|  |
| --- |
| Individual Report |
| Big Data Visualization using Commodity Hardware and Open-Source Software – Streaming Data Sourcing Component |
| Course: ELEN7046 – Software Technologies and Techniques |

|  |
| --- |
| Kyle Trehaeven(Student Number 0602877N)  6-30-2016 |

# Abstract

A data sourcing application has been created in C# in the .net Framework to capture tweets from Twitter. The application leverages off of third party twitter libraries to source tweets relevant to the topic of American elections and South African elections, will verify certain data points provided by twitter and subsequently persist the data. The persisted data will then be available to be converted into a JSON text file and exported to the transformation component of the Big Data Visualisation solution.

Contents

[Abstract 1](#_Toc455214475)

[1. Introduction 3](#_Toc455214476)

[2. Background 3](#_Toc455214477)

[3. Requirements 4](#_Toc455214478)

[3.1. Functional 4](#_Toc455214479)

[3.2. Non-functional 4](#_Toc455214480)

[4. Approach 4](#_Toc455214481)

[5. Challenges 5](#_Toc455214482)

[General Data Sourcing Challenges 5](#_Toc455214483)

[Streaming Data Sourcing Challenges. 5](#_Toc455214484)

[6. Design Overview 6](#_Toc455214485)

[6.1. Conceptual 6](#_Toc455214486)

[6.2. Technologies Used 6](#_Toc455214487)

[6.3. Class Diagram 7](#_Toc455214488)

[6.3.1 MainWindow 7](#_Toc455214489)

[6.3.2 TwitterStreamConnectionManager 7](#_Toc455214490)

[6.3.3 TwitterMessageHandler 7](#_Toc455214491)

[6.3.4 MongoDBManager 7](#_Toc455214492)

[6.3.5 TweetModel 8](#_Toc455214493)

[6.4. Application Process Flow 8](#_Toc455214494)

[6.4.1 Application Initialise 8](#_Toc455214495)

[6.4.2 Connect 8](#_Toc455214496)

[6.4.3 Disconnect 8](#_Toc455214497)

[6.4.4 Data Dump 8](#_Toc455214498)

[7. Testing 9](#_Toc455214499)

[8. Recommendations 9](#_Toc455214500)

[9. Conclusion 9](#_Toc455214501)

[References 10](#_Toc455214502)

# Introduction

Arguably the least alluring aspect of a Data Transformation process, data sourcing is however unarguably a very import component. Whilst in many contexts large samples of data are already availed to potential investigators, often such pools either lack certain key data components that must be references and linked to the initial data pool’s entries; or such data pools may not even exist. In the context of this particular endeavour, a stream of data was available, however it accessible in a less than direct manner, and in such a context the streaming data source app’s primary purpose was to behave in a fashion not dissimilar to that of an aggregator of sensor data.

Data sourcing as a piece of the system had the following responsibilities:

* Sourcing data from Twitter, with the relevant keywords used as a filter.
* To be prototyped as quickly as possible to determine the extent of the provided data and the reliability of the offered data.
* To be a stable source of data, so as to minimise gaps in the data.
* To prepare the data for processing on an “as received” basis where it was best suited to be cleaned.
* To provide a single format to feed into transformation processes.

The standalone application operated on a Personal Computer with an ADSL internet connection, and whilst a few drops in ADSL service were noticed, there was no poor performance from either the line or the machine.

# Background

The formative stages of the project involved each of the components of the system independently develop a proof of concept to explore their proposed technology stack to address the problem. Data sourcing was split into two pieces: a batch process that would source historical data, and an online process that would extract from a streaming data source.

The decision to split the data sourcing was made based on a few key points of interest. In ascending order of importance:

* Data sourcing was considered the easiest component to fragment. It permitted a far greater degree of work allocation and reconsolidation than other components, particularly prior to an agreed upon technology stack. The data sourcing could be completely different technology but distil down to a single JSON output, whereas different visualisations and transformation processes would be more difficult to re-merge.
* An understanding of the type of data that was available to leverage was critical, and the sooner such information was available the better. This is both in the context of what the interface claims to expose, and in the context of how consistent that data may be. By allocating two resources to sourcing some preliminary data, the capacity to ensure an initial prototyping sample data set was prioritised at that stage of the endeavour.
* Twitter imposes a limit on messages solicited, and this can be mitigated by utilising more than one IP address and twitter account.
* The premise of the project was to illustrate that any open source technologies can be involved in the low cost solution to big data processing needs, and by demonstrating two solutions to at least one aspect of the problem, it can be demonstrated that different means of leveraging the technology have been explored.

# Requirements

The following is a list of the key requirements.

## Functional

* Build a solution that can reliable interface with Twitters streaming data source.
* Subscribe to the streaming data source with the appropriate filters (election key words).
* Verify the integrity of Location data, and interface with Google Maps to consolidate any invalid data.
* Persist the data in a data storage solution for data retention and lookup.
* Export the data into Hourly-time stamped JSON files for FTP pushing.

## Non-functional

* The solution should be stable and capable of handling any exceptions such as internet connectivity loss or twitter stream limits reached.
* Retention should retain as much of the initial tweet data as possible, and any marking-down should only occur during the extract process to retain the integrity of the data.
* The solution should present an interface to initiate and terminate the stream, with console outputs for Stream information, Database information and General system information for monitoring.

# Approach

The approach was to use a standalone Windows Powered Forms (WPF) application, developed in the .net framework using C#. This approach would allow a rapid development that could provide prototyping data as soon as possible, as well as allow a much more fluid development cycle rather than deploying a service. A Forms application could also be used to generate a more integrated UI, and meant that more time could be focused on the solution rather than working on periphery niceties. Alternative options explored included running the application as a service in windows, and while there was no definitive problem with this, it would mean that exposing configuration options and message logs would either require developing a WPF that integrated with it anyway, or a Web application that integrated with it; both of which were excessive.

The solution was hosted on a home pc with an ADSL connection. Ideally the solution would run on a PI to further promote the premise of a solution running on low cost hardware, however the hardware was not available. The idea to run the application on some cloud computing service was briefly entertained before realising that such an action would defeat the premise of the project.

The solution leveraged a third party library that behaved as a wrapper for the Twitter API called TweetInvi. Other wrappers exist but this particular option seemed the most refined. Likewise the official Google Services library was used to interface with Google Maps. There was no need to leverage other libraries as they all provided value add features that weren’t necessary for the solution.

Data persistence used Mongo DB, with the official Mongo DB C# driver. This was chosen due to the driver’s availability on the PI, even though at a later stage it was decided to revert to raw JSON for transfer. Mongo was still the best data permanence option as it persists data in a “JSON Like” format, whereas a relational table database could either be an expensive commodity solution or would require reforming the data into a JSON format.

# Challenges

Challenges can be divided into two broad categories:

## General Data Sourcing Challenges

From the perspective of input data in general, some challenges were immediately encountered.

* Data needed to be rapidly sourced to identify the data that could be used as quickly as possible. In order for decisions regarding the final product to be made, an understanding of the data available for use needed to be ascertained.
* The validity of the data had to be checked and, if necessary and possible, repaired. This particular challenge was best demonstrated by the geo-location data, which was very unreliable, but could be accommodated for by leveraging off of Google Maps.
* Format of sourced data had to be seamlessly consolidated. In order for two data sourcing processes to be used, the final output from each of them had to adhere to a single, unified standard of output.

## Streaming Data Sourcing Challenges.

* Connectivity could drop, and no monitoring or administrative powers could be present. A robust solution must be in place in the event of long periods of down time, and in the event of a drop in connectivity, a connection must be re-established as soon as possible.
* Stream Limit overflows could occur, particularly in reaction to significant news events. Any overflows of the limit should be handled, and when the limit period has expired, the solution should continue to listen for relevant tweets.

# Design Overview

## Conceptual

From the perspective of workflow, the application is required to listen for data provided by twitter, inspect and verify the data sourced, then persist the data. Once called upon to unload a range of data, the application will then generate JSON files containing the persisted tweets batched into hourly windows. With these behaviours in mind, understanding the high level view of what the design should incorporate becomes clearer.

A component must be responsible for the interaction with the Twitter stream, handling the incoming tweets as they arrive. This component must be capable of connecting to the stream, disconnecting from the stream, and invoking any data verification that must be done. This component can delegate the data verification to another component that focuses on any tasks related to data integrity verification.

Another component will be used to manage the Tweet persistence, and will be invoked by the stream managing component to persist data, and will also be invoked by the user when a data transfer needs to occur.

Finally, as a means by which to internally represent tweets between these components, a simple model for a tweet can be used to encapsulate all information sourced from Twitter and act a single view of each object whilst it is still within the domain of the application.

## Technologies Used

The application was developed in C#, in the .net Framework 4.5, using the Visual Studio Express IDE. The solution was built in a standalone WPF application.

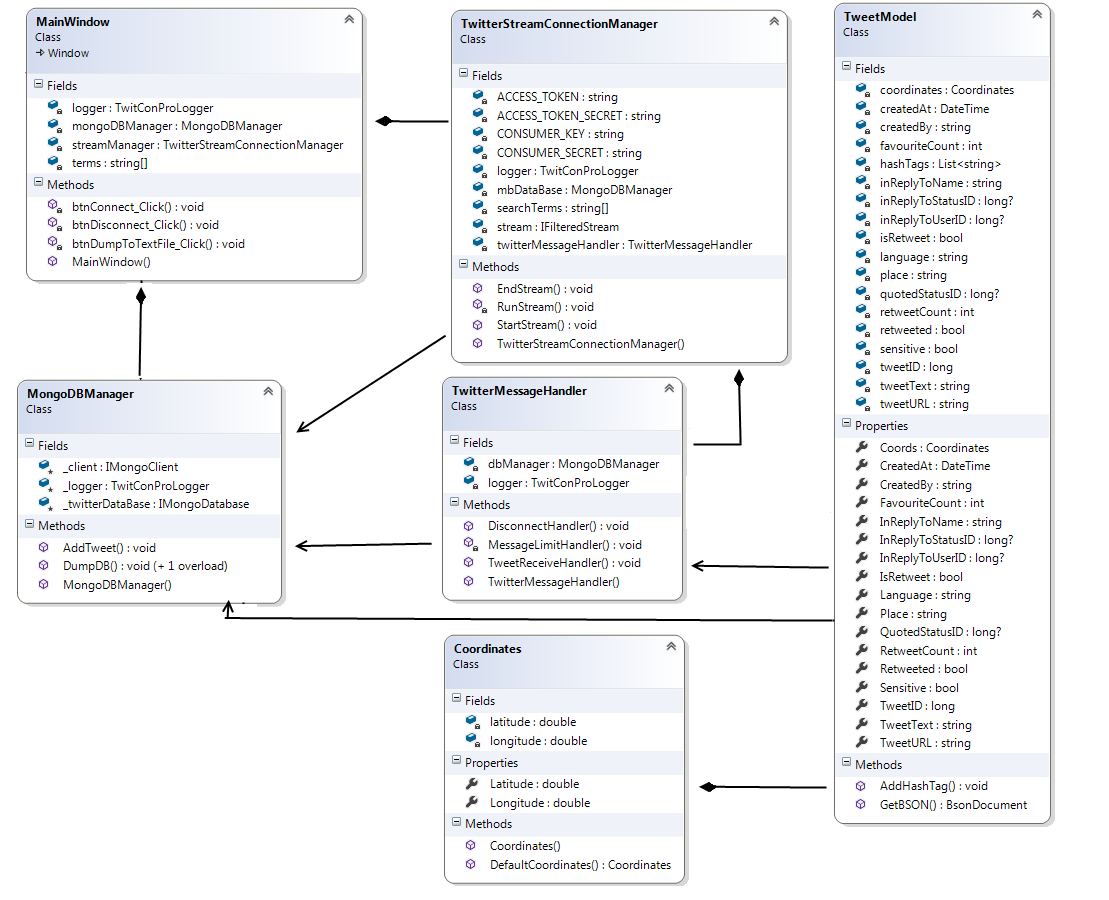
Using the Nuget package manager, an external third party library called TweetInvi was sourced to behave as a wrapper to the Twitter API.

The Nuget package manager was also used to source the Google Services API, which was used to interface with Google Maps to source geolocation data for tweets in the event that any sourced tweets only had city names instead of raw co-ordinates.

Mongo DB was used as a database solution, with the official Mongo DB C# driver sourced from Nuget to interface with the Mongo DB instance.

Raw JSON Files were generated to export to the Transformation solution.

## Class Diagram



### 6.3.1 MainWindow

Main Window is the class behind the user interface. It houses all the event responses to the buttons, and it will be the parent which is composed of the sub components.

### 6.3.2 TwitterStreamConnectionManager

This component will handle the connection to and disconnection from the Twitter Streaming Data source. It has the authentication credentials known, and is associated to and composed of the TwitterMessageHandler class. It will also store search terms used as a filter for the Twitter stream, and will have an association with the MongoDBManager to provide to the TwitterMessageHandler object.

### 6.3.3 TwitterMessageHandler

The Message Handler class will be created on stream connection. It will listen to the stream and verify the validity of location data, and source more reliable data from google maps if needed. It has a reference to the MongoDBManager for object persistence.

### 6.3.4 MongoDBManager

The MongoDBManager is responsible for interfacing with the Database instance, and will be invoked to persist or recall tweet information.

### 6.3.5 TweetModel

The TweetModel is a simple class used to encapsulate information sourced from the TwitterMessageHandler. It is composed of a Coordinates class, which is just a simple longitude and latitude class.

## Application Process Flow

### 6.4.1 Application Initialise

On initialisation, the application will create a Stream Connection Manager and Database manager object. These objects have hard coded sentinel values that are required for their generation, and inversion of dependency is earmarked as a recommended feature. The Mongo DB Manager requires a Mongo DB instance to be running, and will automatically look for a database by the name of “tweetDB2” to begin appending to, and will associate all tweets with a collection called “tweets”. The Stream Connection Manager has sentinel values for the access tokens issued by Twitter, and will prime the TwitterStreamConnectionManager for connection.

### 6.4.2 Connect

On connection, the TwitterStreamConnectionManager will initiate a connection with twitter using provided search terms as filters. It will create a TwitterMessageHandler object which will handle any received tweets and subscribe the TwitterMessageHandler to the event stream. Any tweets the TwitterMessageHandler receives will be verified then submitted to the MongoDBManager for persistence.

### 6.4.3 Disconnect

On Disconnect, the TwitterStreamConnectionManager will terminate the connection.

### 6.4.4 Data Dump

When the dump command is issued with a provided directory, the MongoDBManager will dump all the tweets into the provided directory in a JSON format. The dump will consist of multiple files, each named with a Year-Month-Day-Hour timestamp.

# Testing

Each component was individually tested in a shell “proof of concept” sandbox to understand and verify the concepts being used. Once each component had been assessed as functional, the tool was was piloted overnight to verify that its running conditions were acceptable. Due to the unilateral nature of the application, use cases were very simple and it was simply a matter of verifying the following cases to be true:

* Tweets matched search conditions.
* Location Updates happened.
* Connection Exceptions were handled and logged to the applications text output.
* Data was persisted.
* Data could be retrieved.

All these scenarios were monitored and considered acceptable.

# Recommendations

As highlighted in the design, refactoring the Data Validation into another class would have been better decomposition. Similarly, externalising search terms and access tokens for twitter would of made the application more robust and is a relatively low hanging fruit for extension.

Geolocation caching would have been an interesting update to apply, and would limit calls to the google API. The application could build up an understanding of Country, City and County names and the associate coordinates for each, and lookup internally.

While it wasn’t necessary to have a timed automatic dump to the FTP site for this project, the capacity to automatically dump would be a great feature.

There is a logging mechanism not detailed in this solution, however having remote messaging in the event of significant concerns would be a good feature to consider.

# Conclusion

The application developed achieves its goal acceptably. It sources the required data, refines it as is expected, and persists it, awaiting an invocation to provide it to transformation processes. It maintained a connection for two weeks with no issue save for a single disconnect caused by infrastructure (from which it recovered on its own), and two weeks of twitter data was solicited with no intervention save the occasional monitoring its logs. While there will always be room for improvement, the ultimate criteria by which is should be assessed is by whether or not it achieved its goal, and this it managed comfortably.

# References

Tweetinvi Twitter Wrapper: https://github.com/linvi/tweetinvi/wiki

Google Maps Geocoding API Documentation: https://developers.google.com/maps/documentation/

Mongo DB General Tutorials: https://docs.mongodb.com/manual/tutorial/

Mongo DB C# Driver: https://docs.mongodb.com/ecosystem/drivers/csharp/

Scala Language Tutorials: http://www.tutorialspoint.com/scala/

# Appendix A: Timesheet

