Matt Richards

**Rotten Netflix**

*A film cross-reference utility*

I developed ***Rotten Netflix*** as a utility to cross-reference Netflix titles with corresponding entries in Rotten Tomatoes. I was motivated to create this app because of frustration I had with the Netflix movie rating system. Netflix, rather than supplying the user with an aggregate rating for a title, will instead supply suggestive ratings in which it calculates a custom star value based on what it knows of the user’s viewing history. This is frustrating for me, as I wanted an easy to way to get a slightly more objective view of the movie (at least as objective as an aggregate of subjective views can be). To create this app, I needed access to the open APIs for both Rotten Tomatoes and Netflix. Lucky enough, both companies use Mashery (http://www.mashery.com/) for their key distribution, so I was fairly quickly and easily able to gain developer accounts for both APIs. I developed this app on Java, placing most of the processing in the Java side rather than the UI side to allow for easy porting of this app to other mediums.

**Rotten Tomatoes**

The rotten tomatoes portion was the easiest part of this app. All API calls are unsecured besides the inclusion of the developer key as part of the URL String. Here is an example of an API call to the Rotten Tomatoes server:



JSON

As indicated in the URL. This request returns a JSON object for processing. I then used a generic JSON parser I downloaded from json.org to separate the response items in an Array of RottenTomatoResult objects. Here is a snippet of the JSON returned from the RottenTomatoes server:

**[{"id":"104762093","title":"Fargo","year":1996,"mpaa\_rating":"R","runtime":97,"critics\_consensus":"Violent, quirky, and darkly funny, Fargo delivers an original crime story and a wonderful performance by McDormand.",**

**"release\_dates":{"theater":"1996-03-08","dvd":"1997-06-24"},**

**"ratings":{"critics\_rating":"Certified Fresh","critics\_score":94,"audience\_rating":"Upright","audience\_score":91},**

**"synopsis":""...}**

**Netflix**

Exercising the Netflix API is where things got a lot more complicated.

Oauth

To pull any needed data from the system, they required the use of “Oauth” to properly sign requests. OAuth is an open source security standard that was developed to support the Twitter API as a complement to OpenID. The components needed to send from client to server for OAuth are as follows:

|  |  |
| --- | --- |
| **OAuth\_Consumer\_Key** | This is a security credential that is sent, in plaintext, as a URL parameter (supplied by Mashery) |
| **OAuth\_Nonce** | Typical nonce, generated at runtime to prevent replay attacks |
| **OAuth\_Timestamp** | Number of seconds since the epoch (standard operating procedure) |
| **OAuth\_Signature** | This is a digital signature generated through a standard HMAC-SHA1 hashing algorithm using a shared secret (also supplied by Mashery) |

**Generating the Nonce:** The Nonce is generated using the SecureRandom java class. It generates a pseudo-random byte array using the “SHA1PRNG” algorithm (SHA1 Pseudo-random number generator)

**Generating the Signature:** The digital signature is generated through a standard HMAC-SHA1 hashing algorithm. The base string that actually gets hashed is defined as:

***VERB*** *+ resource\_path +* ***parameters***

For example, a request to retrieve the details for the movie Fargo would yield a pre-hash string of:

**GET&http%3A%2F%2Fapi-public.netflix.com%2Fcatalog%2Ftitles&oauth\_consumer\_key%3D[redacted]%26**

**oauth\_nonce%3D%255BB%25403feb8c6a%26oauth\_signature\_method%3DHMAC-SHA1%26**

**oauth\_timestamp%3D1354460476%26oauth\_version%3D1.0%26term%3Dfargo**

The String above is URL encoded (besides the ampersands separating the three main components). This, then, is the final string that is hashed using the shared secret supplied by Netflix. The generated signature is then appended on to the end of the URL string that was signed, and that resulting String is the URL that defines the GET request to Netflix.

XML

Unlike Rotten Tomatoes, which returns JSON results, Netflix returns standard XML. To handle these responses, I included a JAR for Dom4J, which is an open source XML-parsing utility. I used this to read the XML and store the search results in to an array of NetflixResult objects.

**Cross Referencing the Data**

After generating the array of NetflixResult objects and RottenTomatoResult objects, I then cross-referenced them to find the resulting list to send back to the user. To accomplish the cross-reference, I ran all of the movie titles through a filter that removes all “:”, “-“, and “ “ characters and replaces them, instead, with empty strings. The reason for this additional parser is because I saw a small yet significant disconnect in the title construction between Netflix and Rotten Tomatoes. Namely, Netflix uses a lot of colons (e.g. Star Wars IV: A New Hope) and Rotten Tomatoes uses dashes (e.g. Star Wars IV – A New Hope). A simple text compare of these two titles would not come back positively. By removing all spaces and punctuation, I could do the text compare on only the alphanumeric characters in the title.

**UI**

For the purposes of this academic example, to make the delivery easier and to not spend a ton of time on the presentation layer when that was not the intent of the project, I implemented this using a very simple JFrame. However, I made sure to keep the search results and the presentation as de-coupled as possible so that I could later put a different user interface on it. My future plans include turning this in to apps build on Grails, Android, and iOS.