http://twtt.tarfi.re

Twitter Poem Generator Documentation

malinow5, ford16, snnnbrg2, tokuhir1, vspatel2, osunero1

Table of Contents

- Overview
- Process
- Requirements & Specifications
  - Iteration 2:
  - Iteration 3:
  - Iteration 4:
  - Iteration 5:
- Architecture And Design
- Future Plans
- Installation and Distribution

Overview

Twitter is one of the most prominent social networks today with a huge amount of data posted to it every minute. Although the data sets are large, there seems to be limited usages for it outside of the actual Twitter website. Our project, Twitter Poem Generator, seeks to make use this data to generate several types of poems based on specific hashtags. This documentation outlines the technical features of Twitter Poem Generator as well as the implementation choices made during the process.

Process

We decided to use Scrum for our Software Development Process. In many ways, Scrum is similar to Extreme Programming . It is also an iterative Agile software development framework, but as opposed to XP's rigid guidelines, it has a focus on "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal" as opposed to a "traditional, sequential approach". The philosophy behind Scrum accepts the idea that requirements will change and unpredicted challenges will arise. Instead of trying to plot out every unknown in a problem and define every corner-case, it instead focuses on maximizing a team's ability to adapt to those changes and deliver quickly.

Scrum defines three core roles in software development , that of the Product Owner, the Development Team, and the Scrum Master. The Product Owner represents those who have invested in the project and acts as the voice of the customer, he ensures that the team meets expectations. Either he or his team writes the user stories for the project and then he prioritizes them and adds them to the product backlog. Scrum teams generally have one Product Owner and while he may also be a member of the Development Team, he should never be the Scrum Master.

The Development Team is responsible for creating shippable code in the given timeframe. It consists of three to nine individuals and, as with Extreme Programming, the team is self-organizing and everyone is on the same level.

The Scrum Master is tasked with removing obstacles that may impede a team's ability to deliver on time. He ensures that the Scrum process is being used properly and enforces the rules. While not a team leader in the traditional sense, he is more of a buffer that makes sure the team is free from distraction and ensures that everyone is performing at peak level.

By allowing the team to self-regulate and have only one member, the Scrum Master, responsible for enforcing proper practices, Scrum is much more flexible than Extreme Programming. It focuses more on higher order team management as opposed to trying to influence development practices. The development team chooses from the backlog what features to work on, as opposed to strictly following priority order. Scrum provides a freedom in development practices that XP simply does not.

Requirements & Specifications

The following section outlines the implementation of the completed user stories included in Twitter Poem Generator. Each section is split up into iterations, signifying the work completed in two week intervals. The user stories begin at iteration 2 as Iteration 1 was reserved for project selection. It should also be noted that iteration 2 was completed using Xp's pair programming. User stories whose descriptions are self explanatory are not expanded on in this section.

Iteration 2:

actual	estimated	Story description	NetIDs
4 units	4 units	As a user, I want to store tweets	ford16, malinow5
4 units	4 units	As a user, I want to type words and determine syllables of words	tokuhir1, osunero1
4 units	4 units	As a user, I want to type words and determine rhyming words	vspatel2, snnnbrg2

I want to store tweets:

Mysql database setup and Twython library selected for managing Twitter's api

I want to type words and determine syllables of words:

CMU dictionary used to break words into phonetic parts, these parts are then counted and a syllable count is determined. Words not appearing in the dictionary are evaluated heuristically.

I want to type words and determine rhyming words:

CMU dictionary also used to break words into phonetic parts. The phonetic endings of two words are then checked for equivalence. If the two are equivalent it can be concluded the two words rhyme.

Iteration 3:

actual	estimated	Story description
3	3 units	As a user, I want to search for poems with similar tweets
5	9 units	As a user, I want to provide a hashtag and Construct haiku
4	6 units	As a user, I want to access a basic skeleton website to view poems

I want to search for poems with similar tweets:

This was accomplished using Twitter's API. This functionality is contained in the method get tweets, which returns a collection of tweets about the same hashtage

I want to provide a hashtag and construct a haiku:

Haikus were constructed using the syllable counts from iteration 2. Tweets about a particular hashtag are pulled until three lines are found matching the 5 7 5 syllabic structure.

I want to access a basic skeleton website to view poems:

The website is a python flask application, it is started by running the main.py python script in the main site directory. The style is from Twitter's bootstrap.

Iteration 4:

actual	estimated	Story description
2	2 units	As a user, I want to access a basic skeleton website to view poems
12	12 units	As a user, I want to access the bot's poems through a simple, easy to use website.
2	2 units	Set up an account with GAE and push skeleton code to it
3	5 units	As a user, I want to choose and receive a couplet poem

I want to access a basic skeleton website to view poems:

This user story continued work on the frontend, while the bare bones remained untouched. the website was modeled reflecting the minimal design pattern.

I want to access the bot's poems through a simple, easy to use website.

This user story implemented all the http requests relevant for generating and viewing a poem. The requests can be found in the main file of the project

Set up an account with GAE and push skeleton code to it

Implementation of this sought to improve the packaging of our website. The repository was modified to support google's paradigms and push to an appspot url.

Iteration 5:

actual	estimated	Story description
2	2 units	As a user, I want to choose and receive a couplet poem
2	2 units	As a user, I want to be able to look at the original tweets of a poem
1	1 units	as a user, I can choose poem subject through a searchbar

1	1 units	as a user, I can choose different poem style through a dropdown
3	3 units	As a user, I want to save poems I generate
3	3 units	As a user, I want to view poems that have been generated
1	1 unit	As a user, I want to select from trending tweets from links
3	3 units	As a user, I want to choose and receive a limerick poem

I want to choose and receive a couplet poem:

Implementation of this involved modeling and generating a couplet poem. Lines about specific hashtags are pulled until two lines are found that rhyme.

I want to be able to look at the original tweets of a poem

This user story added meta data to the poems. Instead of just receiving the raw text, urls were added to allow users to see where the content did in fact come from twitter.

I want to save poems I generate

SqlAlchemy was used here in interfacing the website with the backend database. In this implementation the schema is entirely generated in code to improve portability and scalability.

I want to choose and receive a limerick poem

Implementation of the limerick poem type involved matching lines to the limerick type. Lines are pulled from twitter about the specific hashtag until a set of 3 and 2 rhyming lines are found. These lines are then spliced together in the aabba rhyme scheme.

Iteration 6:

actual	estimated	Story description
4	4 units	As a user, I can click a button and upvote poems that I like
2	3 units	As a user, I want to click a button and post my poem to twitter
3	4 units	As a user, I want to see how long I have to wait for the rate limit
3	4 units	As a user, I want to know if I cannot generate a poem due to rate limiting
1	1 units	As a user, I want to see top voted poems
1	1 units	As a user, I want to see recent poems
1	1 units	As a user, I want to see that my poem request is being handled

I can click a button and upvote poems that I like:

This implementation focused on adding scoring to the existing schema. A poems score is determined by its number of likes minus its number of dislikes.

I want to click a button and post my poem to twitter

This user story utilized the twitter api to push user generated poems to a common twitter account. The twitter account is here: https://twitter.com/twttpoet

I want to see how long I have to wait for the rate limit

One implementation challenge of the project was Twitter's rate limiting. Under high load, poems become impossible to generate. After completing this user story user's were provided with information about the down times, as opposed to 500 errors.

Architecture And Design

Our project is written in Python, and uses the Flask/SQLAlchemy framework in addition to Jinja2 and Twitter bootstrap for the front end. We have designed the backend to be composed of many modules that work in a sequential process to generate a poem. The broad idea is to query the Twitter API for relevant tweets, store them in our database, and generate poems using the text of those tweets. Tweets are heavily processed before being added to the database with several attributes appended including its originating URL, the syllable count, etc. A general outline of the generation process, given a hashtag and poem type, is as follows. Note the modules are described in more detail below.

1. The appropriate poem module, using SQLAlchemy calls, gets relevant tweets from the database. If this process is successful on first try, the database has already been sufficiently populated with tweets of that hashtag (from previous queries) and we are done. Otherwise, we throw a 'Not enough tweets' exception.
2. At this point, our Twitter querying module is called to get more recent tweets with the given hashtag. This raw tweet data is processed by the parser module, which cleans up the tweets before calling the rhymer and syllable modules.
3. We retry steps 1 and 2 until we hit a rate limit exception or a hard-coded upper limit of tries (usually set to only 2, as step 2 takes a long time).

As follows are some of the key modules (in the file structure of our project, these are all under the 'models' folder):

`generate`: This is the module called by the front-end to generate the poem. Its algorithm is described above.

`getTweets`: This module uses the Twython library to make API calls to Twitter.

`Rhymer` and `Syllables`: These two modules provide the text analysis functionality needed, independent of the rest of the project. They take in strings and give attributes such as the syllable count, or the phonetic pronunciation, or whether two words rhyme. They operate with an open source dictionary (CMU Dict) and fall back on heuristics to provide this information.

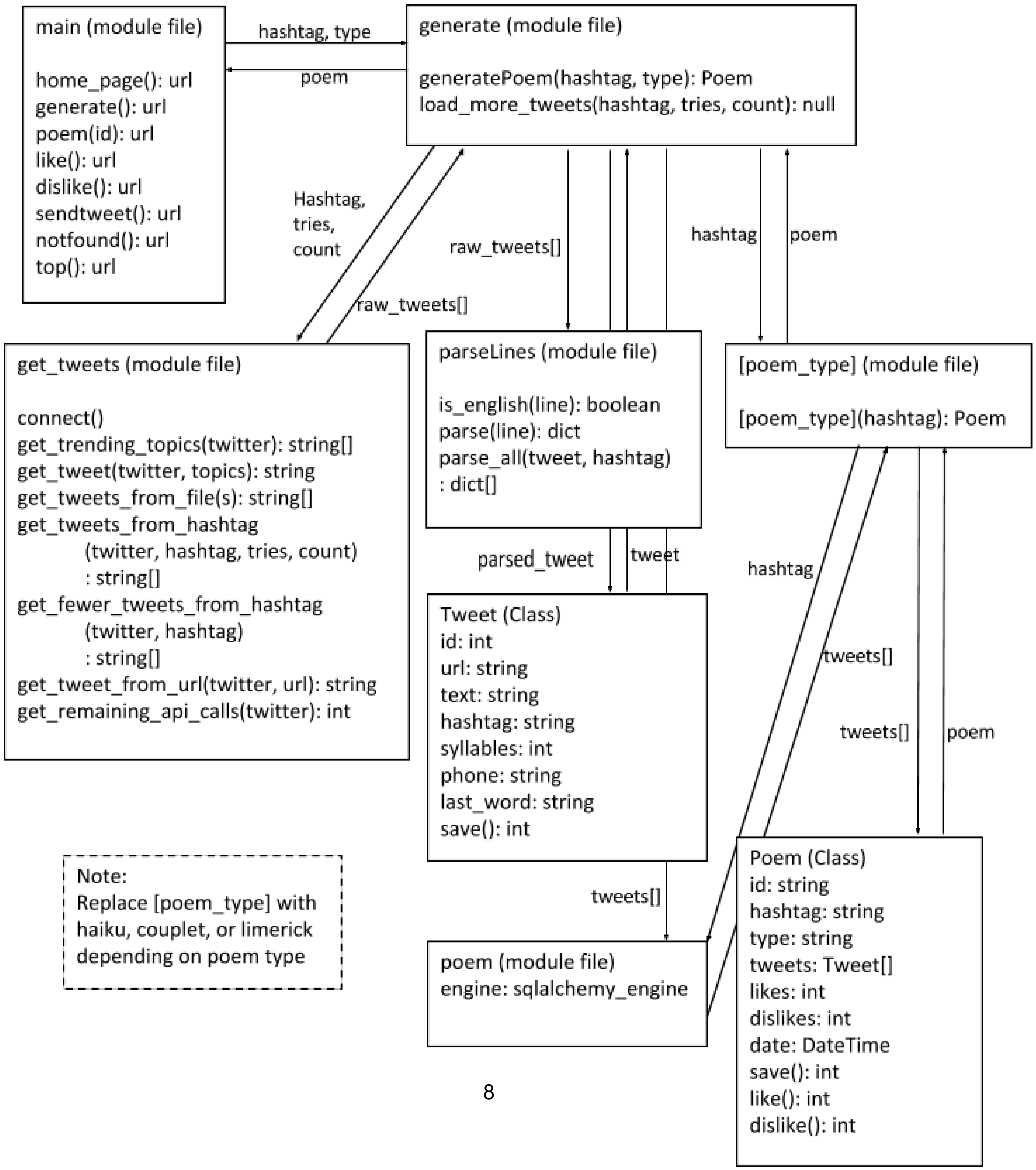
`parseLines`: There is a surprising amount of work needed to clean tweets to a form that is appropriate. For our purposes, non-english tweets are the first to be pruned (which is actually non-trivial). Many tweets are also cluttered with irrelevant and repetitive 'words' like spammed hashtags or long retweet chains ('RT @user1 @user2 @user3'). parseLines attempts to clean the tweets up before calling the above analysis modules.

`Haiku`, `Limerick`, `Couplet`: These three poem modules provide the generation logic specific to the style of their respective poems. For example, the Haiku module would at a point query for tweets with 5 syllables and a given hashtag that have not already been used in a poem.

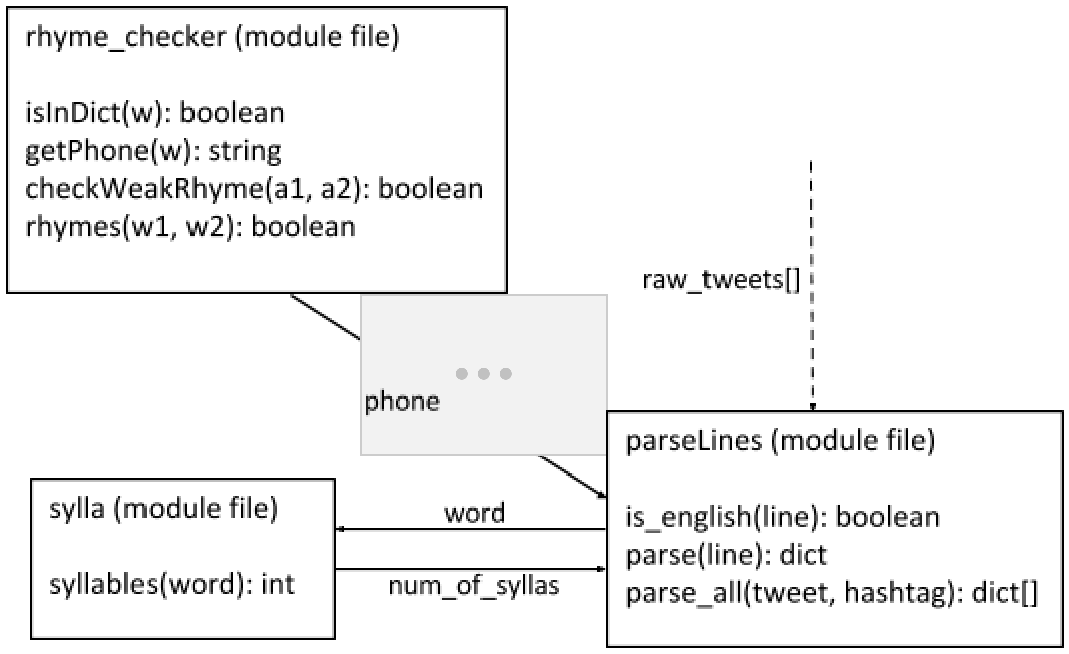
We decided on Python very early on, as the language's built in features make working with text extremely easy. We attempted to make the most out of Python's features to create clean, concise code. Given the extremely procedural, generative process we were to implement, our system forewent major usage of object oriented programming to save on unneeded complexity. Flask provided a simple but powerful framework that worked well with our minimalist front-end.

SQLAlchemy, as an ORM, worked extremely well with our concepts of a ‘Tweet’ and ‘Poem’ models and their relationship.

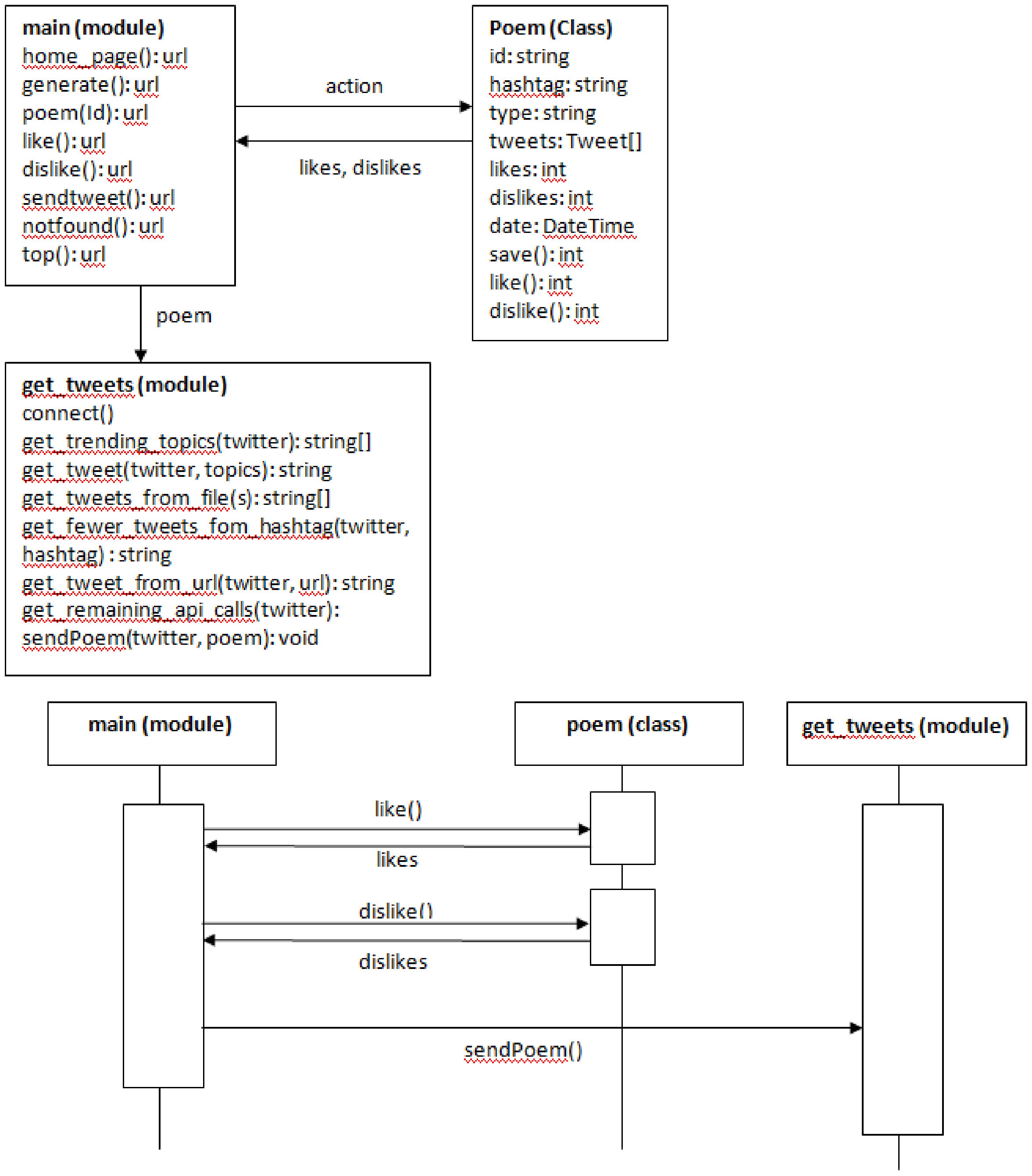
UML Diagram 1: Generating Poem



UML Diagram 2: Parsing Lines



UML Diagram 3: Poem Interactions



Database Design

```
twitterpoem.poem
CREATE TABLE `poem` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `hashtag` varchar(50) DEFAULT NULL,
  `type` varchar(50) DEFAULT NULL,
  `likes` int(11) DEFAULT NULL,
  `dislikes` int(11) DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=27 DEFAULT CHARSET=utf8mb4;

twitterpoem.tweet
CREATE TABLE `poem` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `hashtag` varchar(50) DEFAULT NULL,
  `type` varchar(50) DEFAULT NULL,
  `likes` int(11) DEFAULT NULL,
  `dislikes` int(11) DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=27 DEFAULT CHARSET=utf8mb4;

twitterpoem.tweets
CREATE TABLE `tweets` (
  `poem_id` int(11) DEFAULT NULL,
  `tweet_id` int(11) DEFAULT NULL,
  KEY `poem_id` (`poem_id`),
  KEY `tweet_id` (`tweet_id`),
  CONSTRAINT `tweets_ibfk_1` FOREIGN KEY (`poem_id`)
REFERENCES `poem` (`id`),
  CONSTRAINT `tweets_ibfk_2` FOREIGN KEY (`tweet_id`)
REFERENCES `tweet` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Future Plans

Future additions to this project could include more poem types. It could also incorporate more complicated schemas like iambic pentameter. We'd also like to make the twitter feed more entertaining for followers so if there are short poems it could tweet those out and hopefully get retweets that would introduce more people to the bot.

Yuki Tokuhiro:

I really enjoyed working on this project. I learned a lot about utilizing Agile processes to work with team members and organize work. The freedom to choose our own project also gives an equivalently larger responsibility to push for the features you want to see and allocate resources as necessary. This process gave me a lot of valuable experience in working with a team.

Ryan Sonnenberg:

This project taught me a lot about bottom up and top down design. A big issue at the beginning of the project was we had a lot of sophisticated modules that didn't work well together. Tackling the project from a top down approach, it was amazing to see how the modules could work together when a high level design in the data types they used was common throughout.

Mike Malinowski:

This project gave me more experience with working on a project in a group. At the start we did not have very much synergy but as time went on we were able to work together to build an application that we are all proud of. I learned how to depend on my teammates and trust that they will deliver. I also learned a lot about web development and moving from a test environment to a production environment.

Matthew Ford:

I liked the unpredictable aspect of Twitter Poem Generator: you don't know that content of the poems will be beyond one hashtag, so the output can be very unexpected. During this project I learned the importance of creating diagrams *before* the code is written. It was much easier to figure out what files needed to be created and what they needed to do when we had UML diagrams created. The process of using the scrum board was also very helpful because it made it easy to keep track of what everyone was working on.

Paul Osunero:

I think that this project was really unique and had a really interesting set of problems to solve. We used some fairly complex libraries to do our initial analysis, and we managed to pull it together into one cohesive project. We owe a lot of our productivity to using an agile development method. In particular, keeping our scrum board was really helpful in keeping track of what people were working on.

Vishal Patel:

I enjoyed the novelty of the project idea. The concept was simple, but the result was interesting and fun to use. The application is timeless and I will most likely come back to it in the future. In terms ofs the technical aspect, I had never worked with flask or sqlalchemy before. Though I was familiar with python, I learned a lot about style and design from my teammates. From the development process, I learned about the preparation before (and during) coding. UML diagrams helped focus our design decisions and formatted our code making it easier once we got into the meat of it.

Installation and Distribution

As an end-user, http://twtt.tarfi.re/ points to the production website.

For developers, we use Python 2.7. The following frameworks/libraries must be installed before running a local copy of the server:

- Flask
- Jinja2
- Flask-SQLAlchemy
- Twython

From there, point the database parameters to your own database (located in poem.py) and run with main.py.
Our github repo is located at https://github.com/TwitterPoemBot