Jay Batavia

CP341-Web Services

09/16/2015

NYT-Twitter API

Final Project Write-up

The motivation for this project was to make it possible to have several news sources aggregated on to a single page. Essentially this project is an effort to decrease the number of websites visited just in search for news, and bring the most common news sources together. The New York Times and the Twitter APIs were chosen for several reasons: their RESTful nature, their robust implementations, and feature support. Using a combination of javascript and python client/server classes, a simple web-page can be implemented that can be dynamically updated with news content.

The front-end of the system is just a user-facing web-page that is coded in javascript; this page has a search bar for query search. The user inputs a query they're interested in, and the page updates with news content specifically for that query. The webpage is divided into different sections for different results: New York Times(NYT) article search, New York Times Newswire, New York Times Popular Stories, and related popular tweets. These are also the APIs used in this project.

The NYT article search API allows one to search for headlines and recent stories based on a particular term. The API then further allows for filtering based on parameters like time ranges. All NYT stories have associated with them sections, like business, economics, etc., these sections could be used as a filter for results. This project uses the NYT Newswire API in such a manner; based on the query search performed, the program will check for the sections that occur the most in the query result and then serve up additional newswire stories that are filtered by that section. Thereby making the content on the page more relevant. More granular filtering could be achieved by digging through the NYTTags API and searching based on relevant tags. Finally, the page will display the most shared stories on NYT. Essentially when a search query is entered, the program fetches 10 recent stories matching the query, 15 'mixed' – recent and popular – tweets based on the search term, 5 very recent NYT Newswire stories that match the most frequently appearing section in the 10 NYT stories above, and finally 10 most shared items on the NYT website in the past 24 hours.

Originally meant to be just a served javascript webpage, the project's back-end now has two python servers running side-by-side: one serving the page, the other routing requests to the relevant APIs. Two servers had to be used to get around the 'Same-Origin-Policy'; which means that the web-page only makes requests to the latter server, and that server then makes an API

request for the page. No data is stored on either of the servers. The structure of the 'routing' server follows the dispatcher pattern somewhat loosely, and can be modified to be an endpoint that would mash the twitter and NYT APIs together. There are two different python files that handle NYT API requests and Twitter API requests.

An example query search on the article search API:

Python:

```
def getQueryRecent( query_term ):
	connection = http.client.HTTPConnection( NYT_HOST )
	query = {'q': query_term, 'sort': 'newest', 'hl': 'true','facet_field': 'section_name', 'api-key': '%s' %NYTSEARCH_KEY}

	connection.request( 'GET', SEARCH_PATH+'?'+urllib.parse.urlencode(query))
	try:
		response = connection.getresponse()
		decoded_response = response.read().decode(encoding)
		result_dict = json.loads(decoded_response)['response']
		article_dict = filterResult(result_dict)
	except:
		return []

	return article_dict
```

The article search offers flexibility in terms of optional parameters. There are numerous ones to use to filter the returned content. For example, the presence of the 'facet_field' query parameter and the value 'section_name' (as above) would mean that the articles returned by the query would have an additional field attached to them indicating what NYT section they belong to.

There were a few challenging aspects of this project; the foremost being working around the Same-Origin-Policy. The project ended up being much different than initially planned owing to tight CORS restrictions with some APIs. This was eventually mitigated by using a second server. Another challenge was to make sense of all the data being provided. Twitter, for example, has upwards of 25 fields of metadata for a tweet that is going to be 140 characters long. Picking and choosing between what data is relevant and what is not could be daunting.

The NYT and the Twitter API definitely are powerful tools to consume their data. Given enough time, the goal for this project was to have real-time news updates on the webpage based on trending news topics, or most shared NYT tags, or the tweets being retweeted the most. There are quite a few possibilities.

The biggest take-away from this project, and the course in general, has been that designing robust web services is not easy. There is a lot of thought to be put into how different components come together and make a cohesive, useable framework. And that well designed web-services should not be taken for granted.