**Overview of Geoq's Twitter Stream Integration:**

Features:

- Added a real-time data feed from Twitter
- Allows users to save and delete tweets
- Allows users to flag tweets as irrelevant, in order to build filters based on hashtags (on a per session basis). This is also referred to as the "hashtag blacklist".

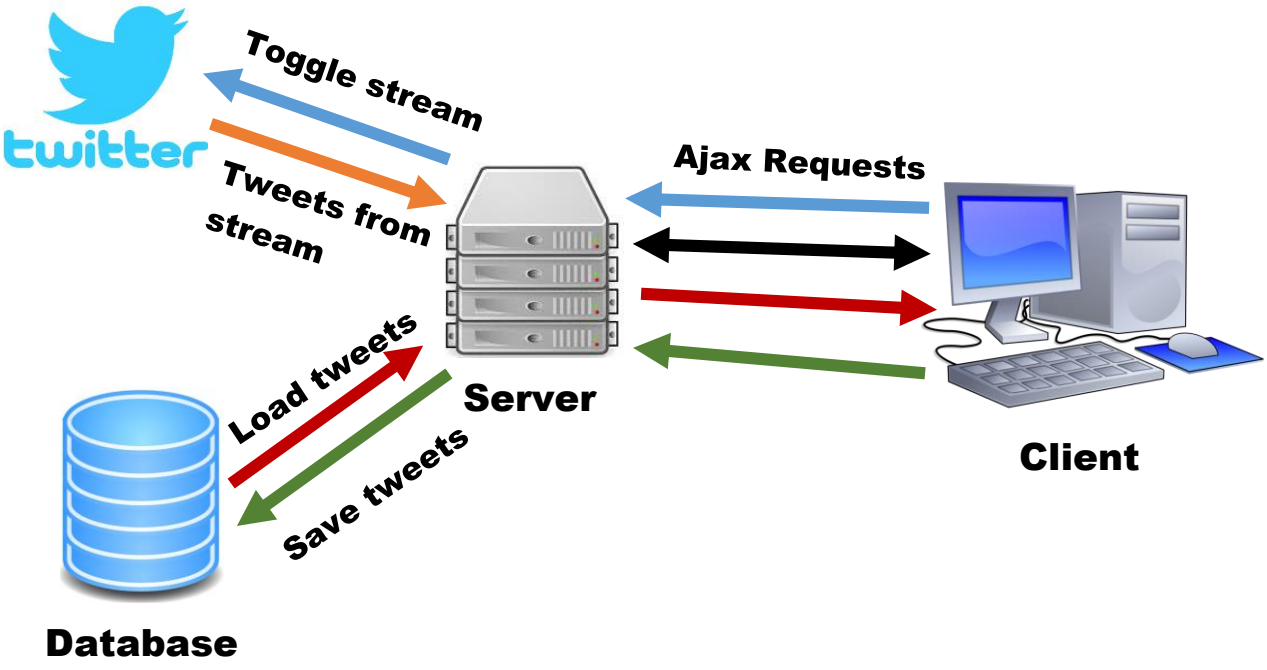
Main Files (relative to manage.py):

- geoq/core/urls.py
    - Added new endpoints to point to streaming functions defined in views.py
- geoq/core/views.py
    - Added 4 new functions designed to be called by AJAX requests
        - twitterfeed – toggles the stream on and off
        - gettweets – called by an ajax request on an interval. Sends blacklisted hashtags to the server and returns all newly streamed tweets to the client (from a temp json file)
        - savetweet – Saves a specific tweet to the database, indexed by workcell id
        - loadtweets – Loads all saved tweets from database that correspond to a specific workcell id
- geoq/twitterstream/*
    - tasks.py – Uses both Celery and Tweepy to asynchronously manage streams and incoming data
    - models.py – Outlines data structure of tweets for when they are saved to the database
- geoq/core/static/core/js/twitter_demo.js
    - Defines all client-side methods for interacting with Twitter and the Twitter-related Geoq endpoints (listed in views.py)
- geoq/core/static/core/js/leafletcontrols/leaflet.layer_tree.js
    - Added a new control layer called Twitter Stream
- geoq/core/templates/core/edit.html
    - Endpoint urls are passed to this page via Django MVC templating. It also defines a few Twitter-related click handlers and CSS schemes
- geoq/celery.py
    - Requirement to use Celery within a Django project.
- geoq/__init__.py
    - Loads Celery when project is launched to more easily use Celery within Django

New dependencies and Django Apps:

- Celery – Python module that provides tasking (multi-threading) to allow the stream returned by Twitter to be non-blocking
- Tweepy – Python module that provides easy methods to connect to the Twitter Streaming API. This module handles the OAUTH handshake and returns all data from the stream
- geoq/twitterstream – A custom Django app that uses both Celery and Tweepy to open/close streams, handles/filters incoming data, and save/load tweets to the geoq database

Data Flow Graph:




Each color represents a specific endpoint outlined earlier in views.py:

- twitterfeed (blue):
  - This AJAX request is sent to the server when the user clicks the "Start/Stop Stream" button. This is designed to open and close streams with Twitter via the Tweepy module. As of now, only one stream can exist at a time on a server. It's also important to note that spamming streams with Twitter will cause their system to block your IP address. Because this endpoint is called via AJAX, it needed to be non-blocking, therefore I utilized Celery to open the stream on a separate thread.

- gettweets (black):
    - This endpoint is called via AJAX once the user begins a stream. An AJAX request is sent on an interval, currently every 15 seconds. This request sends an array of blacklisted tweets, if the user flagged any tweets within the interval. This request returns all tweets collected by the stream (that also "passed" the server-side filters) to the client in json format. This data is then added to the Leaflet map via GeoJson. Note: All tweets collected by the stream are stored temporarily in a json file (stream.json), this methods reads all the data from the json, then empties it to reduce the memory footprint of long-term streaming.

- savetweet (green):
    - This endpoint stored a specific tweet to the database to be loaded later. This method is rather straight forward, except for one nuance. The tweet json object returned from the stream get sent to the client via the gettweets endpoint described above. The tweet data is stored on the client, then sent back to the server to be stored in the database. (Complete path of data, when saving: Twitter -> Server -> Client -> Server -> Database). This could be improved to reduce the amount of data sent between client and server.

- loadtweets (red):
    - This endpoint is also very straightforward and returns all tweets associated with the same workcell-id that the user is viewing. Just like gettweets, this data is published to the Leaflet map via GeoJson. Note: gettweets and loadtweets are on separate GeoJson's (or FeatureGroups) and there have different icons and popup displays.

- Twitter's Streaming API (orange):
    - This just represents data returned to the server from Twitter

How to run it - (for best demo results, open a workcell over a populated area):

- git clone https://github.com/Pkthunder/geoq.git and use the demo branch
- Start it like any other version of Geoq (python manage.py runserver)
- Then navigate to the edit workcell page and click on the Twitter Stream control layer
- Click the Start Stream button to open a stream, but watch the terminal window. If the stream opened correctly, you will see "Tweet received from Twitter…" over and over again. Watch the server and client careful, sometimes they can get out of sync. If this happens, just reset server the ( Ctrl-C + python manage.py runserver) and refresh the browser window (once the server is running again)
- **DO NOT** spam the Start/Stop button. Spamming may cause Twitter to block your IP