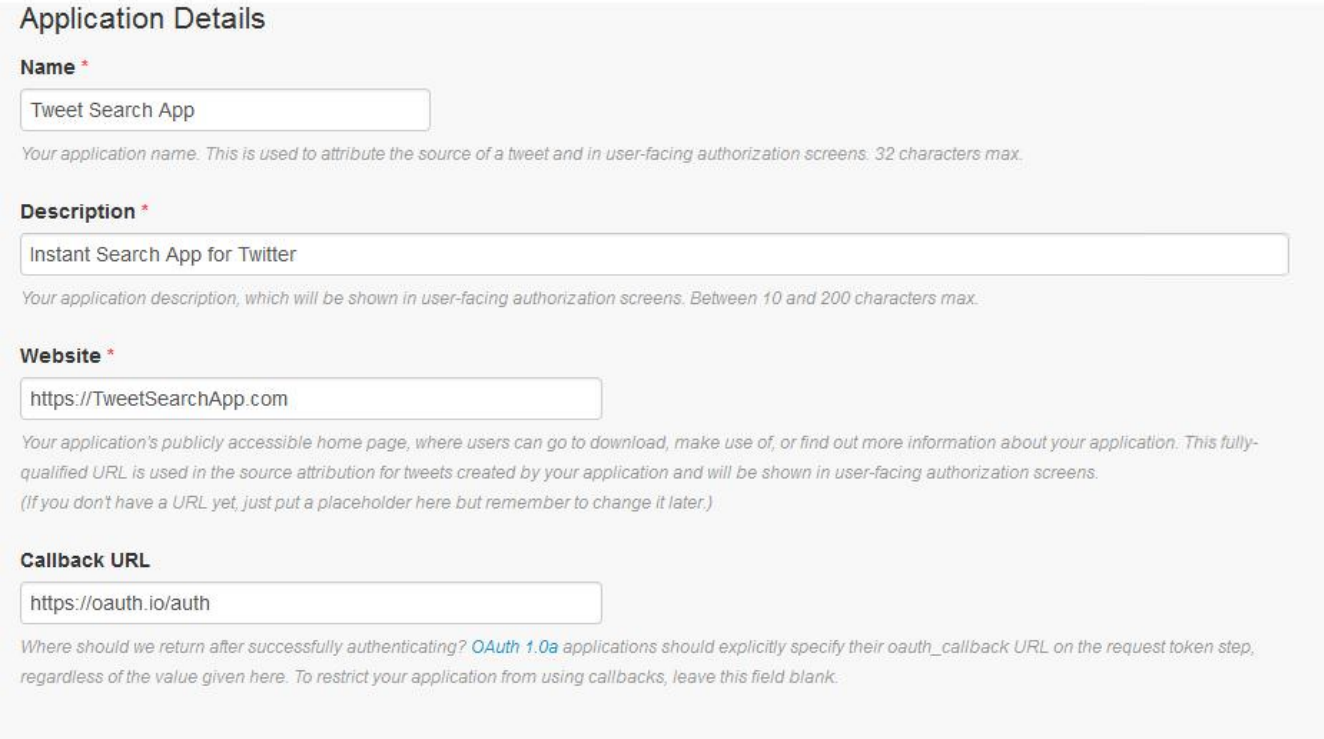
File structure

Now, let's discuss the file structure. All of the files will be in same directory and are set up as follows:

`app.js` : This file will be used to define the main module and the internal services module of the application.
`controllers.js` : This file will handle the various user interactions.
`index.html` : In this file we'll load the main module and provide the user interface.
`oauth.js` : This is the JavaScript SDK provided by OAuth.io (https://oauth.io/) to handle the OAuth authentication.
`services.js` : It'll be used to handle the communication with OAuth.io (https://oauth.io/).

Registering the app

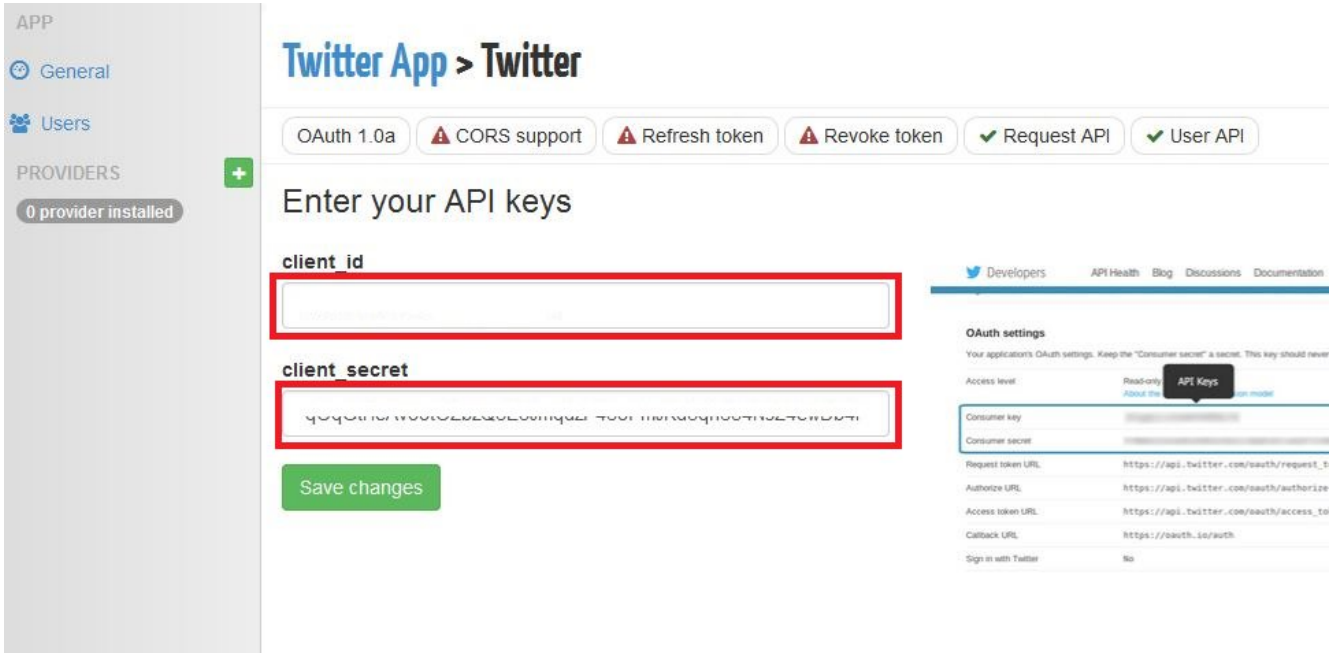
As I mentioned in the previous section, to simplify the Twitter OAuth integration, we'll use OAuth.io (https://oauth.io). Before that we must head to https://apps.twitter.com/app/new (https://apps.twitter.com/app/new) and create a new application.



We also need to be sure that https://oauth.io/auth (https://oauth.io/auth) is used as the callback URL. Once you create the new application, you have to take note of the consumer key and the consumer secret. In the settings page tick the “Allow this application to be used to Sign in with Twitter” field. Finally, you should follow the steps given below to fully setup the application with OAuth.io (https://oauth.io).

We need to create a new account at OAuth.io (https://oauth.io) and create a new app in dashboard. Then, take note of the public key provided.

Now, click on the “Provider” button and select “Twitter”. Insert the consumer key and consumer secret obtained from the step above.



Click on “Try Auth” to test the application.

Creating the Main Module

Let’s now create a main module for our app which will be referred by the `ng-app` directive in the `index.html` file. We’ll call it `twitterApp`. Note that our main module also depends on two different modules: the `ngSanitize` module, used to handle the sanitization of the tweets, and `twitterApp.services`, used to encapsulate the internal services of our app. The following snippet demonstrates how to create the module.

```
1 // twitterApp is dependent on ngSanitize and myApp.services modul
2 var app = angular.module('twitterApp', ['ngSanitize','twitterApp.
```

Communicating with OAuth.io

As mentioned earlier, OAuth is an open standard for authorization that is used for providing secured access to server resources on behalf of a resource owner. The app must perform the following steps in order to use OAuth:

Gain access token to act on behalf of user account

Authorize the HTTP requests to send to Twitter's API

In `services.js` we'll create a module called `twitterApp.services` and add a factory called `twitterService`. This factory will be used to handle the communication with OAuth.io (https://oauth.io). Finally, Angular's `$q` service will be injected into the factory to deal with asynchronous requests.

The public key of the app, obtained from OAuth.io (https://oauth.io), can be replaced here. The following snippet shows various functions present in our factory.

```
angular.module('twitterApp.services', []).factory('twitterServ

    var authorizationResult = false;

    return {
        initialize: function() {
            //initialize OAuth.io with public key of the appli
            OAuth.initialize('19gVB-kbrzsJWQs5o7Ha2LIeX4I', {
                cache: true
            });
            //try to create an authorization result when the p
            // this means a returning user won't have to click
            authorizationResult = OAuth.create("twitter");
        },
        isReady: function() {
            return (authorizationResult);
        },
        connectTwitter: function() {
            var deferred = $q.defer();
            OAuth.popup("twitter", {
                cache: true
            }, function(error, result) {
                // cache means to execute the callback if the
                if (!error) {
                    authorizationResult = result;
                    deferred.resolve();
                } else {
                    //do something if there's an error

                }
            });
            return deferred.promise;
        },
        clearCache: function() {
            OAuth.clearCache('twitter');
            authorizationResult = false;
        },
        getLatestTweets: function(maxId) {
            //create a deferred object using Angular's $q serv
            var deferred = $q.defer();
            var url = '/1.1/statuses/home_timeline.json';
            if (maxId) {
```

```
                url += '?max_id=' + maxId;
            }
            var promise = authorizationResult.get(url).done(fu
                // https://dev.twitter.com/docs/api/1.1/get/st
                // when the data is retrieved resolve the defe
                deferred.resolve(data);
            }).fail(function(err) {
                deferred.reject(err);
            });
            //return the promise of the deferred object
            return deferred.promise;
        }
    }
});
```

The most important functions in the above snippet are `connectTwitter()` and `getLatestTweets()`. The former is used to connect a user with Twitter, while the later fetches the latest tweets from the user's timeline.

As you can see, the method `OAuth.popup()` is used to trigger a popup that asks the users to connect with their Twitter account. If the operation is successful we resolve the promise by calling `deferred.resolve()`. In case of error we reject the promise and pass the error object to the resolve function. Similarly, the function `getLatestTweets()` makes a call to the API endpoint `/1.1/statuses/home_timeline.json` to obtain a list of tweets. Once the operation is successful we resolve the promise.

Responding to User Interactions

In `controller.js`, we will create a controller called `TwitterController`. This will call methods from `twitterService` and respond to user interactions. `twitterService` will be injected into the controller along with `$q` and `$scope`.

```
app.controller('TwitterController', function($scope, $q, twitt
    $scope.tweets = []; //array of tweets

    twitterService.initialize();

    //using the OAuth authorization result get the latest 20 t
    $scope.refreshTimeline = function(maxId) {
        twitterService.getLatestTweets(maxId).then(function(da
            $scope.tweets = $scope.tweets.concat(data);
        }, function() {
```

```
            $scope.rateLimitError = true;
        });
    }

    //when the user clicks the connect twitter button, the pop
    $scope.connectButton = function() {
        twitterService.connectTwitter().then(function() {
            if (twitterService.isReady()) {
                //if the authorization is successful, hide the
                $('#connectButton').fadeOut(function() {
                    $('#getTimelineButton, #signOut').fadeIn()
                    $scope.refreshTimeline();
                    $scope.connectedTwitter = true;
                });
            } else {

            }
        });
    }

    //sign out clears the OAuth cache, the user will have to r
    $scope.signOut = function() {
        twitterService.clearCache();
        $scope.tweets.length = 0;
        $('#getTimelineButton, #signOut').fadeOut(function() {
            $('#connectButton').fadeIn();
            $scope.$apply(function() {
                $scope.connectedTwitter = false
            })
        });
    }

    //if the user is a returning user, hide the sign in buttor
    if (twitterService.isReady()) {
        $('#connectButton').hide();
        $('#getTimelineButton, #signOut').show();
        $scope.connectedTwitter = true;
        $scope.refreshTimeline();
    }
});
```

Firstly, we call `twitterService.initialize()` to set up the service properly. The function `$scope.connectButton()` is called when the user clicks on the “Connect Twitter” button. This initiates the authorization process by calling `twitterService.connectTwitter()`. Upon completion we set `$scope.connectedTwitter = true` so that the search box and the “Load more” button will be visible on the UI. We set the model `rateLimitError` on `$scope` to `true` when error is encountered.

When a user clicks on the button “Get My Timeline”, the function `$scope.refreshTimeline()` is called. It retrieves a list of tweets and sets it to the `scope` model `$scope.tweets`. As a result Angular’s two way data binding kicks in and `ng-repeat` automatically refreshes the UI with a list of tweets from the user’s timeline.

Finally, The function `$scope.signOut()` logs user out of the app and cleans up the tweets.

Creating `index.html`

In `index.html`, we’ll load the main module `twitterApp` via the `ng-app` directive. In the header we’ll include OAuth.js (https://github.com/oauth-io/oauth-js/tree/master/dist) from the official GitHub repository of OAuth.io (https://oauth.io). Also note that we will include Bootstrap to quickly create a layout for our app.

```
<!DOCTYPE html>
<html ng-app="twitterApp">
<head>
    <title>AngularJS Instant Tweet Search Application</title>
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/boc
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/boc
    <script src="https://code.jquery.com/jquery-2.1.1.min.js (
    <script src="//netdna.bootstrapcdn.com/bootstrap/3.1.1/js/
    <script src="oauth.js"></script>
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2
    <script src="//ajax.googleapis.com/ajax/libs/angularjs/1.2
    <script src="app.js"></script>
    <script src="controllers.js"></script>
    <script src="services.js"></script>
    <style>
        .container {
            margin-top: 10px;
            margin-bottom: 10px;
        }

        #results .row {
            margin-top: 15px;
            margin-bottom: 15px;
        }
    </style>
</head>
<body>
    <div class="container" ng-controller="TwitterController">
        <h1>AngularJS Instant Tweet Search Application</h1>
        <div class="row">
```

```
            <div class="col-xs-6">
                <button ng-click="connectButton()" id="connect
                <button ng-click="refreshTimeline()" id="getTi
                <button ng-click="signOut()" id="signOut" type
            </div>
            <div class="col-xs-6">
                <input type="text" ng-model="searchTerm" class
            </div>
        </div>
        <div class="row">
            <div class="col-xs-12" id="results">
                <div class="row" ng-repeat="t in tweets | filt

                    <div class="col-xs-2 col-sm-1">
                        <img ng-src="{{t.user.profile_image_ur
                    </div>
                    <div class="col-xs-10 col-sm-11">
                        <small>{{t.user.name}}</small>
                        <br> <span ng-bind-html="t.text"></spa
                    </div>

                </div>

                <div ng-show="rateLimitError">
                    Rate limit reached. You are making too mar
                </div>
                <div>
                    <br/>
                    <input type="button" class="btn btn-info"
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

The `ng-click()` directives are used to handle the click events in order to carry out the tasks. In our case the “Connect Twitter”, “Get My Timeline”, “Sign Out”, and “Load More” buttons use the `ng-click()` directive to run various tasks in `controllers.js`. These tasks are: open the Twitter authorization popup window, fetch tweets from the user’s timeline, clear OAuth cache, and load additional tweets. When a user exceeds the Twitter’s rate limit, we show an error message via `ng-show="rateLimitError"`. Note that we have used `ngBindHtml` to sanitize the content of the tweets.

Let’s now discuss the search box. We’ve used the `ng-model` directive to keep the model `searchTerm` in sync with the input field. When a search term is entered into the text field, it’s automatically synchronized to the model `searchTerm` via the `ng-model` directive. As you may already know, the `ng-repeat` directive is used to

repeat HTML elements. `filter: searchTerm` has been used to filter the elements according to the value present in the model. So, one can just type into the text field and the tweets will automatically refresh based on the search term. In order to display usernames, profile image, and tweets we have used AngularJS expressions by leveraging one-way data binding. Note the expressions `{{t.user.name}}` and `{{t.text}}` to get the username and the tweet posted by the user.

When the "Load more" button is clicked, `maxId` is passed to the `refreshTimeline()` function. In our case the `maxID` is nothing but the highest tweet ID fetched by our app. Doing so, the `refreshTimeline()` function is able to render the next set of tweets using the data binding defined in the HTML template. Finally, it's worth noting that the `ng-repeat` directive is used to loop through the tweets and append them to the previous set of tweets.

Conclusion

In this tutorial we implemented the initial steps to create a Twitter application and authenticate users into our system in order to fetch their timeline and search instantly using a given search term.

The code of this demo is also available on GitHub (https://github.com/sitepoint-editors/tweet-search). Feel free to download it, experiment, and add some more features. Don't forget to post a comment if you have any questions.

Preetish Panda (http://www.sitepoint.com/author/ppanda/)

Preetish is a consumer internet start up enthusiast who loves anything and everything related to the web. He has 4+ years of experience in programming with Java, JavaScript and 1 year experience as a Business Analyst. While not working on web technologies, digital marketing, analytics, he can be found riding

his super fast bike and listening to music (of course not at the same time).

---

8 Comments SitePoint dongguangming

Recommend 3 Share Sort by Best

Join the discussion…

Buccaneer88 • a month ago

Great article, but I think you should inform everyone following this tutorial that you can't run this from a localhost. The url used when setting up the twitter app on apps.twitter.com must be a live url.

• Reply • Share ›

Art Bergquist • 2 months ago

Great article.

One grammatical opportunity: please replace "searching" with "search" in the following sentence:

Besides, the application will feature a search box to let the user searching among the tweets shown.

The corrected sentence reads as follows:

Besides, the application will feature a search box to let the user search among the tweets shown.

• Reply • Share ›

Wade Bekker • 2 months ago

Thank you for the article. I've used your code and the popup initializes but returns "Cannot find hostname in file:/// from static
code: InvalidHeader
message: Cannot find hostname in file:/// from static"

Any idea why this may be? I added the twitter integration on oAuth.io and it works when I click "Try auth" so that part seems to be working fine. Also put it onto a server but gives the same error. Thanks so much.

• Reply • Share ›

phil_good • 2 months ago

Insightful article! Short and concise.

• Reply • Share ›

Marvin Amador • 2 months ago

I get an error "TypeError: undefined is not a function" at var promise = authorizationResult.get(url) in the twitterService factory I'm using algular 1.4 strict mode

• Reply • Share ›

Aditya Pratap Singh • 2 months ago

Downloaded the code from github but it shows a message "Rate limit reached. You are making too many requests."

Any way good and informative article.

• Reply • Share ›

Aurelio De Rosa Aditya Pratap Singh • 2 months ago

Hi.

Unfortunately this is a very well known and hated Twitter limitation, and there's nothing you can do. If you want to learn more about this limitation, read this page from the Twitter API documentation: https://dev.twitter.com/rest/p...

1 • Reply • Share ›

Vo Quoc Dat • 2 months ago

Next Article

How to Integrate Facebook Login into a Cordova based App ➔ (http://www.sitepoint.com/how-to-integrate-facebook-login-into-a-cordova-based-app/)

Suggestions

Add or upvote an item to make SitePoint better.
(http://sitepoint.com/premium/upvote)

Coming Soon

Check out upcoming books and courses.

(http://sitepoint.com/premium/coming-soon)

Newsletters

SitePoint goodness delivered to your inbox.

(/newsletter)

About

Our Story (/about-us)
Advertise (/advertising)
Press Room (/press)
Reference (http://reference.sitepoint.com/css)
Terms of Use (/legals)
Privacy Policy (/legals/#privacy)
FAQ (https://sitepoint.zendesk.com/hc/en-us)
Contact Us (mailto:feedback@sitepoint.com)
Contribute (/write-for-us)

Visit

SitePoint Home (/)
Forums (http://community.sitepoint.com)
Newsletters (/newsletter)
Premium (/premium)
References (/sass-reference)
Store (/store)
Versioning (/versioning)

Connect

(/feed/) (/newsletter/)

(https://www.facebook.com/sitepoint)

(http://twitter.com/sitepointdotcom)

(https://plus.google.com/+sitepoint)