

**Analysis and Visualisation of Twitter Data for Growth Hacking
using the Raspberry Pi**

Richard Flanagan

A00193644

BSc. Software Design 2016

Athlone Institute of Technology

Declaration

I hereby certify that the material, which is submitted in this report towards the award of BSc. Software Design, is entirely my own work and has not been submitted for any academic assessment other than part fulfilment of the above named award.

Future students may use the material contained in this report provided that the source is acknowledged in full.

Signed

Date

Abstract

Many start-ups and technology companies have started to adopt a new form of marketing strategy dubbed 'Growth Hacking'. Growth hacking takes advantage of modern internet technologies to achieve a self-perpetuating marketing campaign as quickly and inexpensively as possible. Such a campaign can be achieved in Twitter though the use of well-crafted and well-timed tweets targeted at the most responsive audience.

This projects main output will be an online data analysis service run entirely on a Raspberry Pi, marketed at this new generation of Growth Hackers. Users will be able to view the impact of their marketing strategy, and make changes to maximise impact. It will also explore the suitability of the inexpensive Pi in the field of data analysis and growth hacking.

This service is targeted towards users developing or updating their marketing strategy. Locations, times, and volume of audience participation is important to show to the marketer. An ability to access data from anywhere in the world, at any time, as well as filtering options to reduce your area of interest both make sense for a market to have. These feature drove the development of my product, leading to design choices like providing this software as an online service.

Results tend to show that the Pi is capable of small levels of data analysis. Large datasets prove too much for this machine. The analysis-as-a-service model used by the service requires fast response times which the Pi cannot provide. However, given powerful server hardware, this service is most definitely a viable and marketable product, one that could be used by marketing teams worldwide.

Acknowledgements

I would like to thank my supervisor Karol Fitzgerald for guiding and supporting me throughout my project.

I would also like to thank both Karol Fitzgerald and Dr David Scott for reviewing my work and providing feedback.

Finally, I wish to thank all the individuals who helped and encouraged me throughout the months of research and implementation.

Table of Contents

Declaration.....	ii
Abstract.....	iii
Acknowledgements.....	iv
Table of Contents	v
List of Tables	vii
List of Figures	vii
Chapter 1: Introduction.....	1
1.1 Introduction	1
1.2 Rationale.....	1
1.3 Research Aims and Objectives	2
Chapter 2: Background Research.....	3
2.1 Introduction	3
2.2 Hardware	3
2.3 Raspberry Pi Operating System.....	4
2.4 Development System	5
2.5 HTTP Server	6
2.6 Web Framework and Front-end	6
2.7 Persistent Back-end Server	7
2.8 Maven Dependency Management	8
2.9 Database.....	9
2.10 DNS	10
2.11 SSL/TLS and HTTPS	10
2.12 Traffic Routing and Port Forwarding	12
2.13 Twitter	12
Chapter 3: System Design.....	14
3.1 Introduction	14
3.2 Requirements.....	14

3.3 Architecture	16
3.4 Design	18
3.5 Implementation	24
Chapter 4: Testing and Evaluation	32
4.1 Introduction	32
4.2 Testing	32
4.3 Evaluation	32
Chapter 5: Conclusions	33
5.1 Introduction	33
5.2 Results	33
5.3 Reflection	34
5.4 Recommendations	35
References	36
Glossary	41
List of Abbreviations	42
Appendix A:	44

List of Tables

Table 2.2.1: Comparison of Pi Models	3
Table 2.3.1: Raspbian versions available in January 2016	4
Table 2.5.1: Apache Modules.....	6
Table 2.7.1: Server dependencies	8
Table 2.9.1: Database options	9
Table 2.13.1: Comparison of the two Twitter APIs	12
Table 2.13.2: Different data streams supplied by Twitter	13
Table 3.4.1: Server URL paths.....	19
Table 5.2.1: Page load times for a selection of data sets	34

List of Figures

Figure 2.4.1: My Debian8 virtual machine running in my Windows7 PC	5
Figure 2.10.1: The site URL	10
Figure 2.11.1: A valid certificate for my domain. IP addresses censored.....	11
Figure 3.3.1: Data analysis flow	17
Figure 3.3.2: Data analysis flow converted into system architecture	17
Figure 3.4.1: A diagram displaying the design of the operating environment.	18
Figure 3.4.2: Server low level architecture	20
Figure 3.4.3: Basic layout of a Django project.....	21
Figure 3.4.4: My client Django project layout	22
Figure 3.4.5: A mock-up of the collection creation page	23
Figure 3.4.6: A mock-up of the analysis page	23
Figure 3.5.1: Raspberry Pi and external hard drive	24
Figure 3.5.2: Client: The server status page	26
Figure 3.5.3: Client: The create collection page	27
Figure 3.5.4: Client: The existing collections	27
Figure 3.5.5: Client: Viewing a collection.....	28
Figure 3.5.6: Client: Choosing analysis for a collection	28
Figure 3.5.7: Client: The about page	29
Figure 3.5.8: Client: Visualisation showing the location of a selection of tweets from the collection	29
Figure 3.5.9: Client: Visualisation showing the tweets-per-country heat distribution in a collection	30

Figure 3.5.10: Client: Visualisation showing the amount of tweets per hour in a collection	30
Figure 3.5.11: Client: Visualisation showing the hashtag breakdown of all the tweets in a collection	31

Chapter 1: Introduction

1.1 Introduction

This project explores the domain of data analysis and visualisation by creating an analysis and visualisation service using Twitter data. This service provides a platform for users pursuing a self-perpetuating Twitter campaign. They can use the service to discover current local or global Twitter trends and track the popularity of particular keywords tweeted across the world, and from that, view the impact of their social media advertising campaigns. A Raspberry Pi forms the operating environment of this system.

1.2 Rationale

Many start-ups and technology companies have started to adopt a new form of marketing strategy dubbed 'Growth Hacking'. Growth hacking takes advantage of modern internet technologies to grow a customer base as fast as possible, and as inexpensively as possible. A successful campaign using this technique is often called 'going viral', that is, reaching a point where your campaign gains enough momentum to continue to grow without intervention or investment.

"The end goal of every growth hacker is to build a self-perpetuating marketing machine that reaches millions by itself." [1]

A self-perpetuating campaign can be achieved in Twitter through the use of well-crafted and well-timed tweets targeted at the most responsive audience. This service acts as a companion tool to such a campaign. It allows the user to view the impact of their marketing campaign, and observe if or where they have achieved viral status. They are able to measure and quantify the success of their campaign, view statistics by volume, location, and rate-over-time. This gives the advertiser the feedback required to adapt their campaign to achieve greater growth.

This product is envisioned to be available to potential customers as a service. In other words, the service is available online, and is accessible without significant investment in hardware or technical expertise by the end-user. Software-as-a-service business

models have been experiencing a period of growth over the past few years [2], as cloud hosting and distribution becomes more accessible and less expensive compared to alternatives. Quick access to users' analytics data is easily facilitated by this business model, allowing them to update their marketing campaigns at any time, in any place. It is aimed at professionals and semi-professionals, but can be used by amateur users just as readily.

1.3 Research Aims and Objectives

One of the technical goals of this project is to develop a working and eventually marketable service to allow social media advertisers and marketing managers (both professional and amateur) to view the impact of their Twitter advertising campaigns. In doing this, experience with the tools, technologies, and domains relevant to the project is acquired.

Such topics include:

- Django web framework
- Java Enterprise Edition
- Apache2 & Tomcat7
- MongoDB
- Data Analysis techniques and practices
- Charting and visualisation
- JavaScript and accompanying web technologies
- Maven dependency management
- Linux operating system
- DNS, SSL, routing and similar web and networking technologies

As a secondary aim, the system is to test the capabilities of the Raspberry Pi 2 B by running on this limited hardware. This is an investigation to see whether data analysis can be carried out on a smaller scale without requiring large hardware clusters for processing huge data sets. It is also an opportunity to discover the suitability of hosting web services on a Raspberry Pi.

Chapter 2: Background Research

2.1 Introduction

Data Analysis comprises of the “techniques used to describe and illustrate, condense and recap, and evaluate data” [3]. This project deals with the description and illustration of data, and exploring the organisation and visualisation of this data in order to benefit users pursuing a viral advertising campaign.

Research in this project is mainly focused around the design and implementation of this system. Research on Linux, various web technologies, and tools such as Django and Apache are essential in order for this project to succeed.

2.2 Hardware

It was intended for the system to operate on some version of a Raspberry Pi computer. A Pi has greater support and resources compared to other micro-computer alternatives. Chief among these advantages is the Raspbian operating system available for all Raspberry Pi systems. Raspbian will be discussed in a later section.

After some research, it was concluded that the Raspberry Pi 2 Model B was the most suitable computer for this system. This was the most powerful model of the Pi available at the time. As it would be acting as a server, the computer required an Ethernet connection, and available RAM would be a priority. The quad core processor would also be a bonus when processing the data.

Table 2.2.1: Comparison of Pi Models. [4]

Model Name	Model A+	Model B	Model B+	2, Model B
Processor Cores	1	1	1	4
Processor Speed	700 MHz	700 MHz	700 MHz	900 MHz
RAM	256 MB	512 MB	512 MB	1 GB
Ethernet	No	Yes	Yes	Yes

2.3 Raspberry Pi Operating System

Raspbian [5] is a fork of the popular Debian Linux distribution [6]. Debian is noted for its stability and package ecosystem. Packages available through the Debian repositories are usually highly compatible with both the operating system and other installed packages. Compatibility and dependency availability is handled by the Debian package manager dpkg [7].

Raspbian Jessie Lite [sic] forms the operating system. Jessie is the latest major Debian release, meaning it is using the latest security and compatibility patches with the most up to date software packages for Debian systems. The Lite version is a barebones, terminal-only distribution. It does not contain a graphical user interface (GUI), and can be used remotely without the need for a monitor plugged into the Pi, or X11 server running to transmit the desktop.

Table 2.3.1: Raspbian versions available in January 2016

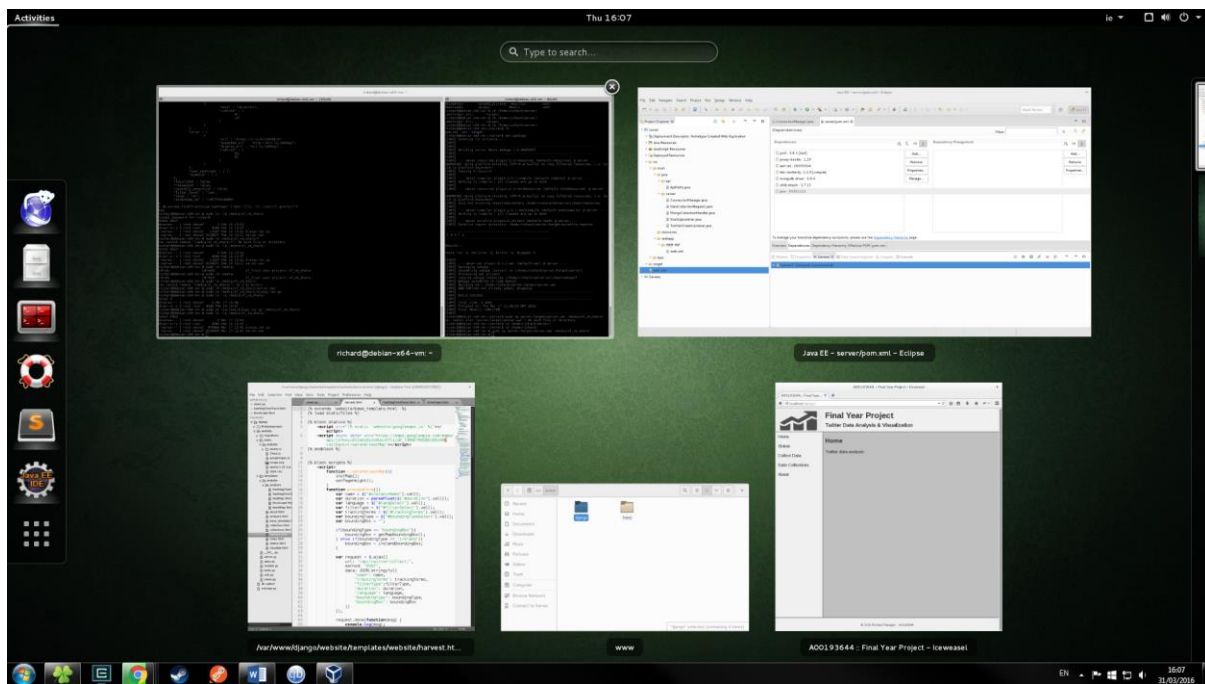
Name	Debian Version	Notes
Raspbian Wheezy	7.x	Out of date kernel No working MongoDB package Unnecessary GUI packages
Raspbian Jessie	8.x	Unnecessary GUI packages Higher system requirements
Raspbian Jessie Lite	8.x	No unnecessary packages Lower system requirements Headless

2.4 Development System

While the “live” or deployed system will be operating on a headless Raspberry Pi, developing on such a system would prove to be prohibitively difficult. Instead, development takes place in a virtual machine with a Debian image. Raspbian is a port of Debian, so this will protect against most compatibility related issues that may have been encountered otherwise.

Using VirtualBox [8], development in the Debian virtual machine can happen almost as if it was my primary operating system. From inside the image, the developer can access the host computer, the local network (important for accessing the Pi), and the internet, all from the virtual machine. The system itself works the same way on both the image and the Pi. One of the only noticeable difference is that the Pi provides external access from the web, while the virtual machine does not.

Figure 2.4.1: My Debian8 virtual machine running in my Windows7 PC



2.5 HTTP Server

A HTTP server is required in order to serve the front-end website to visitors, and to handle the API requests. The choice of software was largely between Apache HTTP Server [5] and Nginx [6].

Apache is the most popular HTTP server used today, and is estimated to serve about 50% of all active websites in the world [7]. It supports all of the features that are critical to the implementation of the system. These features are outlined in table 2.5.1.

Table 2.5.1: Apache Modules

Functionality	Module Name	Documentation
Proxy and Reverse Proxy	mod_proxy	[8]
WSGI support (Web Server Gateway Interface)	mod_wsgi	[9]
SSL and HTTPS support	mod_ssl	[10]
URL redirection and mapping	mod_alias	[11]

Nginx also supports these features. However, Apaches popularity means that there is a greater amount of references, learning resources, and examples available online, which would be invaluable to me. This is why Apache was chosen as the HTTP server.

2.6 Web Framework and Front-end

A web framework is also employed when implementing the front-end. Frameworks have several advantages, not limited to the following:

- provides templates that prevents repeating code
- reducing time spent implementing simple, already supported functionality
- making development both easier and faster

Using a framework allows developers to work on more important areas of their project. They also offer libraries of community-built plugins. Django [15] handles constructing the front-end web pages and responding to page requests, and jQuery [16] streamlines DOM manipulation in those web pages.

Django is an open-source high-level Python Web framework. It supports features such as in-built MVT (Model View Template) design pattern, relational database support, custom URL patterns, administration panels, HTML templates, and a solid template scripting language to dynamically populate HTML files. These features are beneficial, and facilitate development with as little implementation overhead as possible. [17]

jQuery describes itself as a small, fast and feature-rich JavaScript Library. It makes things like HTML document transversal and manipulation, event handling, animation, and Ajax much simpler. [16]

2.7 Persistent Back-end Server

A persistent back-end process manages the stream of data arriving from Twitter. This process needs to be separate from the front-end, so that it is available for use on-demand and for long periods of time. It also requires support for parallel processing. A need for parallel processing means that Python, which is normally locked to a single thread, will not be appropriate.

Java supports multi-threading, and its robust garbage collection strategy allows a java process to operate for long periods of time. As the front-end client communicates with this server process through the web, a REST enabled framework like Java 1.8 Enterprise Edition [18] with Jax-RS [19] was essential. This combination makes a RESTful service in Java possible. In other words, the client can communicate with my process using a series of custom HTML requests, which is proxied through my Apache HTTP daemon to the Java process.

The back-end server process uses a Java Servlet container to run. Apache Tomcat [20], a popular and open source servlet container is used to fulfil this requirement. Tomcat runs as a process on the Pi, and loads any provided Java EE module. All

requests to Tomcat are received by the Apache HTTP server, and subsequently routed to Tomcat using the mod_proxy module.

Several Java libraries are also required as dependencies by the back-end server, such as those in table 2.7.1.

Table 2.7.1: Server dependencies

Name	Use	Reference
mongodb-driver	Connect to the MongoDB instance	[21]
hbc-twitter4j	Hose-bird Client & twitter4j Connects to the Twitter data stream	[22]
slf4j-simple	Simple Logging Facade for Java Application logging	[23]
json	JSON parsing and encoding	[24]
jersey-bundle	Contains Jax-RS and other requirements	[25]
asm-all	Jersey dependency	[26]

Apache Maven [27] manages these dependencies and packages the project into the war file required by Tomcat.

2.8 Maven Dependency Management

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information [27]. In other words, a POM file defines various operations which Maven must complete during each project build cycle. Users can specify dependencies which are to be downloaded by Maven from their central repositories. They also use Maven to automatically run tests, build a war file for Tomcat, and generate documentation for the project.

Maven is a development tool, and is only installed on the development machine. It is not a part of the Raspberry Pi system, but is still a technology that is desirable during the development process of the project.

2.9 Database

A database allows the system to store data received in the Twitter stream. It can then retrieve the data when needed in order to perform analysis on it, or to display it to the user. The choice of database came down to one that was either a relational or NoSQL.

Table 2.9.1: Database options

Name	Type	Notes
MySQL [23]	Relational	<ul style="list-style-type: none">- Does not match twitter data format- Easily integrates into Django- Supported by Raspbian distros
MongoDB [24]	NoSQL - Document	<ul style="list-style-type: none">- BSON storage format matches Twitter data format (JSON)- No Django integration- No pre-Raspbian Jessie support
Cassandra [20]	NoSQL - Column	<ul style="list-style-type: none">- Experimented with by Twitter, proposed to store tweets-Supported by Raspbian distros

A NoSQL database called MongoDB forms the database of this system. Some of the reasoning behind this choice is influenced by the data structure and storage format used by MongoDB. MongoDB records data in a document structure, analogous to the JSON format used by the Twitter streaming API. The documents are then stored on disk using the BSON format, which is the binary form of JSON. In other words, it can receive and store any data I sent by Twitter with very little processing or overhead, as the format of the incoming data matches the storage format.

Unfortunately, unlike relational popular databases like MySQL, MongoDB does not have official integration into Django's MCP design pattern. Django cannot refer to a MongoDB document using a Model object, like what would happen with a MySQL row. To get around this, the client depends on a Python module called PyMongo [26] to interface with my database.

2.10 DNS

DNS stands for Domain Name System. It is a system for naming computers and network services into a hierarchy of domains. It can be used to locate computers and services through user-friendly names. Names and IP addresses of the computers are saved in a public DNS server, and can be accessed by anyone. [15]

Every online service, such as the one being provided by this system, requires a publically accessible address. This allows the user to access the service from anywhere in the world (as long as they have a working and uncensored internet connection!). In order to satisfy this requirement, a DNS service is employed. Duck DNS provide a free, minimum hassle DNS service to their users [16]. Using this, the server has the public URL as in figure 2.10.1:

Figure 2.10.1: The site URL

- <https://flanagan-server.duckdns.org/djano/>

As the server is not online 24 hours a day, the link in figure 2.10.1 may not work. Duck DNS requires a script on the server to broadcast a DNS advertisement to the DNS server every few minutes. This advertisement is broadcast using a shell script and a cron job. If the server is offline, or the system fails to execute the scripts, this broadcast will not occur, and the users DNS request will fail.

2.11 SSL/TLS and HTTPS

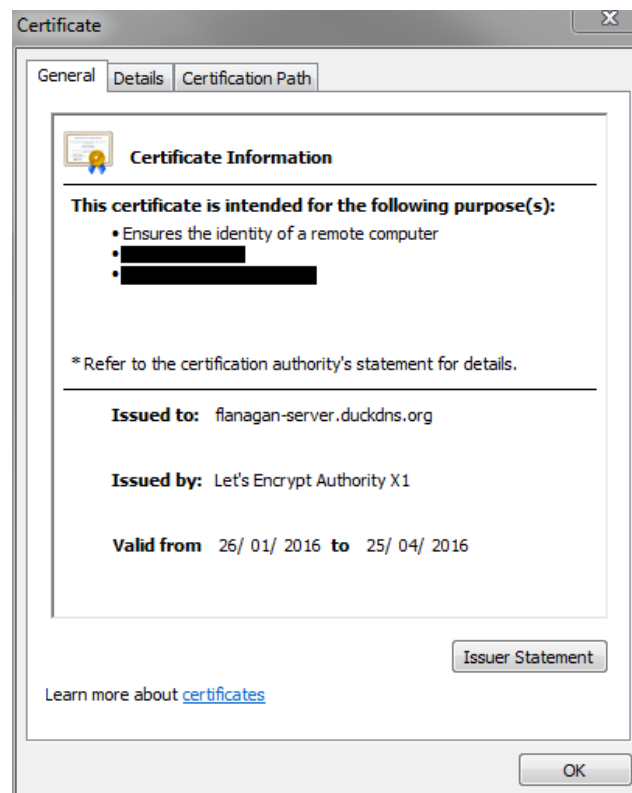
SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a browser [15]. TLS is the more modern

replacement for SSL, and is often referred to as SSL 3.1. TLS is based on SSL, but can be stepped down to SSL if required [16]. HTTP traffic transmitted through an encrypted SSL/TLS connection is known as HTTPS traffic.

As the service is hosted on a personal server which is open to the internet, security is an issue. In order to increase the security of the service, HTTPS and SSL/TLS encryption is enforced on the entire domain. In other words, all unencrypted HTTP connections will be forced into using an encrypted HTTPS connection. This has little negative impact on the end user, aside from a negligible increase in page load times due to the required encryption, decryption and SSL/TLS handshakes. On the other hand, it increases user safety by preventing man-in-the-middle attacks and gives the user peace of mind when they see the green padlock in their browser window.

HTTPS validation is provided for free by the new Let's Encrypt Certificate Authority [15]. They provide a script which generates SSL/TLS keys and authenticates the keys against their own Certificate Authority. Let's Encrypt is supported by Mozilla and the Electronic Frontier Foundation, as well as many other major web organisations.

Figure 2.11.1: A valid certificate for my domain. IP addresses censored.



2.12 Traffic Routing and Port Forwarding

By default, all incoming connections to most modern home routers are normally rejected by the router (with a few exceptions, such as for remote maintenance or configuration). In order to allow traffic to reach the Pi, SSL, HTTP, and SSH traffic to the router need to be routed to the machine. This required changing configurations on the router. A static IP is assigned to the Pi to allow developers or administrators to SSH into the machine remotely with minimal hassle.

2.13 Twitter

Twitter provides two types of API: the RESTful API and the Streaming API [38]. A RESTful API allows users to retrieve archived data from the past, and provides the user with an interface to integrate Twitter functionality into their application or website. This differs from the Streaming API, which provides a live stream of tweets from across the globe to the user.

Table 2.13.1: Comparison of the two Twitter APIs

Name	Notes
RESTful API	<ul style="list-style-type: none">- Request based- Paginated batches of tweets- Historic/archived data- Advances search queries
Streaming API (Public Stream)	<ul style="list-style-type: none">- Connection based- Data rate limits apply (~1%)- Limited connections allowed- Live feed of data- Filter by term, language, location

Twitters streaming API can be further broken down into the categories outlined in figure 2.13.2. These categories cater for different types of users, and return data from different sources. The back-end server connects to the public stream. While the

firehose would be better for a service such as mine, the public stream is more practical for a proof of concept. It is also unavailable for public use, and requires authentication by Twitter.

Table 2.13.2: Different data streams supplied by Twitter

Name	Description
Public Stream	A stream of all public tweets worldwide. It is limited, sending about 1% of all global tweets to the receiver, but can be filtered.
User Stream	All new tweets and events from one specific user.
Site Stream	Organisation orientated stream allowing a large number of tweets from specified users to be received. This is in closed beta and currently inaccessible.
Firehose	A full stream of all public global tweets. This is an enterprise level paid service.

Twitter requires application developers who wish to use the Twitter APIs to register their applications. This can be done on Twitters app portal [39]. After registering your app, you get a set of keys which will authorise you when connecting to the API.

Chapter 3: System Design

3.1 Introduction

This section will outline the requirements and design of my system. Some sections are divided into three sub-sections. This corresponds to the three main components of my system: the environment, the server, and the client.

3.2 Requirements

This section outlines the requirements for the system. System requirements are outlined in a simple format. Each requirement has a unique number corresponding to both the component and the requirement. This number is accompanied by a description of the requirement.

3.2.1 System requirements

1001: The system shall be compatible with Linux Debian systems.

1002: The system shall be able to run on a Raspberry Pi 2 Model B machine.

1003: The system shall be accessible from the internet using a URL.

1004: The system shall use SSL to encrypt all network traffic.

3.2.2 Server Requirements

2001: The server shall allow one active stream connection at a time.

2002: The server shall be able to connect to the Twitter filter data stream and retrieve data based on information provided in the active connection.

2003: The server shall store retrieved data in a MongoDB collection, which is named after the active collection.

2004: The server shall allow end the active stream when it has met its duration time.

2005: The server shall allow the cessation of data collection from the active stream.

2006: The server shall allow the resumption of data collection from the active stream.

2007: The server shall allow the creation of a new stream connection with name, duration, location filter and optional language filter.

2008: The server shall allow the creation of a new stream connection with a name, duration, tracking term filter, and optional language filter.

2009: The server shall provide an API path to allow the cessation of data collection from the active stream.

2010: The server shall provide an API path to allow the resumption of data collection from the active stream.

2011: The user shall provide an API path to allow the creation of new stream connections.

3.2.3 Client Requirements

3001: The user shall be able to view the current status of the server.

3002: The user shall be able to view the details of the current running collection.

3003: The user shall be able to stop a running data collection process.

3004: The user shall be able to restart a stopped data collection process.

3005: The user shall be able to view the list of data collections.

3006: The user shall be able to view the tweets of a single collection in table format.

3007: The user shall be able to search and filter the displayed table of tweets.

3008: The user shall be able to start a new data collection process, providing a name, location information, duration, and optional language filters. The location information shall be defined using an interactive map-based bounding box, or using selectable, pre-defined bounding boxes.

3009: The user shall be able to start a new data collection process, providing a name, tracking terms, duration, and optional language filters.

3010: The user shall be able to access site and owner information in an 'About' page.

3011: The user shall be able to view visual representations and analysis of any data collection in a dashboard format. The dashboard shall comprise of self-contained modules.

3012: The user shall be able to toggle what modules to view in their dashboard.

3013: There shall be a visualisation module which displays tweet locations on a map.

3014: There shall be a visualisation module which displays a heat-map of tweet volume by country.

3015: There shall be a visualisation module which displays tweet numbers per hour.

3016: There shall be a visualisation module which displays a pie chart of hashtags.

3017: There shall be a visualisation module which displays tweets-over-time for a particular hashtag.

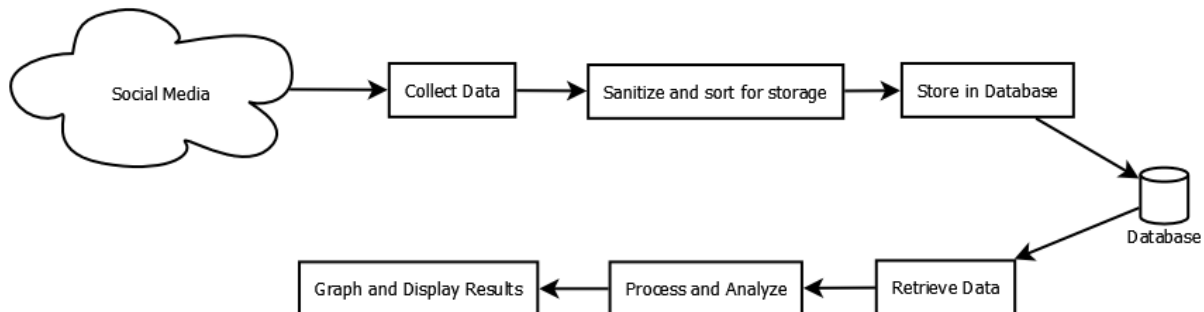
3.3 Architecture

In this section the architecture of the system will be discussed. This will include the high level design of the system, and the main responsibilities of each component. It will not go into detail about the specific design of each component in the system. That will be covered later in the Design chapter.

This system is divided into three parts: environment, server, and client. Environment constitutes the hardware, the operating system, HTTP server daemons, and any dependant software required for the system to operate, such as DNS scripts. Server handles the Twitter data streams, reading and storing data as it arrives. It is controlled by the client. Client takes the form of a website. It allows the user to interact with the server and the data already collected.

A basic flow of the system is outlined in figure 3.3.1. Data will be collected from the source, which in this case is Twitter. Incoming data will be sanitized if required, and then stored in the database. This section of the flow is fulfilled by the server.

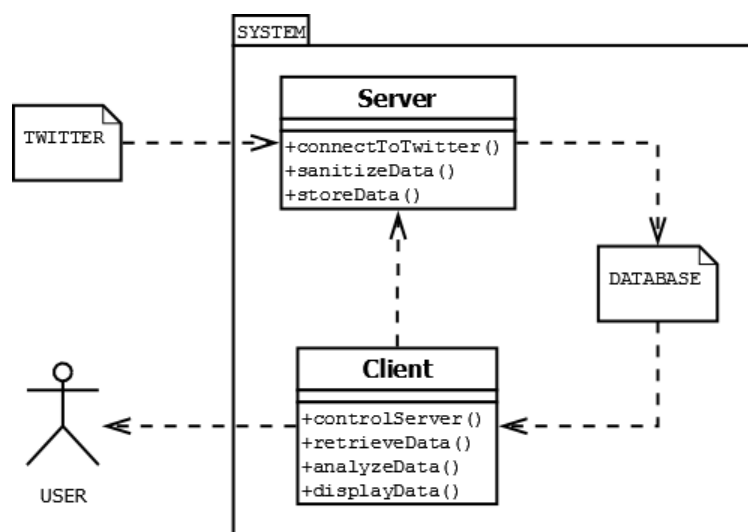
Figure 3.3.1: Data analysis flow



Functionality shown in the lower part of the diagram is handled by the client. It analyses the data collected in the database, and displays the results to the user. One important relationship not outlined in the diagram is the communication between the client and the server. Data collection on the server is controlled by the client. Essentially, the server is simply a persistent dumb worker process to handle the data stream, while the client contains most of the control logic.

Data analysis flow converted into a system architecture might look like figure 3.3.2. It follows the pattern of the data analysis flow in figure 3.3.1, but also includes the relationship between the client and server. Arrows denote the direction of data flow.

Figure 3.3.2: Data analysis flow converted into system architecture



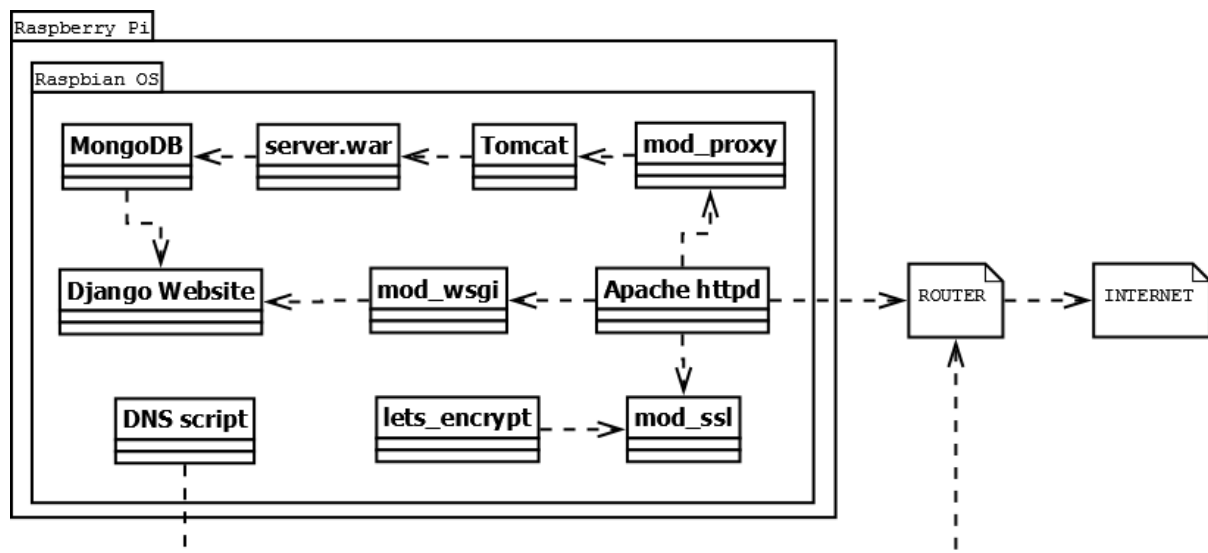
3.4 Design

System design will be discussed in this section. This includes a low level outline of each component and the functionality that each will provide to the system. Any results and issues found when implementing this design will be covered in the Implementation chapter.

3.4.1 Environment Design

As described in the background research section, the live system exists on a Raspbian Jessie Lite operating system on a Raspberry Pi 2 Model B. As outlined in figure 3.4.1, the machine will have a connection to the internet through a router. This router forwards the desired traffic types to the Pi, such as HTTP, SSL and SSH.

Figure 3.4.1: A diagram displaying the design of the operating environment.



An Apache HTTP server daemon manages incoming HTTPS/SSL traffic, and routes it to either the client (a Django website) or the server (a Tomcat module). All HTTP connections are encrypted into HTTPS/SSL connections by Apache before they are routed to their destinations. As a result, all traffic is afforded an extra level of security from third party packet sniffers. The SSL is provided as a free service by Let's Encrypt [15].

A DNS script notifies the DNS server, provided by DuckDNS [23], that the system is online and open to requesting traffic.

3.4.2 Server Design

The server is designed with several goals in mind:

- Be a persistent process
- Allow clients to interact with and control the server (i.e. order new data collections to start, stop ongoing collections etc.)
- Multithreaded to allow concurrent command and data collection
- Maintain a connection to the MongoDB instance

Persistence is an important feature as the server may be online for long periods for time. Twitter also punishes API users who do not obey the rules and limits of their API with rate reductions or disconnections. As a result, connections to Twitter needed to be handled carefully to avoid multiple reconnects or other abuses. This can be accomplished by maintaining one server instance per Twitter application (of which only one is used, in order to maintain simplicity).

Logic is not concentrated in the server. Control data is instead received from the client, and the instructions simply carried out server-side. In order to make this a reality, the server needs to be accessible from the client. This is accomplished using Jersey and Jax-RS, described earlier in this document, to integrate REST functionality into the server. Using this technology, URL paths to the server can be defined, which, when invoked, will accomplish different tasks. These paths are outlined in table 3.4.1.

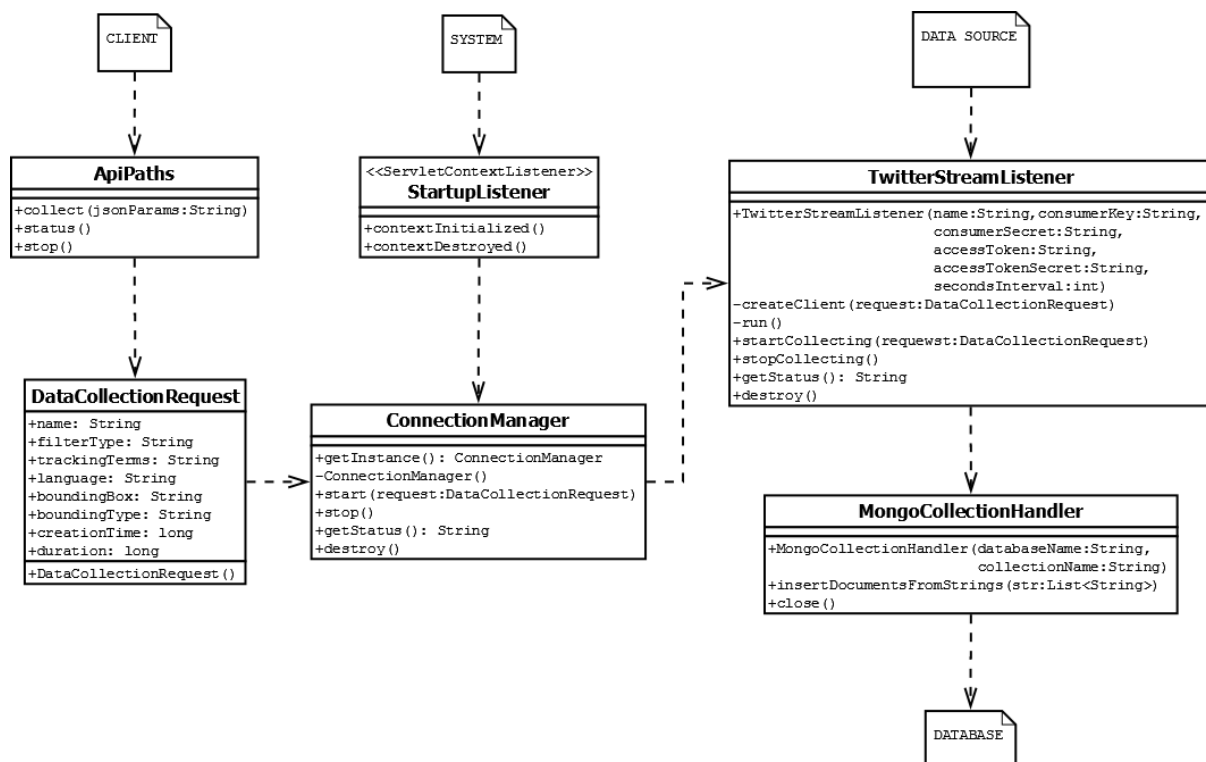
Table 3.4.1: Server URL paths

Path	Type	Description
/api/twitter/collect/	POST	Notifies the server to start collecting data using the supplied parameters.
/api/twitter/status/	GET	Returns the status of the collection process and the details of the active collection, if any.
/api/twitter/stop/	GET	If there is an active collection, stop collecting data.

Each URL begins with the /api location. This notifies Apache HTTPD that the request is to be routed to Tomcat, which will then match it to the server modules paths. Some other advantages in using Apache for this job is to prevent CORS (Cross-Origin Resource Sharing [40]) errors and allow referencing the server from the client as if they were operating on the same port.

Multithreading lets the server control and data collection to happen concurrently. Any control operates on the main thread so that any client actions can be acted upon quickly. Data collection is handled by a separate thread as its life is most likely limited and will be at some point stopped.

Figure 3.4.2: Server low level architecture



Finally, outlined in figure 3.4.2 is the design of the server. A *Connection Manager* is the persistent element, and a *Twitter Stream Listener* is the threaded data handler. A *Data Collection Request* is simply a Java object that encapsulates the data of a request by the client.

3.4.3 Client Design

The client-side website is built using the Django web framework. Code is designed around the Django MVT design pattern. Templates are simply HTML documents

created using Django's templating language. This language can dynamically create the HTML using generic loops and logic, and allows parameters to be passed into the document from the view. A view completes any processing required by the template parameters, and then renders the template as desired. These views are called by the URLs specified by the app. Models act as data objects which directly translate into database rows. For example, a User model might correspond to a row in the users table in your relational database. Unfortunately, as a non-relational database was being used, Django's model functionality could not be used.

Figure 3.4.3: Basic layout of a Django project

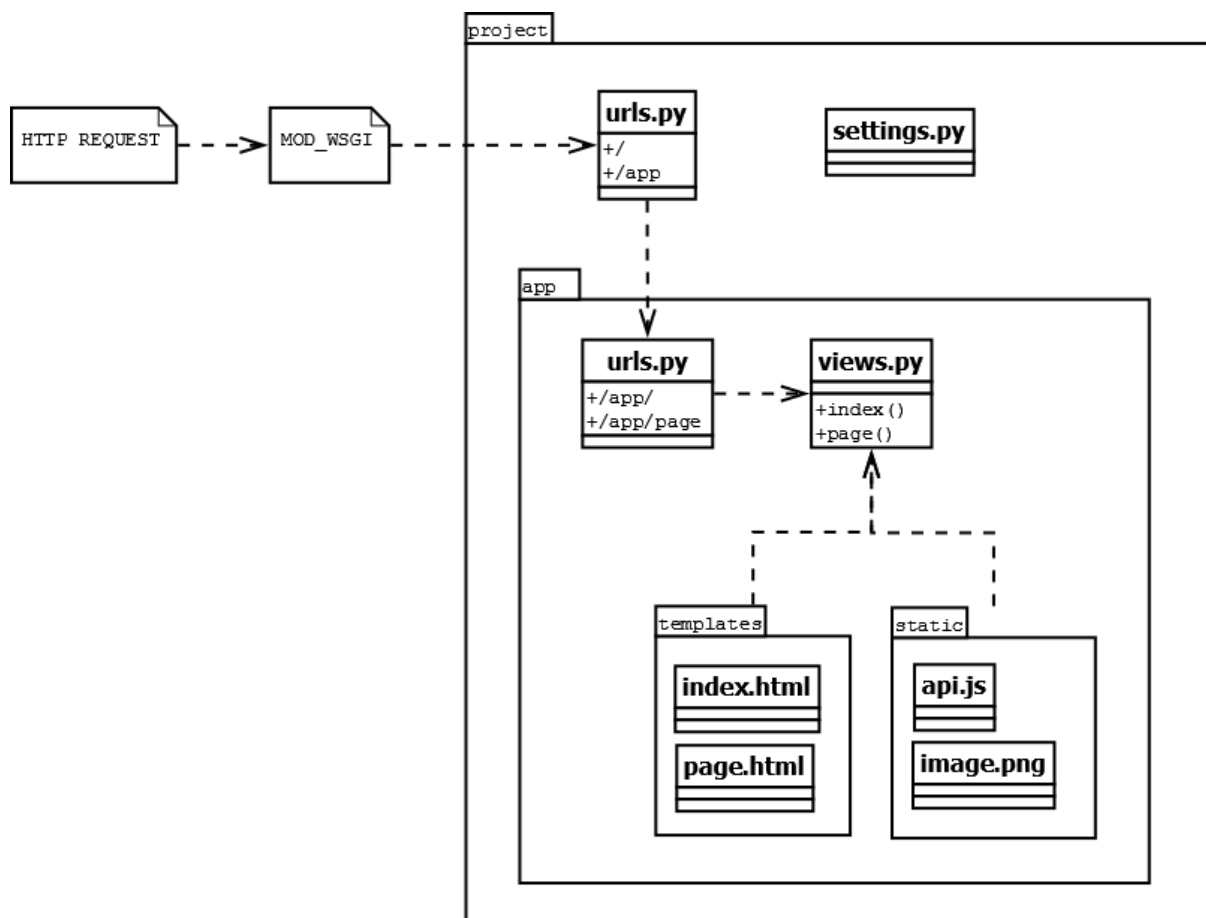
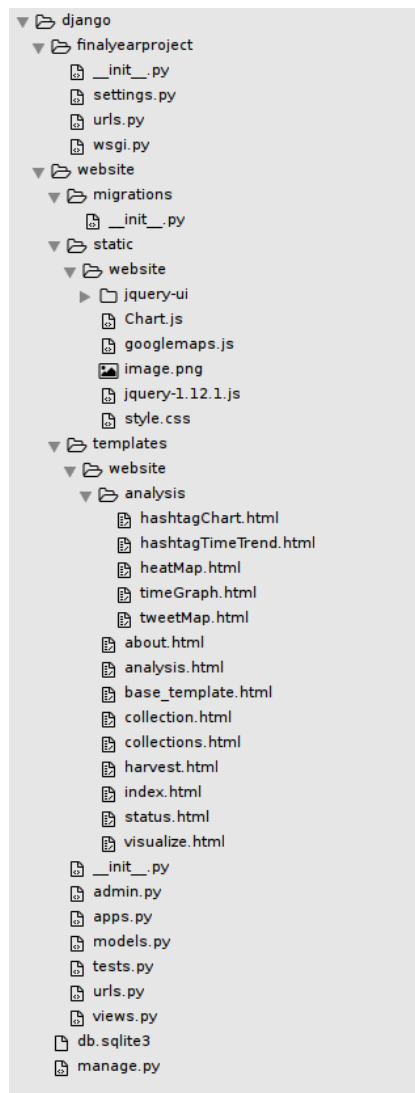


Figure 3.4.4: My client Django project layout



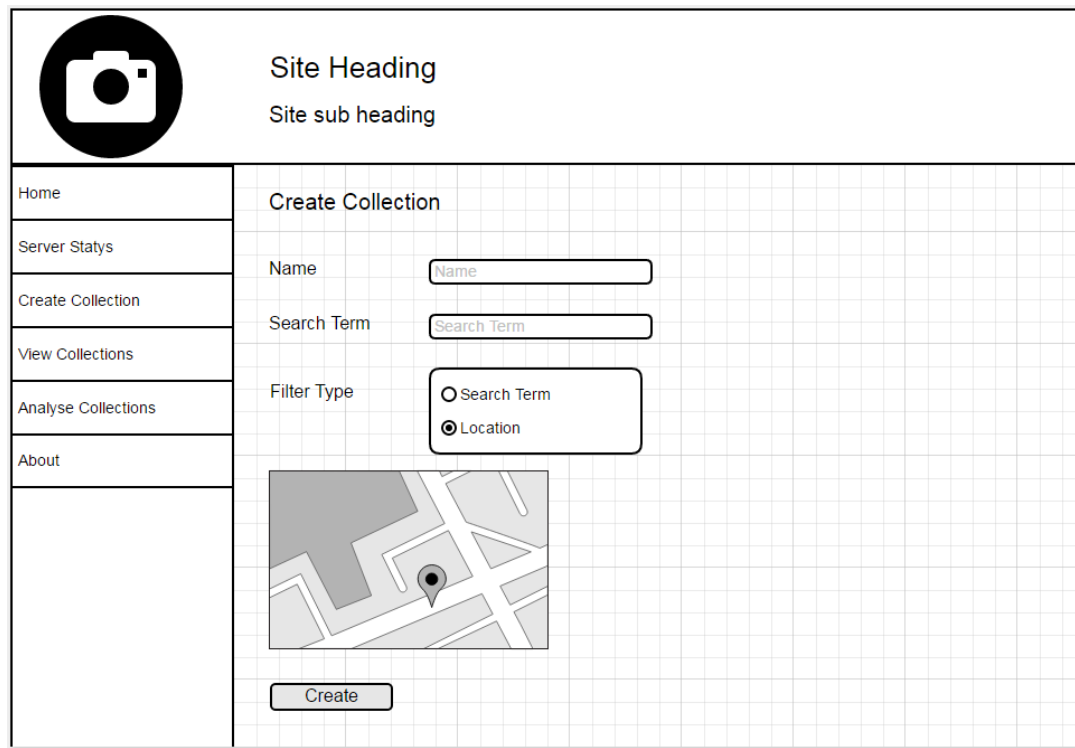
Users can to accomplish various tasks in the client:

- Create a new collection
- View Existing collections
- View analysis on a collection
- View the server status
- Control the server

Design of these pages are mostly straightforward. There is a base HTML template that forms the structure of the page, and includes common features between all pages such as the heading and the sidebar. This base template is then inherited by the actual page templates. An exception to this is the analysis page. It is constructed using several modules which are dynamically included in the analysis page. For example,

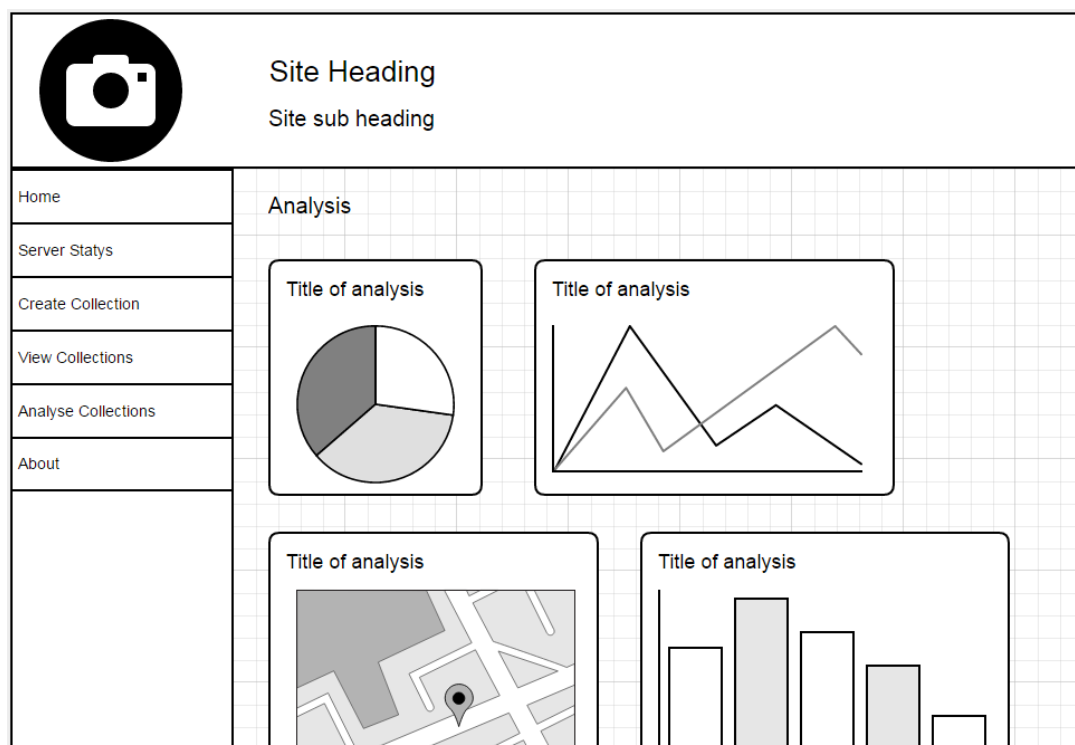
the user can curate a selection of visualisations to show on the page, and exclude any modules that they do not want to see.

Figure 3.4.5: A mock-up of the collection creation page



The mock-up shows a web interface for creating a collection. It features a top header with a camera icon, a 'Site Heading', and a 'Site sub heading'. A left sidebar contains navigation links: Home, Server Statys, Create Collection, View Collections, Analyse Collections, and About. The main content area is titled 'Create Collection' and includes form fields for 'Name', 'Search Term', and 'Filter Type'. The 'Filter Type' section has two radio buttons: 'Search Term' and 'Location', with 'Location' selected. Below the form is a map placeholder with a location pin and a 'Create' button.

Figure 3.4.6: A mock-up of the analysis page



The analysis page in figure 3.4.6 contains several modules, each which can be excluded by the user. This allows the user to focus on the information they want to see, instead of wasting processing time and screen space on what they might consider irrelevant information.

3.5 Implementation

This section will go into detail about any design issues experienced during implementation, such as changes, limitations, or new additions to the design.

3.5.1 Environment Implementation

As already stated, the system operates on a Raspberry Pi 2 Model B. A Router is connected to the Raspberry Pi, allowing access to the Internet. It is also has a 1TB external hard drive connected, which is used to store the data received from the Twitter data stream. An 8GB microSD card in the Pi hosts the operating system.

Figure 3.5.1: Raspberry Pi and external hard drive



The Raspberry Pi is running Raspbian Jessie Lite operating system, a minimal image based on the Debian Operating system [15]. This is a terminal-only version of Raspbian, and the Pi is not connected to a monitor. In order to interface with my Pi, a program called ShellInABox [34] is installed. ShellInABox is a web server and terminal emulator that facilitates access to the terminal of the Pi from a web browser using shell login details. ShellInABox does not replicate Apache HTTPD functionality. However, Apache acts as a proxy for ShellInABox, both directing data to ShellInABox and also encrypting all traffic into HTTPS. This prevents any transmissions from being read by a man-in-the-middle attack.

SSH connections are allowed between the Pi and other machines. SSH provides more advanced functionality such as file transfer and HTTP server configurations. These features were not included in the original design, but were quickly incorporated to expedite development.

Originally, Raspbian Wheezy was chosen as the operating system. It was soon found that there was no official support for running MongoDB in Wheezy. In order to rectify this, there was the option to either build and install the MongoDB code manually, or upgrade to Raspbian Jessie. It was decided to upgrade, instead of the alternative. While the upgrade was an annoyance, building the MongoDB might result in unforeseen problems down the line, making it the least desirable option.

3.5.2 Server Implementation

An earlier server design allowed for several collections to be ongoing at the same time. Due to limitations imposed by the Twitter API, the server was prevented from using multiple stream connects at once without creating new Twitter apps each time. Instead, the design was modified so that the server only supports one active stream at a time. The ongoing connection must be ended before a new connection can be made. In the future, there could be a queuing system for connections, or maybe more Twitter apps, which in turn mean more possible active connections. For now, as an early iteration, the server works as intended.

3.5.3 Client Implementation

Django assisted in quickly creating pages using templates and inheritance, and easily populating them with data. There were no major problems with the client side implementation. The only notable issues that were encountered was moving the site from the development HTTP server provided by Django, to the Apache HTTP daemon. A similar issue encountered during implementation was related to static files such as images and CSS files not working after the move to Apache. Apaches confusing configuration and Linux directory access permissions were the causes of these issues.

Figure 3.5.2: Client: The server status page

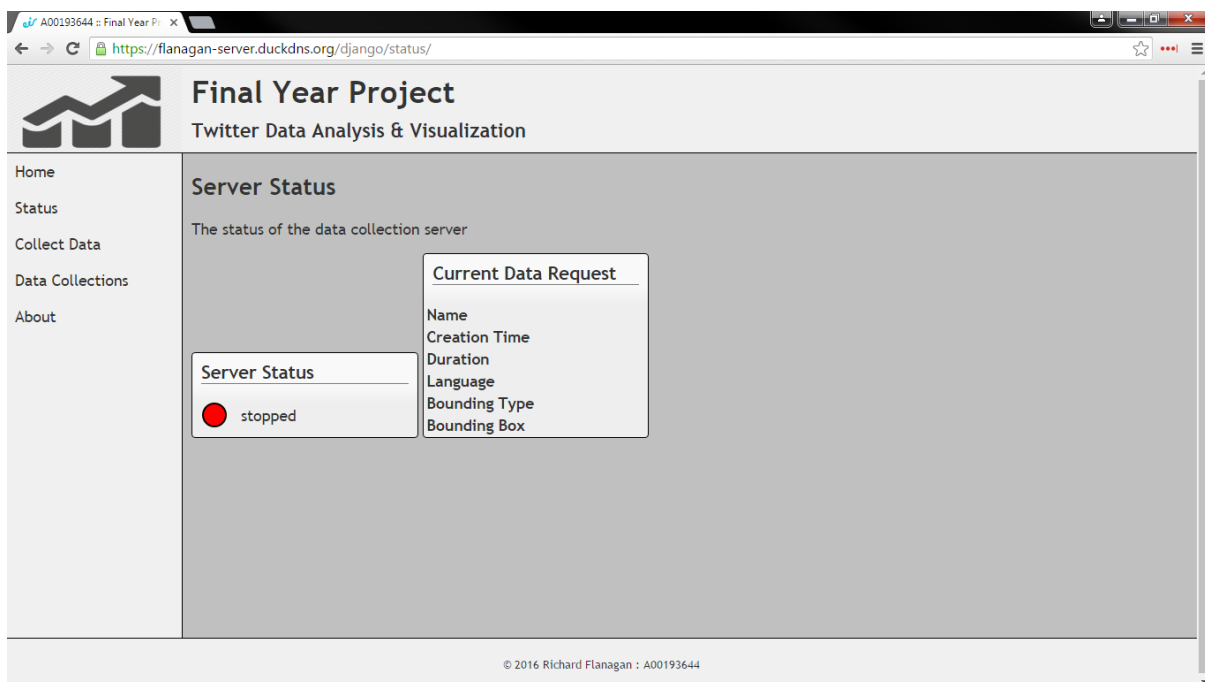


Figure 3.5.3: Client: The create collection page

Final Year Project
Twitter Data Analysis & Visualization

Home
Status
Collect Data
Data Collections
About

Harvest Data

Create and populate a new collection

Instance Details

Instance Name:

Duration (minutes):

Language:

Filter Method:

Tracking Terms:

Location Settings

Collect from:

Figure 3.5.4: Client: The existing collections

Final Year Project
Twitter Data Analysis & Visualization

Home
Status
Collect Data
Data Collections
About

Twitter data collections

Here is a list of previously collected twitter data collections which you can view

Collection Name	View Collection	Collection Analysis
tweets	collection	analysis
spa	collection	analysis
west_europe	collection	analysis
conor	collection	analysis
ire	collection	analysis
isle_of_man	collection	analysis
murica	collection	analysis
prague	collection	analysis
test55	collection	analysis
presentation_test	collection	analysis
athlone	collection	analysis
st_patricks_day_ireland	collection	analysis
hashtag_stpatricksday	collection	analysis

© 2016 Richard Flanagan : A00193644

Figure 3.5.5: Client: Viewing a collection

Final Year Project
Twitter Data Analysis & Visualization

Collection: hashtag_stpatricksday

Here is the data from the hashtag_stpatricksday collection

Show entries

Search:

user.name	created_at	coordinates	lang	text
- Han!	Thu Mar 17 14:30:11 +0000 2016	None	en	My mentions rn
.	Thu Mar 17 13:53:53 +0000 2016	None	en	@ChampionsLeague sure @paulpogba
3five	Thu Mar 17 14:33:25 +0000 2016	None	en	Happy St Patrick's Day from 3five #StPaddysDay #StPatricksDay https://t.co/0gJTwnXTtq
3OIN_	Thu Mar 17 14:15:36 +0000 2016	None	en	The bad thing about doing nothing on a day like this is liking everyone's posts on insta/Fb/twitter and then looking like a stalker . sorry
96.4 FM The Wave	Thu Mar 17 14:15:16 +0000 2016	None	en	Thank to the team @Tenpin_Bowling #Swansea for Siary's latest challenge. Check out the video https://t.co/EMgd43q23X https://t.co/KGLQUPQGq3
	Thu Mar 17			A few highlights from the parade so far... tweet us your

Figure 3.5.6: Client: Choosing analysis for a collection

Final Year Project
Twitter Data Analysis & Visualization

Analysis

st_patricks_day_ireland collection

- ☐ Tweet Map
- ☐ Heat Map
- ☐ Time Graph
- ☐ Hashtag Chart
- ☒ Hashtag-over-Time Trend Graph

© 2016 Richard Flanagan : A00193644

Figure 3.5.7: Client: The about page

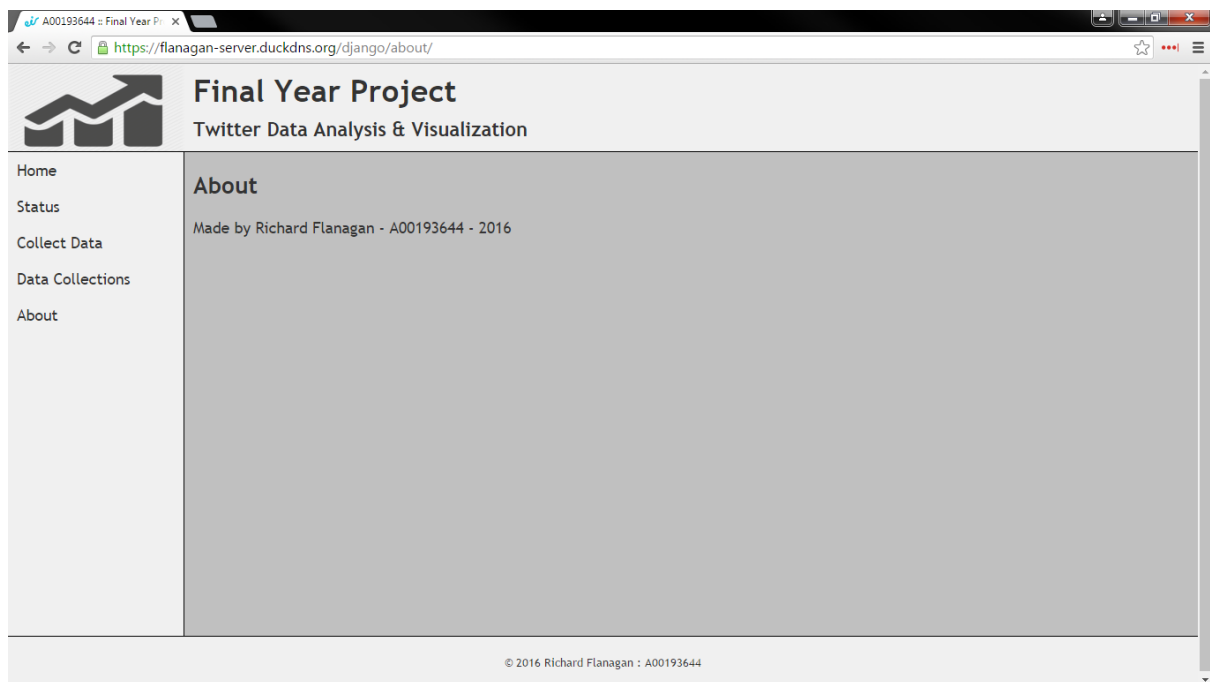


Figure 3.5.8: Client: Visualisation showing the location of a selection of tweets from the collection



Figure 3.5.9: Client: Visualisation showing the tweets-per-country heat distribution in a collection



Figure 3.5.10: Client: Visualisation showing the amount of tweets per hour in a collection

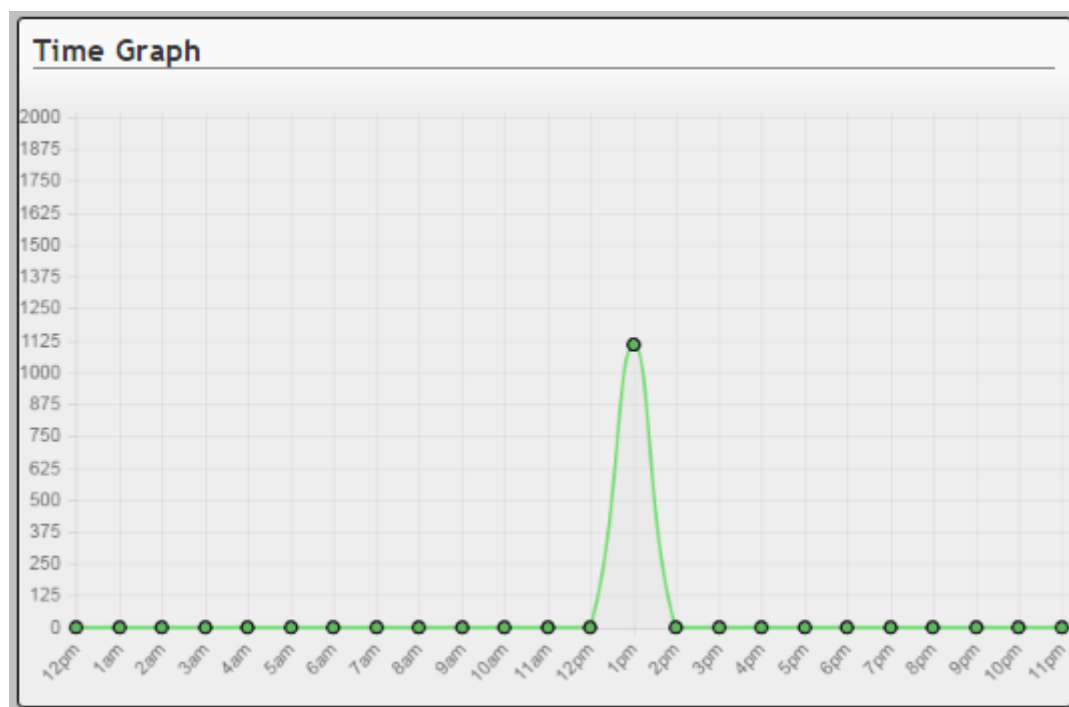
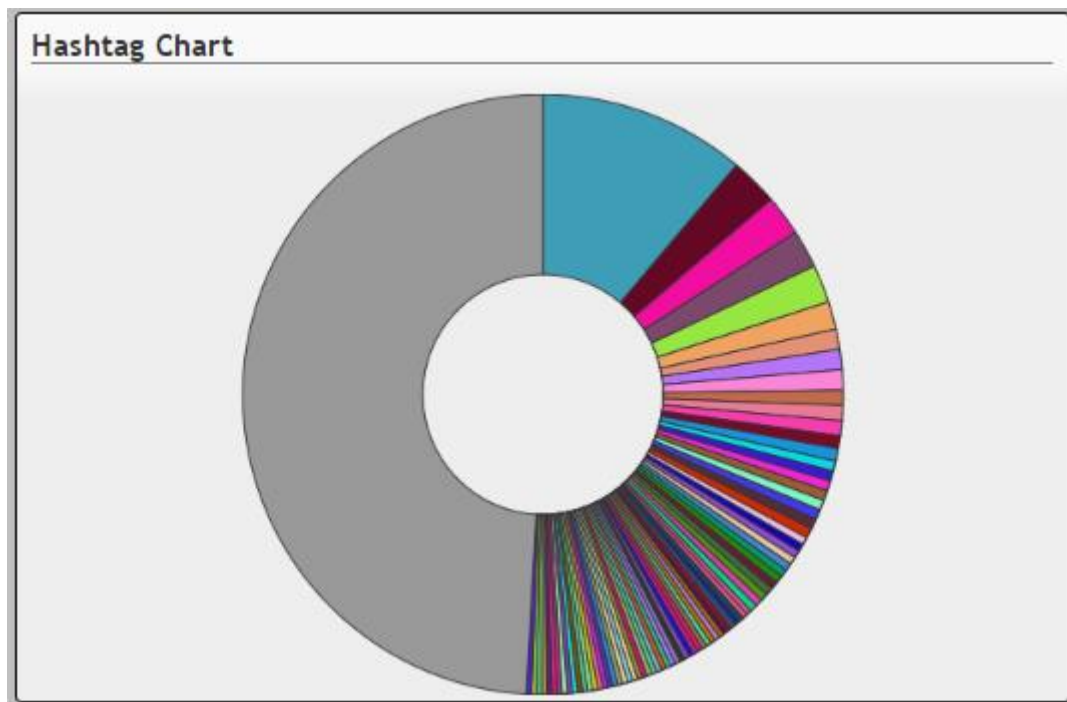


Figure 3.5.11: Client: Visualisation showing the hashtag breakdown of all the tweets in a collection



There are a few bugs in the system, especially in the client. For example, in the heat map in figure3.5.9, the lower end of the colour spectrum is the same as the no-data colour. This means that the United Kingdom appears as if it has no data, when the opposite is true.

There are some other usability issues that could be fixed, such as in the hashtag chart. All hashtags with only one mention are included in the Other section in grey. Other hashtags with two tweets, for example, are still shown, cluttering the pie chart.

Finally, some features are not fully implemented. A collection duration input field in the create collection screen is one example. It was supposed to stop collecting after the specified amount of minutes, but for now it does nothing. Given a few extra weeks, many of these issues might be fixed, and the system would feel more polished. However, they are all low priority issues and do not affect the operation of the system.

Chapter 4: Testing and Evaluation

4.1 Introduction

In this section, testing carried out on the system will be outlined, and the resulting quality of the system will be discussed.

4.2 Testing

Testing took a back seat during this project. No test cases were written for the client or the back end. The focus of the project was on design and implementation, with the aim of producing a working, feature-filled early iteration instead of a fully polished product.

Of course, manual testing of features was carried out to ensure that they worked properly. This manual testing is a type of dynamic black-box testing. It involves running the code to detect bugs or flaws in the program. This covers all the main, mission-critical functionality, but smaller features often to slip through the cracks.

Non-functional testing was also carried out. It involved testing load times of the web pages to attempt to reduce user waiting times. This is discussed further in the Results section.

4.3 Evaluation

As is already mentioned, testing took a back seat to implementation during this project. This choice resulted in some quality problems, as discussed in the client implementation section. This choice was necessary to implement the critical features of the system in time for demoing the product. There simply was not enough time to undergo rigorous testing.

One of the main aims of the project was to produce a working product, essentially as a proof of concept of the final service. All of the main features work mostly as intended, with some exceptions. Given more time, a more polished and feature complete product could be produced, and eventually marketed and sold to consumers.

Chapter 5: Conclusions

5.1 Introduction

This section will talk about the results of the project, and reflect on accomplishments.

5.2 Results

As stated in the project aims, the system was to test the capabilities of the Raspberry Pi 2 B by running the entire system on this limited hardware. This is a proof of concept that data analysis can be carried out on a smaller scale without requiring large hardware clusters for processing huge data sets.

It was found that larger data sets tend to strain on the Pi. Page load times would increase significantly for larger data sets, as is to be expected.

Table 5.2.1 outlines the load time range for several data sets. Data may be skewed due to a variety of factors, such as:

- Browser-side caching (time decrease)
- Network-side (internet provider level) caching (time decrease)
- Other processes requiring CPU time and RAM space in the Pi (time decrease)
- Data set attribute (hashtags-per-tweet, character encoding) (time increase)
- CDN fetch delays (time decrease)

These results should not be considered definitive. They can be considered accurate enough to allow me to draw some conclusions. For example, it shows an increase in data sets result in an increase in load page times.

Table 5.2.1: Page load times for a selection of data sets

Data Set (approx.)	Min and max load times (approx.)	Average load times
150 items	1.1 to 1.5 seconds	1.3 seconds
600 items	1.4 to 1.8 seconds	1.6 seconds
1100 items	1.7 to 2.3 seconds	2.0 seconds
1400 items	2.0 to 2.5 seconds	2.25 seconds
4700 items	4.0 to 4.5 seconds	4.25 seconds

According to a paper published by Fiona Fui-Hoon Nah in 2004 [42], users believe that “Web pages that were downloaded faster were perceived to be more interesting than the slower ones”, and that “satisfaction decreases with increases in response time”. In other words, slow load times equates to a worse user experience, which is a problem for any product or service on the market.

The Raspberry Pi struggles on relatively small data sets (at least in the realm of data analysis). While the Pi can be used to perform this analysis, asking it to complete this task, run Apache, MongoDB and Tomcat at the same, and load a webpage quickly enough to maintain user satisfaction, is too much to ask. Therefore, we can conclude that the Pi is unsuitable for data analysis tasks that require timely responses and feedback, such as in the service outlined by this document.

5.3 Reflection

There are various aspects of the project that I would like to include in the future. I would integrate a user account function into the system. It would allow users to register an account, and tie the data collections to each account. Users could keep a history of their collections and analysis for future reference. Functionality to allow users to export images of their analysis would also be useful. Users could include these images

in their marketing reports or meetings. In the same vein, collection comparisons would be very interesting. I can see value in the ability to compare analysis from two collections.

On the server-side, I would increase the amount of active collections available. At the moment, the server only collects relevant data from Twitter. The server currently operates in this way to save hard-drive space, processing time, and other system resources. Instead, I would like to experiment with collecting all stream data, then perform all user-specific data analysis on the complete dataset.

I think that I have learned a lot over the course of this project. I feel like I have achieved most, if not all, of the goals I set out to meet. I have developed a working service to allow social media advertisers and marketing managers to view the impact of their Twitter advertising campaigns. I have gained much valuable experience with the tools, technologies, and domains used in the development of this project, many of which I have never used before. I have also tested the Pi's compatibility with processing intensive tasks like data analysis, and also the Pi's ability to host a working, web-facing server. Overall, I am happy with my work.

5.4 Recommendations

As discussed in the results, it is clear that the Pi is unsuitable for data analysis tasks such as those desired by this project. On the other hand, if having fast response times is not an end-goal, or if using small data sets, the Pi is a surprisingly capable machine.

There are alternatives to using the Pi for data analysis. A user who needs large data sets processed quickly can rent cloud space for the duration of their analysis, or instead network many Pi systems together, creating a more powerful cluster.

What did prove surprising was the Pi's ability to host live web servers with minimal traffic. It is indeed well suited to being a home or small-business level server hosting a personal website or cloud storage.

References

- [1] A. Ginn, "What is a growth hacker?," [Online]. Available: <http://www.aginnt.com/growth-hacker>. [Accessed 20 March 2016].
- [2] Gartner, Inc., "Gartner Says Worldwide Software as a Service Revenue Is Forecast to Grow 21 Percent in 2011," 7 July 2011. [Online]. Available: <http://www.gartner.com/newsroom/id/1739214>. [Accessed 29 March 2016].
- [3] Responsible Conduct of Research, "Data Analysis," Northern Illinois University, [Online]. Available: https://ori.hhs.gov/education/products/n_illinois_u/datamanagement/datopic.html. [Accessed 20 March 2016].
- [4] makershed.com, "Raspberry Pi Comparison Chart," [Online]. Available: <http://www.makershed.com/pages/raspberry-pi-comparison-chart>. [Accessed 2016 March 20].
- [5] Raspberry Pi Foundation, "RASPBIAN," [Online]. Available: <https://www.raspberrypi.org/downloads/raspbian/>. [Accessed 29 March 2016].
- [6] Debian Project, "debian.org," [Online]. Available: <https://www.debian.org/>. [Accessed 30 March 2016].
- [7] Canonical Ltd., Ubuntu community, "Ubuntu Manuals," [Online]. Available: <http://manpages.ubuntu.com/manpages/lucid/man1/dpkg.1.html>. [Accessed 30 March 2016].
- [8] Oracle Corporation, "Welcome to VirtualBox.org!," [Online]. Available: <https://www.virtualbox.org/>. [Accessed 31 March 2016].
- [9] Apache Software Foundation, "Apache HTTP Server Project," [Online]. Available: <https://httpd.apache.org/>. [Accessed 22 March 2016].
- [10] Nginx, Inc., "nginx," [Online]. Available: <http://nginx.org/>. [Accessed 22 March 2016].

- [11] Netcraft, "November 2015 Web Server Survey," [Online]. Available:
<http://news.netcraft.com/archives/2015/11/16/november-2015-web-server-survey.html>.
[Accessed 22 March 2016].
- [12] Apache Software Foundation, "Apache Module mod_proxy," [Online]. Available:
https://httpd.apache.org/docs/2.4/mod/mod_proxy.html. [Accessed 26 March 2016].
- [13] G. Dumbleton, "readthedocs.org: mod_wsgi," [Online]. Available:
<https://modwsgi.readthedocs.org/en/develop/>. [Accessed 26 March 2016].
- [14] Apache Software Foundation, "Apache Module mod_ssl," [Online]. Available:
https://httpd.apache.org/docs/2.4/mod/mod_ssl.html. [Accessed 26 March 2016].
- [15] Apache Software Foundation, "Apache Module mod_alias," [Online]. Available:
https://httpd.apache.org/docs/2.4/mod/mod_alias.html. [Accessed 26 March 2016].
- [16] Django Software Foundation, "djangoproject.com," [Online]. Available:
<https://www.djangoproject.com/>. [Accessed 27 March 2016].
- [17] jQuery Team, "jquery.com," [Online]. Available: <http://jquery.com/>. [Accessed 27 March 2016].
- [18] Django Software Foundation, "Django at a glance," [Online]. Available:
<https://docs.djangoproject.com/en/1.9/intro/overview/>. [Accessed 27 March 2016].
- [19] Oracle Corporation, "Java EE at a Glance," [Online]. Available:
<http://www.oracle.com/technetwork/java/javaee/overview/index.html>. [Accessed 31 March 2016].
- [20] Oracle Corporation, "Java API for RESTful Services (JAX-RS)," [Online]. Available:
<https://jax-rs-spec.java.net/>. [Accessed 31 March 2016].
- [21] Apache Software Foundation, "Apache Tomcat," [Online]. Available:
<http://tomcat.apache.org/>. [Accessed 31 March 2016].
- [22] Oracle Corporation, "RESTful Web Services in Java," [Online]. Available:
<https://jersey.java.net/download.html>. [Accessed 31 March 2016].

- [23] Twitter, Inc., "A Java HTTP client for consuming Twitter's Streaming API," [Online]. Available: <https://github.com/twitter/hbc>. [Accessed 31 March 2016].
- [24] C. Gülcü, "Simple Logging Facade for Java (SLF4J)," [Online]. Available: <http://www.slf4j.org/>. [Accessed 31 March 2016].
- [25] J.-j. contributors, "A reference implementation of a JSON package in Java," [Online]. Available: <https://github.com/stleary/JSON-java>. [Accessed 31 March 2016].
- [26] Oracle Corporation, "Jersey: RESTful Web Services in Java," 11 March 2016. [Online]. Available: <https://jersey.java.net/>. [Accessed 31 March 2016].
- [27] OW2 Consortium , "ASM," 05 March 2016. [Online]. Available: <http://asm.ow2.org/>. [Accessed 31 March 2016].
- [28] Apache Software Foundation, "Welcome to Apache Maven," 15 March 2016. [Online]. Available: <https://maven.apache.org/>. [Accessed 31 March 2016].
- [29] Oracle Corporation, "mysql.com," [Online]. Available: <https://www.mysql.com/>. [Accessed 30 March 2016].
- [30] MongoDB Inc., "mongodb.org," [Online]. Available: <https://www.mongodb.org/>. [Accessed 30 March 2016].
- [31] Apache Software Foundation, "cassandra.apache.org," [Online]. Available: <http://cassandra.apache.org/>. [Accessed 30 March 2016].
- [32] Oracle Corporation, "PyMongo 3.2.2 Documentation," [Online]. Available: <https://api.mongodb.org/python/current/>. [Accessed 30 March 2016].
- [33] Microsoft, "DNS defined," 21 january 2005. [Online]. Available: [https://technet.microsoft.com/en-ie/library/cc787920\(v=ws.10\).aspx](https://technet.microsoft.com/en-ie/library/cc787920(v=ws.10).aspx). [Accessed 30 March 2016].
- [34] duckdns.org, "Duck DNS," [Online]. Available: <https://www.duckdns.org>. [Accessed 30 March 2016].

- [35] info.ssl.com, "FAQ: What is SSL?," [Online]. Available: <http://info.ssl.com/article.aspx?id=10241>. [Accessed 30 March 2016].
- [36] A. O. Freier and P. C. Kocher, "The TLS Protocol: Version 1.0," January 1999. [Online]. Available: <https://tools.ietf.org/html/rfc2246>. [Accessed 30 March 2016].
- [37] Internet Security Research Group, "letsencrypt.org," [Online]. Available: <https://letsencrypt.org/>. [Accessed 30 March 2016].
- [38] Twitter, Inc., "The Streaming APIs," [Online]. Available: <https://dev.twitter.com/streaming/overview>. [Accessed 31 March 2016].
- [39] Twitter, Inc., "Twitter Apps," [Online]. Available: <https://apps.twitter.com/>. [Accessed 31 March 2016].
- [40] Mozilla Foundation, "HTTP access control (CORS)," 5 February 2016. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS. [Accessed 31 March 2016].
- [41] s. contributors, "shellinabox," [Online]. Available: <https://github.com/shellinabox/shellinabox>. [Accessed 29 March 2016].
- [42] F. F.-H. Nah, "A study on tolerable waiting time: how long are Web users willing to wait?," 2004. [Online]. Available: http://sighci.org/uploads/published_papers/bit04/BIT_Nah.pdf. [Accessed 01 April 2016].
- [43] R. King, "Cassandra at Twitter Today," 10 July 2010. [Online]. Available: <https://blog.twitter.com/2010/cassandra-at-twitter-today>. [Accessed 30 March 2016].

Glossary

Term	Description
Back-end	The server-side implementation of the system. Handles requests by the front-end
Daemon	A computer process running in the background
DNS	A naming system that matches URLs and IP addresses
Framework	A reusable set of libraries, classes and resources on which to build an application
Front-end	The web page what is displayed to the user
HTTPD	A web server daemon that serves html documents to visitors
REST	REST delivers content to a requester over HTTP
SSL/TLS	A protocol which allows for secure communication between two points

List of Abbreviations

Abbreviation	Description
API	Application Programming Interface
BSON	A binary form of JSON
CDN	Content Delivery Network
CPU	Central Processing Unit
DNS	Domain Name System
DOM	Document Object Model
GB	Gigabyte
GUI	Graphical User Interface
HTML	Hyper-Text Mark-up Language
HTTP	Hyper-Text Transfer Protocol
HTTPD	HTTP server Daemon
HTTPS	Hyper-Text Transfer Protocol Secure
JSON	JavaScript Object Notation
MB	Megabyte
MHz	Megahertz
MVT	Model-View-Template
POM	Project Object Model
RAM	Random Access Memory

REST	Representational State Transfer
SSL	Secure Sockets Layer
TB	Terabyte
TLS	Transport Layer Security
URL	Uniform Resource Locator
WSGI	Web Server Gateway Interface

Appendix A:

A.1