JumpyPotatoes (Runmin Lu, Raymond Wu, Jerry Ye, Ivan Zhang)
SoftDev pd7
P02 -- The End
T 2018-01-09

PoliTracker

Project Manager: Raymond Wu

Project Description:

The goal of PoliTracker is to allow its users to follow and keep track of what actions their political representatives are taking. By taking advantage of the News API, The New York Times API, Twitter API, and Google's Civic Information API, based on how many posts and articles are written about a politician, we can determine how active any given politician is. By combining the Twitter API and the TwinWord Sentiment Analysis API, we can determine public sentiment and support toward any given politician. On our user interface, the user can see all of this information, as well as recent articles about a politician for more details. We can make the data we aggregate even stronger by storing the data in a database, and analyzing it for our users (public sentiment/activity across states, political parties, etc.; *additional feature*).

We will be using Bootstrap because it has a grid system that is built into Flexbox. In addition, it provides simplified and easily usable forms. There are features of Foundation that are still in their Beta version whereas Bootstrap has long supported it. Bootstrap's forum is also populated by a massive community of other developers.

__**Program Components**__:

Back-end

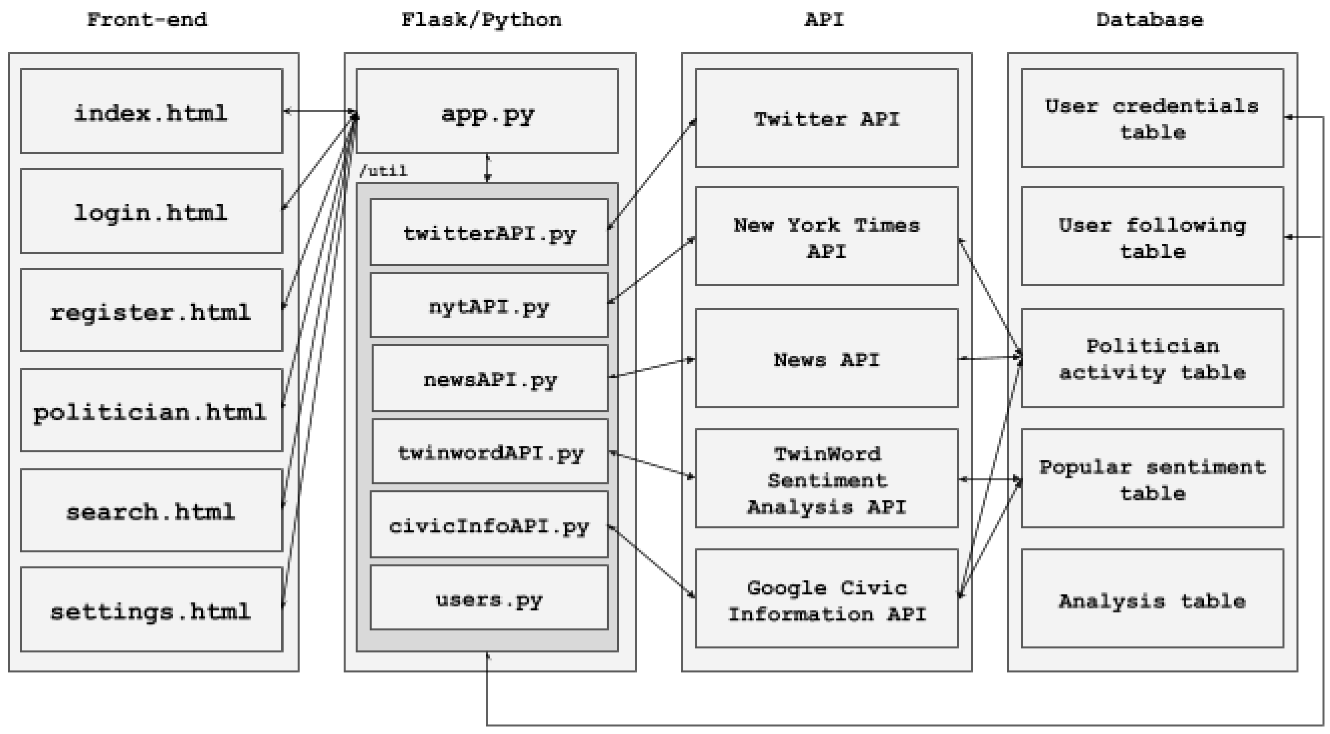
- SQLite database, stores data about:
  - User account credentials
  - Users following politicians (*additional feature*)
  - Politician activity (quantified)
  - Popular sentiment toward politician (aggregated data, *additional feature*)
  - Analysis of data (*additional feature*)
- Use of APIs, analysis (/util)
- HTML templating (/templates)
- Register/log in/log out capability
  - Logged in allows users to follow politicians
- Flask app (front-end views, connect use of multiple APIs)

Front-end

- Landing page
  - If user is not logged in, shows an alphabetical list of all politicians with a link to their Politician Page. Also provides links to the login and register pages.
  - If user is logged in, shows an alphabetical list of the politicians they have followed followed by alphabetical list of all of their representatives, **or** if they have not followed any, shows an alphabetical list of all of their representatives, with a link to their Politician Page.
- Login page
- Registration page
- Politician Page

    - General stats about politician (political affiliation, jurisdiction, term, etc.), most recent news articles, activity, public sentiment, analysis)
    - Ability for logged in user to follow the politician (*additional feature*)
- User settings
    - For user to adjust their zip code (affects list of representatives)
    - List of politicians being followed by current logged in user and the ability to unfollow (*additional feature*)

Component Map:



<u>Database schema</u>:

user_credentials

id	username	password
INTEGER, UNIQUE, NOT NULL, PRIMARY KEY	STRING, UNIQUE, NOT NULL	STRING, NOT NULL

user_following (for additional feature)

id	user_id	politician_name
INTEGER, UNIQUE, NOT NULL, PRIMARY KEY	STRING, NOT NULL	STRING, NOT NULL

politician_activity

id	politician_name	number_articles	number_media_outlets
INTEGER, UNIQUE, NOT NULL, PRIMARY KEY	STRING, UNIQUE, NOT NULL	INTEGER, NOT NULL	INTEGER, NOT NULL

popular_sentiment (for additional feature)

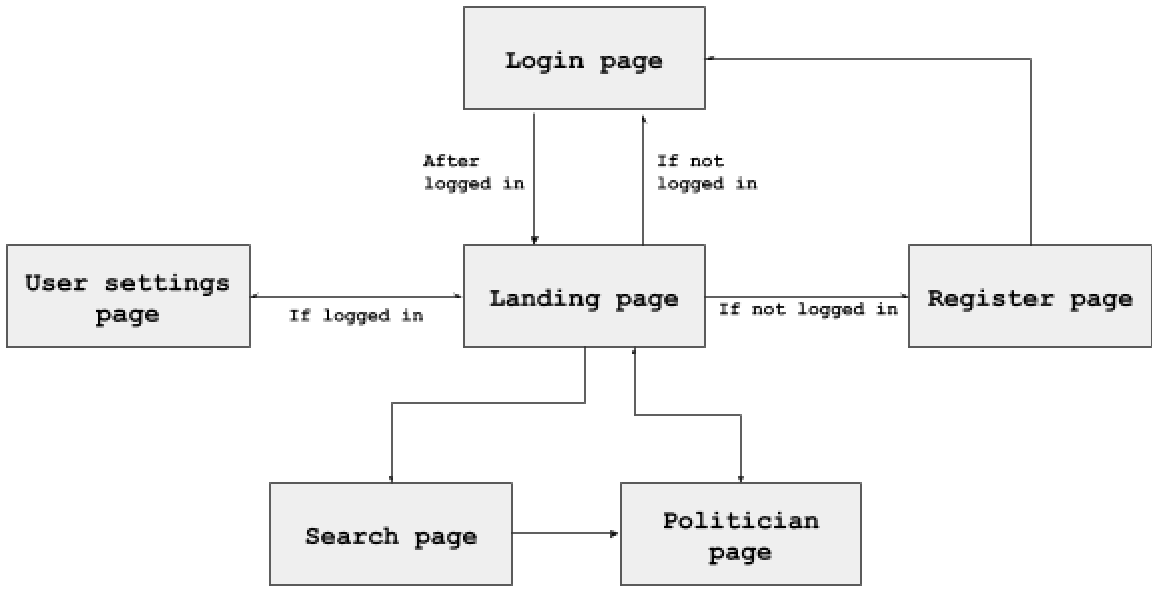
id	politician_name	number_articles	sentiment
INTEGER, UNIQUE, NOT NULL, PRIMARY KEY	STRING, UNIQUE, NOT NULL	INTEGER, NOT NULL	REAL

analysis

id	politician_name	data_point_one	data_point_two
INTEGER, UNIQUE, NOT NULL, PRIMARY KEY	STRING, UNIQUE, NOT NULL	...	...

** This will be part of an additional feature and its schema will reflect which data points will be calculated once it is decided, if time permits.*

Front End Site Map:



Breakdown of tasks:

0. Create GitHub repo, submodule **(Raymond)**
1. Start with Flask Starter Kit **(Raymond)**
2. Basic Landing Page **(Jerry)**
3. Database creation, table for user functions **(Raymond)**
4. Database table for politician data **(Runmin)**
5. Account creation/registration form and page **(Ivan)**
6. Account login/authentication, login form **(Jerry)**
7. HTML templating for front-end views **(Runmin)**
8. HTML templating for Politician Pages **(Jerry)**
9. Show all politicians and general info
   a. Use Google Civic Information API to return relevant information **(Ivan)**
   b. Display relevant information in front-end **(Jerry)**
10. Show politician activity for each politician
   a. Use News API, The New York Times API, Google Civic Information API to gather relevant data about activity **(Ivan)**
   b. Display relevant information in front-end **(Runmin)**
   c. Store data in database **(Jerry)**

11. Allow a logged in user to follow politicians (*add'l feature)*
    a. Implement a front-end follow/unfollow button on all mentions of politician names (homepage, Politician Page, user settings page) **(Runmin)**
    b. Add the user_following table in the database and add appropriate methods in the database work util/ .py file to get/set user-politician followings **(Jerry)**
    c. Allow user to see following politicians on top of list of other politicians on the homepage **(Ivan)**
12. Show public sentiment information for each politician (*additional feature*)
    a. Use News API, The New York Times API, Twitter API, Google Civic Information API, and TwinWord API to gather relevant data about public sentiment **(Ivan)**
    b. Display relevant information in front-end **(Runmin)**
    c. Store data in database **(Jerry)**
13. (*Other additional features*) Decide on data points/analysis that can be derived from data we already have/form algorithm **(ALL)**
    a. Create a database table for analysis **(Raymond)**
    b. Implement algorithm for one form of data analysis (store data in analysis table) **(Jerry)**
    c. Display relevant information in front-end **(Runmin)**