CrackerJack

You vote, we Post!

Frida Kiriakos
John Movius
Annette Ruiz
Eduardo Rojas
Denice Ron Valbuena

1. Project Description

In this project, we have implemented a web service that allows users to post to Twitter as a group. This service can be used by clubs or organizations to better coordinate and organize their social activity on Twitter. A member of the group can post what he/she wants to tweet on our service, then the other members can vote on it to decide if they want to publish it publicly on their twitter account.

Our service also allows users to edit their posts before publishing, it also provides the ability to shorten the post to conform with the 140 characters limit imposed by Twitter. Users can also view the published tweets on their account.

2. Features

Our web service was implemented using the following features and technologies:

1) Express Framework and Node.js: We have used Node.js in our backend in addition to Express, the web framework for Node.js. We have used the express generator to create the file structure for the project and to follow the Model View Controller architecture. This made it easier for us to arrange the code and to know where to find each function in the project.

2) MongoDB and Mongoose: Since our service supports adding posts and voting on them, we decided to create a persistent environment to preserve this data when the server is restarted, which is why we used MongoDB as out data store. To simplify storing and retrieving data from MongoDB, we have used Mongoose, a mongoDB object modeling for Node.js. It provided us with a simple method to create object schemas, in addition to

adding references and validation on the data. It also simplified querying the collections

to retrieve the required data.

As an example of this, we have the User and the Post objects which are defined in

user.js and post.js models:

```
var usersSchema = new Schema({
  _id    : Number,
  username: String,
  password: String,
  email: String,
  groupList: Array,
  posts: [{ type: Schema.Types.ObjectId, ref: "Post" }],
  date: { type: Date, default: Date.now },
  submitted: Number,
  published: Number,
  admin: Boolean
});
module.exports = mongoose.model("User", usersSchema);
```

The same for the Post object:
```
var postsSchema = new Schema({
        author: { type: Number, ref: "User" },
        body: String,
        upvotes: Number,
        downvotes: Number,
        upvoters: [{ type: Number, ref: "User"  }],
        downvoters: [{ type: Number, ref: "User"  }],
        published: Boolean
});
module.exports = mongoose.model("Post", postsSchema);
```

Now whenever we need to retrieve data from mongoDB, in the routes for example, we

would require the User and Post models and use the APIs provided by mongoose to

create queries. For example, in the route for the index page, routes/index.js, we retrieve

the list of posts stored in the posts collection to display them in the view. This is performed by first requiring the model:

```
var Post = require("../models/post");
```

Then using the following query:

```
Post
.find()
.populate("author")
.sort({upvotes: -1, downvotes: 1})
.exec(function (err, posts) {
        if (err) {
                console.log(err);
                res.render("index");
        }
        res.render("index", {
                .
                .
                .
                posts: posts
        });
});
```

This query retrieves all the posts, sorts them by the number of votes and it also populates the author object, such that for each post, the author object and all its attributes can be accessed by using post.author.

3) Authentication

4) Twitter APIs and widget

5) Feathr.it API

3. Deployment

To deploy and run our app, you can follow the below steps:

First, clone our repo from github:

$ git clone https://github.com/frida-kiriakos/crackerjack.git

We use mongoDB as our data store, so it needs to be installed and running, this can be done using the following, for Ubuntu (Install MongoDB on Ubuntu, n.d.):

$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10

$ echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release

-sc)"/mongodb-org/3.0 multiverse" | sudo tee

/etc/apt/sources.list.d/mongodb-org-3.0.list

$ sudo apt-get update

$ sudo apt-get install mongodb-org

$ sudo service mongod start

Since we are using express generator, all dependencies are specified in package.json file, and can be installed using the following command:

$ npm install

The Twitter API requires a credentials file to authenticate our application and allow us to publish to the Twitter account, we did not include this file in the repository for security reasons, copy the credentials.sample file into credentials.json. You can use your own authentication secrets and tokens and we will also send a copy of our credentials file with the submission.

Then to run the service:

$ DEBUG=crackerjack_express:* ./bin/www

or if you don't want to run it in debug mode:

$ ./bin/www

4. Walkthrough

Users must be initialized prior to logging in by typing the following link:

> localhost:3000/users/new/denice

In this case, denice is the username being created.

Once username has been created, it will display the following message. Follow the same steps

for the other three users, john, frida, and test. (ex. localhost:3000/users/new/john)
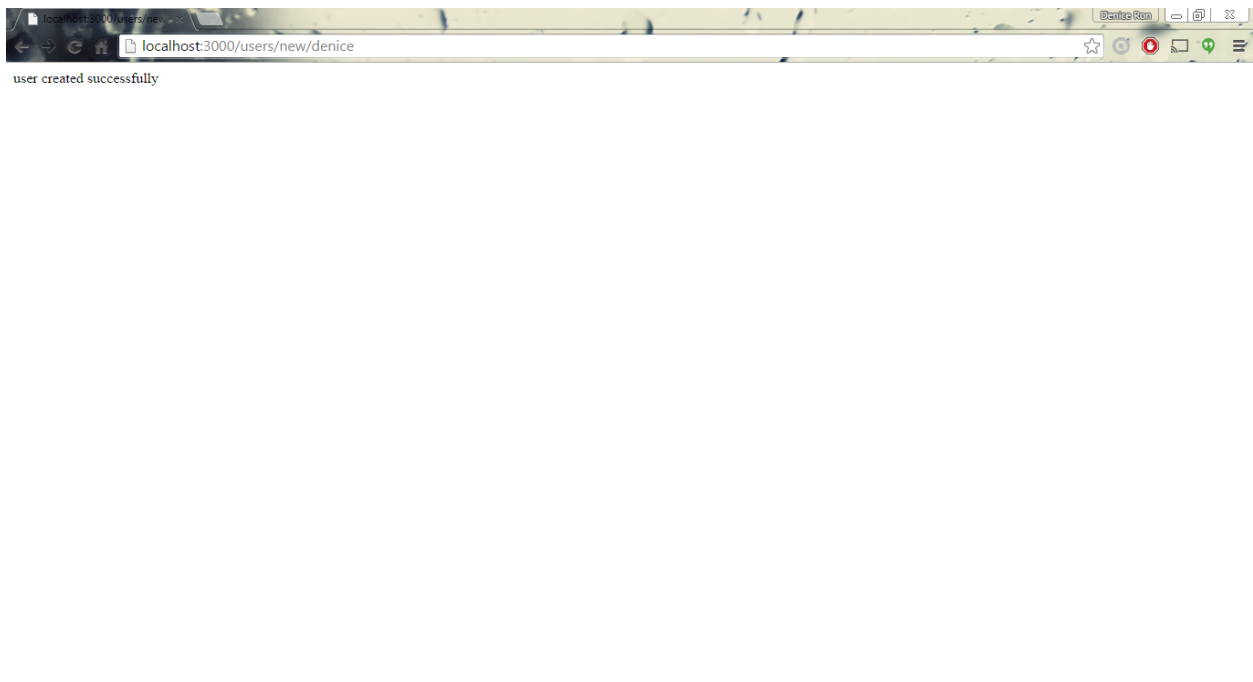
After you have initialized the users, use one of the following logins:
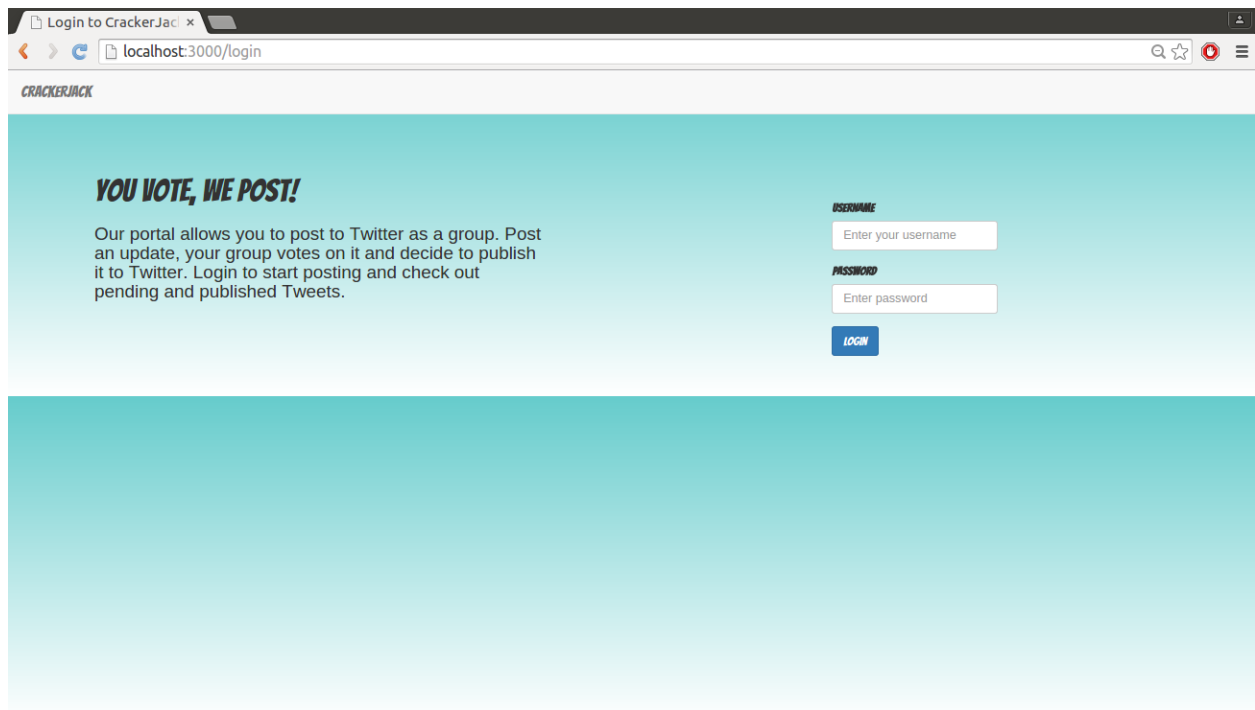
Username: frida (admin)
Password: test123

Username: test
Password: test123

Username: john(admin)
Password: test123

Username: denice(admin)
Password: test123

After logging in, you will be redirected to the main page containing the group's pending tweets and its current Twitter feed. Logging in as the admin will display the additional "Publish to Twitter" feature, which is not seen by normal users. Once a post has passed the necessary amount of votes, the admin must approve it before it gets published to Twitter.
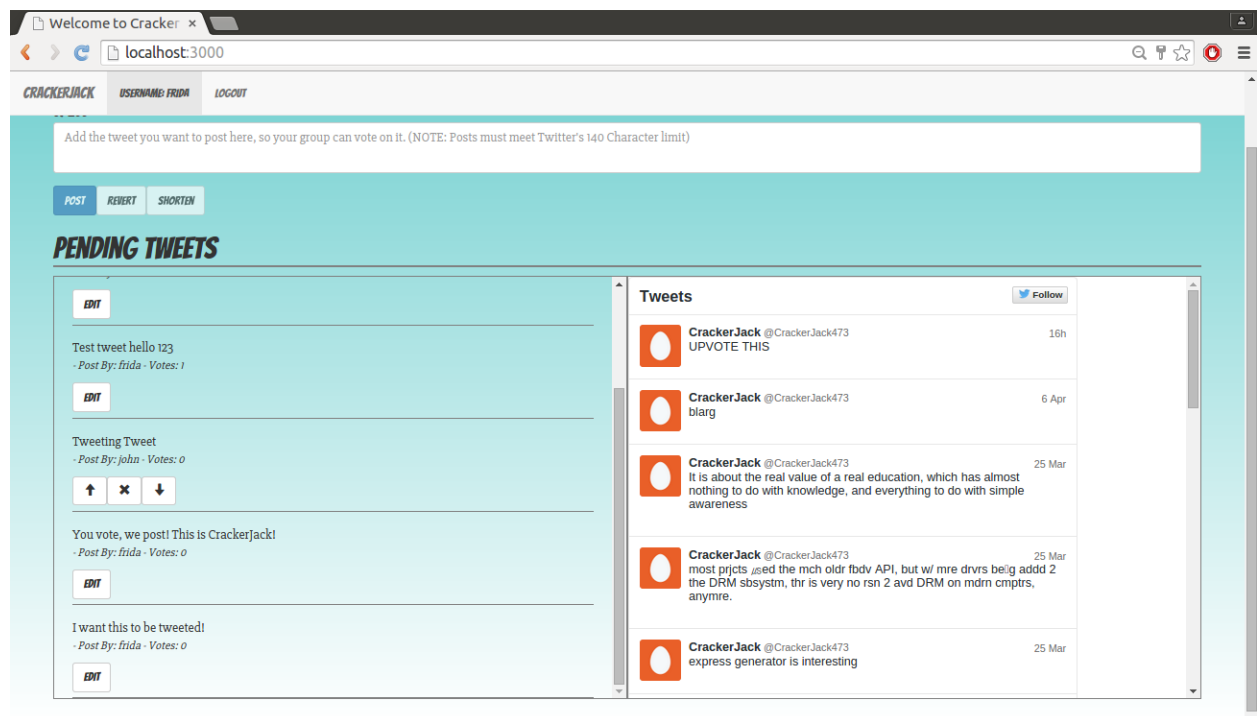
All users are able to vote up or down on other user's posts, but not their own. Users are only able to edit their own posts.
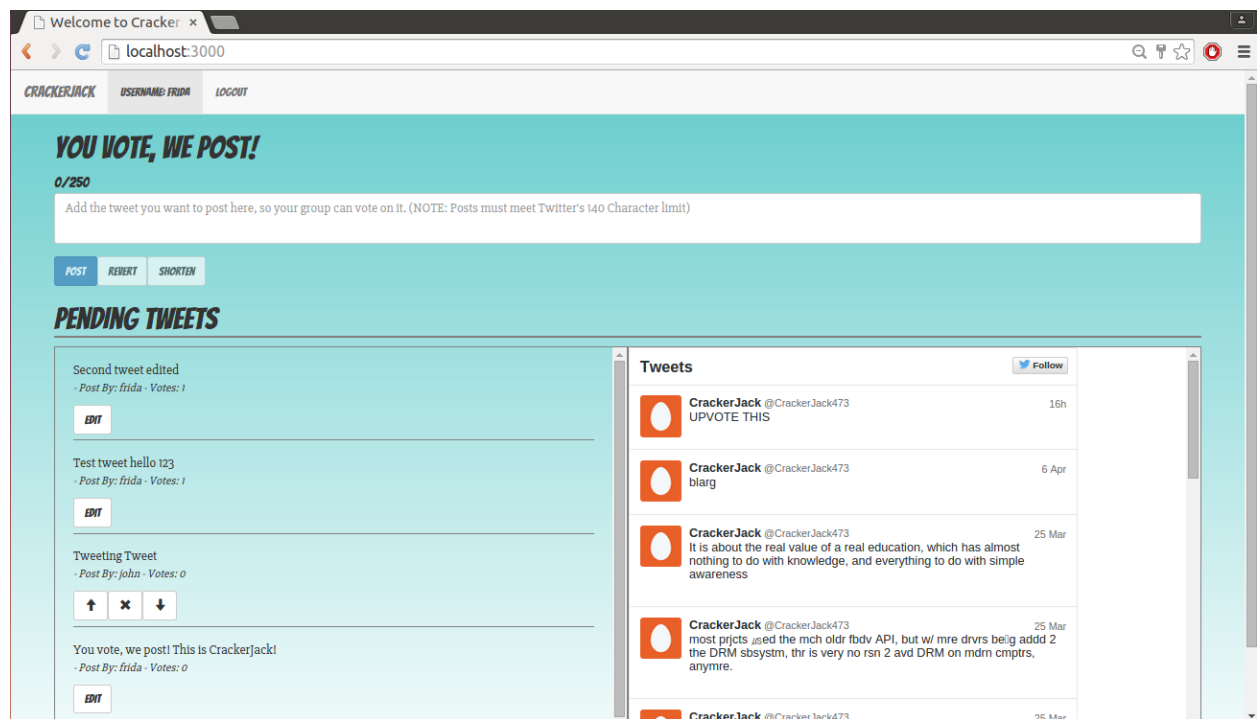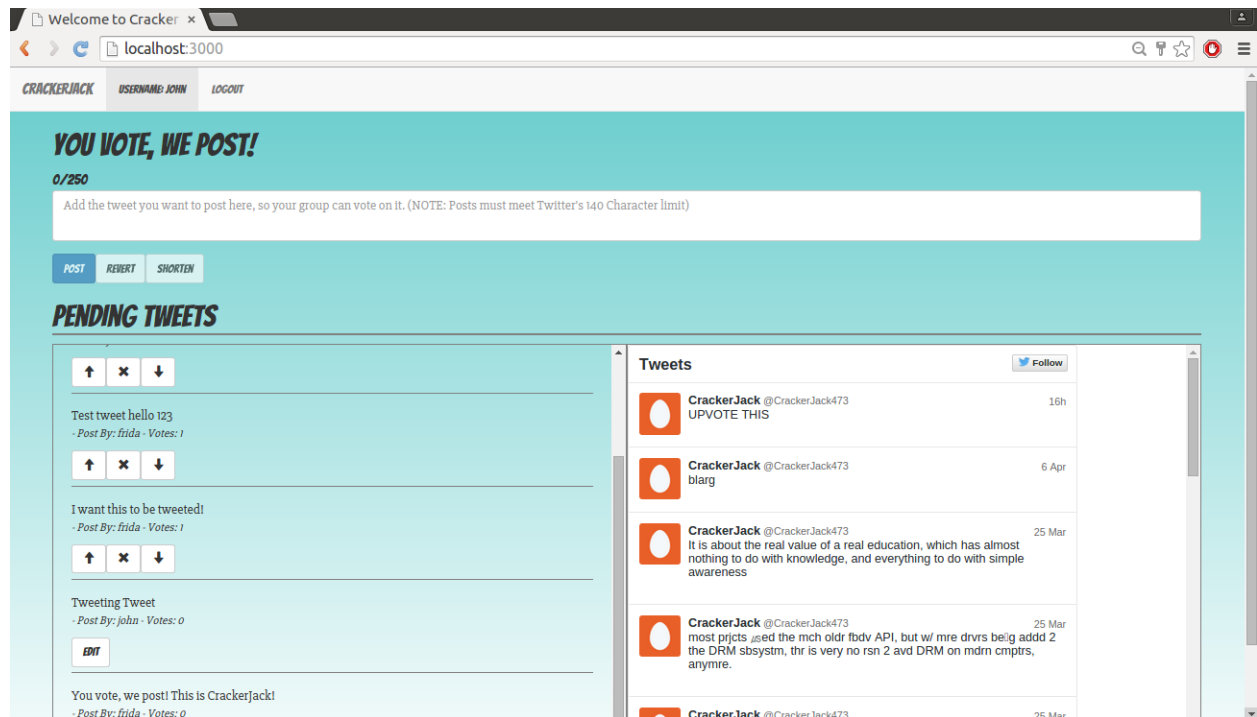
Below is a screenshot prior to posting a new pending tweet.
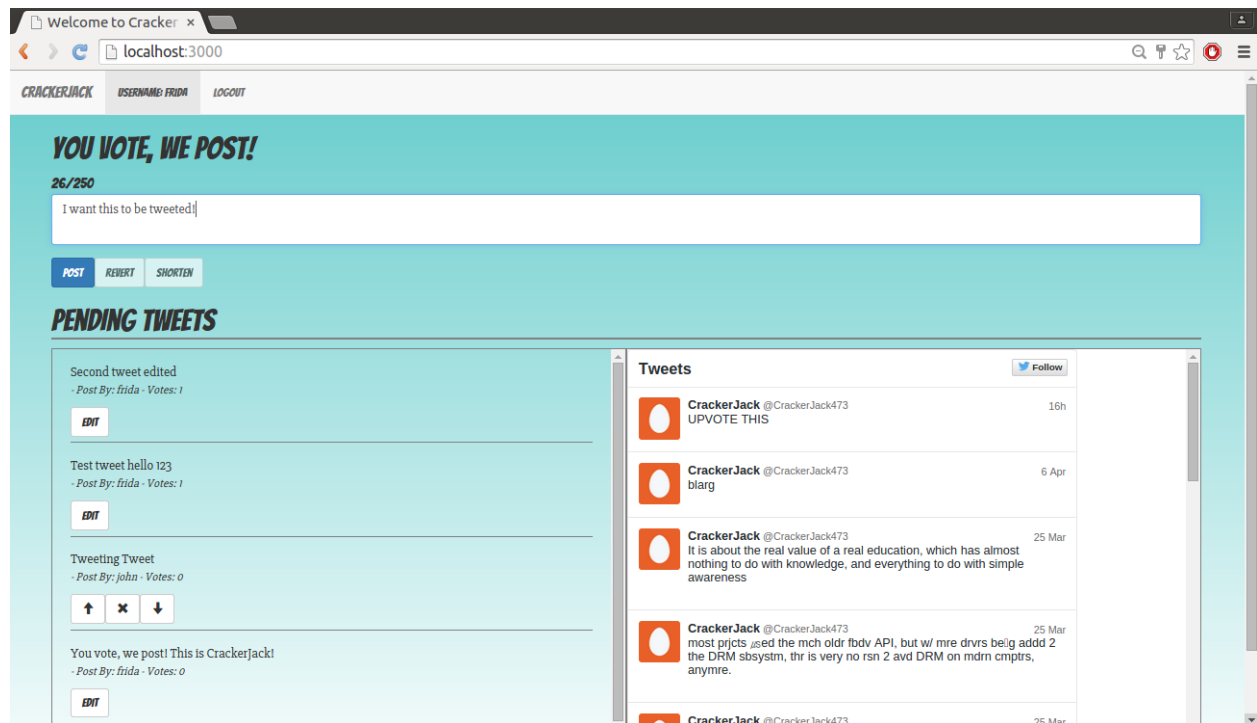
After posting, the new pending tweet is displayed at the bottom of the list.

Logged in as a different user, you can upvote or downvote on other user's posts. Notice the vote count on the new pending tweet that was added in the example above.

5. APIs Documentation

The following documents the APIs provided in the code and their description:

Get /feed, in routes/feed.js:

function: retrieves the tweets for a certain account.

parameters: takes account screen name and needs the twitter credentials for the twitter

API

return value: returns a json object of the tweets body.

restrictions: user must be logged in.

Get /posts, in routes/posts.js:

function: retrieves all posts in the posts collection.

parameters: none

return value: returns a json object that contains an array of posts including the author object for each post.

restrictions: none.

Get /posts/:username, in routes/posts.js:

function: retrieves all posts in the posts collection by a specific author.

parameters: username

return value: returns a json object for the specified author that contains an array of posts by that user.

restrictions: none.

Post /posts/edit, in routes/posts.js

function: edits the body of the post

parameters: post_id which is the id of the post to be edited and post_body the updated post body.

return value: the message "success" on success and "error" otherwise.

restrictions: none so far, can add a restriction on index.ejs to display the edit button only for the user who owns the post.

Get /posts/upvote/:id, in routes/posts.js:

function: increments the upvotes for the posts with the specified id.

parameters: id of the post.

return value: none, redirects to index page on success.

restrictions: user must be logged, user must not be the author of the post, user cannot upvote more than once on a single post

Get /posts/downvote/:id, in routes/posts.js:

function: increments the downvotes for the posts with the specified id.

parameters: id of the post.

return value: none, redirects to index page on success.

restrictions: user must be logged, user must not be the author of the post, user cannot

downvote more than once on a single post.

Post /posts/new, in routes/posts.js:

function: creates a new post and adds it to the database.

parameters: username which is retrieved from the session (req.session.username),

tweetBody submitted from the input form.

return value: none, redirects to index page on success

restrictions: user must be logged in, tweetBody must be less than 140 characters.

Get /publish/:post_id, in routes/publish.js:

function: publishes the post with id equals to post_id to group twitter account

parameters: the id of the post to be published.

return value: redirects to index page on success.

restrictions: the user must be logged in and must have admin privileges, also needs the

credentials for the twitter account to access the twitter API.

The following APIs are necessary to manage users, this can be modified to be more dynamic by

allowing users to register on the website or by creating an admin page. Currently users are

created manually.

Get /users/new/:username, in routes/users.js:

function: creates a user in the users collection for the specific username

parameters: takes a username to retrieve user information from an array hardcoded in the file.

return value: the message "user created successfully" on success.

restrictions: there must be an object in the users hash for the specified username.

Get /users/:username, in routes/users.js:

function: retrieves the specified user from the database.

parameters: username

return value: json object for the user retrieved.

Get /users/delete/:id, in routes/users.json:

function: deletes the specified user from the database.

Get /users/create_group/:group, in routes/users.json:

function: creates a group with the specified name in group parameter

6. Recommendations

Our service currently supports one group to post to twitter, so as a next step, we recommend supporting multiple groups, by associating users with their groups when they sign up on our website. Each user can view the page for the group that he/she selects to post and vote on updates.

7. References

1) Install MongoDB on Ubuntu. (n.d.). Retrieved April 8, 2015, from

http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/