Tools for Supporting Community Growth in Open Source

Bruntmyer J. Author, OSU, Goossens M. Author, OSU, Nguyen H. Author, OSU

Abstract

For 13 weeks our group has been working on a project that is creating tools that gives users the ability to look for open source community leaders that are hosting events about projects that they are leading. These tools will allow users to have the opportunity to find these events in order to become a contributor to an open source project. This is done in a form of a website that will have features for finding certain events dealing with open source projects so that it can be easily accessible by people with a passion for wanting to contribute to projects. Throughout this document, we look at what this team has accomplished for each of the requirements we have laid out, discussing problems that have halted our progression through the project, and what we have left to do before completion of the project. We are currently at a strong alpha level stage for this project as we continue to develop for our beta and version 1.0 stages. Also included are important images of the user interface we have decided to use, along with pieces of code that we have completed and worked with thus far. By the end of this document, you will get a complete picture of how we reached our alpha level prototype and what we have left to finish before we have a version 1.0 ready to be used.

I. PROJECT PURPOSES AND GOALS

After working on the project for about five weeks now, the perspective on what this community development tool wants to accomplish became clear. After taking more time to read and understand the code that was previously written for the prototype, we can see clearly on how exactly the data is organized and connected using Django and its tools. First and foremost, the purpose of the community development tool remains the same. The purpose is to gather information from meetup.com and parse that information into a list of upcoming events related to Apache and open source projects so that developers in the open source community have an easy to access environment where they can hope to participate in those events.

II. CURRENT PROGRESSION THROUGH REQUIREMENTS

A. Fix the People page where the list of community leaders are shown

1. Current Progress: The People page currently takes all of the people in the database and lists them onto the page. Normally, if the prototype is hosted on a local machine and the database is relatively small, then the page loads fine in a minimal amount of time. The issue is nested in the actual hosted site by Apache where hundreds of thousands people are imported into the database daily and dramatically slowing down the loading time of the page. With our current progression of the project, we have not made significant progress into improving the loading time of the page. We use our own local host to import a small amount of members at a time and that requirement is set to be worked on shortly for Beta implementation. The guideline for working towards accomplishing this requirement is to limit the amount of people loaded at a time onto the page. For Beta, we have rearranged where the table is generated for the people page in the function within views.py. This change specifically was introduced because a bug was found where when the table is generated, then if the amount of people were too many, then the table would crash and not build. With the rearrangement, now the table does not break through a large build and now tends to load faster.

2. Things to Complete: Currently, the implementation of the fix for the People page is local. We have yet to test it on the host that Apache is using to run the prototype currently. We predict that the fix will work but it will need to be approved and patched into the live site for clarification.

Fig. 1: The people page

B. Tweet at a person listed in the database

1. Current Progress: Currently, a person is generated their own profile page based off of their Meetups ID and information from meetups about the persons profile is also parsed in the community development tool. Now those profiles include displaying the twitter handle of the person. This was done by changing the Meetup API URL request so that we could get the correct information and then storing that information in our database. This can be seen in code snippet 1 located below with the URL shown along with the 'if' statements to locate the twitter handle. This allows the user to identify a way to get in contact with the person in the profile. Above the twitter handle is a button that has the Twitter symbol which allows the user to click and send a tweet at the person via Hoot-suite. The hoot-suite app is given the twitter handle of the person the user wants to tweet at and the URL of the persons page on our application for reference. The user signs in to compose the tweet and sends it successfully. The code snippet 2 located below shows the HTML encoding of the button used to create the Hootsuite connection and shows the retrieval of the twitter handle form the database with 'person.service'.

2. Things to Complete: The tweeting button we have implemented currently shows up on a persons profile even if they do not have a twitter handle registered with the Meetup website thus having no purpose. To complete this requirement completely we would like to hide that button from the user when a person does not have a twitter handle registered.

```python
        url = "https://api.meetup.com/2/members?offset=0&format=json&group_id=" +
        str(group.meetupID) + "&photo-host=public&page=500&sig_id=148657742&key=" +
        MEETUP_API_KEY
        response = urllib2.urlopen(url)
        result = response.read()

        data = json.loads(result)
        members = data['results']

        for member in members:
            try:
                person = Person.objects.get(meetupID = member['id'])
            except Person.DoesNotExist:
                person = Person()

            try:
                person.meetupID = member['id']
                person.name = member['name']
                #person.service = member['other_services']['twitter']['identifier']
                person.country = member['country']
                if 'other_services' in member.keys():
                    if 'twitter' in member['other_services'].keys():
                        if 'identifier' in member['other_services']['twitter'].keys
():
                            person.service = member['other_services'}
                                                    ['twitter']['identifier']
```

Code Snippet: 1: Views.py file where twitter handle is identified and stored

```html
    <a href="{{ person.service }}" title="{{ person.service }}"
                class="_hs_socialshare" > Tweet at this Person </a>
      <script>
        var _hs = {
                size: 5,
                partner: "community.apache.org"
        };
        (function() {
                var h = document.createElement('script'); h.type =
                'text/javascript'; h.async = true;
                h.src = ('https:' == document.location.protocol ? 'https://' :
                'http://') + 'dtirydke3kdq7.cloudfront.net/hootlet.js?v=1';
                var s = document.getElementsByTagName('script')[0]; s.
 parentNode.insertBefore(h, s);
        })();
      </script>
```

Code Snippet: 2: View.py where the button for tweeting at a person is created and twitter handle is displayed.

Fig. 2: The button that is displayed when a user has a twitter handle that is publicly available on Meetups that allows a user to tweet at the person.
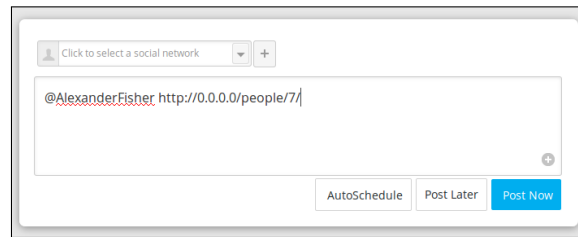

Fig. 3: The HootSuite service is used to send the tweet at the person as we supply the twitter handle and it is up to the user to use their Twitter account to tweet.

C. Add user accounts to application and track when a user has tweeted an event

1. Current Progress: The purpose of adding user accounts to the applications is to be able to track when users tweeting about events or people. Currently, the account creation is working along with being able to sign in successfully with a confirmation of signing in by displaying a welcome message along with the user's username. There is also a login and logout button located in the top right section of the website on all pages allowing the user to login or logout at anytime. We decided to make action functions within the application which include importing meetups, importing members, marking events as not applicable, and marking groups as not applicable to be behind being signed in. This means if you are not logged in you can't perform these actions.

2. Things to Complete: The creating account page does not currently save the first and last name to the database. We will need to make sure to get those stored when accounts are created. We also need to track the events of the user tweeting an event from our application, which is the event trigger of pressing the tweet at button. This will be helpful for warning people of events they have already tweeted about.

```python
def login(request):
    state = "Please log in below..."
    username = password = ''
    if request.POST:
        username = request.POST.get('username')
        password = request.POST.get('password')

        user = authenticate(username=username, password=password)
        if user is not None:
            if user.is_active:
                auth_login(request, user)
                state = "Welcome " + username + "!"
            else:
                state = "Your account is not active, please contact the site admin."
        else:
            state = "Your username and/or password were incorrect."
    template = loader.get_template('login/login.html')
    context = RequestContext(request, {
                            'state': state,
                            'username': username
    })
```

```python
    return HttpResponse(template.render(context))

def logout_view(request):
    logout(request)
    return render(request, 'login/login.html')

def createAccount(request):
    if request.POST:
        username = request.POST.get('username')
        password = request.POST.get('password')
        email = request.POST.get('email')
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')

        user = User.objects.create_user(username, email, password)
    return render(request, 'login/createAccount.html')
```

Code Snippet: 3: Views.py file that handles the creation of accounts and the login page view

```python
url(r'^accounts/login/$', views.login, name='login'),
url(r'^logout', views.logout_view, name='logout'),
url(r'^login/$', views.login, name='login'),
url(r'^createAccount/$', views.createAccount,  name='createAccount'),
url(r'^index/$', views.index, name='eventIndex'),
```

Code Snippet: 4: urls.py file where the destinations are stored for login and logout

Fig. 4: Example usage of the create account form.

Fig. 5: Showing created user actually signing into the website.

D. List tweets about events and/or people via the application

1. Current Progress: We cannot currently track when a user tweets from our application, but we have implemented a round-about way to do so. The website makes a call to Twitter's search API, requesting all tweets from a certain user that contain a certain hashtag. The hashtags are the same as the ones used to get events from meetup.com. Once it has the tweets, it uses the id from each one to make another call to Twitter's OEmbed API, which sends back HTML that is used in the page's template to present embedded tweets to users.

2. Things to Complete: We still need to implement a form of tracking users tweeting from the application so that we can pull only the tweets that they created, instead of potentially unrelated tweets.

E. List tweets about events and/or people not via the application

1. Current Progress: The website now has a tweet parser. Currently it sends a call to Twitter's search API, requesting all tweets with a certain hashtag. The hashtags are the same as the ones used to get events from meetup.com. Once it has the tweets, it uses the id from each one to make another call to Twitter's OEmbed API, which sends back HTML that is used in the page's template to present embedded tweets to users.

2. Things to Complete: The search results currently contain all instances of the hashtag requested, even when they are not relevant to any event, or even open source. The search needs refinement, however, this will be difficult, and may not be ready in time for the 1.0 release
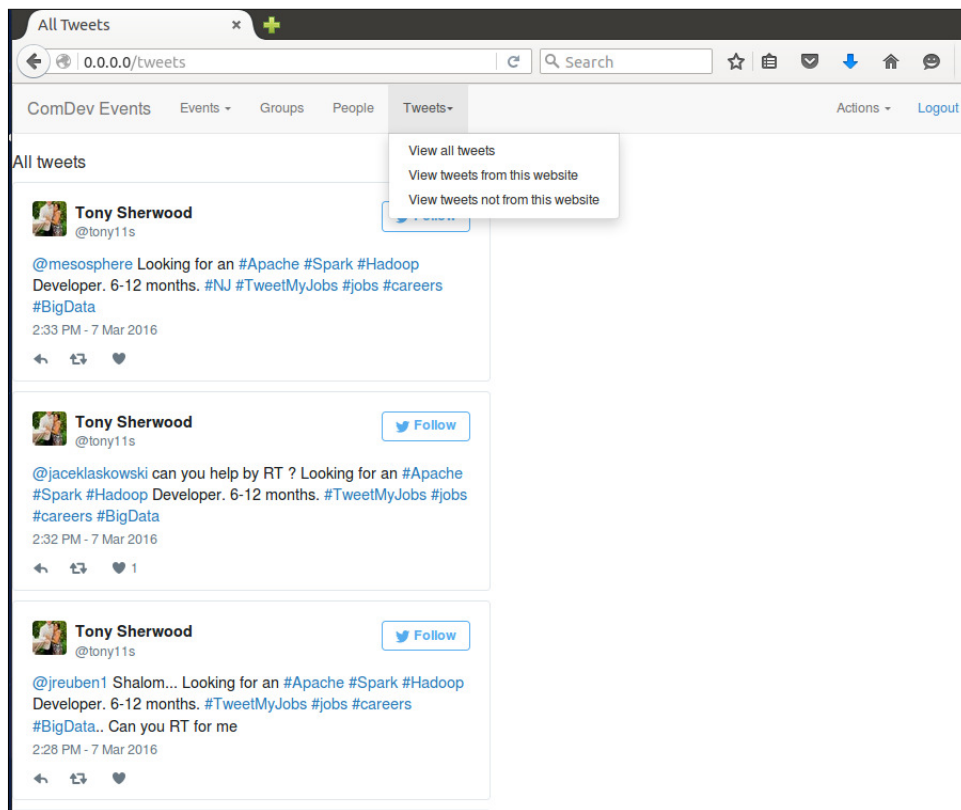

Fig. 6: Showing our application listing the tweets that are located on twitter.

F. Export a list of people with Information

1. Current Progress: When taking on this requirement we quickly realized that the Meetup API would not provide email address for its users which was understandable. We then looked at what other information would be useful to extract about the people that were loaded into the database. This lead us to export information such as name, twitter handle, bio, Meetup ID, URL, country, state, and city. The export can be executed by clicking on the 'Export Info' button located in the top left of the people page and creates a file in a XLSX format which can be directly opened or saved.

2. Things to Complete: This requirement is complete. For the 1.0 version release we were planning on looking into designing a better button for aesthetic purposes but other than that nothing needs to be done.

```
def WriteToExcel(person_list):
    output = StringIO.StringIO()
    workbook = xlsxwriter.Workbook(output)
```

```python
worksheet_s = workbook.add_worksheet("People")

#write title
person_text = ugettext("everyone")
title_text = u"{0} {1}".format(ugettext("Information for"), person_text)
#merge cells
worksheet_s.merge_range('B2:I2', title_text, title)

#write header
worksheet_s.write(4, 0, ugettext("No"), header)
worksheet_s.write(4, 1, ugettext("Name"), header)
worksheet_s.write(4, 2, ugettext("Service"), header)
worksheet_s.write(4, 3, ugettext("Bio"), header)
worksheet_s.write(4, 4, ugettext("Country"), header)
worksheet_s.write(4, 5, ugettext("State"), header)
worksheet_s.write(4, 6, ugettext("City"), header)
worksheet_s.write(4, 7, ugettext("MeetupID"), header)
worksheet_s.write(4, 8, ugettext("URL"), header)

#column widths
bio_col_width = 25

#add data to the table
for idx, data in enumerate(person_list):
    row = 5 + idx
    worksheet_s.write_number(row, 0, idx + 1, cell_center)
    worksheet_s.write_string(row, 1, data.name, cell)
    worksheet_s.write_string(row, 2, data.service, cell)
    worksheet_s.write_string(row, 3, data.bio, cell)
    worksheet_s.write_string(row, 4, data.country, cell)
    worksheet_s.write_string(row, 5, data.state, cell)
    worksheet_s.write_string(row, 6, data.city, cell)
    worksheet_s.write_number(row, 7, data.meetupID, cell)
    worksheet_s.write_string(row, 8, data.url, cell)

workbook.close()
xlsx_data = output.getvalue()            #xlsx_data contains the Excel file
return xlsx_data
```

Code Snippet: 5: excelutils.py file where the worksheet is generated and exported as an xlsx file
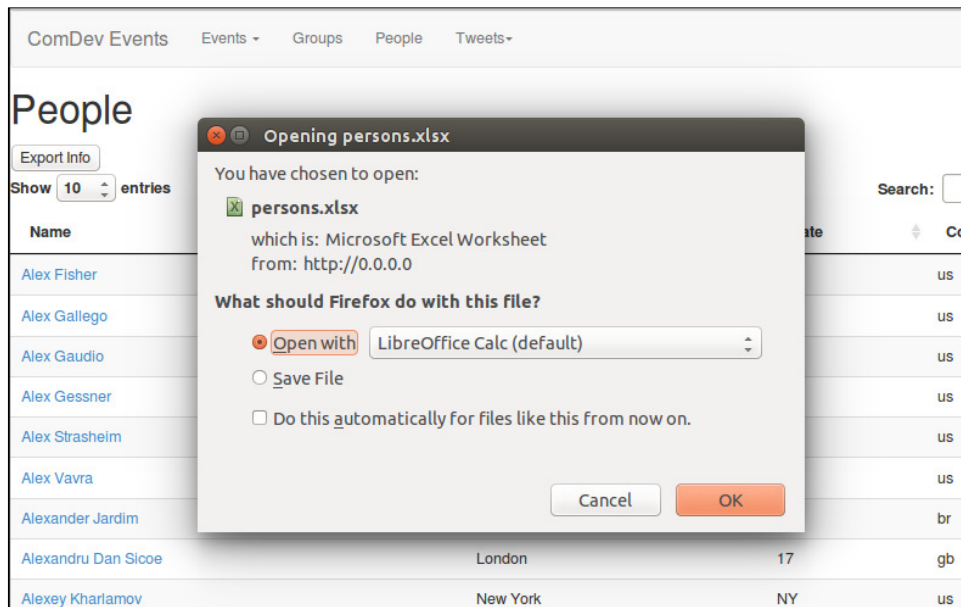
Fig. 7: Showing export button executing.

Fig. 8: Showing the list of exported information about people that were imported.

G. Improve hashtag searching of application to improve finding more relevant events

1. Current Progress: The web application currently successfully parses information from events that are in any way related to Apache and open source development. The issue at hand is that around 50 percent of the events parsed are also completely unrelated meetup events that do not have the same development intention of the application such as mountain bike racing and hiking events. These types of events are also parsed through the application which thus calls for the improvement of the searching algorithm for finding more relevant events. Currently the webpage has the tool to mark events not applicable which would remove the event completely from the list of events. This is a good short term solution for the current day, but it is not an automated process and marking relevant events not applicable is also very possible. In order to improve the available sorting of more relevant events, we need to develop a more viable option still.

2. Things to Complete: The intended solution looks at how the information is gathered. Currently the searching of the application looks for keywords. In this case the keyword is the hashtag of Apache. Any event that has the word Apache in it will thus then show up the in the event list. Oddly enough, there are some events even still that show up on the list which do not even contain the keyword Apache. Our solution is to create filter keywords that remove events that have a keyword that is not related to the intention of the application. We will also create a more broad range of keywords to search for to view events as well. This will hopefully improve the amount of events that are more relevant to open source development.

H. Improve the visuals of the tool's look as a whole

1. Current Progress: The application itself is fairly organized at the moment. The navigation bar implementation really helps the user keep track navigating between each page. When viewing the list of events, the events are listed in chronological order starting with the most recent. There is a search bar available for the user to type in for a certain event that they wish to view. There are other sorting mechanisms to view those events in another sorting order. However, there are some elements missing that could still be improved to help users navigating on the webpage. There are a few improvements that still should still be implemented to overall help the feel of the website.

2. Things to Complete: When viewing a single event, all the relevant information is parsed and organized well for the user to view, but it is in a plain text format. Functionally, this is great, but the page itself can still be improved to make the page look more appealing and easier to view. This can be applied to viewing a specific person as well. There a few pages missing still that need to be created such as a page that contains information about a specific group. This page would be especially useful for users wishing to learn more about a specific group or community to start to get involved in.

Upon improving the visuals of the website we will want to get feedback on the changes we have made to figure out if those changes have appealed to users of the tool. In order to figure this out we have decided to create A user study where we sit the user down with our website and ask them to complete a list of tasks. We will be interacting with the user and helping them if they have any questions and we will be taking notes as well during the experience. Once the set of tasks are attempted or completed we will then have a set of questions that we will ask the user. The list of tasks and questions are shown below:

a) Tasks:

    i) Look through all pages and play with the navigation.
    ii) Create an account and login.
    iii) Import upcoming open source events.
    iv) Import members/hosts of a group.
    v) Browse 2-3 people profiles.
    vi) Browse 3-4 groups.
    vii) Tweet about an event.
    viii) Tweet at a person from the website.
    ix) Look for events based on location.
    x) View tweets that are on twitter on the website.
    xi) Export some information about people in groups.
    xii) Logout.

b) Questions:

    i) On a scale from 1 to 10 how easy was it to navigate this website? (10 being very easy, 1 being not easy at all)
    ii) What was the feature you found most useful?
    iii) What was the feature you found least useful?
    iv) Would you use this website in the future (Please explain why or why not)?
    v) Would you recommend this website to a friend or colleague?
    vi) If you could add one feature to this website, what would that feature be? Why?
    vii) How do you feel about the overall look of the website?
    viii) What would you change visually in the website? Where?

We decided to go with a questionnaire based user study because we feel for functionality it would be great to get feedback on how the user experiences the tools early on. With the scale of how easy it was to navigate the website we will be able to get a sense of how everything flows and if getting the user to the tool they want to use is happening as smoothly as possible. When asking about a feature the user liked most and least we can get a better sense of which tools are actually being utilized in the website to either build on them or decide they are not useful and how we can improve on them. Asking if the user would either recommend this website to a friend or if they will use it themselves in the future gives us great feedback on if the user generally enjoyed the experience with the tool or if the user had a horrible time and would not return to use this tool again. The last question about the user adding a feature for the website gives us great feedback to see if we are missing something crucial that a lot of people would like to use during their experience using our tool.

I. Implement system of improved sorting of finding events by nearby location within radius of the user

1. Current Progress: The current tool has the function to create a list of all events parsed through the application and the ability to search for a specific event by sorting or an individual search. An additional option is added for the user to be able to locate an event based on the location that they input within a certain mile radius of that location. The current implementation is nearly complete as we are using the Google Maps API to display a map and allow the user to enter a location. Once the user has

entered a location the map is populated with markers which indicate event locations. When these markers are hovered over it will display the name of the event. This will allow the user to jump to a location on the map and see if the event's are nearby. The event search page also lists all events along with latitude, longitude, and start time. When no event's are imported the user can still use the map but there will be no markers displayed and the list will display the message 'No events available'.

2. Things to Complete: We will be adding more details to the markers themselves which feature a short summary of the event along with the local start time displayed both will appear when click on. We would also like to have the map be centered on the location of the machine the user is using such as taking advantage of location services just for a nice starting point. As of right now the centered beginning location is on Corvallis, Oregon.

```python
def eventSearch(request):
    #return render(request, 'events/eventSearch.html')
    now = datetime.now()
    upcoming_events_list = Event.objects.filter(is_applicable = True).filter(
local_start__gte=now).order_by('local_start')
    template = loader.get_template('events/eventSearch.html')
    context = RequestContext(request, {
                               'upcoming_events_list': upcoming_events_list,
                               'can_import': _canImport()
    })
    return HttpResponse(template.render(context))

url = "https://api.meetup.com/2/open_events?&sign=true&photo-host=public&state=ky&
city=lexington&country=usa&topic=" + hashtag.name + "&radius=10000&sign=true&key=" +
MEETUP_API_KEY

    try:
        #getting location of the events instead of groups show below - Justin
Bruntmyer
        if 'venue' in meetup.keys():
            if 'city' in meetup['venue'].keys():
                event.city = meetup['venue']['city']
            if 'country' in meetup['venue'].keys():
                event.country = meetup['venue']['country']
            if 'state' in meetup['venue'].keys():
                event.state = meetup['venue']['state']
            if 'address_1' in meetup['venue'].keys():
                event.address_1 = meetup['venue']['address_1']
            if 'lat' in meetup['venue'].keys():
                event.latitude = meetup['venue']['lat']
            if 'lon' in meetup['venue'].keys():
                event.longitude = meetup['venue']['lon']
        #end getting locaiton info - Justin Bruntmyer
```

Code Snippet: 6: Views.py shows the eventSearch funciton that allows the view in the template for the events page to pull information from the database. This also shows the URL used to get the info about events along with how it is stored.

```javascript
function initAutocomplete() {
    var map = new google.maps.Map(document.getElementById('map'), {
        center: {lat: 44.5645659, lng: -123.2620435},
        zoom: 13,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    });

    // Create the search box and link it to the UI element.
    var input = document.getElementById('pac-input');
    var searchBox = new google.maps.places.SearchBox(input);
    map.controls[google.maps.ControlPosition.TOP_LEFT].push(input);

    // Bias the SearchBox results towards current map's viewport.
```

```javascript
map.addListener('bounds_changed', function() {
    searchBox.setBounds(map.getBounds());
    });

var markers = [];
// Listen for the event fired when the user selects a prediction and retrieve
// more details for that place.
searchBox.addListener('places_changed', function() {
    var places = searchBox.getPlaces();

    if (places.length == 0) {
        return;
    }

    //Clear out the old markers.
    markers.forEach(function(marker) {
        marker.setMap(null);
    });
    markers = [];

    {% for even in upcoming_events_list %}
        var myLatLng = {lat: {{even.latitude}}, lng: {{even.longitude}}};
        var marker = new google.maps.Marker({
            position: myLatLng,
            map: map,
            title: '{{ even.name }}'
        });
    {% endfor %}
    // For each place, get the icon, name and location.
    var bounds = new google.maps.LatLngBounds();
    places.forEach(function(place) {
        var icon = {
            url: place.icon,
            size: new google.maps.Size(71, 71),
            origin: new google.maps.Point(0, 0),
            anchor: new google.maps.Point(17, 34),
            scaledSize: new google.maps.Size(25, 25)
        };

        // Create a marker for each place.
        markers.push(new google.maps.Marker({
            map: map,
            icon: icon,
            title: place.name,
            position: place.geometry.location
        }));

        if (place.geometry.viewport) {
          // Only geocodes have viewport.
            bounds.union(place.geometry.viewport);
        } else {
            bounds.extend(place.geometry.location);
        }
      });
      map.fitBounds(bounds);
    });
}
```

Code Snippet: 7: The eventSearch.html has a mixture of JavaScript and HTML that create the Google Map and search bar along with creating markers for events.
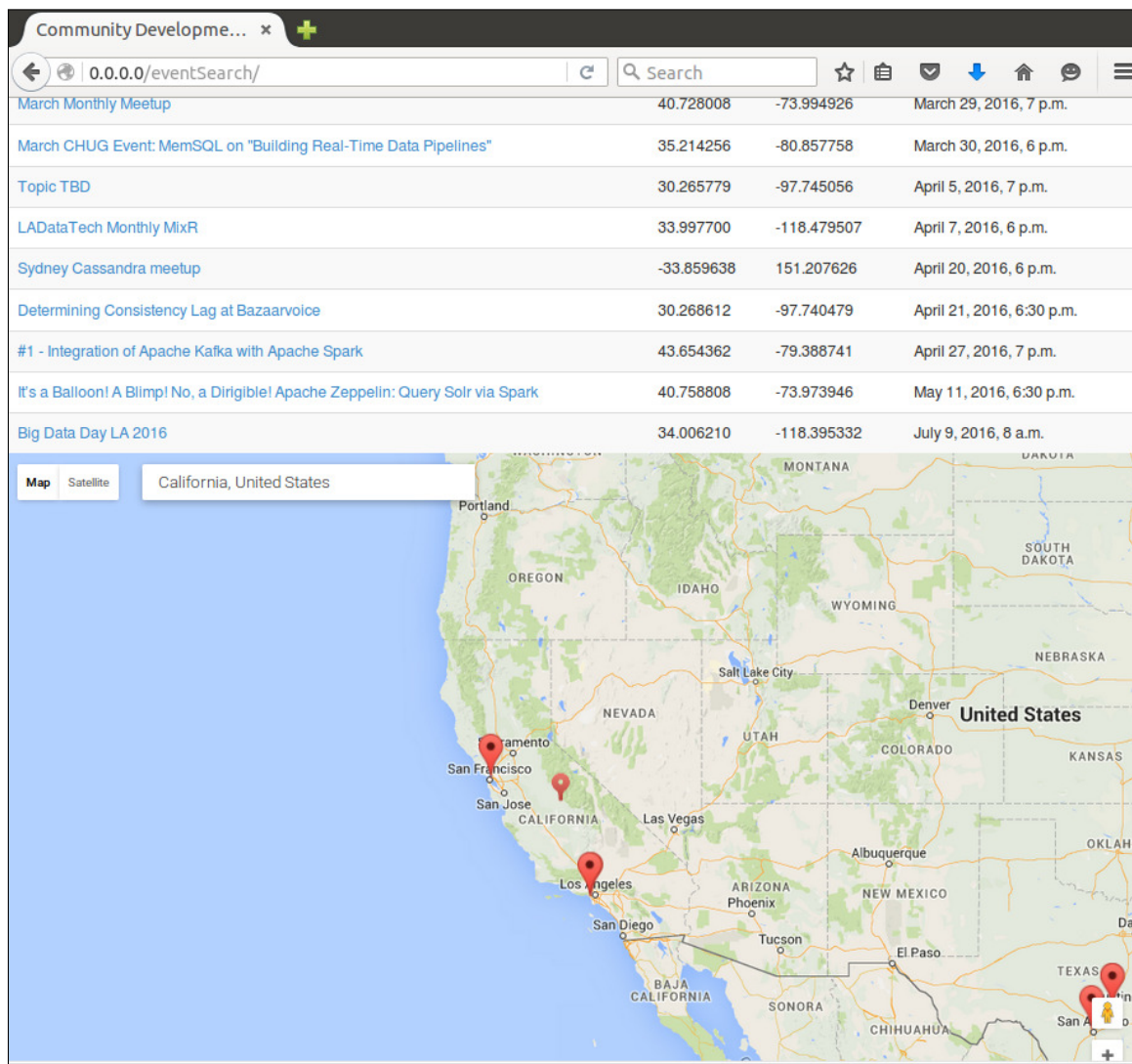
Fig. 9: Showing the map of where you can search a location to see nearby events, also listing events locations.

J. Add feature to generate a profile for people and have it display a way to contact the person if a method is available

1. Current Progress: The main goal of the tool is to promote community development in the open source scene. A very important feature is then to have the best information available for users who would like to start development to be easily viewable. With a generated profiles for people within active groups contributing to Apache projects and project alike that displays a method of contacting said person then it will give people a chance to get involved with the project. Currently, the people generated profiles show the name, a bio, their twitter handle if they have one, location, topics interested in, a picture, source linked to profile on meetup.com, last activity, and the group they are associated with. Since we are not able to pull email address from Meetup we have decided to focus on the service twitter handle as a method of contact which is displayed for the user.

2. Things to Complete: During the implementation of this requirement we came up with the possibility of adding a page just for people who hose events so that users can have a chance to contact organizers directly. This will require pulling the organizer of each event from the Meetup API, storing it, and displaying just as the people page works now. This page will have organizer profiles for users to go to.
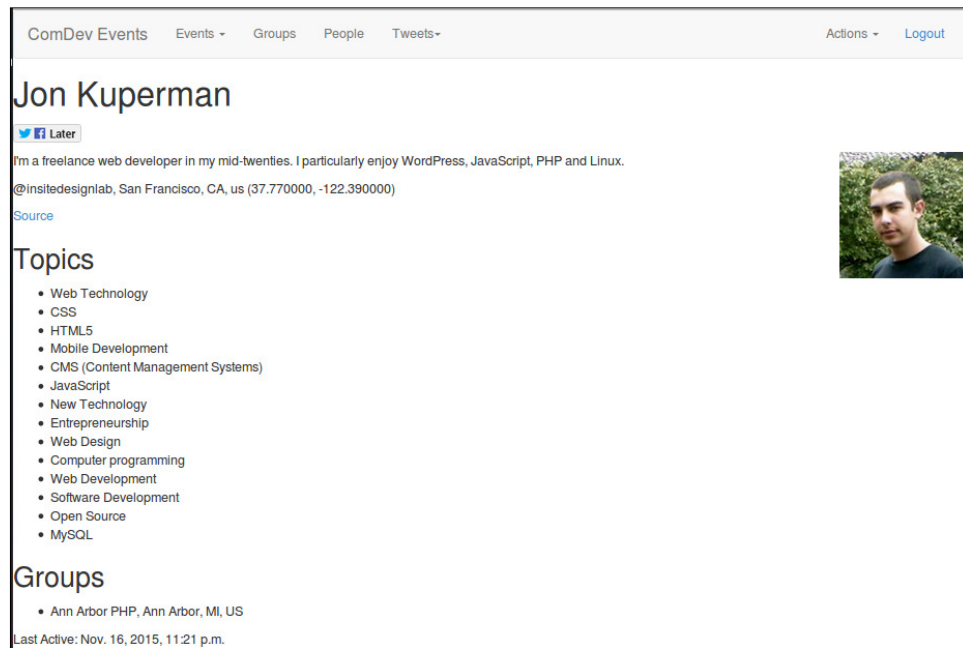
Fig. 10: Showing profile generated for person along with a twitter handle for a method of contacting.

III. FEATURE REQUESTS

A. Implement system of improve sorting of finding events by nearby location within radius of the user

When this tool was presented to our client he was pleased to see what was done and came up with some feature requests he would like to see for this tool. It is important to note that these are not required they are just something our client would like to see in the future. Whether it be from us or by someone else in the future.

1) Have map and list of events and start times be side by side on the page.
2) Calculate distance from where the user has searched to where the location is and display how far away each location is and sort by distance.

IV. PROBLEMS ENCOUNTERED

Throughout this term we have encountered many problems that have impeded our progress for this project. This was to be expected as with all projects events occur that can halt progress and bring forth challenges that, as a group, we needed to overcome. The problems that we have faced are listed below along with the methods we used to get through the situation.

A. Problem 1

Deciding which platform we were going to be developing on using Docker. This was a major issue as we originally planed to work with the Windows operating system however we could not get the application to run locally on a Windows platform. We continually ran into errors with creating a local database along with having the right Docker tools to support the application. We had available resources such as a README file that gave insight on the problem but whatever we tried did not seem to work. We eventually shifted gears and decided to try Docker and the application on Linux, specifically Ubuntu 14.04. Thanks to the Linux knowledge of Megan Goossens and plenty of online documentation we were able to get Docker installed successfully along with running the application on a local host. From this point we decided to continue developing in the Linux environment.

B. Problem 2

At the end of the Fall 2015 term we set up a weekly meeting with our client throughout Winter 2016 to discuss implementation details for the week and planned to utilize this time to make sure everyone is on the same page. Due to some miss-communication we were unable to meet with our client for the first two weeks. This halted our progress with because we had a lot of issues with getting Docker to work with Windows and we were counting on a meeting with our client to resolves those issues as soon as possible. We eventually got in contact with our client and figured out what was happening as the first week our client was on an unexpected trip to the UK and in the second week our client did not realize that these meetings occurred every week throughout the term. These things happen and once we all had a chance to get on the same page every weekly meeting is going smoothly.

C. Problem 3

During week three of Winter 2016 we ran into the issue of meetings being canceled due to illness and injury. One of our team members experience an injury that caused a full group meeting with a TA to occur which halted our progress in having to get everyone caught up on the same page. This same week another group member became sick and could not make it to two meetings for the week which meant that two people could not make it so two meetings were canceled out of the weekly three meetings we have as a group. This was not too untactful as we were able to work individually at home but it was still a noticeable disruption from the normal work we produce in a week. It did not seem like it was going to effect the group at first however when we began to get back on track it took some adjustments to makes sure everyone was on the same page and try to make up for the week we missed.

D. Problem 4

During the first week of the term we began developing based on the requirements we had listed in our requirements document. The problem we ran into here was that we had issues understanding what are requirements were trying to say thus we went through the document and changed the language used for the requirements. This did not change the requirement but it made it easier to understand if someone is reading through it. This took a days worth of progress which was frustrating due to the fact that we believed to have this done last term. We currently are happy with the updated requirements document as it has been approved by our client, professor, and TA. This halted our progress by being an unnecessary step in the implementation process as it should have been completed last term.

V. TIMELINE UPDATE

When we began the development side of the project we realized that it was best to strive for achieving alpha level functionality with all of our features that we would like to have. This required us to take a different approach then what our timeline originally suggested. With that change we were able to have the structure ready for us when we began beta level implementation. With all requirements completed at the beta level we then began to look ahead to see when we want to accomplish the tiny tweaks and bug fixes for the 1.0 release.

We are going to focus our attention on choosing dates to fully version 1.0 complete each requirements. This means that we will move on from the expected completion date for a new data that we will except to have no bugs and tiny tweaks finished. Below shows our future timeline for getting these items completed for version 1.0 release.

REQ#	Requirement	Expected Completion (V 1.0)
1	Fix the "People" page where the list of people are shown from groups	Completed
2	Tweet at a person listed in database	3/29/2016
3	Add user accounts to the application and track when a user has tweeted an event	4/1/2016
4	List tweets about events and/or people via the app	4/12/2016
5	List tweets about events and/or people from twitter, but not via the app	4/14/2016
6	Export a list of people with information	Completed
7	Improve hashtag searching of application for better results on relevant events	4/19/2016
8	Implement a system of finding events nearby a location entered or within radius of the user	4/21/2016
9	Add feature to generate a profile for community developers to have contacting information easily visable	4/28/2016
10	Improve the visuals of the tool and how it looks as a whole.	5/18/2016

VI. CODE SNIPPETS

A. List tweets about events and/or people via the application

1) views.py

```
def _twitterAuth():
    # Encode the keys
```

```python
        key = base64.b64encode("5fqpzXtaoZmwF29KAHc0Grit3:
uDgra72MDCg42CMooGGw1pIlRFdwHr9srjIRjezPZgvZkHMw8G")

        # Set needed values
        authURL = "https://api.twitter.com/oauth2/token"
        content_type = "application/x-www-form-urlencoded;charset=UTF-8"
        body = "grant_type=client_credentials"

        # Create the header
        authHeaders = {'Content-Type': content_type, 'Authorization': "Basic " +
key}
        # Get auth
        auth = requests.post(authURL, headers=authHeaders, data=body)
        # Get the response in a useable format
        authJSON = auth.json()

        return authJSON['access_token']

    def _oembedTweets(tweets):
        oembed = []
        for tweet in tweets:
            url = "https://api.twitter.com/1/statuses/oembed.json?id=" + str(tweet[
'id'])
            embededResponse = requests.get(url)
            embeded = embededResponse.json()
            oembed.append(embeded['html'])

        return oembed

    def tweetsApp(request):
        # Auth with Twitter
        accessToken = _twitterAuth()
        # Get people
        person_list = Person.objects.all()
        hashtags = Hashtag.objects.all().exclude(name = "Meetup")

        oembed = []
        allTweets = []
        for person in person_list:
            for hashtag in hashtags:
                url = "https://api.twitter.com/1.1/search/tweets.json?q=from%3A" +
person.service[1:] + "%23" + hashtag.name + "&src=typd"
                headers = {'Authorization': "Bearer " + accessToken}
                response = requests.get(url, headers=headers)
                tweetsJSON = response.json()
                for tweet in tweetsJSON['statuses']:
                    allTweets.append(tweet)

        oembed = _oembedTweets(allTweets)

        return render(request, 'tweets/app.html', {'tweets': oembed})
```

2) app.html

```html
    {% for tweet in tweets %}
      {% autoescape off %}{{ tweet }}{% endautoescape %}
    {% endfor %}
```

This is the code for viewing tweets from the app. It consists of two parts: the view, and the template. The view consists of three parts: Twitter authorization, getting the tweets, and embedding the tweets. Authorizing with Twitter is merely a matter of constructing a request properly, and sending it to their authorization API. This is contained in the _twitterAuth() method. First, it encodes the keys in the way Twitter requires, then sets several variables that contain other information that Twitter requires. It crafts a header, then uses the Python requests library to send the data to Twitter. The requests library returns unicode, which is then converted to JSON, so that it is easy to get the access token that is returned to the tweetsApp method for future requests.

The next step is in the tweetsApp method. First, it gets a record of all people in the database, then all the hashtags. For every person, it goes through all of the hashtags, crafting a search call to send to Twitter using the requests library. Once the repsponse has be converted to JSON, the tweets are pulled out and added to a seperate dictionary, as a lot of unneeded data is sent by Twitter.

Next, Twitters OEmbed API is called. This is done in the _oembedTweets method. This method goes through the library from the last step, and crafts a url using the tweet id. This is sent to Twitter via the requests library, which returns another dictionary. This is converted to JSON, and the HTML that creates an embeded tweet is added to a new dictionary, which is returned to the tweetsApp method.

The last step is to send that dictionary to the template so that it can be presented. In the template, the dictionary is looped through, presenting each tweets. The key here is the turning off of autoescaping. This allows the HTML from the variable to be displayed properly, allowing the tweets to be embedded nicely in the webpage.

B. List tweets about events and/or people not via the application

1) views.py

```python
def _twitterAuth():
    # Encode the keys
    key = base64.b64encode("5fqpzXtaoZmwF29KAHc0Grit3:
uDgra72MDCg42CMooGGw1pIlRFdwHr9srjIRjezPZgvZkHMw8G")

    # Set needed values
    authURL = "https://api.twitter.com/oauth2/token"
    content_type = "application/x-www-form-urlencoded;charset=UTF-8"
    body = "grant_type=client_credentials"

    # Create the header
    authHeaders = {'Content-Type': content_type, 'Authorization': "Basic " +
key}
    # Get auth
    auth = requests.post(authURL, headers=authHeaders, data=body)
    # Get the response in a useable format
    authJSON = auth.json()

    return authJSON['access_token']

def _oembedTweets(tweets):
    oembed = []
    for tweet in tweets:
        url = "https://api.twitter.com/1/statuses/oembed.json?id=" + str(tweet[
'id'])
        embededResponse = requests.get(url)
        embeded = embededResponse.json()
        oembed.append(embeded['html'])

    return oembed

def viewTweets(request):
    return render(request, 'tweets/view.html')

def tweetsNotApp(request):
    # Auth with twitter
    accessToken = _twitterAuth()

    # Get the hashtags
    hashtags = Hashtag.objects.all().exclude(name = "Meetup")
```

```python
            oembed = []
            allTweets = []
            for hashtag in hashtags:
                url = "https://api.twitter.com/1.1/search/tweets.json?q=%23" + hashtag.
    name + "&src=typd"
                headers = {'Authorization': "Bearer " + accessToken}
                response = requests.get(url, headers=headers)
                tweetsJSON = response.json()
                for tweet in tweetsJSON['statuses']:
                    #tweet['created_at'] = datetime.datetime.strptime(tweet['created_at
    '], "%a %b %d %H:%M:%S +0000 %Y").isoformat()
                    allTweets.append(tweet)

            # Supposed to sort the tweets so they will be in order of posting, doesn't
            # really work. Does at least kinda mix up the tweets so they're not blocks
            # of one hashtag
            #sortedTweets = sorted(allTweets, key=lambda k: k['created_at'])
            sortedTweets = sorted(allTweets, key=itemgetter('id'))

            oembed = _oembedTweets(allTweets)

            return render(request, 'tweets/notApp.html', {'tweets': oembed})
```

2) notApp.html

```
        {% for tweet in tweets %}
          {% autoescape off %}{{ tweet }}{% endautoescape %}
        {% endfor %}
```

This is the code for viewing tweets that aren't from the app. It consists of two parts: the view, and the template. The view consists of three parts: Twitter authorization, getting the tweets, and embedding the tweets. Authorizing with Twitter is merely a matter of constructing a request properly, and sending it to their authorization API. This is contained in the _twitterAuth() method. First, it encodes the keys in the way Twitter requires, then sets several variables that contain other information that Twitter requires. It crafts a header, then uses the Python requests library to send the data to Twitter. The requests library returns unicode, which is then converted to JSON, so that it is easy to get the access token that is returned to the tweetsNotApp method for future requests.

The next step is in the tweetsNotApp method. First, it gets all the hashtags in the database. It goes through all of them, crafting a search call to send to Twitter using the requests library. Once the repsponse has be converted to JSON, the tweets are pulled out and added to a seperate dictionary, as a lot of unneeded data is sent by Twitter. The dictionary is then sorted by id, which is supposed to put the tweets in order. However, neither this nor the attempt to sort by creation date work quite right, and need improvement.

Next, Twitters OEmbed API is called. This is done in the _oembedTweets method. This method goes through the library from the last step, and crafts a url using the tweet id. This is sent to Twitter via the requests library, which returns another dictionary. This is converted to JSON, and the HTML that creates an embeded tweet is added to a new dictionary, which is returned to the tweetsApp method.

The last step is to send that dictionary to the template so that it can be presented. In the template, the dictionary is looped through, presenting each tweets. The key here is the turning off of autoescaping. This allows the HTML from the variable to be displayed properly, allowing the tweets to be embedded nicely in the webpage.