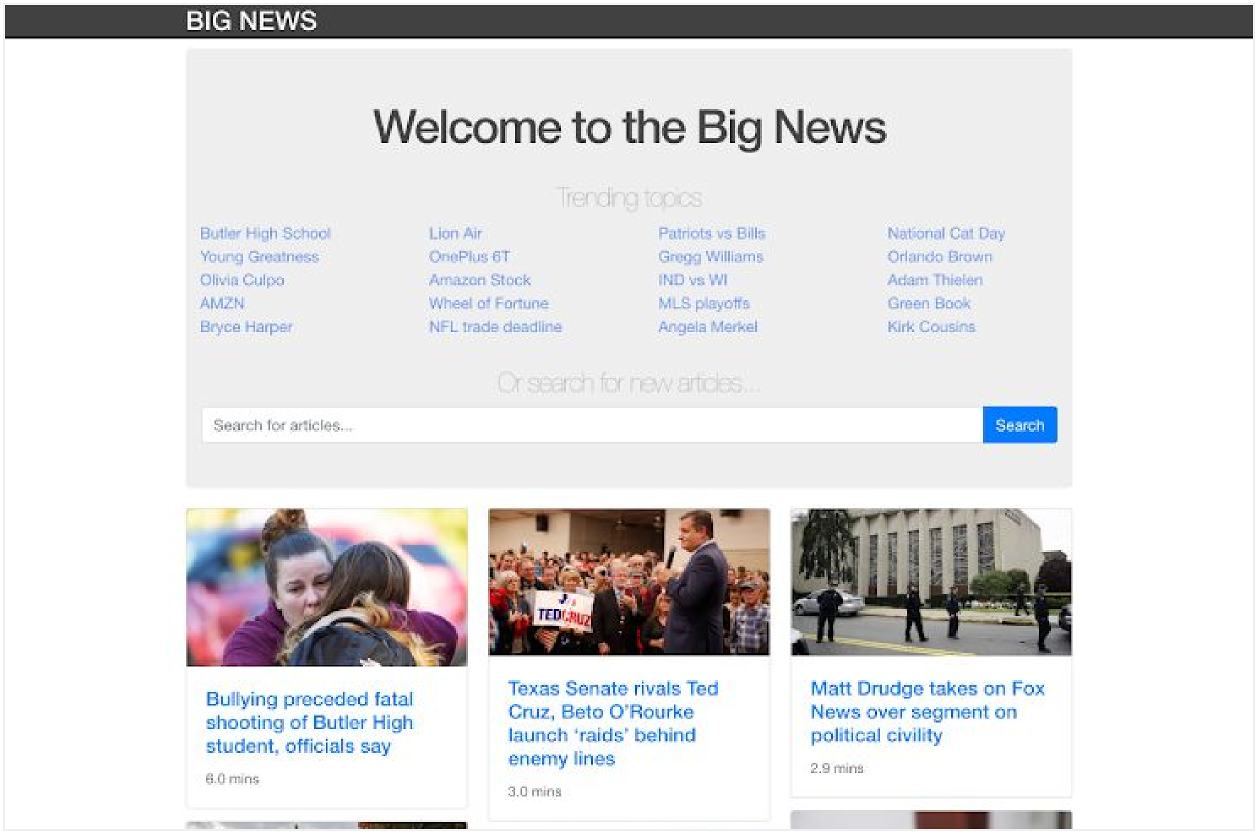
Big News
Yuanji Huang (yh3059) Siyang Yin (sy2820) Mengyao Li (ml4042) Vivien Ngo (vn2260)

First Iteration Demo



BIG NEWS

Welcome to the Big News

Trending topics

Butler High School	Lion Air	Patriots vs Bills	National Cat Day
Young Greatness	OnePlus 6T	Gregg Williams	Orlando Brown
Olivia Culpo	Amazon Stock	IND vs WI	Adam Thielen
AMZN	Wheel of Fortune	MLS playoffs	Green Book
Bryce Harper	NFL trade deadline	Angela Merkel	Kirk Cousins

Or search for new articles...

Bullying preceded fatal shooting of Butler High student, officials say
6.0 mins

Texas Senate rivals Ted Cruz, Beto O'Rourke launch 'raids' behind enemy lines
3.0 mins

Matt Drudge takes on Fox News over segment on political civility
2.9 mins

1 Demo

Date/Time: Tuesday, November 6, 2018, 2:00pm

Challenges

- We did not have a coverage report of the unit testing.
- Our pre-commit was not properly configured.
  - As part of our pre-commit, we should have a coverage report appended to our pre-commit.
  - It would have been ideal to have a block from commiting/pushing if we failed test.
  - We had an infinite loop because of the pre-commit logs.
- We had some errors with git tracking changes because of the CI.

2 User Stories

Time Estimation

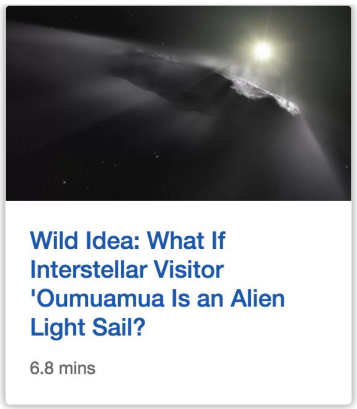
Story

As a student, I am interesting in the topic about tech news. I want to spend 30 minutes every morning to find out the latest news in the industry.

Implementation

When we start the server, we use News API to produce the articles based on what are most trending via its `/top-headlines` endpoint. From the URLs that we are given, we use the goose3 python package in order to parse the contents of each article. As part of goose3, we are given the text contents of the article, and from that, we can use a set words per minute (200 wpm) to estimate how long the average user will take to read the article.

Calculation formula :
Estimation time = words / 200 (words per minute)



Wild Idea: What If Interstellar Visitor 'Oumuamua Is an Alien Light Sail?

6.8 mins

This estimated time is then rounded to the first decimal place and displayed on the article card.

Conditions of Satisfaction

When I see a card that says it will take me 6.8 minutes to read, reading it at 200 words per minute should take me the listed amount of time. That is, the article is 6.8 * 200 = 1360 words.

Testing

We test this in the `test_time()` function of test_article.py, which mainly focuses on testing the parsing functionality. Namely,

- **Input:** article of known length (about 500 words) via its URL
- **Output:** 2.5 minutes for someone who reads 200 words per minute.

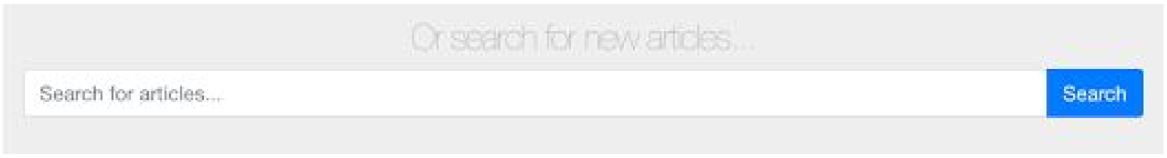
Search

Story

As a entrepreneur, I am interested in industry dynamics. I want to keep up-to-date on all the news about my industry competitors.

Implementation

In the search box, the user can input a word or words they would like to search for:



This is sent to the `/search` endpoint of our Flask application, which then uses the News API's [`/everything` endpoint](), which searches for all articles using the input string as the `q` request parameter. This means that we hit the News API every time the user enters a new search; there is currently no persistent corpus of articles.

When we hit the News API `/everything` endpoint, we have the parameter `{'sortBy': 'publishedAt'}`, meaning the News API returns the most recent articles, but we can also change this the `{'sortBy': 'relevancy'}` to ensure that our results are more related to the user's search query.

Note: This story has changed from our revised proposal. We are not trying to determine the intent of the user from their search queries. Rather, we just show results based on an API.

Conditions of Satisfaction

Based on a list of company names that I give, I want the application to show me articles about those companies. For example, this means that if I give it the keywords "Apple, Google, Microsoft", it will show me popular articles about these keywords.

Testing

There are two things we want to check. First, we want to check that *something* is getting returned. Then, we want to check that the results are actually accurate and related to the keywords. We can do this by making sure that the queried words are present inside the body of the article. So, for example:

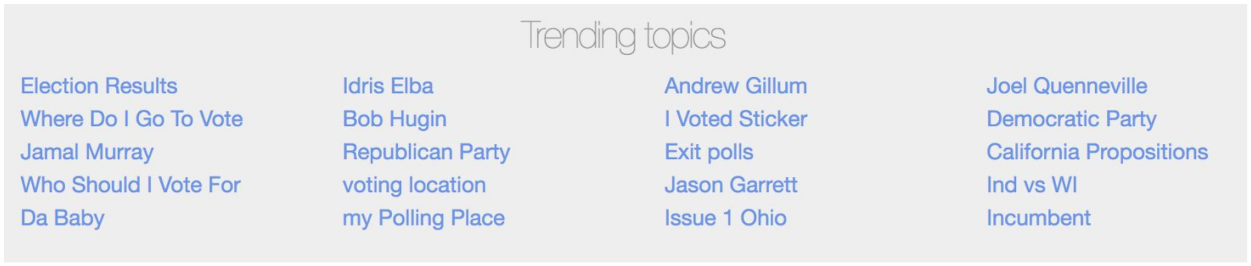
- **Input:** "Apple company" is our search query
- **Output:** a list of URLs — each of these URLs contains an article that contains at least one of the words "Apple" and "company"

Trending Topics

Story

As a busy person, I want to know what to focus on because I have too many news articles to read. I want the app to show me top key words, so that I can pay attention to the most trending ones. This may take the form of, for example, a word cloud that highlights key words from a group of articles.

Implementation



Trending topics

Election Results	Idris Elba	Andrew Gillum	Joel Quenneville
Where Do I Go To Vote	Bob Hugin	I Voted Sticker	Democratic Party
Jamal Murray	Republican Party	Exit polls	California Propositions
Who Should I Vote For	voting location	Jason Garrett	Ind vs WI
Da Baby	my Polling Place	Issue 1 Ohio	Incumbent

This is currently implemented by static pulling from Google Trending API on server startup and storing them on Mongo DB. We serve the same trending words to any user that logs in, but in a future iteration, we would like to make sure these results are periodically refreshed.

As a user interface upgrade, we would also like to make the sizes of the words change depending on the popularity of each words. This can be determined based on the Google Trending API, which also returns the number of searches (for example, `200,000+`).

Conditions of Satisfaction

I can get a rough idea of what is most popular in the news right now from a quick glance in real time (similar to Twitter trending topics maybe), so that I can dig deeper later.

Testing

This is a bit tricky because once again we rely on an API. Possibly, we can test this like so:

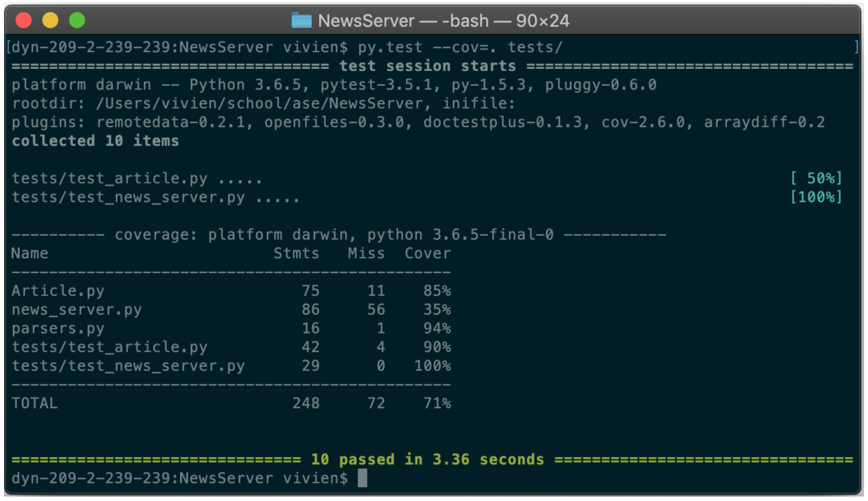
- **Input:** current date
- **Output:** a list of terms, which all appear on the Google Trends website.

3 Continuous Integration

Pre-commit

As in the example posted by Professor Kaiser, we set up our pre-commit hook in /bin/git-hooks. As shown in the demo, we write to a file kept in the /logs folder, but it currently does not work since it is written before the commit, resulting in the "infinite loop" that we discussed. We test using the pytest framework in the tests folder of our repository, and in order to produce the test coverage report, we use the pytest-cov package.

Coverage Report



```
[dyn-209-2-239-239:NewsServer vivien$ py.test --cov=. tests/
================================== test session starts ===================================
platform darwin -- Python 3.6.5, pytest-3.5.1, py-1.5.3, pluggy-0.6.0
rootdir: /Users/vivien/school/ase/NewsServer, inifile:
plugins: remotedata-0.2.1, openfiles-0.3.0, doctestplus-0.1.3, cov-2.6.0, arraydiff-0.2
collected 10 items

tests/test_article.py .....                                                        [ 50%]
tests/test_news_server.py .....                                                    [100%]

---------- coverage: platform darwin, python 3.6.5-final-0 -----------
Name                          Stmts   Miss  Cover
-------------------------------------------------
Article.py                       75     11    85%
news_server.py                   86     56    35%
parsers.py                       16      1    94%
tests/test_article.py            42      4    90%
tests/test_news_server.py        29      0   100%
-------------------------------------------------
TOTAL                           248     72    71%


=============================== 10 passed in 3.36 seconds ================================
dyn-209-2-239-239:NewsServer vivien$ 
```

Post-commit

We used Travis CI for continuous integration. As in the pre-commit, it tests using the pytest framework in the tests folder of our repository. After running on the cloud at travis-ci.com, the reports are pushed to GitHub.

4 GitHub Repository

URL: https://github.com/XJBCoding/NewsServer

This is a public repository.