Tracking Interconnected Facebook Links - Project Report

ELEN4009 - Software Engineering

School of Electrical and Information Engineering,
University of the Witwatersrand,
Johannesburg, South Africa

Back-End Pair:
Julian Zeegers (704582)
James Allingham (672732)

Front-End Pair:
Joseph Gage (751052)
Nathan Haag (873666)

Abstract

The Tracking Facebook Interconnected Links project is fully documented after the first prototype is developed. A detailed Software Requirement Specification is described that covers both the back-end and front-end requirements. The front-end and back-end's design and implementation of the first increment of the project is discussed in detail as well as the goals that were achieved. The agile Scrum methodology approach that was used for this project is also reviewed with a full description of the first sprint of the project. The sprint retrospective summarises the achievements of the first sprint and the improvements that can be made in the next sprint.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

Facebook is one of the most popular social networks used today. Due to the fact that it has over 1.5 billion users, it produces an incredibly large amount of data [1]. This data could be used in a number of ways, listed below.

- Data scientists and analysts could make use of this data to learn about social trends.
- Individuals can use the data for personal analytics.
- Sociologists could use the data to make and test new hypotheses.
- Marketing and advertising agencies could create specialised and targeted adds depending on a set of user's likes and status updates.

There are a large number of potential applications and the possibilities are only constrained by human creativity.

The Tracking Interconnected Facebook Links project is intended to visualise the links and connections of a Facebook user with other users. Initially, these visualisations are focused on identifying the relationships of a Facebook user (the end user of this product) with the network of friends this Facebook user has. The visuals are also intended to further illustrate the relationship connections of the user's friends of friends and an overview of the user's friend network. Some personal analytics such as analyses of the user's Facebook post contents will also be implemented. There is a myriad of additional features that could be added to this tool and therefore the solution must be dynamic and flexible. For example, degrees of separation analyses or more personal analytics could be added. At a later point the project could be expanded to include the large scale data analytics used by data scientists, sociologists and marketers.

The project team consists of four Software Engineering students, divided into two groups: one for the front-end and one for the back-end. This document serves as a description of the project. The software requirement specification will be laid out. The design and implementation of the front and back ends

will be discussed. Finally, the Scrum planning and retrospective will be presented.

1.2 Project Objectives

In order to both quantify the success of the project as well as properly create tasks with correct priorities, the objectives of the project must be laid out. They are as listed below.

- Create a prototype system for visualising a person's Facebook connections, social trends and personal analytics.
- Document the requirements for the system.
- Document the design of the system.
- Document the implementation and testing of the system.
- Document the group work aspects of the system including the SDLC.

1.3 Stakeholders

The Project Management Body of Knowledge describes stakeholders as individuals or organisations that may be impacted in any way by the decisions or outcomes of a project [2]. There are a number of stakeholders whose interests must be considered when carrying out the objectives above. Often the stakeholders can have conflicting interests which must be compared and carefully weighed up in order to resolve the conflict. The following stakeholders are considered to be of paramount importance:

- **The users**. The users of the system will be individuals who wish to explore their own personal Facebook friend networks and perform intimate personal analytics on their data. At this point the users **do not** include data scientists, analysts, sociologists and marketing/advertising professionals - a later version of the software will be targeted at these groups. The reason that the individual was chosen as the first target market is that the requirements for the individual are much less

resource intensive and thus are a more realistic goal for the first version of a product. Every decision that is made should improve the user's experience when interacting with the product.

- **The development team**. The development team consists of four software engineering students. These four developers are divided into two teams - the front-end team and the back-end team. It is important that the work in the project is divided appropriately between the teams. Note that the Scrum SDLC does not make distinctions between different developers and so it is up to the developers themselves to choose the appropriate tasks that best fit their skill sets. The success or the failure of the project will have a large impact on the development team. On the other hand, the development team will have a large impact on whether or not the project is a success. Whenever a decision is made, it should be one that makes both the best use of the team as well as improves the likelihood that the project will be a success.

- **Facebook**. Although Facebook will not itself be involved in the project, the perception of the company could be influenced by the product if it is successful. This could be as a result of people learning more about the amount or type of data about them on Facebook.

1.4 Abbreviations, Acronyms and Definitions

1. **ACID** - Atomicity Consistency Isolation Durability. A model for reliable database systems.

2. **Apache**. A popular web server.

3. **API** - Application Program Interface. A set of tools for building for software products.

4. **Client-Server**. An architecture or model for distributed computing via the Internet or other networks.

5. **CSS** - Cascading Style Sheets. A language for describing the presentation of a website.

6. **Cypher**. The query language for Neo4j graph databases.

7. **d3.js**. A popular JavaScript library for visualisation.
8. **DBMS** - Database Management System. A database program.
9. **Django**. An MTV web application framework, written in Python.
10. **Git**. A program for software and document version control.
11. **GitHub**. A website for hosting git repositories.
12. **GUI** - Graphical User Interface. A visual tool that allows a user to interact with an application.
13. **HTML** - Hyper Text Markup Language. A language for creating web pages.
14. **HTTP** - Hyper Text Transfer Protocol. An Application level protocol that makes use of HTML from sending and receiving messages in a client-server architecture.
15. **JS** - JavaScript. A scripting language often used in building interactive applications in the Web 2.0.
16. **JSON** - JavaScript Object Notation. A data interchange format used extensively in web applications.
17. **mod_wsgi**. The Django module for Apache Web Server.
18. **MTV** - Model Template View. Another name for the popular MVC framework. This is the nomenclature used by the Django web framework.
19. **MVC** - Model View Controller. A framework for building web applications based on a three layer abstraction of the data, the presentation and the interaction between them.
20. **Neo4j**. A graph database written in Java.
21. **Py2Neo**. A Neo4j driver for the Python programming language.
22. **Python**. A popular, powerful and flexible scripting language.
23. **Slack**. An application for team communication.

24. **SDLC** - Software Development Life Cycle. The process or approach the software development team adheres to.

25. **SDD** - Software Design Description. A document detailing the design of a software project.

26. **SVG** - Scalable Vector Graphic. A vector 2D image format.

27. **Trello**. A web application for managing project tasks.

28. **UI** - User Interface. A tool allowing a user to interact with an application.

29. **Web 2.0**. The informal name for websites that emphasize user generated content such as Facebook.

2 Software Requirement Specification

2.1 Introduction

This project is intended to visualise the links and connections of a Facebook user with other users. Initially, these visualisations are focused on identifying the relationships of a Facebook user (and the end user of this product) with the network of friends this Facebook user has. The visuals are also intended to further illustrate the relationship connections of the user's friends of friends and an overview of the user's friend network. Additional features could potentially be added to this tool and therefore the solution must be dynamic and flexible. This section describes the software requirements that will ensure that the end product is of the highest quality, produced most efficiently and fulfils its requirements.

2.1.1 Requirements

The requirements for the application are that it:

1. consists of a front-end that runs client side and a back-end that runs server side,

2. has a user friendly front-end which is easy to use for non-technologically inclined people,

3. is scalable - with a back-end that supports multiple clients,

4. is fast and responsive leading to an enjoyable user experience,

5. provides attractive and useful visualisations of Facebook data which allow the user to explore their Facebook networks in an intuitive and practical manner which facilitates discovery,

6. is extensible - allowing for additions and modifications to be made quickly and with ease,

7. is secure - keeping the users personal data safe and respecting their privacy, and

8. allows flexibility for data acquisition - the application should allow users to upload their own data or access their data via the Facebook API.

2.1.2 Purpose

The purpose of the project is to provide a tool to be used by individuals (and organisations at a later stage) to learn more about friend networks and how people are connected. This could be for marketing purposes, sociological research, finding friends or even for personal interest. Facebook provides a wealth of useful information about individuals, groups of individuals and companies. This project is a tool for visualising this information dynamically so that it will be advantageous to any user.

2.1.3 System Overview

This project will be made up of various software systems with the back-end and front-end working together to form a web-based visualisation application. The overview of the entire software system is illustrated in Figure 1. This figure shows how the various components of the system interact with each other. Figure 1 shows how the client (using a web browser) interacts with an Apache server and the Django framework. It describes how data from the

Neo4j database is fetched by the Django framework and sent to the user via the Apache server.

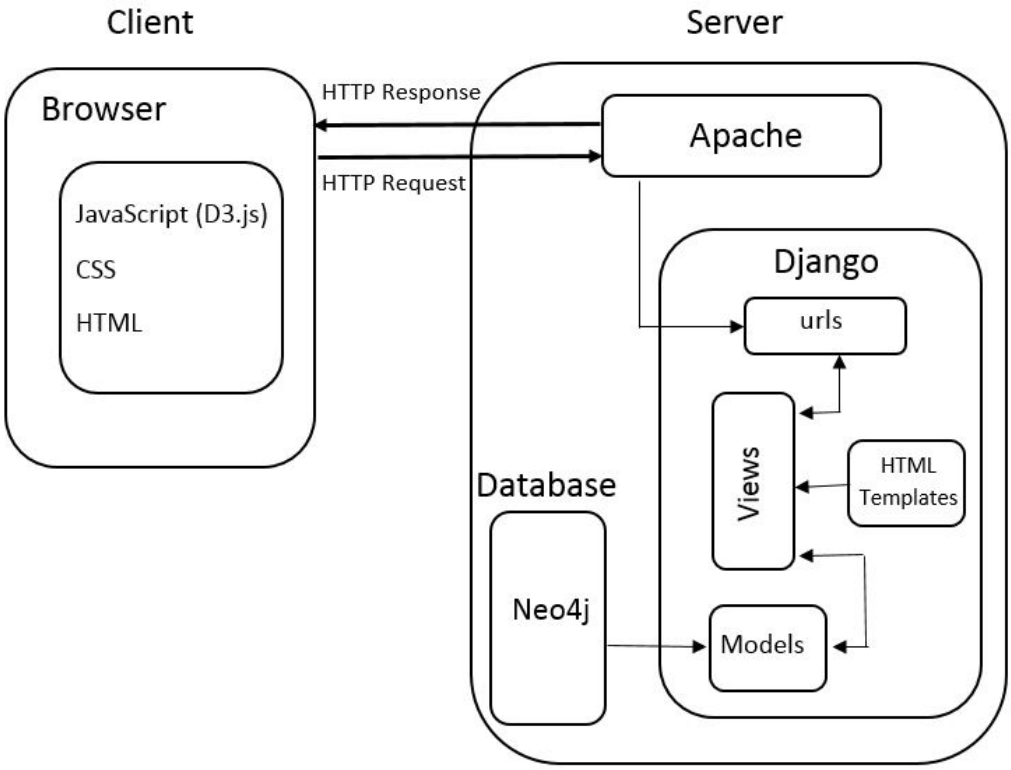


Figure 1: Diagram of System Overview

2.2 Software Development Life Cycle Choice

The SDLC is the process or approach the software development team adheres to throughout the project's development. Choosing the correct SDLC for the project is an important decision at the start of a software project as it could determine whether a project is successfully completed within the time frame, budget and scope requirements. A number of common SDLCs were considered and are listed below.

- **Waterfall model** - A sequential development model that lacks feedback. This technique consists of the following steps: requirements defi-

nition, design, implementation, verification and maintenance. However, due to the lack of feedback, this approach is not feasible unless the requirements are not completely defined in advance, the scope is fixed and the technology is well understood by the development team.

- **Iterative and incremental process** - A development model in which features are gradually added. Unlike the waterfall approach, this approach makes small changes at a time which can allow for more flexibility. However, this approach still requires the whole system to be well defined at the project start.

- **Agile** - A number of development principles aimed at facilitating the evolution of requirements and solutions in a collaborative process.

For this particular project, the fast pace method of the Agile SDLC is the best approach as speed in producing a working solution is a priority. The requirements for this project have not yet been fully defined and therefore the chosen Agile SDLC must be flexible and able to deal with requirement changes of various sizes.

The Agile SDLC also has varying methodologies that allows the achievement of an Agile software development movement. These methods include the Dynamic System Development Method, the Scrum Method, Rapid Application Development, extreme programming and the Feature-Driven Development Method. The Scrum methodology is the chosen method for this project as it allows the project progression to be agile and flexible with continuous feedback sessions to ensure the requirements are dynamically followed [3].

In the Scrum process, prioritised project tasks (referred to as sprints) are defined in short daily meetings (typically 15 minutes) with all the project team members. These meetings allow the team to communicate project progress and identify the important features that need to be added in order to increase the project progress pace. For this project, these sprints will allow for a working product to be produced in the shortest amount of time while additional features can be added at a later stage. This process also allows the client to closely keep track of progress and adjust the requirements in order to produce the most high-grade product possible. The daily meeting process occurs for up to 30 days by which time the first release of the developed software should be ready.

2.3 Architecture Choice

The choice of software architecture is influenced by the tools and libraries used for both the front-end service and the back-end service (detailed in Sections 2.4 and 2.5 respectively), network communication and team composition.

A two-tier, client-server architecture has been selected. The UI is only possible with logic on the server. Based on user requests, the logic decides on the correct data to be sent from the server to the client. Having the logic stored on the same server as the database is an important step towards decreasing response time between receiving and fulfilling user requests [4].

Two-tier means that any instance of a client utilising the provided UI will be separated from the server via a network through which all communication will occur. For the purpose of tracking interconnected Facebook links, the network between the client and the server will be the Internet. This design choice follows the need for multiple users to have access to the software simultaneously [5].

It is important to take note that in this architecture, the client and the server are tightly woven together, as communication between the two entities occurs directly over the network. This means that there is no middle tier to interpret and translate data correctly. In terms of team composition, this design choice is beneficial and reduces the amount of code needed to ensure compatibility.

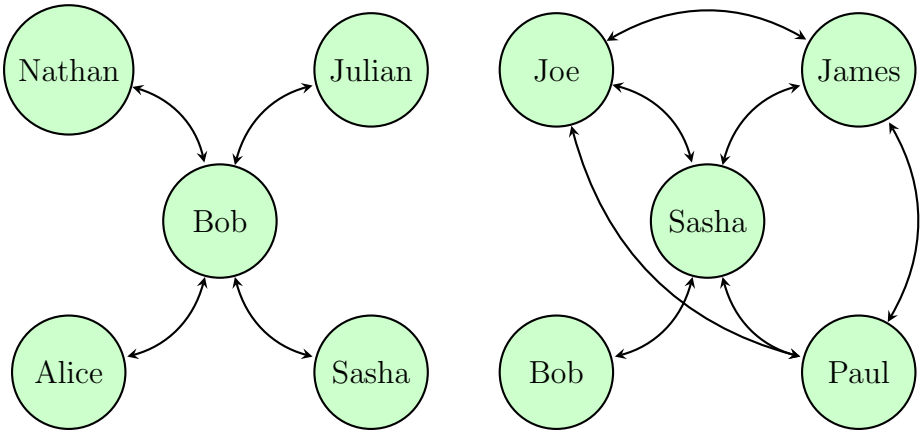
2.4 Front-End Service

2.4.1 UI Functionality

The front-end UI will contain several tools for visualising Facebook friend networks. The most basic tool will show a simple network diagram to view all friends and friends of friends for the user. This friendship diagram will include several functionalities.

Firstly, a user may select any of his friends and the diagram will transition to the selected person's friend network. This is pictured in Figure 2.

Secondly, the user may select certain information to be displayed on the diagram. For example, friendship data can be selected as in Figure 3 or



(a) Before click (b) After click

Figure 2: Transition of friend network after clicking on Sasha

relationship data as in Figure 4.

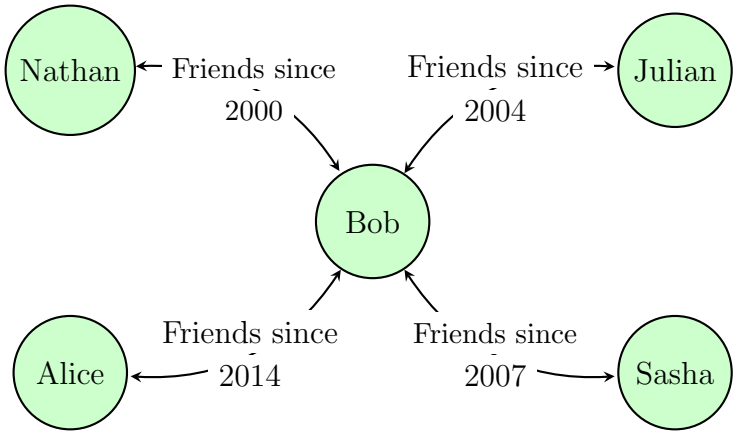


Figure 3: Friend network showing friendship details.

Thirdly, the user will be able to show additional information about a user such as their personal information. This is shown in Figure 5.

Finally, the user will be able customise the types of paths which he or she would like to show. For example, he or she may want to view only friends in the network who have a certain occupation or gender or age and or any other user attribute. Although it is not shown here the views must be able

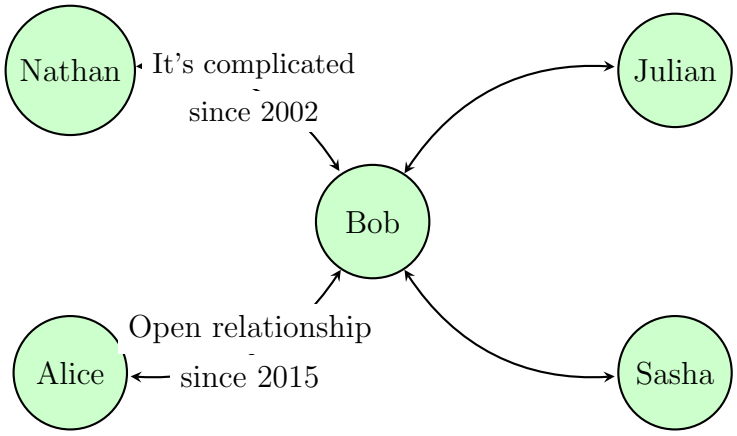


Figure 4: Friend network showing relationship details.

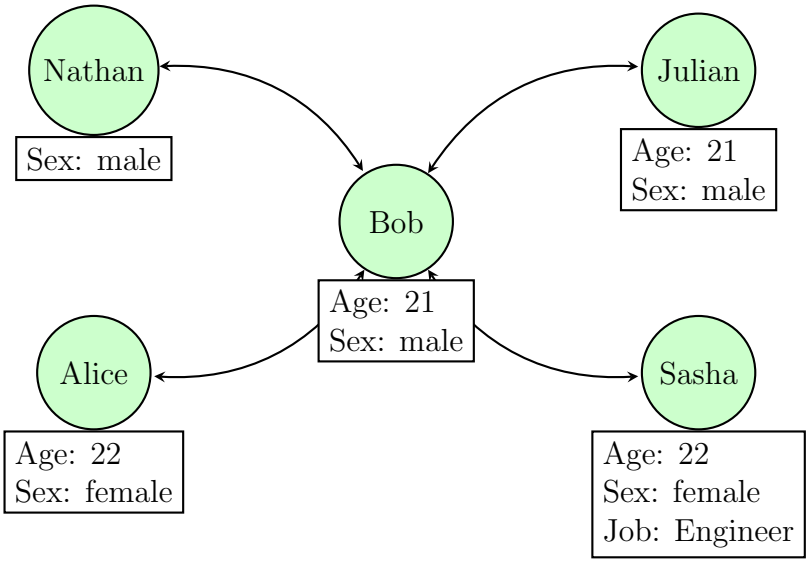


Figure 5: Friend network showing personal details.

to display the Facebook profile pictures rather than generic circular nodes.

2.4.2 UI Implementation

The UI will be operated from a web browser. The following sections detail the tools the UI implementation will take advantage of.

HTML and CSS

The application will use HTML and CSS to present its content to users. Several HTML pages will be implemented including a landing page, a registration page, a login page, a personal information page, a link to Facebook page and the actual data visualisation pages displaying the friend network and other visualisations.

Bootstrap

Bootstrap will be used to style the HTML and CSS pages [6]. This will make the UI more user friendly and aesthetically pleasing.

JS and d3.js

The application will use JS and d3.js for visualising the data [7]. The friend network diagram will use the D3.js force directed graph. Other d3.js visualisations will also be used for a number of views. An example of these views is word bubbles for analysis of a user's status messages.

2.5 Back-End Service

The back-end will be based on the Django web framework running on an Apache Web Server [8, 14]. The DBMS will be Neo4j, that will be interacted with via the Python driver Py2neo [10, 11].

Django is an open source high-level Python web framework. It is based on speed, security and scalability. It is also commonly used, with companies

such as Mozilla, Pinterest and National Geographic building their websites with it [8].

Although it has its own nomenclature, Django can be considered an MVC (Model View Controller) framework and has been compared to the popular Ruby on Rails web framework. An MVC framework is based on a three layer abstraction. The first layer abstracts the data access of the web application and is called the Model layer. The second layer is responsible for data display and is called the View layer. The final layer regulates the Views and is called the Controller layer. In Django's nomenclature, the view is equivalent to the standard controller, the template is equivalent to the standard view and the model is much the same as usual. For this reason Django is often referred to as an MTV [12]. It is important to note that Django is not a programming language, it is a programming pattern designed to streamline web development in the Python programming language.

Django has its own lightweight web server designed to be used for testing and development. However, for production the Django Software Foundation recommends using Apache with the mod_wsgi module [13].

2.5.1 Apache

Apache is a free HTTP web server that has been in production since 1995. It is an extremely sophisticated and flexible tool which supports the newest HTTP standards and a large number of platforms [14]. Apache has a large number of compiled modules that greatly extend it's functionality.

Apache processes any incoming HTTP requests from the clients and then sends the appropriate HTTP responses back. This will be done by interfacing with the Django web framework which will make the appropriate decisions based on the requests, and return responses containing content such as information from the Neo4j database.

Apache has many functions such as virtual hosting which allow one Apache installation to serve multiple websites. This feature allows developers of small websites to save on server costs. Apache also provides many useful security features required when a website goes into production such as: password and digital certificate authentication, SSL and TLS support, and much more. For larger websites Apache also allows load balancing.

2.5.2 Neo4j

Neo4j is a graph database written in Java. It is widely used and has drivers for many languages including Python. Neo4j follows the ACID model which means that it is extremely reliable. These statements are evidenced by the use of Neo4j in the systems of large companies such as Walmart and Ebay [10].

Graph databases are a type of NoSQL or Not Only SQL databases. They excel at managing large complex sets of data which can be described as nodes and relationships. Figure 6 shows an example of the type of data that can be stored in a graph database.

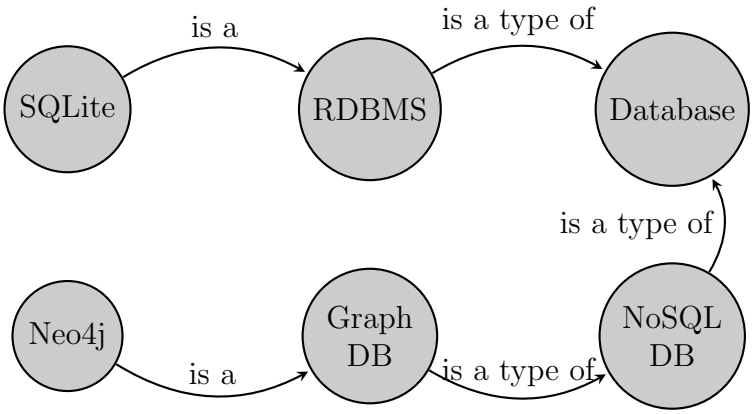


Figure 6: Example of data stored in a graph database

In this application the data being stored will be that of Facebook relationships. Graph databases are orders of magnitudes faster than RDBMS for queries on this kind of data, such as finding extended friend networks of individuals [15]. Figure 7 shows an example of the kind of data that will be stored in the database.

Note from Figure 7 that both the relationships as well as the entities themselves have associated data. In fact, each entity acts as a key value store. Queries can be performed to find all friends since a certain date or all friends of friends (as discussed above). These queries can be performed using Neo4j's Cypher query language or via language drivers.

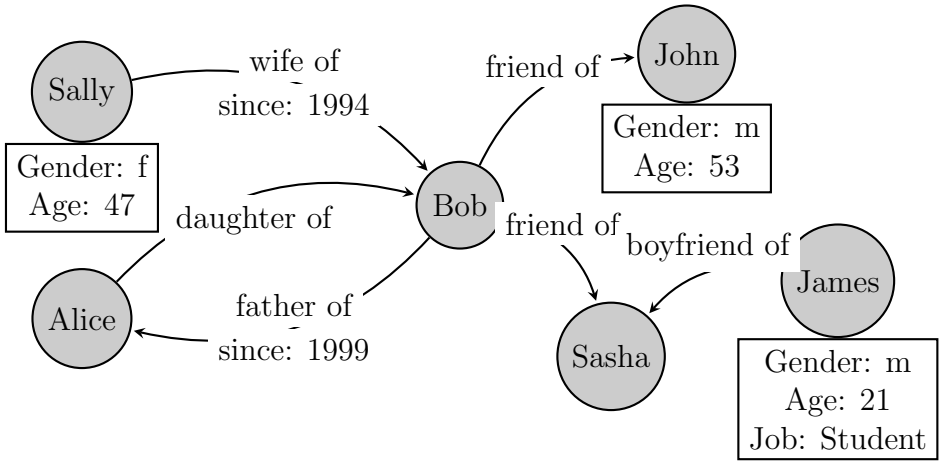


Figure 7: Example of facebook type data in a graph database

2.6 Supporting Software

The Interconnected Facebook Links project is intended to be used by a Facebook user and the assumption upon which all user interfaces will be built is that the user is not technologically advanced. Therefore the final product needs to be user friendly. The supporting software mentioned in this section describes the technologies that will be used to ensure the aforementioned user-friendliness and functionality of the product. The framework and internal structure of the project in terms of the architecture of both the front and back ends has been described in the previous sections. This section describes the software that will be used to support these mentioned structures.

In order to ensure that the end program is easy to use and looks professional, the visual design and theme needs to have a consistent, well designed layout and be visually appealing. Bootstrap is an HTML, CSS and JS framework that allows this charming front-end visual design to be achieved [6]. Bootstrap provides various templates that will maintain a consistent theme throughout the final website. It also provides navigational links for the website and the template is customisable so it can be used to create the desired theme.

Another technology that will support the functionally of the website is the d3.js graphical library [7]. This library contains JS code for many different visualisation graphs that can be used to visualise the data stored in the

Neo4j database. Considering that the main requirement for this project is to visualise the interconnections, the 'force-directed graph' from the d3.js library will be used. As mentioned, other visualisations from the d3.js library can also be used to the further explore the database. These other visualisations include dynamic bar and line charts, geographical heat-maps and dc.js cross-filter graphs as well as a variety of other options that may be used if they match the functionality appropriately.

2.7 Use Cases

A number of important use cases describing the base functionality of the product are listed and described in Table 1. The relationships between the use cases are shown in Figure 8.

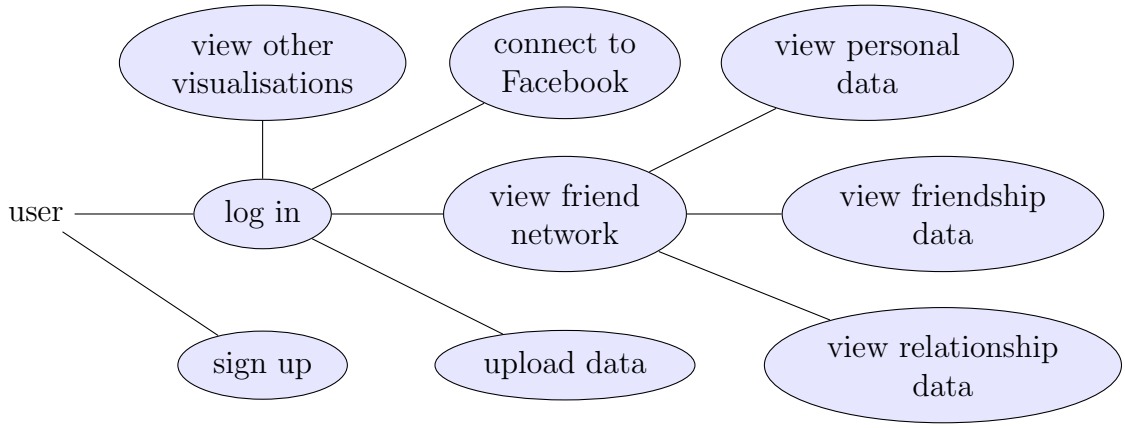


Figure 8: Use case diagram

There are a number of other use cases for the application. These would include the use of more visualisations as well as use by other types of users such as data scientists, sociologists and marketing professionals. The other visualisations were not highlighted as they do not form a part of the core functionality of the application which is to view interconnected Facebook links. The other users were not included because they do not form a part of the basic functionality in the first iteration of the application - they are not within the scope of this document. These use cases are expanded in the Appendix Section A.

Table 1: Crucial use cases for the product

Use case	**Description**
log in	The user has to log in in order to access the various views and analytics for their data
sign up	The user has to sign up to be able to log in for the first time
connect to Facebook	The user can connect to their Facebook account to pull data from the Facebook API
upload data	The user can manually upload their friend network data
view friend network	The user views their friend network, showing friends and friends of friends
view friendship data	The user views the details of the friendship links in the friend network view
view relationship data	The user views the details of the relationship links in the friend network view
view personal data	The user views the personal details of their friends in the friend network view
view other visualisations	The user can view other visualisations of their data

3 The Front-End

3.1 Design Document

This section was written in accordance with the IEEE Std 1016-1987 for SDDs [16].

The front-end design consists of a landing navigational web page, a friend status word bubble chart web page and a friend network diagram web page. These web pages together represent the front-end aspect of the web application for viewing interconnected Facebook links.

This section is divided into subsections representing the different views of the design, namely the home page view, the friend network diagram view and the friend status update word chart view. Each of these views represent a single interlinked web page.

3.1.1 Landing Navigational Web Page

The landing page was designed using a mark-up methodology for visualisation. This section describes each element of the page as well as its dependencies.

The web page consists of several elements: a navigation bar, a body and a footer. The navigation bar contains links to each of the other web pages. The body contains two visual links to the other two web pages. The visual links are screen shots of the other pages which when selected navigate to the corresponding page. The footer contains copyright information.

The page has the following dependencies: HTML, Bootstrap and the screen shot image files [6]. HTML was the mark-up language used for the page. Bootstrap was used for styling the page.

3.1.2 Friend Status Update Word Bubble Chart Web Page

The friend status update word bubble chart web page was designed using a mark-up methodology for visualisation along with some functional dependence. This section describes each element of the page, the script it uses as well as its dependencies.

The web page consists of several elements: a navigation bar, a body and a description. The navigation bar contains links to each of the other web pages. The body contains the word chart visualisation. The description simply describes what is being visualised. The visualisation shows circles known as bubbles (each containing a common word from recent status updates). The more times a word has occurred, the larger the bubbles are. If hovered over by a mouse pointer, the bubble also reveals the exact number of times the word has appeared in recent status updates.

The web page uses a script for creating the visualisation it displays. The general operation of the script is pictured in Figure 9.

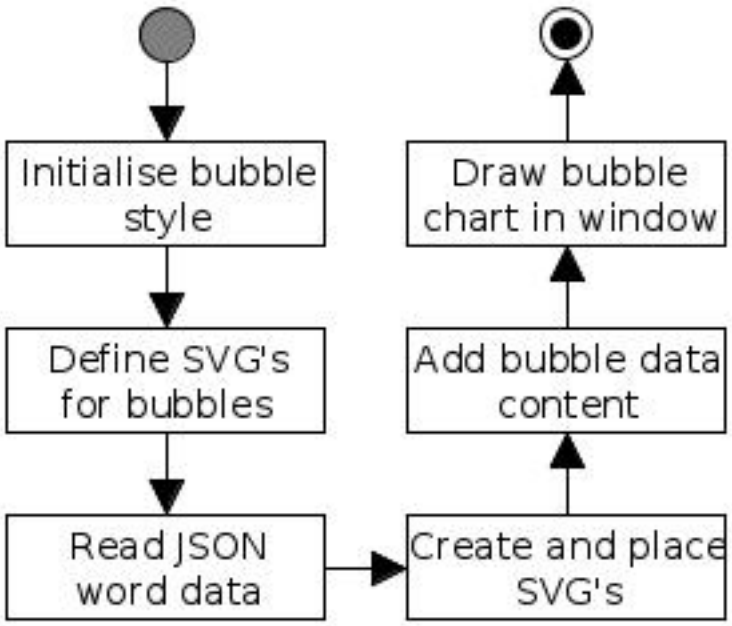


Figure 9: Code structure for word bubble chart drawing script.

The page has the following dependencies: HTML, Bootstrap, d3.js, JS and JSON formatted word data [6, 7, 17]. HTML was the mark-up language used for the page. Bootstrap was used for styling the page. d3.js was used for the providing the framework to create the word chart visualisation. JS was used

to write the script for creating the visualisation on the page. The word data was also used by the script to create the visualisation.

3.1.3 Friend Network Diagram Web Page

The friend network diagram web page was designed using a mark-up methodology for visualisation along with some functional dependence. This section describes each element of the page, the script it uses as well as its dependencies.

The web page consists of several elements: a navigation bar, a body and a description. The navigation bar contains links to each of the other web pages. The body contains the friend network visualisation. The description simply describes what is being visualised. The visualisation shows profile pictures of friends in the network with lines between them representing friendship. The pictures can be dragged around the screen and stuck in place. The images also grow slightly while a user's cursor is over it.

The web page uses a script for creating the visualisation it displays. The general operation of the script is pictured in Figure 10.

The page has the following dependencies: HTML, Bootstrap, d3.js, JS and JSON formatted word data [6, 7, 17]. HTML was the mark-up language used for the page. Bootstrap was used for styling the page. d3.js was used for providing the framework to create the word chart visualisation. JS was used to write the script for creating the visualisation on the page. The word data was also used by the script to create the visualisation.

3.2 Implementation

3.2.1 Introduction

The implementation of the front-end user interface formed the simpler aspect of the project. The front-end was created with HTML, using Bootstrap styling to create a desirable look and feel. The visualisations themselves, which were created using d3.js, were called in the HTML as script elements through Django's *static files* functionality. In order to provide developers with the ability to add new functionality easily and to ensure scalability, a

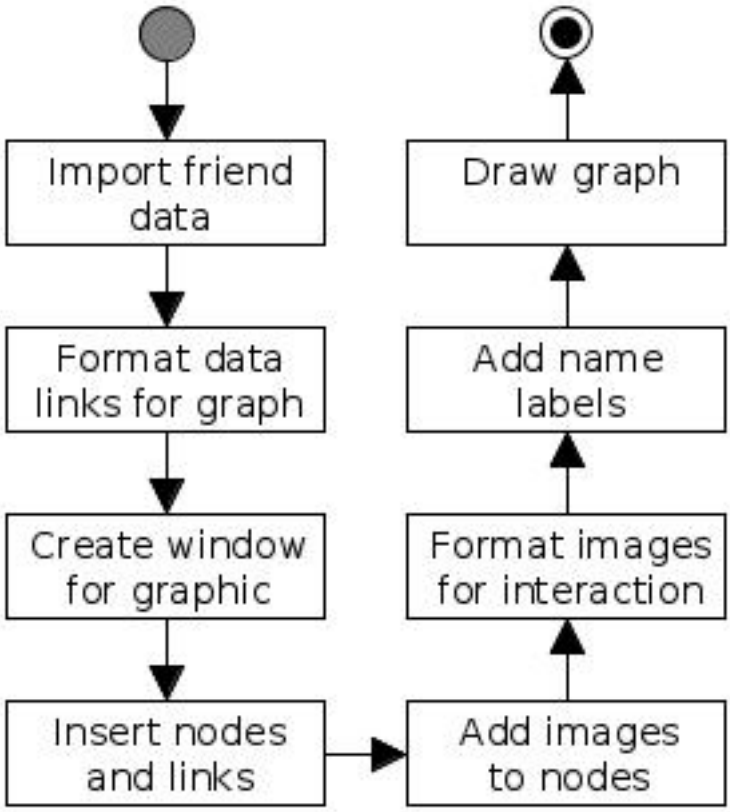


Figure 10: Code structure for friend network drawing script.

separate module exists for each visualisation that the software presents to end-users. This means that to add a new visualisation, three separate steps need to occur:

1. Create an HTML page to present the visualisation.
2. Implement the script that produces the visualisation.
3. Add the new HTML page to the index allowing users to navigate between visualisations.

3.2.2 Implementing HTML pages with Django

Django refers to HTML pages as *templates*. The initial implementation of the web pages consisted of coding a page using HTML and Bootstrap, and testing that the inserted scripts worked correctly in producing the visualisations. Once the scripts had been tested, the pages were integrated with Django. Instead of referring to the normal Bootstrap files, the page referred to the Bootstrap file found within Django's static files environment.

Static files are loaded into the HTML page as follows:

```
{% load staticfiles %}
```

To direct the HTML page to the Bootstrap styling file, the following link is called:

```
<link rel="stylesheet" href="{% static 'js/bootstrap.min.css' %}">
```

To direct the HTML page to the d3.js functionality, the following script is called:

```
<link rel="stylesheet" href="{% static 'js/bootstrap.min.css' %}">
```

Data for the friend network and for the word bubble chart is accepted in a JSON format. It is then filtered in the scripts and used to create the visualisations. Both visualisations were created using SVGs, which is beneficial due to its interactive nature and powerful ability to customise. Visualisations are created dynamically and on demand, and so SVGs are perfectly suited to the context of the project. A major advantage of using these graphics is that they are relatively small and so users will not have to wait long to receive their visualisations [18].

3.2.3 Friend Network

The friend network was implemented using the d3.js force directed graph. To create the graph, explicit parameters were defined in the script to control the size of the graph, setting the width at 960px, the height at 500px and the radius of the nodes in the graph to 6px. The HTML code imports the data through back-end functionality to create the friend network.

The script accepts graph data with two concepts: 'nodes' and 'links'. The node data is used to create the nodes on the graph representing the Facebook friends of the current user. Each node has an id, name, surname, occupation, age and img_url assigned to it. The link data maps nodes to each other, describing their relationship level, type and duration. This mapping is done by defining a link as the relationship between a source node and a target node.

3.2.4 Word Bubble Chart

The word bubble chart was implemented using the d3.js pack layout. The d3.js pack layout allows developers to represent elements as bubbles, and the size of the bubble is determined by the frequency (also known as the size attribute) of the elements. The word bubble chart was implemented with a flattened hierarchy, meaning that there is only one level of bubbles as in Figure 11, as opposed to 'bubble charts within bubble charts,' shown in Figure 12.

The script accepts graph data with two attributes: 'name' and 'size'. Each element in the JSON file gets its own bubble. The radius of the bubble is determined by each element's associated size attribute.

Figure 11: Bubble Chart with a flattened hierarchy

3.2.5 Front-End Testing

In order to ensure implementation occurred correctly, the following testing was performed after implementation:

General testing involved ensuring that the project workflow ran smoothly. This meant following all existing links and testing that they actually worked, in other words, redirecting users to the correct pages.

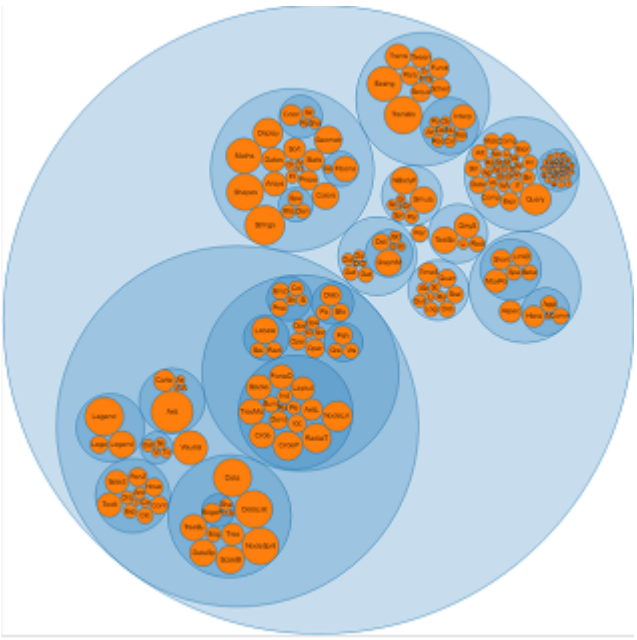


Figure 12: Pack Layout, showing hierarchical composition.

The friend network graph page was tested first by using dummy data to ensure that the graph behaved as expected. The graph's border, node characteristics and interaction capabilities was constantly adjusted until they were considered satisfactory. After this the network graph was tested with real data from the Neo4j database to ensure integration worked as expected. All of these tests were successful.

Testing of the friend status update word analysing bubble chart page involved using various sets of dummy data to see how the bubble chart changed when the size and content of the data set changed.

4 The Back-End

4.1 Design Document

This section was written in accordance with the IEEE Std 1016-1987 for SDDs [16].

The back-end design consists of a graph database system containing supporting functionality for making friend network queries as well as a web application framework. Both views were designed using a functional methodology. The particular functionalities of the design are documented below. These views together represent the back-end aspect of the web application for viewing interconnected Facebook links.

The section is divided into sub sections representing the different views of the design, namely the database view and the web application framework view.

4.1.1 The Web Application Framework

This section will discuss the functions, dependencies and details of the web application framework. The general operation of the system is shown in Figure 13.

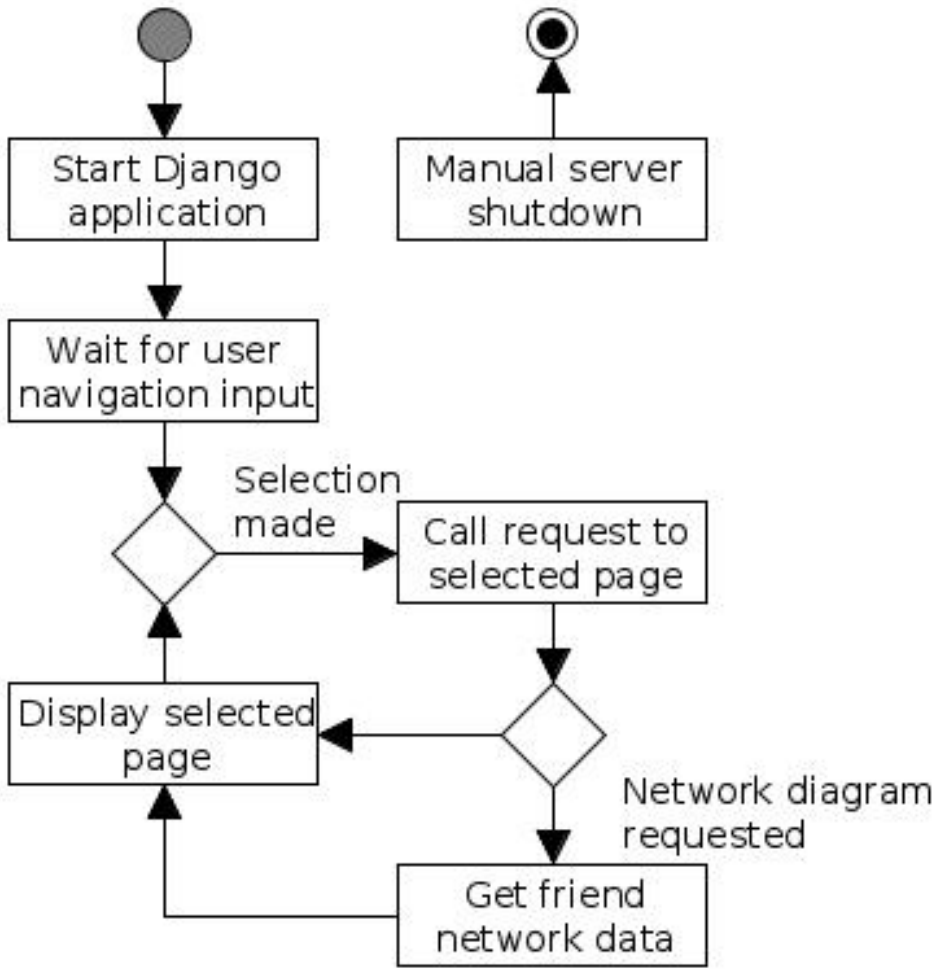


Figure 13: Code structure for running of web application and processing user input.

The following functions were used in the web application framework: `home()`, `friendNetwork()` and `wordChart()`. These functions refer to the different pages served by the web application. Each of these functions simply return the file path to the requested page which is the input to each function.

The `friendNetwork()` function, however, also calls a database supporting function to retrieve data which the page supported by the `friendNetwork()` function requires. The function not only returns the file path but also the content for the page to use.

The web application framework has the following dependencies: Django web framework, Apache HTTP server, mod_wsgi, Python and the page documents [8, 14, 19]. Django was the framework used for integrating the source code used in the web application. Apache was the HTTP server used to host the web application. mod_wsgi integrates the Django framework and the Apache server. Python was the programming language used by the framework. The page documents were stored within the Django framework and were required by the functions described above.

Several other details of the web framework are important. Firstly, the web application file system contains all of the dependencies of the functions mentioned as well as their supporting functions. Secondly, the file system also contains several configuration files specific to Django. Finally, several other scripts are required to run to start the server and web application - these are standard and available with the dependencies used.

4.1.2 The Graph Database

This section will discuss the functions, dependencies and details of the the graph database. The general operation of the supporting functions of the system is shown in Figure 14.

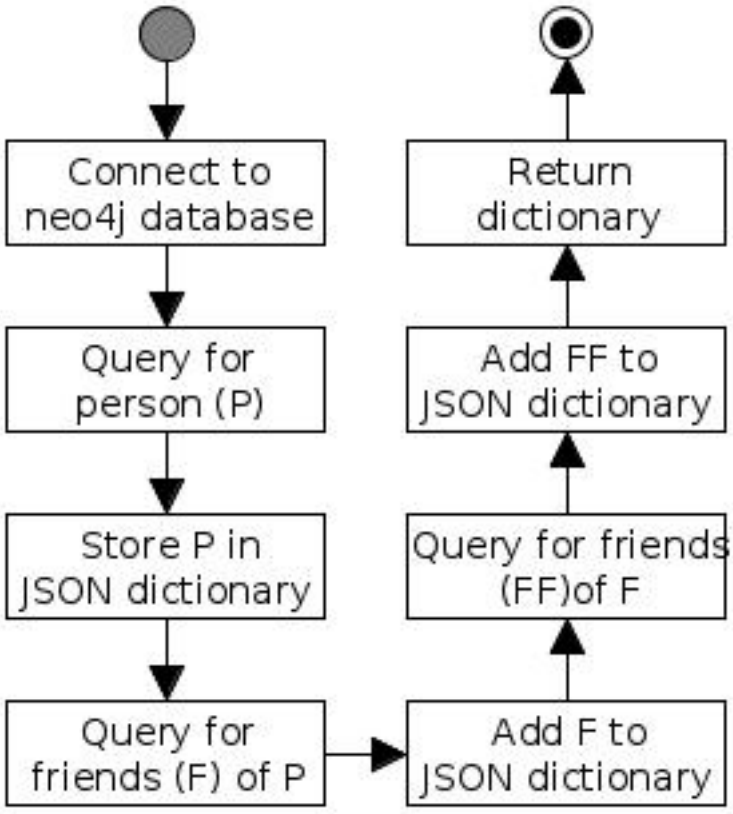


Figure 14: Code structure for fetching friends of friends network from Neo4j database.

The following functions were used to support the graph database: `getPerson()`, `getFriendNetwork()` and `getFoFNetwork()`. These functions each take as inputs the name and surname of the person being queried and return a dictionary containing the query results.

The `getPerson()` function uses the name and surname input to query the database for the requested person. The function returns a dictionary containing the person in a node.

The `getFriendNetwork()` function uses the name and surname input to query the database for the requested person and the person's friends as well

as the links between them. This function makes use of the `getPerson()` function. The function returns a dictionary containing all of the queried persons in nodes and the relationships between them in links.

The `getFoFNetwork()` function uses the name and surname input to query the database for the requested person, the person's friends and the friends of the person's friends as well as the links between them. This function makes use of the `getFriendNetwork()` function. The function returns a dictionary containing all of the queried persons in nodes and the relationships between them in links.

The graph database has the following dependencies: Neo4j, Py2neo and JSON [10, 11, 17]. Neo4j was the graphical database used in the software. Py2neo was the Python API which was used in order to enable use of the database by the Python programming language. The web framework required use of Python. The dictionaries returned by the functions mentioned above were in JSON format.

Two other details of the database are important. Firstly, the fields in the dictionaries returned by the above functions were nodes and links where each node and link had sub-fields. An example of their format is shown in Listing 1. Secondly, the database requires an initialisation and migration to start it up. Standard code is provided for these functionalities.

```
{
    "links": [
        {
            "level": "None",
            "since": "2014",
            "source": 248,
            "target": 247,
            "type": "FRIENDS_WITH"
        },
        {
            "level": "dating",
            "since": "2015",
            "source": 248,
            "target": 252,
            "type": "IN_RELATIONSHIP_WITH"
        },
```

```
            {
                "level": "None",
                "since": "2013",
                "source": 248,
                "target": 249,
                "type": "FRIENDS_WITH"
            }
        ],
        "nodes": [
            {
                "age": null,
                "id": 247,
                "img_url": "{% static 'images/paul.jpg' %}",
                "name": "Paul",
                "occupation": "None",
                "surname": "Cresswell"
            },
            {
                "age": 21,
                "id": 248,
                "img_url": "{% static 'images/james.jpg' %}",
                "name": "James",
                "occupation": "Student",
                "surname": "Allingham"
            },
            {
                "age": 21,
                "id": 249,
                "img_url": "{% static 'images/jules.jpg' %}",
                "name": "Julian",
                "occupation": "Data Scientist",
                "surname": "Zeegers"
            }
        ]
}
```


Listing 1: test

4.2 Implementation

4.2.1 Introduction

All implementation discussions are considered to occur in a Linux environment. The implementation of the back-end consisted of setting up Django as the main framework that provided means for integration of the front-end templates and visualisations with back-end data modelling and logic.

4.2.2 Django

The first step in the implementation of the software was installing Django. In order for the software to run, the Django source code, containing all front-end and back-end code, needed to be copied into the path /var/www. According to the Filesystem Hierarchy Standard Group, /var is the folder in which variable data files are placed, including 'spool directories and files, administrative and logging data and transient and temporary files' [20]. The Friend Analyser software constantly displays dynamically generated content, and so it is fitting that the web application is stored in a file location created for variable data.

4.2.3 Web Server

The Apache HTTP server hosts the web application, and is simple to implement. In order to host a Python web application, the mod_wsgi package is implemented. Once the Apache server and the mod_wsgi package have been installed, the Apache configuration files need to be changed to direct traffic through the mod_wsgi package.

This is done by replacing the default Apache configuration file with a new configuration file directing Django to run on the Apache server through the mod_wsgi package. Listing 2 is essentially telling Apache that the application needs to be served from the friend analyser directory, and that Apache needs to serve any request that is received from the application using the WSGI application defined in the WSGIScriptAlias path.

```
DocumentRoot /var/www/friendAnalyzer
```

```
WSGIScriptAlias / /var/www/friendAnalyzer/friendAnalyzer/wsgi.py
```

Listing 2: Apache Configration File Snippet

```
os.environ.setdefault ("DJANGO_SETTINGS_MODULE", "
    friendAnalyzer.settings")
application = get_wsgi_application()
```

Listing 3: wsgi.py

The WSGI application callable is the interface that is used for communication between the server and the code. As seen in listing 3, it is an object named *application*. The DJANGO_SETTINGS_MODULE is used by Django to locate the settings for the software. The settings module for the friend analyser application is found in the friend analyser folder, and is titled *settings.py*. In the settings file, the WSGI_APPLICATION variable is set to be the application object defined in listing 3.

4.2.4 Views

Django uses a *views* file to return data to the web pages to be rendered. There are three functions which have been named in the back-end design documents: `home()`, `friendNetwork()` and `wordChart()`.

The functions `home()` and `wordChart()` return the path to where the HTML page is stored. The home page is statically created and thus requires no generated content. The current implementation of the word bubble chart in the prototype consists of dummy data, and so no dynamic data is generated for this function either. The function format is shown in listing 4.

The `friendNetwork()` function generates content in the form of a JSON file and then passes it to the web page. The parameters passed to the `getFoFNetwork()` function is the name and surname of the current user, as seen in 5.

```
def home(request):
        return render_to_response("/var/www/
            friendAnalyzer/friendAnalyzerApp/
            templates/index.html")
```

Listing 4: home()

```
content = {"FriendGraph": json.dumps(db
    .getFoFNetwork("Joseph", "Gage"))}
return render_to_response("/var/www/
    friendAnalyzer/friendAnalyzerApp/
    templates/network.html",content)
```

Listing 5: friendNetwork()

4.2.5 Py2Neo Queries

As mentioned in the back-end design document, referring specifically to the graph database, there are three functions which support the database: `getPerson()`, `getFriendNetwork()` and `getFoFNetwork()`.

For the `getPerson()` function, the query is implemented as shown in listing 6. The query finds a match in the database based on the input parameters N (Name) and S (Surname) and then appends the return values in the form of a *node* element to the function return statement.

```
query = "MATCH (p {name:{N}, surname:{S}})
   RETURN ID(p) AS id, p.name AS name, p.
   surname AS surname, p.age AS age, p.
   occupation AS occupation, p.img_url AS
   img_url"
...
result["nodes"].append({"id":record.id,"name":
    str(record.name),"surname":str(record.
    surname),"age":record.age,"occupation":str(
    record.occupation),"img_url":str(record.
    img_url)})
```

Listing 6: getPerson()

`getFriendNetwork()` and `getFoFNetwork()` are carried out in a similar fashion. The difference is that where `getPerson()` returns the details of a single person, `getFriendNetwork()` and `getFoFNetwork()` uses a for loop to run through all the friends and all the friends of friends, respectively, that a user has, and appends them to the return result. An example of this from the `getFriendNetwork()` function is shown in 7.

```
for record in graph.cypher.execute(query, {"N":
    name,"S":surname}):
        result["nodes"].append({"id":record.idf
            ,"name":str(record.f["name"]),"
            surname":str(record.f["surname"]),"
            age":record.f["age"],"occupation":
            str(record.f["occupation"]), "
            img_url":str(record.f["img_url"])})
        result["links"].append({"source":record
            .idp, "target":record.idf, "since":
            str(record.r["since"]), "type":str(
            record.r.type), "level":str(record.r
            ["level"])})
```

Listing 7: getFriendNetwork()

5 The Sprint Planning

The Scrum process is an agile approach to projects (particularly software development projects) that encourages incremental progress, teamwork, collaboration with clients and allows for sudden requirement changes [21]. This process is comprised of short project sprints (typically 30 days) with each sprint focusing on the succeeding project requirements. This incremental approach requires functionally of aspects of the project after each sprint. This section discusses the sprint planning progress and the Scrum approach taken for the Tracking Interconnected Facebook Links project. Figure 15 illustrates the Scrum process and the labeled numbered sections 1, 2 and 3 are discussed in Sections 5.1 , 5.3 and 6 respectively.

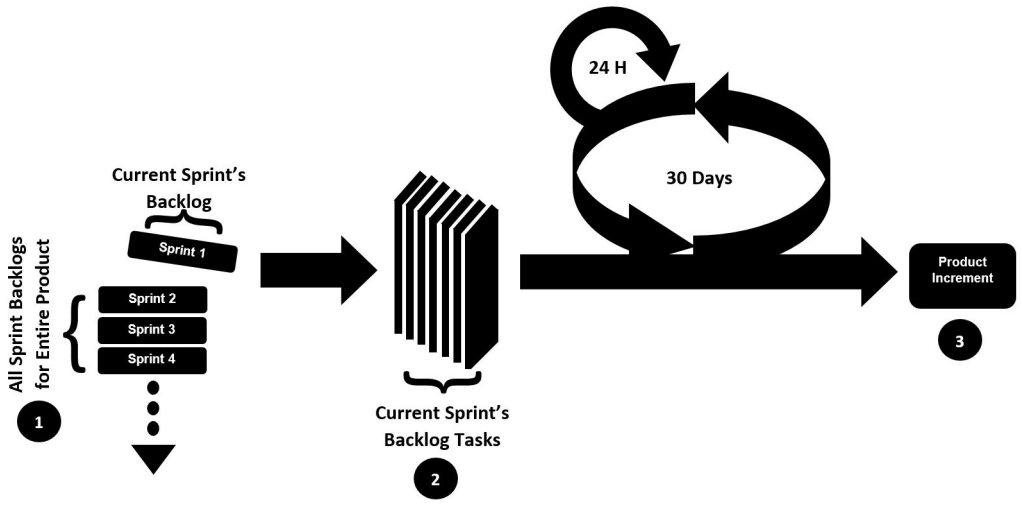


Figure 15: Diagram of the entire Scrum Process adapted from [22]

5.1 The Scrum Backlog

The Scrum backlog (labelled '1' in Figure 15) is a backlog that contains all the tasks for the entire project that are required to be completed. The formation of the Scrum backlog is compiled by the Scrum team and the product owner to ensure that the defined tasks are closely in line with the requirements for the project. The backlog comprises of the various task definitions, a time estimation of each task and the task's priority. The Scrum backlog is made at the beginning of the project during the planning stage. This backlog is flexible as tasks can be added, changed and re-prioritised by the product owner, the scrum team or other stakeholders during the SDLC. The Scrum Product Backlog for the Tracking Facebook Interconnected Links is shown in Table 2 which includes all the tasks, the estimated time and priority.

Table 2: The Scrum Product Backlog

ID	Task Description	Estimate (days)	Priority
2	Start a Django project and change to required settings	2	1
1	Create Neo4j database	2	2

3	Make an HTML landing page view. This should also be a template for the other views and therefore must make use of a Bootstrap theme and have navigational links.	1	3
4	Source Facebook user data and load it onto the database. Obtain the required permissions to access this data. The data must be stored in a logical and structured fashion in order to ensure the data can be accessed as required.	3	4
10	Write Python scripts and functions that return data required by the different views in a correct format by using database queries.	3	5
5	Create a view that contains a d3.js 'Force-diagram' that is used to visualise a friend network diagram that is interactively used by the end user.	5	6
6	Create a view that contains a d3.js 'Word bubble chart' that visualises the popularity of words used by Facebook friends of the end user in their Facebook status.	4	7
7	Set-up an Apache server on a server host computer and host the Django project that contains the various views.	2	8
8	Integrate the first phase systems required for the first sprint. This includes linking all the views to their URLs in the Django project framework, connecting the Neo4j database to the Django project and ensuring that the views are able to pull required data from the database and that the Apache server successfully hosts the project.	3	9

9	Do various software tests to ensure the integration is successful. These tests include testing the database queries return expected data, tests that the graphs in the views visualise the correct information and test that the Apache server host all of the URLs.	3	10
11	Register for a domain name and host the Apache server to that domain in order to allow the project to be accessed on the Internet.	2	11
15	Add Python scripts and functions for the new views	3	12
12	Create a view that makes use of a d3.js library called crossfilter.js. This view must contain charts that visualise the user's various Facebook friend relationship data using bar charts, line charts and pie chart. Each chart in this view must filter the other charts	3	13
13	Create a view that illustrates the degrees of separation between Facebook users. This view must take two names as an input and calculate the degree of separation between them.	4	14
14	View that can be used by a user to plan global trips to his friends.	6	15
16	Write a Python script that implements machine learning that the user can use to predict potential friend pairings.	10	16
17	Perform additional tests on the new views created.	2	17
18	Create a login interface where a user can create an account and login into the web application. An additional table to the database must be added to store these details.	5	18

19	Make a Facebook API for automatically pulling user information from the Facebook website. This should allow all views to be constantly up to date.	4	19
20	Implement the HTTPS protocol on the server to ensure that the web application is more secure	3	20
21	Create automated tests that continuously tests the software to ensure that it is always working as required.	4	21
	Total	74 Days	

The Scrum backlog shown in Table 2 was compiled at the beginning of the project by the Scrum team and product owner during the Scrum planning session. All tasks that are deemed necessary for the entire project are recorded in the backlog. As mentioned, this backlog is adjustable as there is a possibility that new tasks may arise during the project. Smaller meetings with the product owner and Scrum team will be conducted when the backlog needs to be adjusted. During the Scrum planning meetings, the logistics of the entire project in terms of the working environment and communication forums is also discussed and is described in the next section.

5.2 The Scrum Logistics

The project logistics and the formation of the working environment are discussed in the Scrum planning meeting. Firstly, the created project's software needs a central submission repository where all the project's software can be accessed. A Github repository is created and each member of the Scrum team is invited to join this repository and are given access to this repository. This repository is also used for the software version control and is a technology which allows collaborative software development.

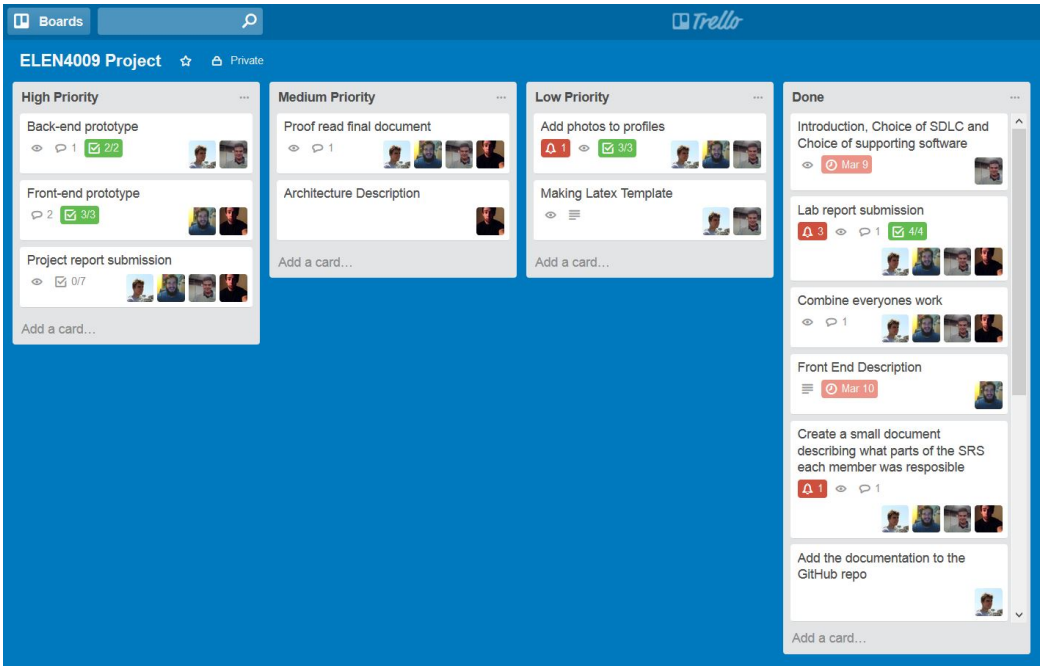


Figure 16: Screen shot of the Trello web application taken from [23]

In terms of communication, there are various platforms used to organise, ask questions or give comments with regards to the project. These platforms include Github, Trello and Slack. Github is used as a form of communication as the commit messages are used to keep track of completed tasks. Another platform called Trello is also used to keep track of progress and it also servers as a task organisation tool [23]. A screen shot of the Trello web application is shown in Figure 16. This application has prioritised "boards" that are ranged in three levels of priorities; namely high, medium and low. The Scrum master adds each task in the form of a "card" to a prioritised board with information associated with the task such as the member or members responsible for the task, the deadline and a task check list. Once a task card is completed, the team member who completed the task moves the task to the done board. This web application can can be downloaded as a mobile application by each Scrum team member and is useful for the Scrum master and team members to keep track of project progress. Figure 16 show a screen shot of the Trello web application and shows an example of the 'boards' and 'cards' using for the project.

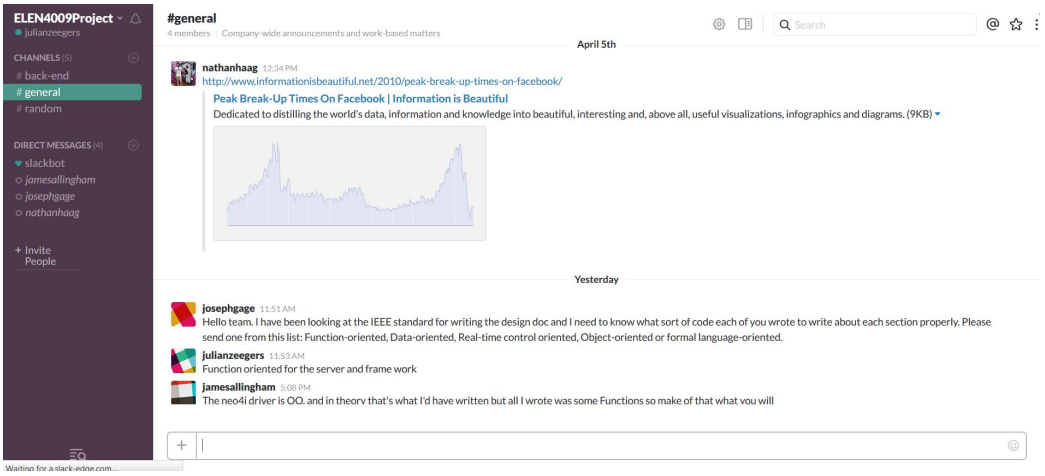


Figure 17: Screen shot of the Slack messaging application taken from [24]

Another communication forum used for this project is called Slack and this messaging platform is used by the Scrum team members to ask questions and post comments about the project [24]. This platform is different from other messaging platforms as users can also post code and get help from the other members for certain coding problems. Slack is also used to organize project events such as Scrum meetings, demonstration events and to communicate deadlines. A screen shot of the Slack messaging platform is shown in Figure 17 and it also shows an example of a typical project conversation.

These technologies and communication tools are used extensively during each sprint of the Scrum to ensure the sprint runs smoothly. The first sprint for the Tracking Facebook Interconnected Links project is discussed in the next section.

5.3 The Sprint Backlog

A sprint is typically two to four weeks long and it should produce a shippable increment of the product at the end of the sprint. A sprint is comprised of tasks defined in a sprint backlog and is labelled '2' in Figure 15. The sprint backlog is made up of certain tasks already defined in the Scrum product backlog (discussed in Section 5.1) and these tasks are the most prioritised tasks remaining within the product backlog.

The first sprint for the Tracking Facebook Interconnected Links project is planned in a sprint planning meeting conducted between the Scrum members. This meeting is run for about 6 hours because the sprint is planned to run for about 4 weeks. During this meeting a sprint product backlog is drawn up and is shown in Table 3. A short description of each task is defined as while as the duration of each task. It also shows the number of days in each week of the sprint each task should be worked on for. The overall sprint goals are defined for the sprint in this meeting which outline the expectations of the sprint's end product. For this first sprint, the following sprint goals are set out:

- Create a project prototype that includes one landing page and two functional views.
- Create and populate the database.
- Set up the system's architecture such as setting up the framework and server.

Table 3: The Sprint Product Backlog

		Days Task Worked on per Week			
ID	Short Task Description	Week 1	Week 2	Week 3	Week 4
2	Start a Django project	2			
1	Create Neo4j database	2			
3	Make an HTML landing page view with a bootstrap theme and navigational links	1			
4	Source Facebook user data and load it onto the database	2	1		
10	Write Python scripts and functions that return data required in a correct format		2	1	
5	Create a view that contains a d3.js 'Force-diagram' that is used to visualise a friend network diagram		2	2	1

6	Create a view that contains a d3.js 'Word bubble chart' that visualises the popularity of words used in status		1	2	1
7	Set up an Apache server that hosts the Django project			2	
8	Integrate the first phase systems			2	1
9	Do software testing				3

Each task is also allocated to a Scrum team member during this sprint planning meeting. The team members choose the tasks that they would like to do with the front-end and back-end team members choosing their respective tasks. Once the tasks are scheduled, a new Trello board is created with the new task allocations (as discussed in Section 5.2). The tasks described in Table 3 are subject to change, however these changes must not change the sprint's goals that were outlined. There were minutes that were taken for the Sprint planning meeting and these minutes can be found in Appendix B. In Appendix B, some project roles were simulated such as the Product Owner, Project Administrator, Scrum Master and Marketing Manager in order to add authenticity to the meeting that took place.

5.4 The Daily Scrum Meetings

Once the sprint begins, there is a daily Scrum meeting that occurs in the morning of each working day. During this meeting, each Scrum team member is given the opportunity to report on their daily progress. Each member is also required to talk about the following items:

- The tasks that they completed since the previous meeting
- The tasks that they plan to work on after the current meeting
- Any project challenges that they are currently struggling with

These daily Scrum meetings are meant to be relatively short (typically 15 minutes) and the objective of these meetings is to provide a platform for the

Scrum team to keep track of progress. These meetings also allow for the members to obtain help or get ideas on how to solve small problems and can be used to communicate small project plan changes.

5.5 Managing The Sprint Process

The sprint process of the project is an effective methodology that allows rapid software development that can be flexible in terms of the requirements. However, the fast paced nature of the sprint can be chaotic and disorganiszed and it is therefore very important to manage the project well. An organised, disciplined Project Manager (can also be the Scrum Master) must be appointed to ensure the project is well organised. Various project managing tools must be utilizsed in order to ensure that the sprint achieves the set out sprint goals. Tools that have already been mentioned include conducting daily meetings and using the communication platforms effectively to maintain project progress. Another project managing tool that can be used is a Gantt chart. A Gantt chart shows the various tasks defined in the sprint backlog on a time-line. It illustrates to the Project Manager when certain tasks must be completed by, which team members are responsible for each task and the critical path. The critical path contains tasks which are crucial to the deadline as delaying these tasks could cause delays for the entire sprint. The Gantt chart for the first sprint is shown in Figure 18

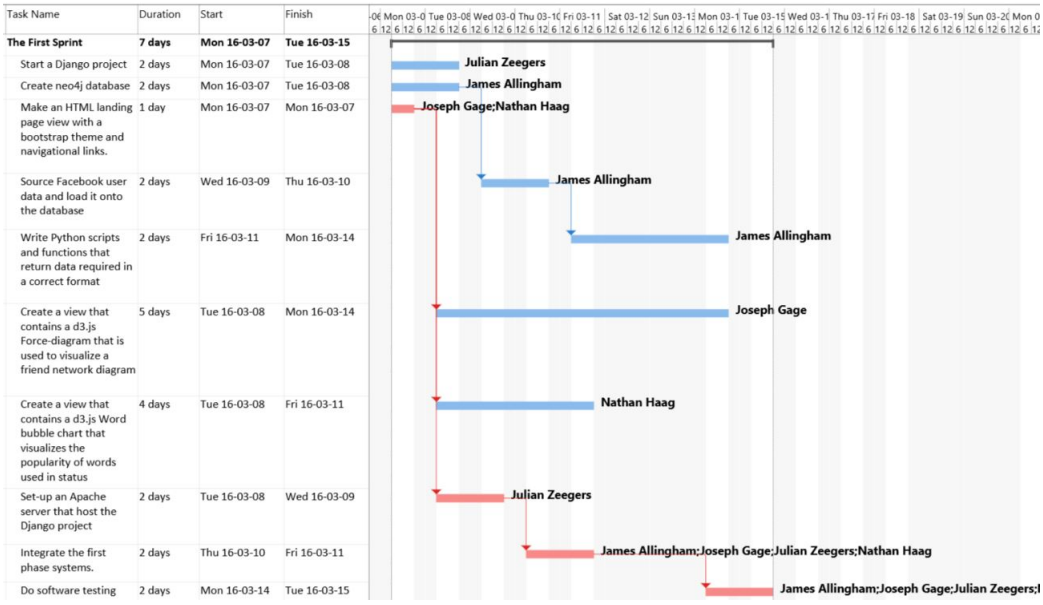


Figure 18: Screen shot of Gantt Chart

6 Sprint Retrospective

Once a sprint is completed, there is a meeting conducted in which the sprint is reviewed called the Sprint Retrospective Meeting. During this meeting, many items related to the sprint are discussed which including the successful goals achieved, the goals that were not achieved, the challenges that were faced and how well the Scrum team worked together. As mentioned, at the end of a sprint, a shippable product must be created that achieves the basic sprint goals (as shown by the label '3' in Figure 15). The sprint product is reviewed in this meeting and the Scrum team members are also given an opportunity to express there opinion. This section discusses all these aspects of the sprint retrospective.

6.1 The Goals Achieved and Positive Outcomes

After the first sprint, a shippable prototype of the Tracking Facebook Interconnected Links project was produced by the Scrum team. The following

goals were successfully achieved:

- A web application with a landing page and two functional views was created.
- The data for the views was stored in a Neo4j graph database and the required data was pulled for the database using Cypher queries and Python functions.
- The web application was hosted on an Apache web server.

The sprint's end product is functional and a detailed user guide document was written up which instructs the user on how to use the product. All the initial system environments were successfully set up such as the database, server, Django project and bootstrap theme which will not need to be done again in future sprints. This allows the next sprint to be able to focus more on creating more views and increasing functionality.

In terms of the working environment, the sprint process was relatively well organised which allowed these goals to be achieved. The Scrum team made effective use of the Trello and Github platforms. In particular, the Trello board was updated regularly with completed and new tasks which allowed the team to keep track of progress. The Github repository was also used effectively for integrating and storing the various tasks.

The sprint was also well documented which will make progressing in the next sprint easier. A fully detailed SRS was created and can be used in future sprints to ensure that the progress is in line with what is required. A design document, implementation document and user guide was written which will also help with future project progression.

6.2 The Negative Outcomes

Although the first sprint achieved the goals that were set up for the sprint, a few negative outcomes occurred during the sprint. It is important to identify these negative aspects in order to improve them for the next sprint. These negative outcomes include the following:

- The overall layout, design and chosen bootstrap theme used for the web application lacks attractiveness.
- There is not enough data in the database. Although the sprint's end product is only a prototype, much more data is required to make the graphs for each view more functional.
- The created views are not as interactive as they should be. More functionality must be added to them.
- Full use of the Django project as the 'models' aspect of Django was not utilised.

One negative aspect of the working environment was a lack of communication between the Scrum members at times. The Slack messaging platform was not fully utilised by the members to communicate. Often communication about challenges and problems faced by a member was dealt with in person with only one or two other members present. This meant that many times the entire team was not involved in that communication and therefore input and insight from the missing members could not happen.

Another problem faced was that the tasks were sometimes completed after the planned deadlines. This caused a slight build up of work towards the end of the sprint and final integration tasks were rushed. These mentioned negative outcomes need to be considered when planning the next sprint as this will improve the overall final project product.

6.3 The Sprint Retrospective Meeting Feedback

The sprint retrospective meeting gives the scrum team an opportunity to reflect on the previous sprint and and serves as an opportunity for team members to give their own input on how to improve for the next sprint. Table 4 shows the feedback given by each member during this meeting.

6.4 The Improvements For the Next Sprint

One of the main purposes of the sprint retrospective meeting is to highlight improvements that can be made for the next sprint and suggest plans on

Table 4: The Sprint Retrospective Meeting Feedback

	Feedback Questions		
Member's Name	What went well during the last sprint?	What did not go well during the last sprint?	How can the team improve for the next sprint?
James Allingham	The team communicated well and the work was all completed on time.	The deadline for the prototype was very stressful as it was almost missed.	Work should be better integrated from the start to prevent last minute integration problems.
Joseph Gage	Delegation of tasks was completed early on and each member was committed to finishing his tasks well and on time.	A proper code structure and design was not discussed before the commencement of work which led to inefficiencies in integration and testing.	More thorough project planning in terms of code structure and architecture can be planned before the project begins.
Nathan Haag	Team communication and delegating of tasks was efficient and clear cut. That being said, team interaction and help was still a contributing factor to team success.	Clearly defined requirements as to what was needed from the "client" was not a part of the sprint, which was frustrating.	Integrate all front-end visualisations with back-end, retrieving real data and create visualisations with bigger data.
Julian Zeegers	Each member of the team knew their respective responsibility and the Trello web application work well to ensure that was the case.	There was not a lot of time left for integrating the components and the integration was therefore unnecessarily stressful.	A better estimation for the time each task takes is required. A better visual design of the web application must be implemented.

how to implement these improvements. The negative outcomes mentioned in Section 6.2 must be improved and the following are suggestions that could improve them:

- The design and layout can be improved by using a better bootstrap template. It is suggested that the 'Dashboard' template should be used.
- In order to obtain more Facebook data, the database should use a Facebook API to continuously pull data from the Facebook site.
- Hosting the Apache server on a dedicated host computer server.

During the next sprint meetings, better task time estimation must be made, especially tasks that involve integrating the project components. The use of the Slack messaging platform must also be encouraged to allow the entire team the opportunity to help fix problems. Although the first sprint has these mentioned improvements that must be implemented, the outcome of the first sprint was successful and it has produced a good software development foundation for the future increments of the project.

7 Conclusion

The requirements, design, implementation and testing of a two-tier web-based application for visualising interconnected Facebook links has been presented. The group work aspects of the project, including SDLC as tools for communication and organisation, have been discussed. The prototype application showed that the selected technologies and tools are capable of delivering the required features and specifications. Future work for the project as well as improvements in the performance of the development team have been suggested.

References

[1] Statista. `http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/`. Last accessed 18 February 2016.

[2] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Newtown Square, Pa: Project Management Institute, 2004. ch 13. p 390.

[3] H. van Vliet, *Software Engineering: Principles and Practice* Wiley, 2007.

[4] N. Liyanage. *Client/Server Architecture: Advantages and Disadvantages of the architectures*. 2013. `http://clientserverarch.blogspot.co.za/2013/03/advantages-and-disadvantages-of.html` Last accessed: 9 March 2016

[5] R. Stephens, Beginning Software Engineering, 1st ed. Indianapolis: John Wiley And Sons, Inc, 2015, pp. 94-95.

[6] *Get Bootstrap - 3.3.6* `www.getbootstrap.com` Last accessed: 9 March 2016.

[7] *d3.js - Data Driven Documents* `www.d3.com` Last accessed: 9 March 2016.

[8] Django Software Foundation. *Django Overview*. `https://www.djangoproject.com/start/overview/`. 2016. Last accessed: 9 March 2016.

[9] Apache Software Foundation. *Apache - HTTP Server Project*. `https://httpd.apache.org/ABOUT_APACHE.html`. 2016. Last accessed: 9 March 2016.

[10] neo4j. `http://neo4j.com/`, Last accessed 18 February 2016.

[11] Py2Neo. `http://py2neo.org/2.0/`, Last accessed 18 February 2016.

[12] Adrian Holovaty, Jacob Kaplan-Moss, et al. *The Django Book*. `http://www.djangobook.com/en/2.0/index.html#`. Ch 3. Last accessed: 9 March 2016.

[13] Django Software Foundation. *How to install Django*. `https://docs.djangoproject.com/en/1.9/topics/install/`. 2016. Last accessed: 9 March 2016.

[14] Apache Software Foundation. *Apache - HTTP Server Project*. `https://httpd.apache.org/ABOUT_APACHE.html`. 2016. Last accessed: 9 March 2016.

[15] Robinson I, Webber J, Eifrem E. *Graph Databases* O'Reilly Media. ch 2. pp 21 - 22. June 2013.

[16] IEEE Standards Board. *IEEE Guide to Software Design Descriptions*. IEEE, New York, 25 May 1993.

[17] JSON. http://www.json.org/, Last accessed 10 April 2016

[18] *The Advantages of using SVG* http://www.todaysoftmag.com/article/1067/the-advantages-of-using-svg-scalable-vector-graphics Last accessed: 10 March 2016.

[19] mod_wsgi. https://modwsgi.readthedocs.org/en/develop/, Last accessed 7 April 2016.

[20] Filesystem Hierarchy Standard Group. *Filesystem Hierarchy Standard*. http://www.pathname.com/fhs/pub/fhs-2.3.html. 2004. Last accessed: 10 March 2016.

[21] M, Cohn *What Are Agile & Scrum*, Available from: https://www.mountaingoatsoftware.com/, Last accessed: 10/04/2016

[22] S, Haunts *Agile Software Development Succinctly*, Technology Resource Portal, Syncfusion Inc, Morrisville USA (2015)

[23] Trello. https://trello.com/, Last accessed 18 February 2016.

[24] Slack. https://slack.com/, Last accessed 18 February 2016.

A Use Cases

pre-requisites: It is assumed that the server is operating correctly and that the client is running a modern web browser compatible with the versions of HTML, CSS and JS used in the application. It is also assumed that the client has an internet connection.

A.1 Use Case 1: Log In

A.1.1 Basic Flow

1. The user launches a browser.
2. The user navigates to the URL of the web application.
3. The user clicks the *log in* button on the default home page.
4. The user enters their user name and passwords in the appropriate fields.
5. The user clicks the *log in* button on the login page.
6. The user is taken to their personal landing page.

A.1.2 Alternate Flows

6a. The user enters the incorrect login details:

1. The user is notified of their mistake.
2. The user enters the correct details.
3. The user is taken to their personal landing page.

6b. The user enters the incorrect login details and is unable to correct them:

1. The user is notified of their mistake.
2. The user requests that their password be reset.
3. A password reset link it emailed to the user.

A.2 Use Case 2: Sign Up

A.2.1 Basic Flow

1. The user launches a browser.
2. The user navigates to the URL of the web application.
3. The user clicks the *sign up* button on the default home page.
4. The user enters their account details on the sign up page. These include: user name, password and email address.
5. The user is informed that they have been sent a verification email address.
6. The user goes to their email and clicks the validation link sent to them.
7. The user is taken back to the default home page, where they can now log in.

A.2.2 Alternate Flows

7a. The user does not receive the verification email:

1. The user is able to request a second email.

7b. The user's email address is incorrect:

1. The user is able to change their email address and request a second email.

A.3 Use Case 3: Connect to Facebook

A.3.1 Basic Flow

1. The user clicks the *connect to Facebook* button on their personal landing page.

2. The user fills in their Facebook log in details.

3. The user clicks the *submit* button.

4. The user is returned to their personal landing page, where they can access visualisations of their Facebook data.

A.3.2 Alternate Flows

4a. The user enters the incorrect login details:

1. The user is notified of their mistake.

2. The user enters the correct details.

3. The user is taken to their personal landing page, where they can access visualisations of their Facebook data.

4b. The user enters the incorrect login details and is unable to correct their mistake:

1. The user is notified of their mistake.

2. The user is taken to their personal landing page, where they can only access visualisations of old data.

A.4 Use Case 4: Upload Data

A.4.1 Basic Flow

1. The user clicks the *upload data* button on their personal landing page.

2. The user selects the file path of the data stored in JSON format.

3. The user clicks the *submit* button.

4. The user is returned to their personal landing page, where they can access visualisations of their Facebook data.

A.4.2 Alternate Flows

4a. The file the user selects is invalid:

1. The user is notified of their mistake.
2. The user enters the correct details.
3. The user is taken to their personal landing page, where they can access visualisations of their uploaded data.

4b. The file the user selects is invalid and is unable to correct their mistake:

1. The user is notified of their mistake.
2. The user is taken to their personal landing page, where they can only access visualisations of old data.

A.5 Use Case 5: View Friend Network

A.5.1 Basic Flow

1. The user clicks the *view friend network* button on their personal landing page.
2. The user is taken to the friend network visualisation page where they have additional options for visualisation.

A.5.2 Alternate Flows

2a. The user does not have any data to visualise

1. The user is notified of the problem.
2. The user is taken back to their personal landing page.

A.6 Use Case 6: View Friendship Data

A.6.1 Basic Flow

1. The user clicks the *view friendship data* button on the friend network visualisation page.
2. The visualisation on the page changes to show the extra information about friendship links.

A.6.2 Alternate Flows

2a. The user does not have any extra friendship data to visualise

1. The user is notified of the problem.
2. No changes occur to the visualisation.

A.7 Use Case 7: View Relationship Data

A.7.1 Basic Flow

1. The user clicks the *view relationship data* button on the friend network visualisation page.
2. The visualisation on the page changes to show the extra information about relationship links.

A.7.2 Alternate Flows

2a. The user does not have any extra relationship data to visualise

1. The user is notified of the problem.
2. No changes occur to the visualisation.

A.8 Use Case 8: View Personal Data

A.8.1 Basic Flow

1. The user clicks the *view personal data* button on the friend network visualisation page.
2. The visualisation on the page changes to show the extra information about the friend nodes.

A.8.2 Alternate Flows

2a. The user does not have any extra personal data to visualise

1. The user is notified of the problem.
2. No changes occur to the visualisation.

A.9 Use Case 9: View Other Visualisations

A.9.1 Basic Flow

1. The user clicks any other visualisation buttons on their personal landing page.
2. The user is taken to the appropriate visualisation page where there will be visualisation specific options.

B The Minutes from the First Sprint Planning Meeting

Tracking Interconnection Facebook Links Project
The First Sprint Planning Meeting

Internal Meeting Notes: 5 March 2016

Attendance: *J. Zeegers, J. Allingham, J. Gage, N. Haag , The Product Owner, The Scrum Master, The Project Administrator*
Apologies: The Marketing Manager
Absent: -
Minutes noted by: J. Zeegers
Meeting at 4th Year Common Room,
University of Witwatersrand

Opening 10h00

- The Scrum Master chairs meeting

Project Description

Item	Person Responsible	Projected Date to Complete by	Done
The project brief was discussed and project objectives highlighted.	-	-	
The Sprint goals were made and include the following: • Create a system prototype that includes one landing page and two functional views. • Create and populate the database. • Set-up the system's architecture such as setting up the framework and server.	-	-	

The Task Allocation

Item	Person Responsible	Projected Date to Complete by	Done
The tasks for the back-end and front-end teams were defined.	-	-	
The following tasks were chosen from a project back-end team member : • Create neo4j database • Source Facebook user data and load it onto the database	*James Allingham*	1 April 2016	

• Write Python scripts and functions that return data required in a correct format • Integrate the first phase systems. • Do software testing			
The following tasks were chosen from a project back-end team member : • Start a Django project • Set-up an Apache server that host the Django project • Integrate the first phase systems. • Do software testing	*Julian Zeegers*	1 April 2016	
The following tasks were chosen from a project back-end team member : • Make an HTML landing page view with a bootstrap theme and navigational links. • Create a view that contains a d3.js "Force-diagram" that is used to visualize a friend network diagram • Do software testing	*Joseph Gage*	1 April 2016	
The following tasks were chosen from a project back-end team member : • Make an HTML landing page view with a bootstrap theme and navigational links. • Create a view that contains a d3.js "Word bubble chart" that visualizes the popularity of words used in status • Do software testing	*Nathan Haag*	1 April 2016	

The Work Environment

Item	**Person Responsible**	**Projected Date to Complete by**	**Done**
A Github repository for the sprint must be set-up	*The Project Administrator*	6 March 2016	
Create a new Trello board for the first Sprint. Add all the tasks and the associated team members responsible for the task as well as the dead line.	*The Scrum Master*	6 March 2016	
Ensure all project members have joined the Slack messaging group for the first Sprint.	*The Project Administrator*	6 March 2016	
There will be a daily Scrum meeting each working day of the sprint. The venue for this meeting is the 4th Year Common Room. Each meet will start at 9 am daily.	-	-	

The Documentation

Item	Person Responsible	Projected Date to Complete by	Done
Write a Full Software Requirements Specifications	*James Allingham, Joseph Gage*	1 April 2016	
Write the Design Document	*Joseph Gage, James Allingham*	1 April 2016	
Write the Implementation Document	*Nathan Haag*	1 April 2016	
Write user guide for product prototype	*Julian Zeegers*	1 April 2016	

General Comments

- There is a Sprint Retrospective Meeting planned for the 4th of April to review the first sprint.
- The second sprint is planned to commence during the week of the 4th of April.

Meeting Closed 16h15