Twitter News Generation - Final Report

Rafał Szymański Maciek Albin Sam Wong Suhaib Sarmad
Jamal Khan {rs2909, mja108, sw2309, sss308, jzk09}@doc.ic.ac.uk

Department of Computing - Imperial College London

January 8, 2012

Contents

1 High Level Overview **3**

1.1 Product Functionality 3

2 Technical Overview **5**

2.1 Analysis 5

2.2 API 7

2.3 Server and Production Environment 7

3 Software Engineering Issues **9**

3.1 Technical challenges, choices and risks 9

3.1.1 Analysis 9

3.1.2 API 11

3.1.3 UI 12

3.2 People and Team Management 14

3.3 Collaboration Tools Used 16

3.3.1 Git 16

3.3.2 Trello 16

3.3.3 Skype 16
3.4 Development Techniques 16
3.5 Testing 19
3.5.1 Unit and Mock Testing 20
3.5.2 Logging and Debugging 20
3.5.3 Unit and Mock Testing 20
3.5.4 UI Testing 21
3.6 Summary of each members contribution 21
3.6.1 Maciek Albin 21
3.6.2 Jamal Khan 22
3.6.3 Suhaib Sarmad 22
3.6.4 Rafał Szymański 23
3.6.5 Sam Wong 23
3.7 Ethical and Environmental Impact of our Project 24

4 Validation and Conclusions **25**

4.1 User Validation 25
4.2 Code Validation 25
4.3 Conclusion 25

5 Bibliography and Tools used **26**

6 Appendix **28**

6.1 Team Meetings 28

Chapter 1

High Level Overview

1.1 Product Functionality

Twitter News Generator - An webapp that pairs current news with relevant tweets.

1.2 Our Aim

Traditionally, the media has been presenting news stories biased with the writers' point of view or agenda. Recruiting readers to join their agendas. This webapp presents a balanced view to the readers. We pair up relevant tweets to news emerging around the world. Our aim is to create a new kind of news reader that also presents the discussion behind the news.

1.3 How we planned to achieve this

When a news story breaks, our backend performs a textual analysis on the story to extract keywords. These keywords are then used as the seed to fetch relevant tweets. Once we have a list of relevant tweets, we perform 1) sentiment analysis on these tweets to gauge positivity of each tweet, 2) textual analysis to extract keywords and 3) using 1 and 2 to gauge which tweets should be shown first.

Figure 1.1: Minimalist UI of our end product

Once these analyses are done, they are presented in our minimalist yet visually appealing news reading interface. The interface will self-updated periodically, and infinite scroll was implemented to look at past news. Figure 1.1 shows the final end product.

Chapter 2

Technical Overview

We decided to divide the project into two separate technical parts: *Analysis* and *UI*, that would be connected through our own *API* server[1].

Analysis was written in Python and its main tasks were to gather current news, fetch relevant tweets, analyse them in bulk and save the results into a database.

API was also written in Python. It provides easy access to the database for the *UI*.

UI was written in HTML5 and JavaScript. It utilises the API to display the news.

We kept *Analysis* and *UI* as separate as possible. Thanks to that we achieved low coupling and extending the application became quite easy. We use MongoDB as the our database. What follows are short descriptions of how each part of the project works.

2.1 Analysis

We decided to divide the Analysis process into three separate threads:

News Fetcher responsible for fetching current news stories. Getting information relevant to us and generating keywords

[1] A general overview of the information flow in the application can be seen in Figure 2.1.

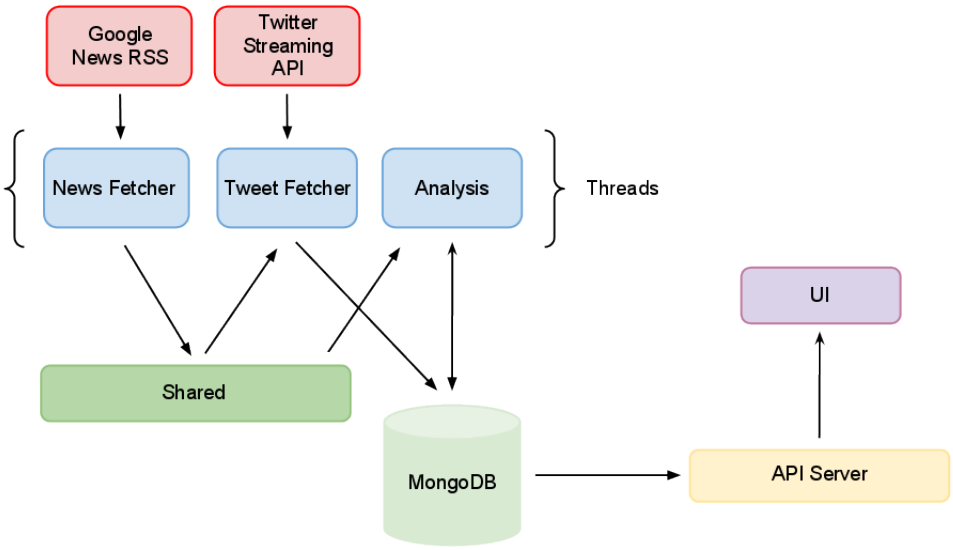

Figure 2.1: General overview of the information flow in the application. Arrows point into the direction information flows.

Tweet Fetcher responsible for fetching tweets using keywords generated by *News Fetcher*

Analysis uses tweets gathered by *Tweet Fetcher* and info gathered by *News Fetcher* to calculate Twitter's reaction to news stories

This is the basic sequence of events that happens in the analysis process:

1. Start three threads: *News Fetcher*, *Tweet Fetcher* and *Analysis*
2. *Tweet Fetcher* waits for the *News Fetcher* to get current news
3. *News Fetcher* gets current news from the RSS feed
4. *News Fetcher* extracts basic information from the RSS feed
5. *News Fetcher* generates keywords using AlchemyAPI[2]
6. *News Fetcher* saves all the info generated into *shared* and goes to sleep for 500s

[2]See Chapter 5.

7. *Tweet Fetcher* wakes up gets all the keywords from *shared* and starts fetching tweets mentioning those keywords restarting every time *News Fetcher updates*

8. *Analysis* wakes up every 5 minutes and gets current stories from *shared*

9. *Analysis* checks if the stories in *shared* are already in DB, if not it inserts them

10. *Analysis* gets the tweets saved by *Tweet Fetcher* in last 5 minutes and attaches them to stories in the current period in DB

11. *Analysis* performs word cloud generation on the tweets attached to stories

12. *Analysis* performs sentiment analysis on the tweets attached to stories using Twitter Sentiment[3]

2.2 API

API provides a very thin layer between the *UI* and the data stored in the Database. It's written in Python using Flask[4] and provides the following calls returning results in JSON:

- `/api/news` returns the list of 10 current news stories and timestamps for the earliest and the newest stories
- `/api/news/before/<timestamp>` returns a list of 10 news stories before the `<timestamp>` and the timestamp of earliest story
- `/api/news/after/<timest22amp>` returns a list of 10 news stories after the `<timestamp>` and the timestamp of newest story
- `/api/story/<id>` returns a detailed description of the story requested

2.3 Server and Production Environment

It is essential that all used technologies used be glued together to work on a production environment. For this, on the outer server layer resides Nginx, which proxies requests to the lower level uwsgi process[5] that serves the API and the main page. Static content, such as javascript files or images, are directly

[3] See Chapter 5.
[4] See Chapter 5.
[5] This allows Nginx to talk to python and flask

handled by Nginx to reduce overhead. Nginx was chosen over the other alternative, Apache, for its lightweight memory footprint, and existing documentation regarding the integration of Nginx, uwsgi, and python.

Continuous deployment was set up to allow us to code, push a change to the repository, and see it live nearly instantly. Whenever a code push occurs, we have a git post-receive hook that copies the static files appropriately so that they can be served, and restarts the uwsgi process[6]. This means that as soon as a code push occurs, the new version is live.

[6]This is the process that serves the python code

Chapter 3

Software Engineering Issues

3.1 Technical challenges, choices and risks

What follows is a more detailed description of different parts of the project together with analysis of choices made.

3.1.1 Analysis

Analysis consists of everything up to and including MongoDB in Figure 2.1.

Programming Language We decided to use Python as the main developing language for this part of the project. We needed a relatively lightweight language that would be quite good with connecting to many different web based services and analysing data. Other languages we considered were Ruby and JavaScript (using Node.js).

Most of us had used Node.js before, when working on last years group project. It's a great language for fast and easy realtime communication and it's quite simple to start writing web applications using frameworks like Express[1]. The problem JavaScript had was a lack of threads. We knew we wanted to separate the tweet fetching, news fetching and actual analysis parts out into separate components that could still communicate with each other. It would have been possible with JavaScript by creating 3 different processes, but communication

[1] `http;//www.expressjs.com`

between them would be much more complicated. The other reason we opted out of using Node, was the relative youth of the platform. There just weren't that many useful libraries available.

The choice between Ruby and Python was harder, as they are both fantastic web languages with thriving communities and lots of libraries. It was made easy after doing some more research on analysing twitter and the *social web* in general. We found two great books on the topic[2] and both of them advocated using Python for the task. They also came with some tested solutions for fetching tweets and analysing them in bulk, so we decided to use the same language they do.

We are very happy with the choice we've made. Not only have we learned a very popular language but also found lots of help and tools online, which made the project a little bit easier.

Database We decided to use MongoDB as our main data store. Most of us had used it before and we were very happy with how it worked. We opted for a NoSQL Document-oriented database, rather than for a classical SQL solution for a couple of reasons.

The data we get for twitter stream is not always the same. Very often there are fields missing, some of them sometimes contain numbers sometimes strings. It would be much harder to save all of that data in a SQL database.

The other advantage MongoDB has here is that it natively operates in JSON, the format twitter data is supplied to us and it makes it extremely easy to save the tweets and retrieve them later.

Our project itself also didn't have a rigid structure initially. We were changing what we write into the database all the time and everything was quite fluid until the very end. The fact that MongoDB didn't impose any structure helped a lot.

Other tools We decided that in order to attach relevant tweets to news stories we will need to generate keyword relevant to the articles. That turns out to be much trickier than we expected. Keywords need to be specific enough so that the tweets we attach to the story are actually relevant but not so specific that there is no tweets mentioning them. It's quite easy for a person with all the cultural knowledge we have to do that, but writing a program to do that efficiently is actually quite hard.

[2] *Programming Collective Intelligence* and *Mining the Social Web*, for more details see Chapter 5

We initially used TermTopia's TermExtractor[3]. It was a simple tool that just analysed the title of the article and tried to decide which word in there could be keywords and rank them in order of importance.

We quickly realised that this will not be enough, very often the relevant keywords don't appear in the title, they appear in the text of the news article. We decided to use AlchemyAPI's keyword extractor[4]. We emailed them and got special university access that increased the amount of API calls we can make.

We initially wanted to do sentiment analysis ourselves, but we quickly realised it's going to be extremely hard. We did find some interesting papers[5], but realised this approach would involve diving really deep into machine learning and teaching our system to recognise tweets for some time. Fortunately authors of the paper also provide a service for analysing tweet sentiment and an API that goes along with it[6] and we decided to use that.

3.1.2 API

From the start we wanted API to be a very thin layer on top of the database. The communication between the API server and the *UI* was supposed to be done using JSON a format that's very easy to use with JavaScript. WE also wanted to keep all of the backend in the same language. It would have allowed possible code migrations from *Analysis* to *API* and vice-versa and also wouldn't require us to learn extra tools.

When looking for a framework we initially considered Django[7], but quickly dismissed it as being way too big for our needs. It requires major setup and is set up for more classic web pages, not live updating, JavaScript based websites.

AFter some research we ended up with two possible choices Bottle[8] and Flask[9]. After quickly going through the documentation it seemed to us that Flask has better support for returning well formed JSON data, so we decided to go with it. API is a relatively small part of the project, so we didn't want to spend too much time making this choice.

[3] `http://pypi.python.org/pypi/topia.termextract/`
[4] `http://alchemyapi.com`
[5] `http://www.stanford.edu/~alecmgo/papers/TwitterDistantSupervision09.pdf`
[6] `http://twittersentiment.appspot.com/`
[7] `http://www.django.org`
[8] `http://bottlepy.org/docs/dev/`
[9] `http://flask.pocoo.org/`

3.1.3 UI

The UI connects to the server's API and displays the information in a clean and appealing tiled layout. These tiles dynamically rearrange themselves and adjust to the size of the users' window or screen. Each tile displays an image related to the article aswell as the title of the article in question. On hover/click of a tile, the image folds away (CSS3 eye-candy) revealing a short summary of the article, a word cloud made from common words found in relevant tweets, a list of top relevant tweets and the overall sentiment derived from the relevant tweets.

We use a HTML frontpage with CSS3 for styling and animations. This allows for a clean and visually appealing user interface. Javascript, JQuery and its various plugins are used to fetch the necessary information from the server API and to make the page dynamic, user friendly and more enjoyable to use.

Specifically we use:

- CSS3 for the fold-away image tile
- Javascript/JQuery for fetching news and stories, and adding the news boxes to the page
- JQuery Masonry for the dynamically rearranging tiled page layout
- JQuery WordCloud for on the fly/dynamic word cloud creation
- JQuery Infinite Scroll for nice/clever way of getting archived news stories
- Various other JQuery plugins for other smaller tasks including nice scrolling

For article images, we originally wanted to extract the image from the linked article but because we get our news from google which links to several different news sites, writing an extraction method which worked with all these sites proved too difficult and time consuming to be worth it.

Instead we decided to the Google Images API and do an image search with the article's title which almost always resulted in a relevant image. Due to the nature of Google Image search, sometimes the resulting image is too low resolution or the link may be broken. When this occurs, we iterate over the search results until we find an satisfactory image.

The front page is dynamic, every few minutes it checks with the server if there are any new articles to display and if any changes (tweets/word cloud/sentiment) have occurred to existing articles. It then updates the news tiles accordingly.

Scrolling to the bottom of the page will invoke a call to fetch past/archived articles. This creates a clean and continuous experience for the user.

We considered but did not use PHP (or any other server-side scripting language) and opted for Javascript because this relieves some of the workload from the server and also makes it easier to make the page dynamic and update in real time.

We also considered using SVG and d3.js[10] for rendering additional information such as twitter following/followed networks but due to a design change we decided this was no longer necessary.

3.2 People and Team Management

We deem it important to discuss how we have managed our team and worked as a team. As it often happens with group projects, problems relating to group interactions and dynamics surface. Firstly, in our group, everyone knows each other well and is good friends with everyone else, which reduces the authoritative attitude of any one group member towards others. In a real life project, you have a manager with an authoritative position to do things such as lay you off, which certainly induces at least a little bit of motivation. In a university project where all the members are good friends, it is harder to establish an authority that is able to motivate others, especially during a time where everyone has a lot of other personal tasks at hand; motivation has to come from within the group members.

During this term all of us had lot of other personal responsibilities, including a lot of coursework for other subjects, setting up and studying for internship interviews for next year, and teaching PMT classes, among many others. Considering every one of us has so many personal tasks that directly affect just him and not the group, we saw decreased motivation towards the project from all group members. We have experienced *Social Loafing*[11], which is "the phenomenon of people exerting less effort to achieve a goal when they work in a group than when they work alone." We have had problems keeping to the iteration task assignment on Trello[12], ie were not, as individuals, completing the iteration subtasks within the timeline. The root cause of this was likely related to the above explained *social loathing*, and diffusion of responsibility within a group without a clear authoritative figure.

Our proposed solution was rationalising to all group members the relative im-

[10] `http://mbostock.github.com/d3/`
[11] `http://en.wikipedia.org/wiki/Social_loafing`
[12] `http://trello.com`

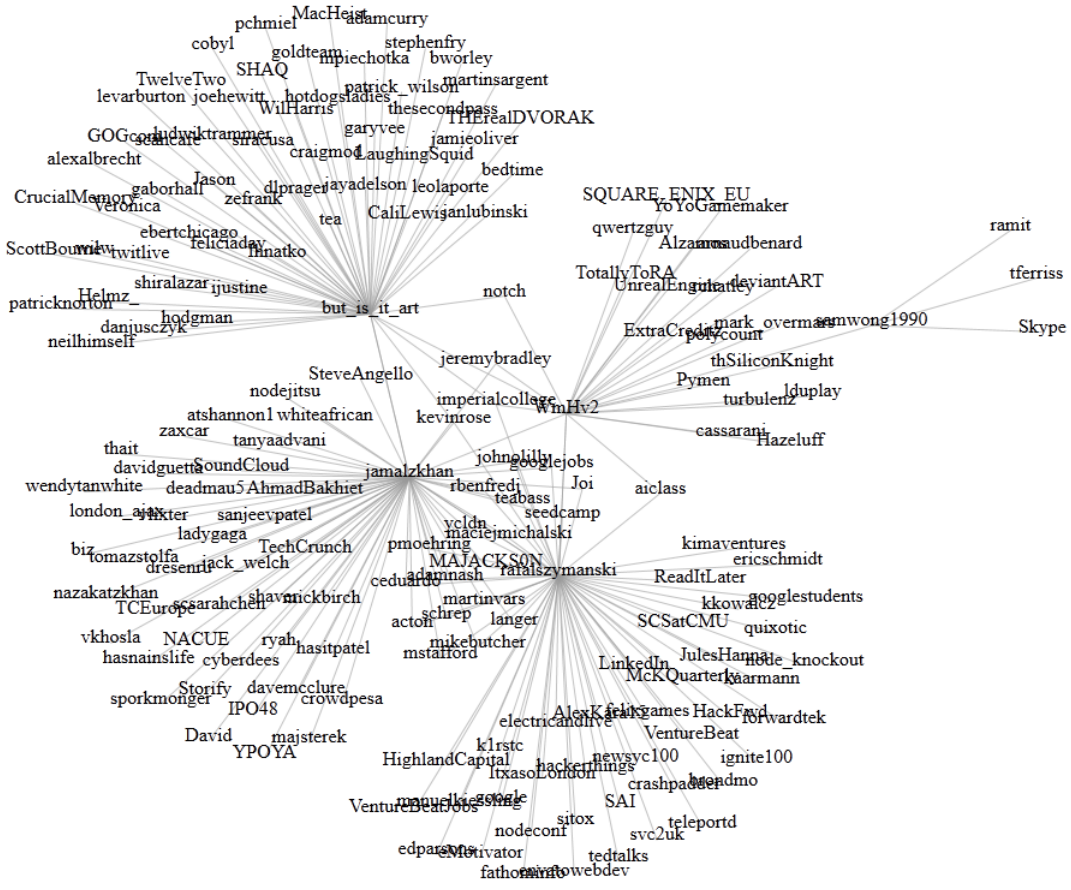

Figure 3.1: Twitter network created from our group members' accounts using d3.js

portance of this project compared to all other coursework - 440/1700 points. Receiving a **C** for our first report was disappointing and a blow to our motivation, as none of us had gotten a C in Imperial before, but on the other hand it was a sign we need increase the quality of our work and approach the project more seriously. This, combined with a meeting with Robert Chatley to discuss problems with report number 1, ensured grades of A+, and A, for reports 2 and 3 respectively.

Even though over the course of the project, we, as a team, have improved in terms of teamwork, there is still a lot to be improved. We see three important features that we lacked, but that would characterize a more efficient team. Firstly, it is necessary to have an authoritative figure which will watch over the group and assign tasks. It is important that task assignment is followed by the team members. Of course, all topics should be open to a reasonable debate, but in the event progress stalls, the authoritative figure is there to take a decision and lead the project forward. Secondly, it would be beneficial if, when working on the project, the team members looked at each other as co-workers and not simply as school friends. If everyone is good friends outside of the project, it is necessary to introduce a change in attitude, so that, while remaining friends, team members can work as co-workers towards a common goal - the completion of the project. Without a change in attitude, we risk the teamwork project to be a considered a fun 'game' of not much importance, and the impact of this will be visible on the final output. Thirdly, as said by Peter Drucker[13], we claim that is vital to measure progress frequently. Trello was a big help in establishing who is to do what, but what was lacking was an enforcement of these assignments. We have planned to have frequent scrum meetings regarding the completion of tasks, but instead, since we all saw each other during lectures anyways, we usually discussed the progress without having had a dedicated meeting. This was an unsatisfactory decision - we should have gotten into the habit of having dedicated progress meetings, where everyone has to showcase what they achieved in the previous 2 or 3 days. Under the knowledge that one has to have something to present at a team meeting, there is a much higher chance that said team member will have produced an amount of work worth discussing. Otherwise, with no set meetings, there is less incentive for individuals to contribute fairly and frequently.

We now discuss the assignment of tasks and group subdivision. We have separated our group of five people as follows: 2 for UI, and 3 for backend, so everyone is doing what they are more comfortable with. From time to time, but maybe not as often as we could have done, both of these subteams utilized pair programming. Many times one of us knew something useful, for example a specific MongoDB query syntax, and can help the other without having to resolve to Google. This ensures a good flow of information between the team, and levels out our knowledge. A visible problem was the following scenario: we

[13] *"What gets measured, gets managed"*

are working in the same location, but not currently pair programming. Person A is concentrating and has a train of thought, while person B, interrupts to ask for a triviality such as the aforementioned MongoDB query syntax, which one easily find on google within 30 seconds. Now A has lost concentration and needs to get back on their train of thought. We see that B has shortened the amount of work for himself, but has lowered the total productivity and output of the group. This is to be avoided at all costs. There is a time to discuss and ask questions, and there is a time to work quietly. The group most not get itself into a state of perpetual back-and-forth questioning of, for example, API syntax, or else group productivity is greatly diminished. Nevertheless, when we set to do a session of pair programming, output and learning were both increased.

We discussed that the group was separated into frontend and backend, but we still thrived to have a *bus count*[14][15] of the size of our group - we wanted everyone to understand, and if needed, to work on any part of our stack. In the end, this has not fully materialized, as the frontend group don't know the intricacies involved in the threading of the backend processes, and the backend team is not aware of the various UI hacks the frontend team had to develop to have a nice cross-browser and pretty UI.

3.3 Collaboration Tools Used

3.3.1 Git

We have used the git version control system for keeping track of the project, for easily reverting if there is a problem, and for having a very quick deploy mechanism. Whenever a code push occurs to our git repository, we have a git post-receive hook that copies the static files appropriately, and restarts the appropriate processes. This means that as soon as a code push occurs, the new version is live right away.

3.3.2 Trello

Trello[16] is a very good piece of software by FogCreek. It is a digital board that allows you to create post-its and write the product backlog, and move tasks between the product backlog, the current iteration, and the finished tasks. We

[14] `bit.ly/j1zquA` - quite interesting article.

[15] Bus count - "how many people in your team have to get hit by a bus before you're all dead in the water"

[16] `http://trello.com`

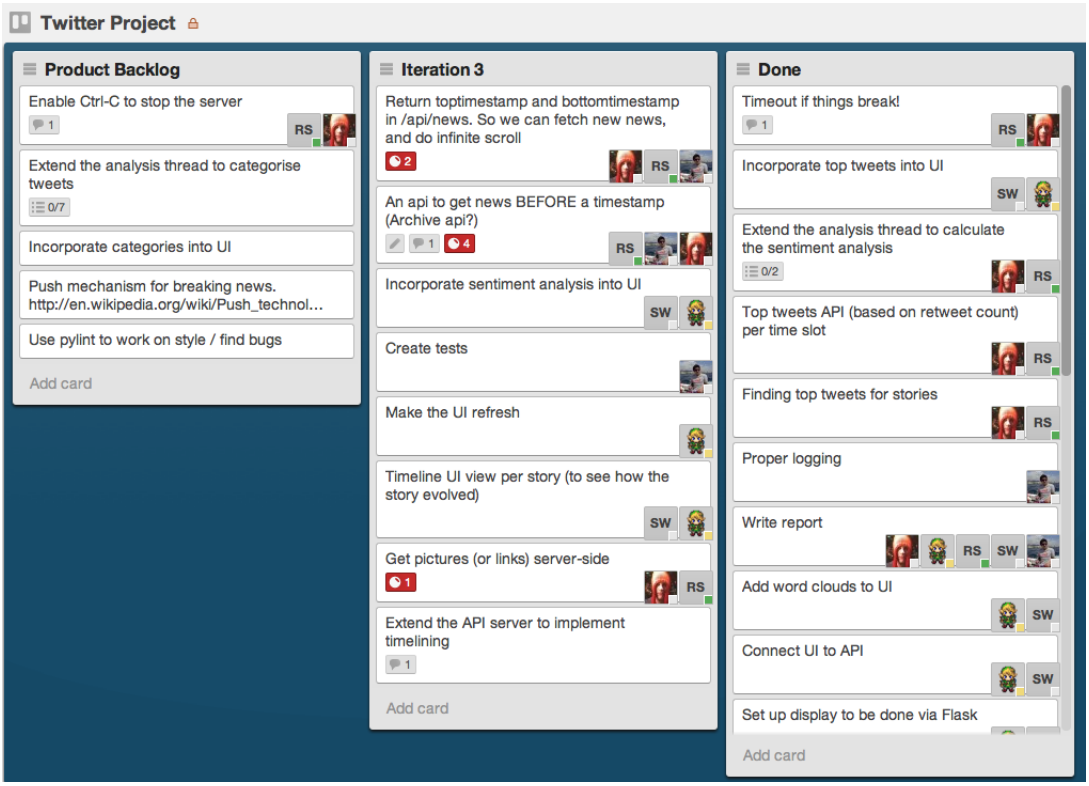

Figure 3.2: Trello project management board.

have used Trello throughout and found it very useful. See Figure 3.2 to have an overview of how Trello operates.

3.3.3 Skype

We extensively used Skype to discuss the project and set up team meetings.

3.4 Development Techniques

We adopted agile development methodologies, but as discussed in the section on people management, we have had some problems in having set project meetings. Nevertheless, we have adopted the use of iterations and developed using iterations plans.

First iteration of iteration plan

- First Iteration MVP - basic news augmenting functionality and live updates.
  - Setup DB and server to receive Tweets and RSS
  - Research suitable DB (MongoDB) – Select frameworks and architecture – digging into twitter API and RSS
  - Design algorithms to generate keywords (Look into using natural language processing libraries, there are a lot available for Python)
  - UI Mockups and UX Research
  - UI Implementation: Framework and plugins to use (eg jQuery, jQuery plugins, Twitter Bootstrap, etc)
  - Getting the web app hosted
- Second Iteration
  - Polished and Efficient implementation including Debugging, refactoring and optimisation
- Third Iteration
  - Extensions including: • Sentiment Analysis algorithm • Categorization algorithm
- Fourth Iteration
  - Final Product

After submitting our first report we realised the Iteration plan we proposed was flawed. In many parts instead of focusing on the user facing features we have reported on the technical aspects of what we will do. One whole iteration was dedicated to "Debugging, refactoring and optimisation".

Therefore, we drafted a new iteration plan and described what the project goals are in terms of user-facing features.

Second iteration of iteration plan

- First Iteration
  - Setup DB and server to receive Tweets and RSS
  - Research suitable DB (MongoDB) – Select frameworks and architecture – digging into twitter API and RSS

  - First attempt at the UI. It will be polished and new features will be added throughout iterations.
  - Generate keywords from news stories and fetch relevant tweets. This is a necessary part of the entire system.
  - Display current news stories
  - Getting the web app hosted
- Second Iteration
  - Wordclouds generated from tweets related to news stories.
  - Influential tweets related to news stories.
- Third Iteration
  - Add the ability to analyse whether the tweets are positive or negative and present the aggregated result in the UI.
- Fourth Iteration
  - Categorise tweets and based on these categories categorise the news stories themselves. Add that to UI.
- Fifth Iteration
  - Fetch pictures relevant to news stories and present them in the UI.
- Sixth Iteration
  - Final application polish and validating if everything is really really working before final submission.

We were satisfied with this plan. We skipped Iteration 4 because we thought it doesn't align with our project aim, but apart from that, we followed the plan and made accountable progress along the way.

3.5 Testing

Testing our project was challenging. Since it is more of a real-time system that just analyses data we did not have any models to test. Nevertheless we needed a way to check if the program is working correctly.

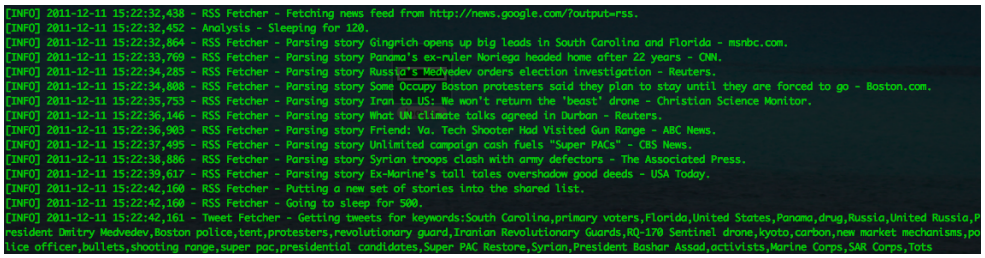

Figure 3.3: Customised logger.

3.5.1 Unit and Mock Testing

We used the PyUnit[17] module for incorporating unit tests in our code. An example of where we used this was in the RSS thread where we were testing functionality of all the functions written for this module. For this we had to create a mock rss fetcher object and run tests on it.

Having said this, it was very difficult to test some parts of the system as there is a lack of clear models, for example having unit tests for keyword generation or tweet fetching is difficult to implement as they use external APIs.

3.5.2 Logging and Debugging

Instead of using standard print statements to output the state of our system, which is composed of the RSS, Analysis and Twitter Thread, we incorporated a flexible event logging system.
We are able to leave our system and if an error occurs we can trace exactly which thread caused it and at which point in the program as opposed to the usual debugging. The logger was also incorporated so that we have different levels of importance in logging i.e. have logging for general information of what is going on in the system and logging for errors. We are also able to log messages to different output sinks including the console and files.

3.5.3 Unit and Mock Testing

We used the PyUnit[18] module for incorporating unit tests in our code. An example of where we used this was in the RSS thread where we were testing

[17] `http://pyunit.sourceforge.net/`
[18] `http://pyunit.sourceforge.net/`

functionality of all the functions written for this module. For this we had to create a mock rss fetcher object and ran tests on it.

3.5.4 UI Testing

To debug the UI, Google Chrome's Developer tools and Firebug were used. These tools were used to inspect the requests the UI issued, the network activities, debug layout and monitor performance.

3.6 Summary of each members contribution

We have tried to establish fair contribution by everyone by assigning to everyone tasks on Trello, and believe that it has worked well and ensured a rather fair individual contribution. As stated beforehand, we had a clear distinction between the people working on the frontend and backend.

3.6.1 Maciek Albin

Date	*Comments*	*Hours*
12/10/2011	Advanced network generation for follower/folowee relations.	3
20/10/2011	More work on the network generation.	4
28/10/2011	Initial internal API definition and basic implementation.	2
28/10/2011	Fixes to inter thread communication.	2
02/11/2011	Refactoring.	3
02/11/2011	Implemented keyword generation using Alchemy API.	2
03/11/2011	Implemented basic analysis thread.	4
04/11/2011	Assigning tweets to stories in analysis thread.	3
04/11/2011	Added short summaries and links to stories DB.	2
08/11/2011	Added wordclouds to the API.	1
10/11/2011	Refactoring.	3
30/11/2011	Fixed bugs in the logger.	2

3.6.2 Jamal Khan

Date	*Comments*	*Hours*
11/10/2011	Initial creation of the RSS Fetcher using python module feedparser.	2
31/10/2011	Keyword extraction completed for each story using Alchemy API.	4
4/11/2011	Implemented Flask Server creating API for the front end for the site as well as relevant database calls to fetch stories.	7
10/11/2011	Added functionality of getting stories by timestamp for the API.	2
25/11/2011	Completed flexible logger for each thread, so we can see exactly what is going on.	7
12/11/2011	Unit testing for RSS Fetcher completed.	4

3.6.3 Suhaib Sarmad

Date	*Comments*	*Hours*
27/10/2011	Resource research and UI mockup	4
28/10/2011	Basic grid UI proof of concept	3
09/11/2011	Implementation of basic jQuery Masonry (tiled) UI	2
10/11/2011	Create tiles in UI from server news API	4
11/11/2011	CSS/HTML Tile layout, split into title/summary, picture, sentiment, word cloud, tweet list	5
11/11/2011	UI debugging and cross-browser compatibility: works on all tested browsers except IE	2
12/11/2011	Added tweets to tweet list using Twitter REST API and keywords from server API, added custom jQuery scrollbars	1
12/11/2011	Added pictures to tiles using Google Images API and article titles	2
12/11/2011	Generating html5 word cloud from keywords from server API	3
19/11/2011	CSS3 transitions research and testing for big picture UI	2
01/12/2011	Automatic refreshing and fetching of new news articles from server	4
09/12/2011	Sentiment bar prototype	1

3.6.4 Rafał Szymański

Date	*Comments*	*Hours*
10/10/2011	Initial commit for the project. Basic implementation in Python of a script that given a keyword fetches the live streaming API for the given keyword and writes it to the Mongo database.	6
11/10/2011	Bug fixes and a short readme.	2
12/10/2011	Fetching the follower/folowee graph and saving to a file. Basic analysis with NetworkX.	3
20/10/2011	More working on generating the user graph.	2
21/10/2011	Big update. Gets headlines from Google news RSS, generates keywords using Termtopia, sets up twitter stream, saves to Mongo and repeats every 5 minutes.	7
01/11/2011	Improving the threading of the program - added condition variables and fixed previous threading bugs.	4
04/11/2011	Working on instant deployment environment using Flask, Nginx, uwsgi, and a couple other technologies. Took forever to do but afterwards, after every git push, the new version is live straight away.	10
07/11/2011	Improvements to the instant deployment - new git hook, and a control script that allows remote restarting and clearing of the database.	4
08/11/2011	Wordcloud generation. Goes through each tweet, finds the most occurring words and adds that data for each analysis period.	4
11/11/2011	Improvements to control script.	2
30/11/2011	Sentiment analysis for each period. Gets sentiment using an API and adds to Mongo.	5
04/12/2011	Top tweets based on the number of retweets.	2
04/12/2011	Added all my new data to the frontend API.	2

3.6.5 Sam Wong

Date	*Comments*	*Hours*
12/10/2011	Python development	1
02/11/2011	UI Prototype 1	3
09/11/2011	UI Prototype 2	4
20/11/2011	UI Prototype 3	5
21/11/2011	UI debugging	1
25/11/2011	UI features experimentation	1
01/12/2011	UI features refinement	1
01/12/2011	Auto-refresh	2
01/12/2011	infinite scroll proof of concept	3

3.7 Ethical and Environmental Impact of our Project

Ethical and environmental impact was a point of discussion in one of the previous reports, and we deem it interesting to add this discussion to this final report. We see the following possible ethical and environmental issues:

- All technologies that we are using are open source. Python has an OSI-approved license[19], our server is written using Flask which is BSD Licenced[20] and MongoDB is also open source[21]. The server is the widely used Nginx, which has a BSD-like license.
- We are storing tweets, which are not our property. We copy the whole tweet to our database, which originally is the property of the original author and of Twitter. This is not exactly a compromising situation, but it nevertheless should be considered from an ethical standpoint.
- We are possibly aggregating information and presenting it in such a fashion that could reveal new and interesting patterns - patterns that might be surprising or ones that others don't agree with. We would also probably need to state that the results are merely the results of our algorithmic analysis, and not our personal viewpoint regarding the current news story.
- In addition, the news stories we are aggregating from Google News may be biased as it is based on what Google believes to be the top stories, and other peoples point of view as to what news is top may differ around the world.
- Our project currently only works for just top 10 stories from google news, and the volume of tweets is already quite high (can be as high as 50 a second). This means that the database is growing large quite quickly. If this project were to be extended to support many more news services, it would have to be distributed over more machines, along with more database instances to support the larger volume of data. This would be something to consider if machines or EC2 instances were to be bought. Already, after running the product over the night, there was a notification from the VPS provider notifying about high disk IO rate. These type of problems could be classified as an environmental issues.

[19] `http://docs.python.org/license.html`
[20] `http://flask.pocoo.org/docs/license/`
[21] `http://www.mongodb.org/display/DOCS/Licensing`

Chapter 4

Validation and Conclusions

4.1 User Validation

From time to time we asked for feedback on the user facing products. This was done so that we could make design and functionality choices. We talked to our supervisor Jeremy Bradley, and he liked our user interface but wanted us to add pictures for each news story, we consequently went ahead with this and received very positive feedback.

4.2 Code Validation

We used Pylint4 to validate the Python code which we wrote. This was done to check for errors such as incorrect indentation, bad naming conventions for class names and functions, among others. Checking for errors and validating the UI side of the project was done using the W3 Validator at `http://validator.w3.org/`.

4.3 Conclusion

Chapter 5

Bibliography and Tools used

Taking into account the fact that this wasn't a research project requiring extensive external material, we do not provide a full bibliography, but provide links to sources we have used and consulted when working on this software engineering project.

- Backend Resources
  - `http://pyunit.sourceforge.net/` - Unit testing
  - `http://nginx.org/` - Backend server
  - `http://projects.unbit.it/uwsgi/` - For nginx to speak with python
  - `http://alchemyapi.com` - API for keyword extraction
  - `http://pypi.python.org/pypi/topia.termextract/` TermTopia's TermExtractor
  - `http://twittersentiment.appspot.com/` Twitter Sentiment Analizer
  - `http://mongodb.com` - Data store used
  - `http://www.logilab.org/857` - Pylint
- Frontend Resources
  - `http://validator.w3.org/` - HTML Validator
  - `http://jquery.com/` - jQuery
  - `http://masonry.desandro.com/` - jQuery Masonry Plugin

- Other Resources
  - `http://trello.com` - Online task board
  - `http://www.python.org/doc/` - Python documentation
  - `http://git-scm.com/` - Version control system used
  - `https://github.com/` - For storing code + wiki
  - `http://flask.pocoo.org/` - Python web framework
  - `http://news.google.com/` - News source
  - Programming Collective Intelligence, Toby Segaran, OReilly Media, 2007
  - Mining the Social Web, Matthew A. Russell, OReilly Media, 2011

Chapter 6

Appendix

6.1 Team Meetings

Date	*Venue*	*Subject*	*Attended*
14/10/2011	Lab Round Table	Choose which project to do	G
18/10/2011	Skypeland	Choose which project to do	G
21/10/2011	Lab Round Table	Discovered the original plan is infeasible due to constraints set by Twitter, draft backup plan	G
27/10/2011	Lab Round Table	Plan approved, Design Architecture	$G \setminus \{\texttt{Suhaib}\}$
3/11/2011	Lab Round Table	Progress Report (Backend, and UI Prototype I)	$G \setminus \{\texttt{Rafal}\}$
10/11/2011	Lab Round Table	UI Prototype II presentation, API requests and design	$G \setminus \{\texttt{Maciej}\}$
17/11/2011	Lab Round Table	UI Prototype III. Post-JB-meeting discussion	$G \setminus \{\texttt{Jamal}\}$
24/11/2011	Skypeland	Incremental improvements	$G \setminus \{\texttt{Sam}\}$
1/12/2011	Skypeland	Extensions ideas	$G \setminus \{\texttt{Suhaib}\}$
8/12/2011	Skypeland	Features prioritisation	G

$G = \{\texttt{Rafal}, \texttt{Sam}, \texttt{Suhaib}, \texttt{Jamal}, \texttt{Maciek}\}$