



National College of Ireland

BSc in Computing

2016/2017

Graham Murray

13504987

graham.murray@student.ncirl.ie

Supervisor: Mr Vikas Sahni

Making social network analysis available to everyone. A social
network analysis platform in the cloud.

Exograph

Technical Report

Exograph Technical Report

Project Declaration

SECTION 1

Name: Graham Murray
Student ID: 13504987
Supervisor: Vikas Sahni

SECTION 2: Confirmation of Authorship

I confirm that I have read the College statement on plagiarism and that the work I have submitted for assessment is entirely my own work.

Signature: _____

Date: _____

Exograph Technical Report

Table of Contents

Executive Summary.....	8
1 Introduction.....	9
1.1 Background.....	9
1.2 Aims.....	10
1.3 Technologies.....	10
1.3.1 Python.....	10
1.3.2 Neo4J.....	11
1.3.3 NodeJS.....	11
1.3.4 Jade.....	11
1.3.5 Data Driven Documents (D3.js).....	12
1.4 Structure.....	12
2 System.....	13
2.1 User Requirements.....	13
2.1.1 Friendliness.....	13
2.1.2 Self Intuitive.....	13
2.1.3 Pages Must Load Within 5 Seconds.....	13
2.1.4 Delete and Restore Graphs.....	13
2.1.5 Interactive Visualisations.....	13
2.1.6 Work Seamlessly Across Major Browsers.....	14
2.1.7 Export a Graph as an Image or in Raw Format.....	14
2.1.8 Provide a Charted Summary of a Graph.....	14
2.2 Functional Requirements.....	15
Requirement 1 <User Authentication>.....	15
Requirement 2 <Import Data>.....	18
Requirement 3 <Manipulate Graph>.....	21
Requirement 4 <Export Graph to Image>.....	24
Requirement 5 <View Charted Summary>.....	27
Requirement 6 <Edit User Account>.....	29
2.3 Non-Functional Requirements.....	31
2.3.1 Performance Requirement.....	31
2.3.2 Recovery Requirement.....	31

Exograph Technical Report

2.3.3	Scalability Requirement	31
2.3.4	Reliability Requirement	32
2.3.5	Security Requirement	32
2.3.6	Maintainability Requirement.....	32
2.3.7	Usability Requirement	32
2.3.8	Reusability Requirement.....	33
2.3.9	Interoperability Requirement	33
2.4	Design and Architecture.....	34
2.4.1	Design Approach and Standards Overview	34
2.4.1.1	Bottom up Approach	34
2.4.1.2	Service Encapsulation	34
2.4.1.3	Single Responsibility	34
2.4.1.4	Data Representation Standards.....	35
2.4.1.5	Communication Standards	35
2.4.2	Architecture Design	36
2.4.2.1	Class Diagram.....	36
2.4.2.2	Sequence Diagram	37
2.4.2.3	Hardware Architecture	38
2.4.2.4	Software Architecture.....	39
2.4.2.4.1	UI Server.....	39
2.4.2.4.2	Extractor Service	40
2.4.2.4.3	Security Architecture	42
2.4.2.4.4	Communication Architecture	43
2.4.3	Database Design	44
2.4.3.1	Neo4J Schema.....	44
2.4.3.2	MySQL Extractor Service Schema	45
2.5	Implementation.....	46
2.5.1	Twitter Crawler	46
2.5.2	LinkedIn Crawler	48
2.5.3	Bulk Loader	50
2.5.4	UI Server Generating a D3 Graph Model.....	52
2.6	Graphical User Interface	53

Exograph Technical Report

2.6.1	Import Graph	53
2.6.2	Graph Dashboard.....	54
2.6.3	Graph View	55
2.6.4	Graph Analytics.....	56
2.6.5	Graph Trash	57
2.6.6	User Profile	58
2.7	Testing.....	59
2.7.1	Unit Testing.....	59
2.7.1.1	Extractor.....	60
2.7.1.2	UI Server.....	61
2.7.2	Integration Testing.....	62
2.7.2.1	Extractor.....	62
2.7.2.2	UI Server.....	63
2.7.3	Customer Testing.....	64
2.7.3.1	Import your LinkedIn Network.	65
2.7.3.2	View the Newly Imported Graph.....	66
2.7.3.3	Send a Graph to Trash	68
2.7.3.4	Final Customer Testing	69
2.8	Evaluation.....	70
2.8.1	Performance Testing.....	70
2.8.1.1	LinkedIn Crawler Import without Caching.....	70
2.8.1.2	Twitter Crawler Import without Caching.....	71
2.8.1.3	Twitter Crawler Import with Caching	72
2.8.1.4	Twitter Import Caching Comparison	73
2.8.1.5	Neo4j Query Performance	74
2.8.2	User Satisfaction March.....	75
2.8.3	User Satisfaction May	76
2.9	Deployment.....	77
3	Conclusions.....	78
4	Acknowledgements	79
5	Further Development	80
5.1	Advanced Analytics.....	80

Exograph Technical Report

5.2	Micro Services.....	80
5.3	Graph Merging.....	80
5.4	Custom Graphs	80
6	References	81
7	Appendix.....	82
7.1	User Manual.....	82
7.1.1	Importing a Graph	82
7.1.2	Viewing an Imported Graph	83
7.1.3	Deleting a Graph.....	84
7.1.4	Accessing Graph Analytics	85
7.1.5	View Executed Jobs	86
7.1.6	Updating Account Information.....	87
7.1.7	Changing a Profile Image.....	88
7.2	Project Proposal.....	89
7.3	Monthly Journals	94
7.3.1	September	94
7.3.2	October.....	96
7.3.3	November.....	100
7.3.4	December	103
7.3.5	January.....	104
7.3.6	February.....	107
7.3.7	March.....	110

Exograph Technical Report

Table of Figures

Figure 1: User Authentication Use Case	15
Figure 2: Import Data Use Case Diagram.....	18
Figure 3: Manipulate Graph Use Case Diagram.....	21
Figure 4: Export Graph to Image Use Case Diagram.....	24
Figure 5: View Charted Summary Use Case Diagram	27
Figure 6: Edit User Account Use Case Diagram	29
Figure 7: System Class Diagram	36
Figure 8: System Sequence Diagram	37
Figure 9: System Hardware Architecture.....	38
Figure 10: UI Server Software Architecture.....	39
Figure 11: Extractor Service Architecture	40
Figure 12: Exograph Security Question.....	42
Figure 13: System Communication Architecture	43
Figure 14: Neo4J Database Schema.....	44
Figure 15: Extractor Service Database Schema	45
Figure 16: Gathering a User’s Twitter Friends	46
Figure 17: Retrieving Twitter Friends of Friends	47
Figure 18: Creating a List of Edges.....	47
Figure 19: Configuring Local Cookie Opener	48
Figure 20: Automated LinkedIn Login	49
Figure 21: Requesting a User’s LinkedIn Connections.....	49
Figure 22: Bulk Loader Generating Node Insert Statements.....	50
Figure 23: Bulk Loader Generating Edge Statements	51
Figure 24: UI Server, Generating a D3 Graph Representation	52
Figure 25: Import Graph Page.....	53
Figure 26: Dashboard Page	54
Figure 27: Viewing a Graph.....	55
Figure 28: Graph Analytics Page	56
Figure 29: Trash Page, Delete and Restore.....	57
Figure 30: Profile Page with Profile Image Upload.....	58
Figure 31: Extractor Python Unit Test.....	60
Figure 32: UI Server User Model Unit Test	61
Figure 33: Extractor Integration Test.....	62
Figure 34: UI Server Selenium Login Integration Testing	63

Exograph Technical Report

Table of Tables

Table 1: Customer Testing LinkedIn Import Results	65
Table 2: Customer Testing View Graph Results	67
Table 3: Customer Testing Sending a Graph to Trash Results	68
Table 4: Total LinkedIn Network Import Time no Caching	70
Table 5: Total Twitter Network Import Time no Caching	71
Table 6: Total Twitter Network Import Time with Caching	72
Table 7: Twitter Import Time Caching Performance Improvement	73
Table 8: Neo4j Sub Graph Retrieval Response Time	74
Table 9: User Satisfaction Ratings on Key Pages in March	75
Table 10: User Satisfaction Ratings in May	76

Exograph Technical Report

Executive Summary

This project involved the implementation of a social network analysis system that facilitates Recruitment Consultants when finding potential candidates for a role. The primary aim of the project was to implement a functional system that meets the needs of a Recruiter in a high paced role. This report describes and evaluates a range of different areas relating to the overall chosen solution. A lack of commercially available solutions to the problem were identified early on during various brainstorming sessions. The solution is called Exograph.

Initially, differing sources that would provide data to the application were identified. Small proof of concept programs written in Python were then developed to test the application and estimate the level of effort required to carry out the task on a production scale. Problems were encountered gaining access to the required data from LinkedIn during the early stages. User Interface prototypes were developed to test the integration of a data visualisation framework and ensure it had all the required functionality to meet the needs of the functional requirements outlined in the Requirements Specification. During this time, alternative solutions were evaluated. An alternative solution to the original problem was to change the targeted end-user if access to the LinkedIn professional network was not acquired. The alternative solution allows anyone interested in visualising their social network to have access to the system to perform such a task. As the size of a person's network can differ depending on their popularity, this is a factor that has an impact on performance in a production environment.

Exograph Technical Report

1 Introduction

1.1 Background

Social Network theory has been in existence for a long time. People have been analysing social networks since medieval times. With the rise of the digital age, the capabilities of such analytics have become more accessible. The application of the area is used extensively by historians, sociologists, and economists alike.

In today's world, millions of people connect with each other using social network mediums such as Facebook, Twitter, LinkedIn and Instagram. Social Networks can be described as a graph of people who come in contact with each other through such a medium. Everyday there are massive amounts of User generated data as a result of such relationships. People constantly rely on these mediums to keep in contact with each other and meet new people, forging new relationships. From a Computer Science aspect this opens a world of possibilities to exploit such data. Graph theory is a very powerful mathematical concept which can be applied to solving such problems and representing relationships between groups of people.

From the point of view of a Recruitment Consultant looking to hire someone, it can be a tedious process manually attempting to see who they are connected to and finding people who are connected with the same profession to find a suitable person for a role. A Recruiter would be a prime example of a person who would connect with people in a cohesive way. As the old saying goes, good people know other good people.

Data visualisation is the graphical representation of data in an all-inclusive way. It enables us to perceive things that would not be otherwise possible. By combining such social network graphs and visually representing them, we can get new insights from data about relationships that we did not even know existed.

Exograph Technical Report

1.2 Aims

The main aim of the Project was to develop an application that enables Recruitment Consultants and others to visually analyse their social connections. The solution helps Recruiters to understand how they are connected with other people. From the perspective of a Recruiter, the solution facilitates them when finding people who are commonly connected to either themselves or others in order to identify a suitable person for a job they may be hiring for. The system is also available to anyone wishing to explore their networks. The solution utilises multiple channels such as LinkedIn and Twitter as data sources. The project demonstrates multiple Computer Science concepts such as Graph Theory, Parsing, Distributed Computing, and Data Visualisation. The process of gathering and performing computations on the data should be transparent to the end-user. From the time they launch an import to when the graph is rendered they are unaware of anything regarding the process. The application is based around Software as a Service model in a cloud environment and is available to Users at anytime, anywhere.

1.3 Technologies

1.3.1 Python

Python is a functional programming language that is both easy to learn and use. The first part of the project focused on extracting data from third party APIs. This required a powerful functional language that could process data swiftly in an environment with minimal resource requirements. This component is the most vital as it needs to perform efficiently and scale as the load on the application grows. There is an abundant amount of community developed libraries for interacting with APIs such as those provided by Twitter. An example is Tweepy which provides a wrapper around the Twitter API and abstracts away the complexities associated with it (Roesslein, J. et. al 2009). There are other functional programming languages such as Lisp and R. They were not chosen as research revealed that there are more Python libraries available that meet the needs of the project.

Exograph Technical Report

1.3.2 Neo4J

The domain concept of the project focuses on graphs and graph oriented data structures. Neo4J is a scalable graph database management system that provides the data persistence tier in the application stack. It works most effectively when storing highly connected elements and therefore meets the needs of the non-functional requirements of the system as it supports a distributed clustering architecture and exhibits low latency (Holzschuher, F. and Peinl, R. et. al 2013). Neo4J is a core component in the stack, the data that exists within the Exograph system is highly connected data. For this reason, Neo4J was selected over other solutions such as MySQL and MongoDB. Mongo was the primary potential databases researched in depth but did not fit the needs of the project due to the nature of the data being stored. The same is true for MySQL, query performance would have been an issue as a large amount of foreign key relationships would have been required which would diminish query performance due to complex joins.

1.3.3 NodeJS

Python and Neo4J perform the roles of extracting the data and storing it, but this is only a small part of the solution. A server side technology is needed to support the client side interactions with the system. This is where NodeJS comes in, Node is a JavaScript based runtime environment that allows server side components to be written in JavaScript. It provides a diverse range of packages that could be integrated into the heart of the application to support technologies such as Neo4j and the D3 visualisation framework. Various other server side technologies were evaluated but Node came out on top. PHP was considered but eventually discarded due to the fact it is a low level programming language compared to Node.

1.3.4 Jade

The above technologies cover the essential server side components, but Jade is another key component that provides a view into the system for a User. Jade is a server side templating engine that runs on top of Node allowing mark-up to be written in a simple, elegant manner. The components built with Jade are fully reusable and allow views to be developed quickly while supporting concepts usually seen in programming languages such as loops and conditional statements. Other templating engines such as Handlebars and Eco were investigated and tested. Jade was selected due to its maturity and performance benefits over the other technologies.

Exograph Technical Report

1.3.5 Data Driven Documents (D3.js)

D3 is a powerful data visualisation library that produces interactive client side visualisations. It integrates excellently with Node as there is a set of packages built specifically around incorporating D3 into the ecosystem. D3 provides a rich set of features that gives power to the developer to create complex, custom visualisations. A number of different data visualisation libraries were analysed during the early stages of the project. Vis.js was a potential option but focused on many different visualisations and tried to do everything well. D3 was selected because of its rich set of functionality and high level of control over the end visualisation result.

1.4 Structure

Section 2 (System) outlines the system, describing what it should do and how it should do it. It includes User Requirements, Functional Requirements, Non-Functional Requirements, Design and Architecture and the Graphical User Interface. There is a sub section that focuses on testing, mainly in the areas of Unit Testing, Integration Testing and Customer Testing. An evaluation section presents the results of system analysis in a number of areas.

Section 3 (Conclusions) summarises the findings outlined in the paper.

Section 4 (Acknowledgments) a section dedicated to thanking the people who made the project possible.

Section 5 (Future Developments) discusses the possible future developments of the project if more resources were available

Section 6 (References) is a list of all references used during the development of the project. The style of referencing used is the Harvard style.

Section 7 (Appendix) consists of four sub sections. The User Manual, Project Proposal document, Monthly Learning Reflection Journals and the Project Plan that was followed to complete the project.

Exograph Technical Report

2 System

2.1 User Requirements

2.1.1 Friendliness

The system shall be friendly for the end-user. This should be achieved by having an attractive and elegant User Interface (UI). The UI should be menu drive such that it empowers a User to find everything they require through a menu driven interface.

2.1.2 Self Intuitive

The system shall not require any complicated manuals. A User should be familiar with the system before they use it. This should be achieved by designing the system using UI design patterns which a User would have seen previously while using other consumer applications.

2.1.3 Pages Must Load Within 5 Seconds

The system shall ensure that all pages' load within a maximum of 5 seconds or less. Users often get annoyed if the page response time is too long. Therefore, the system must perform within a defined threshold of 5 seconds.

2.1.4 Delete and Restore Graphs

The system shall allow a User to send items to a trash bin and subsequently allow them to choose further actions including delete and restore. The trashed graphs should be viewable in one location to compliment the requirement of friendliness.

2.1.5 Interactive Visualisations

The system shall allow Users to interact with graph visualisations by highlighting neighbouring nodes along with their edges to help easily view the relationships. Nodes should be moveable when a User drags them.

Exograph Technical Report

2.1.6 Work Seamlessly Across Major Browsers

The system shall work in the exact same manner across different supported browsers. The browsers the system should work on are Google Chrome, Firefox and Internet Explorer 12. The behaviour and interactions should not change across different browsers.

2.1.7 Export a Graph as an Image or in Raw Format

The system must allow a User to export the current state of a graph. The supported export formats should be Scalable Vector Graphics (SVG). An exported image should be of acceptable quality such that all the contained components are preserved.

2.1.8 Provide a Charted Summary of a Graph

The system shall provide a consolidated summary of the information contained within a graph. This should be achieved using graphs and tables to display metrics such as total number of nodes, categorize different groups, degree of connectivity, reciprocity, and degree of separation.

2.2 Functional Requirements

Requirement 1 <User Authentication>

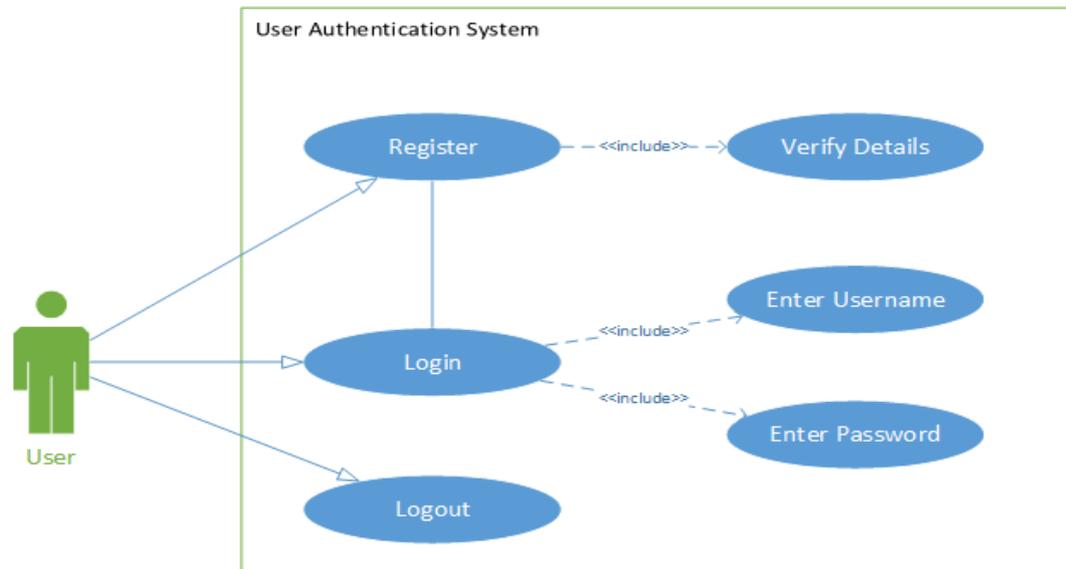


FIGURE 1: USER AUTHENTICATION USE CASE

Description & Priority

The system shall provide an authentication and registration mechanism to provide Users with access to the system. This priority has the highest priority of 10/10.

Use Case

Scope

The scope of this Use Case is to demonstrate the behaviour of the system for a User who needs to gain access to the system prior to using it.

Description

The Use Case describes how a User will register if they do not have an account and shows how they will login to be authenticated.

Exograph Technical Report

Use Case Diagram

The actor in this case is a User who is wishing to authenticate. They appear as a green silhouette. The blue ovals indicate the actions performed by the actor.

Flow Description

The User first must first register if they do not already have an account, they will then be required to enter their credentials before being authenticated.

Precondition

1. The system currently does not have a User already authenticated from a previous session.
2. The User has an active internet connection.

Activation

The Use Case begins when a User needs to login to the system to use any of the provided functionality.

Main flow

1. The system is at the login screen <See A1>.
2. The User enters their username and password.
3. The User clicks the login button.
4. The system checks the user exists (See E1).
5. The system verifies the password is correct (See E2).
6. After the User is finished they logout.

Alternate flow

A1: <Unregistered User>

1. The User enters their details.
2. The system checks they do not already have an account.
3. The main flow continues at #2.

Exograph Technical Report

Exceptional flow

E1: <Incorrect Username>

1. The system is at the login screen.
2. The system generates a message warning the User.
3. The User re-enters their username.
4. The main flow continues at #2.

E2: <Incorrect Password>

1. The system is at the login screen.
2. The system displays a warning message.
3. The User re-enters their password.
4. The main flow continues at #2.

Termination

The system takes the User to the Dashboard.

Post condition

The system waits for a User to perform an action.

Exograph Technical Report

Requirement 2 <Import Data>

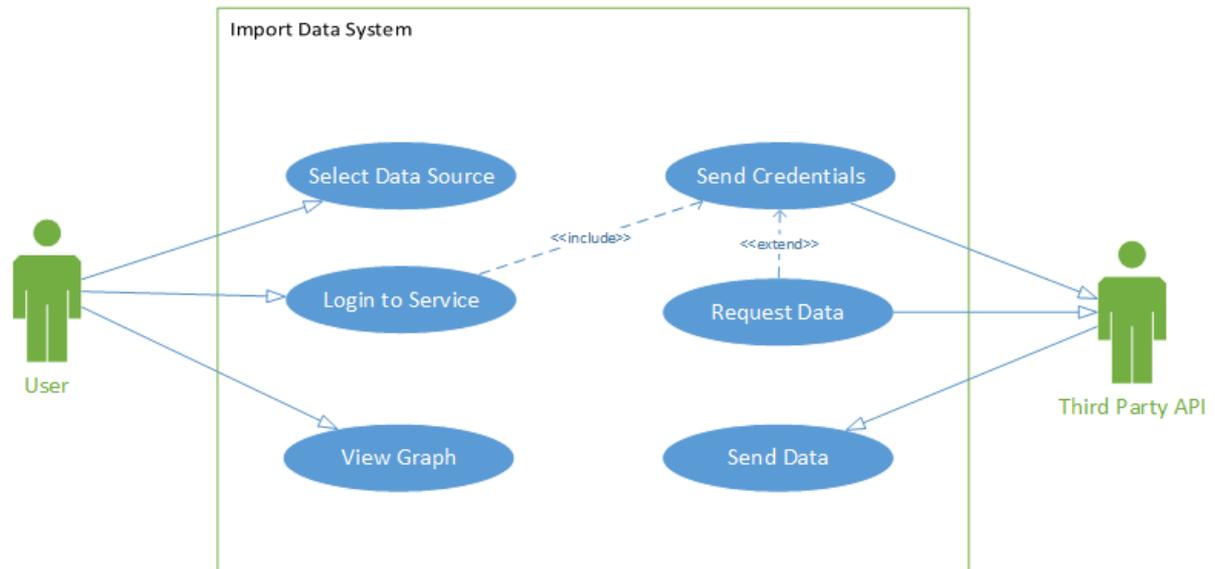


FIGURE 2: IMPORT DATA USE CASE DIAGRAM

Description & Priority

This requirement shall allow a User to import their data from supported sources. The procedure of importing data should appear the same for each data source and be transparent to a User. This requirement has a priority of 9/10.

Use Case

Scope

The scope of this Use Case is to illustrate how a User can import data into the system by detailing the flow of events that are required to fulfil the requirement and describes the state of the system after each action is performed by a User.

Description

This Use Case demonstrates the actions a User can perform and the behaviour of the system when they want to import data into Exograph from third party data sources.

Use Case Diagram

The actor in this case is a User who wishes to import data into the system for analysis. They appear as a green silhouette, blue ovals indicate the actions performed.

Exograph Technical Report

Flow Description

The User can choose a data source which is supported by the system. They will need to provide their account credentials in the case of a LinkedIn import. When a target is selected, the data is pulled into the system via the Extractor.

Precondition

1. A User must be authenticated within the Exograph system (Requirement 1).
2. A valid internet connection is required.

Activation

This Use Case starts when a User chooses to import data.

Main flow

1. The system is awaiting a User to choose a data source.
2. The User selects a data source.
3. The system makes a call to the API for the User to authenticate.
4. The User enters their details for the third party.
5. The system sends the credentials to the third party.
6. The system is notified whether authentication was successful or not. (See E1).
7. The system makes a request for the data (See E2).
8. The third party API then returns the requested data.
9. The system stores the data.
10. The User then views a generated graph.

Exograph Technical Report

Exceptional flow

E1: <Incorrect Credentials>

1. The system has received a notification that third party authentication failed.
2. The system displays a warning message to the User.
3. The User re-enters their credentials.
4. The main flow continues at #4.

E2: <Data Request Failure>

1. The system receives a warning message.
2. The system goes into a waiting state.
3. After 5 seconds it sends another request.
4. If it fails again a warning is displayed to the User.
5. The flow terminates.

Termination

Upon termination the graph is store and an email notification is sent to the User.

Post condition

The system waits for a User to perform an action.

Requirement 3 <Manipulate Graph>

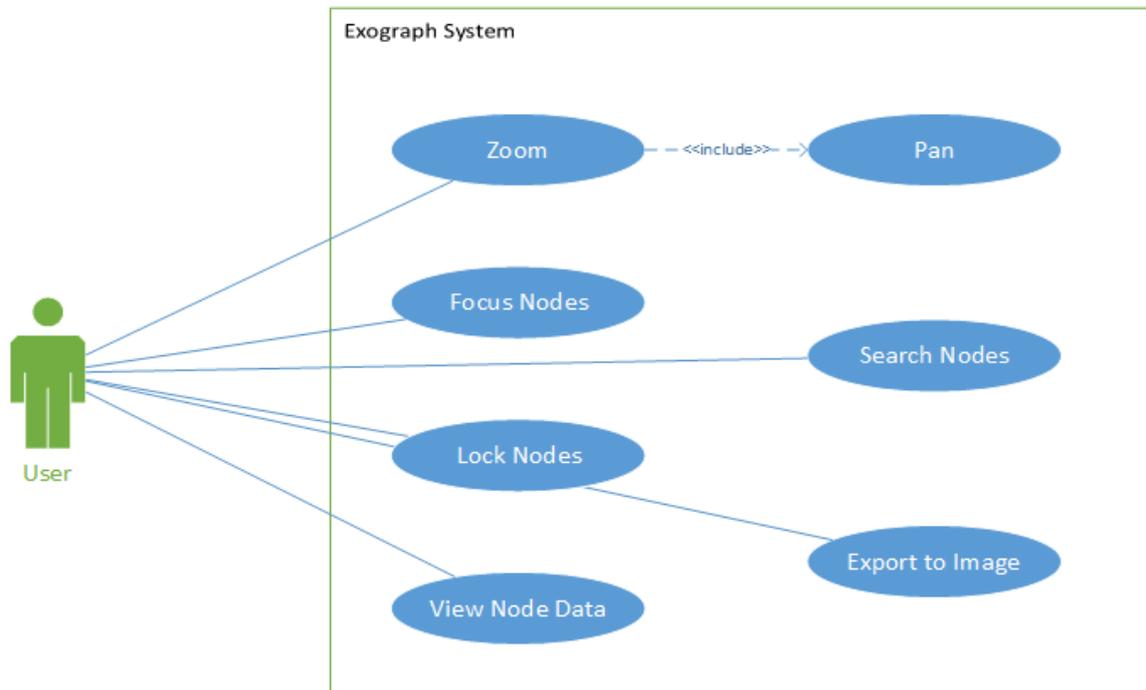


FIGURE 3: MANIPULATE GRAPH USE CASE DIAGRAM

Description & Priority

The system shall allow the manipulation of a rendered graph. This gives the end User greater power when visualising graphs and allows them to explore their networks in depth. This requirement is rated 8/10.

Use Case

Scope

The scope of this Use Case is to show the actions a User can perform on the system while interacting with a generated graph. It details the flow of events that are required to complete this requirement and describes the state of the system after each action is performed.

Exograph Technical Report

Description

This Use Case details the actions that can be performed by a User when viewing a graph in order to improve their understanding of the relationships contained within a graph.

Use Case Diagram

The actor in this case is a User who wishes to manipulate the graph that is initially generated upon the successful importation of data from a third party.

Flow Description

Precondition

1. A User must be authenticated with the Exograph system. (Requirement 1).
2. A valid internet connection is required.

Activation

The Use Case starts when a User selects a data source by either importing new data or retrieving a previously stored social graph.

Main flow

1. The system is waiting for a User to select a data source.
2. The User selects to import a new data source (See A1).
3. The User selects to retrieve a stored data source from a previous session.
4. The system generates a graph.
5. The User then edits the graph.
6. The system makes the required changes.
7. The system updates the graph model.
8. The system updates the rendered graph.

Exograph Technical Report

Alternate flow

A1: <Import from Existing Source>

1. The system takes the User to the import data section (Requirement 1)
2. The User then chooses the newly imported data
3. The Use Case continues at main flow #4

Termination

Upon termination the system will take the User to view all graphs within the system.

Post condition

The system waits for a User to perform an action.

Requirement 4 <Export Graph to Image>

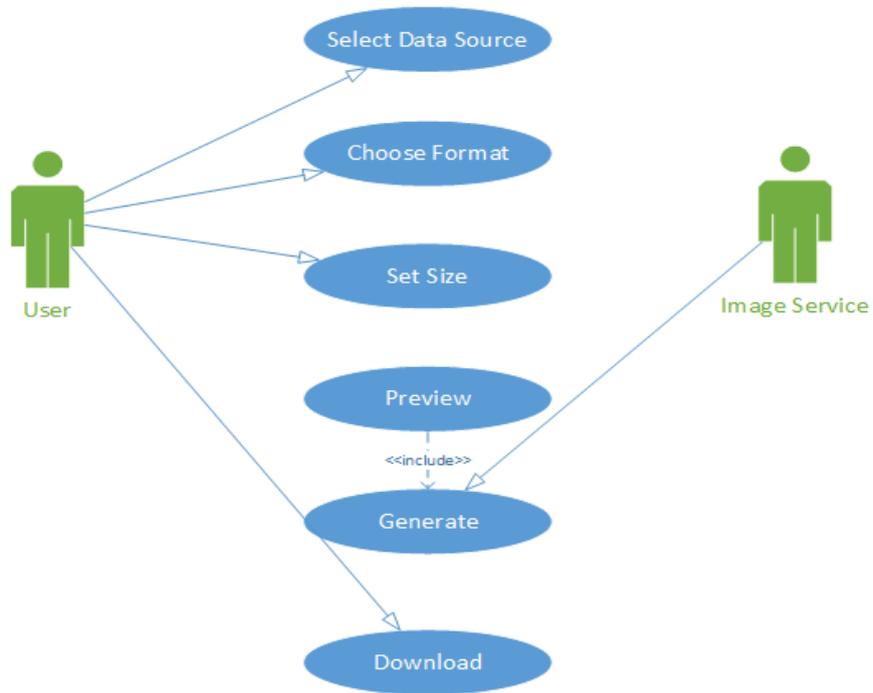


FIGURE 4: EXPORT GRAPH TO IMAGE USE CASE DIAGRAM

Description & Priority

The system shall allow an end-user to export a graph in a supported image format. This is an important requirement that was requested by potential Users. It facilitates the ease of sharing graphs so they can be included in other external documents such as Word documents. This requirement has a rating of 7/10.

Use Case

Scope

The scope of this Use Case is to demonstrate the flow of actions and behaviours when a User is exporting a graph as an image. It shows a high level overview of the flow detailed below.

Exograph Technical Report

Description

Exporting of images is important as it provides a means of viewing a graph at anything without needing to login to the system to view it. In this Use Case there are two main Actors. The User who is requesting an image and the service that generates the image.

Flow Description

The main flow initiates after a User selects a data source, next they can choose various options for customising the final image before downloading it.

Precondition

1. A User must be authenticated with the Exograph system. (Requirement 1).
2. A User must have a graph already in the system.
3. A valid internet connection is required.

Activation

The Use Case begins when a User selects the export to image option while viewing a graph.

Main flow

1. The system is waiting for a User to select a data source.
2. The User selects a data source and chooses to "Export to Image".
3. The system prompts the User to set the size of the image.
4. The User then sets the size (See E1).,
5. The system generates a preview of the image.
6. The User selects if they want to generate it (A1).
7. The image service generates the image.
8. The User then downloads the image.

Alternate flow

A1: <Re Edit Image>

1. The system is prompting the User if they are satisfied with the preview.
2. The User selects No.
3. The system takes the User to main flow #3.

Exograph Technical Report

Exceptional flow

E1: <Invalid Image Size>

1. The system receives and image size too large.
2. The system warns the User.
3. The Use Case return to main flow #3.

Termination

Upon termination system returns to the visualisation.

Post condition

The system waits for a User to perform an action.

Requirement 5 <View Charted Summary>

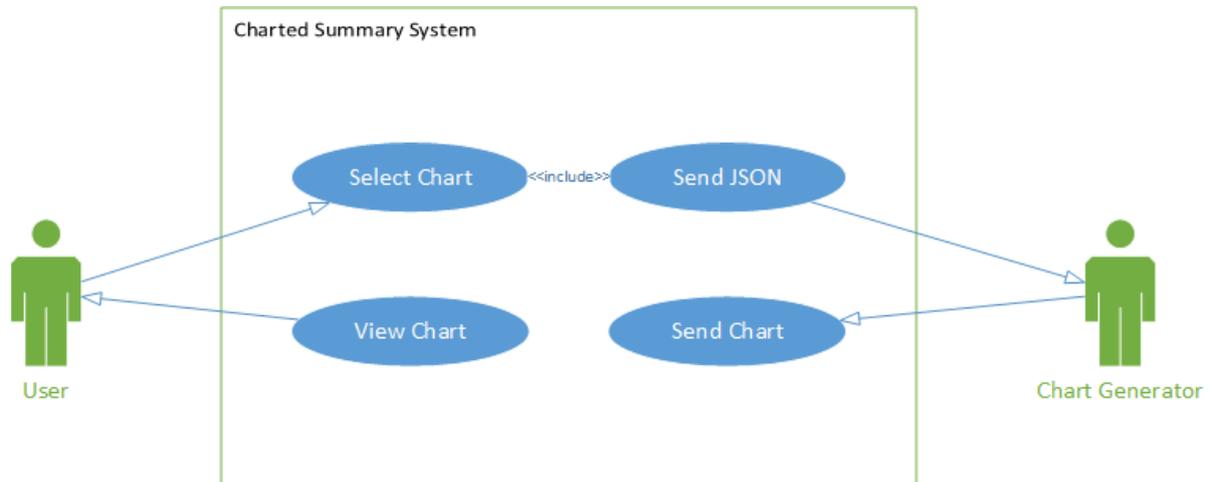


FIGURE 5: VIEW CHARTED SUMMARY USE CASE DIAGRAM

Description & Priority

The system shall provide a charted summary of graphs for the purpose of gaining an overview of the data contained within. This should be achieved using charts and tables to display metrics such as total number of nodes, categorize different groups, degree of connectivity, reciprocity, and degree of separation. This requirement has a priority of 6/10

Use Case

Scope

The scope of this Use Case is to illustrate how a User can interact when wishing to view a graphs metrics. It details a high level overview of the process. Below is a description of the flows and the state after each action is performed.

Flow Description

The main flow initiates after a User selects the “Analytics” option on a graphs related actions. The system uses a third party service to generate the charts before rendering them.

Exograph Technical Report

Precondition

1. A User must be authenticated with the Exograph system. (Requirement 1).
2. A User must have a graph already in the system.
3. A valid internet connection is required.
4. A graph must be selected.

Activation

This Use Case begins when a User selects the “Analytics” option on a graph.

Main flow

1. The User selects “Analytics”.
2. The system sends the data to a Google Charts (See E1).
3. The chart service generates the charts.
4. The chart service sends the charts back to the system (See E2).
5. The system renders the charts.
6. The User views the charts.

Exceptional flow

E1: <Chart service unavailable>

1. The system tries to send the data to the chart service.
2. The chart service doesn't respond.
3. The system retries.
4. The main flow continues at #3.

E2: <Failed to receive chart>

1. The system didn't receive a chart back from the chart service.
1. The system retries.
2. The main flow continues at #3.

Termination

Upon termination the system will return to the Dashboard.

Post condition

The system waits for a User to perform an action.

Requirement 6 <Edit User Account>

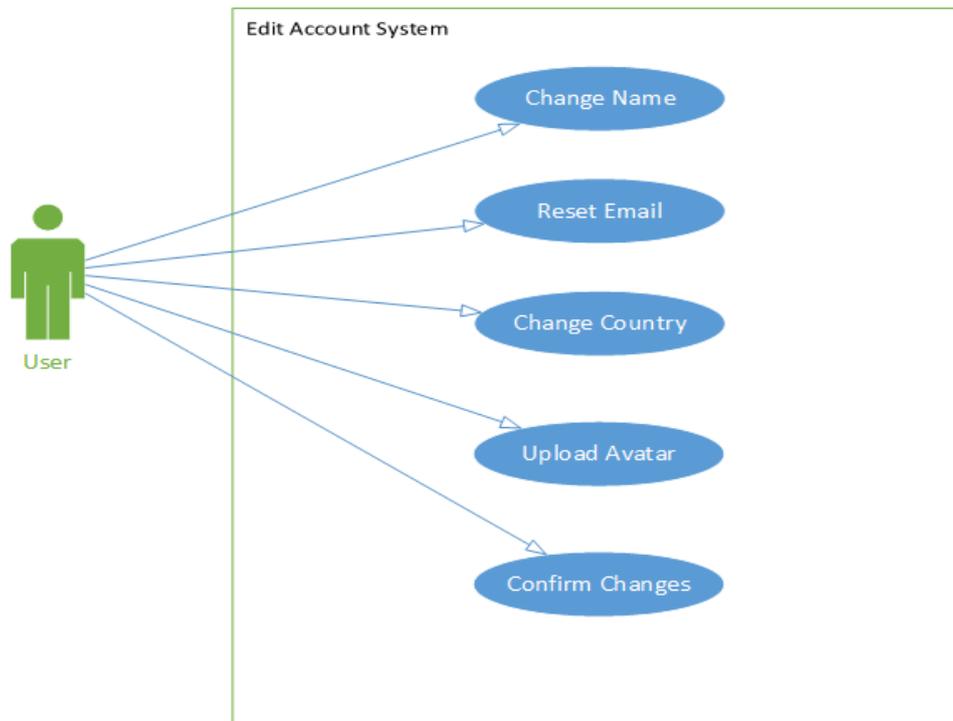


FIGURE 6: EDIT USER ACCOUNT USE CASE DIAGRAM

Description & Priority

The system shall provide the required functionality to empower end-users with the functionality to modify their existing accounts. This includes uploading an avatar, change their name and other associated information. This requirement will only be available to Users who already hold a valid active account. This requirement has a priority of 5/10.

Use Case

Scope

The scope of this Use Case is to illustrate how the User can interact with the system to make changes to their existing Exograph account. It is beyond the scope of this Use Case to detail how the system will perform the process in-depth.

Flow Description

The main flow initiates after a User chooses the “Account” option located in the header of all pages.

Exograph Technical Report

Precondition

1. A User must be authenticated with the Exograph system (Requirement 1).
2. A valid internet connection is required.

Activation

The Use Case begins when a User selects an option to change their account.

Main flow

1. The system displays all the current data to a User.
2. The User changes their name (E1).
3. The User changes their country.
4. They User then uploads an avatar (E2).
5. The User confirms the changes.

Exceptional flow

E1: <Invalid Input>

1. The system receives invalid input.
2. The system warns the User.
3. They User makes the required change.

E2: <Unsupported Image Format>

1. The system receives an unsupported image format.
2. The User is shown an error.
3. The User uploads a supported format.

Termination

Upon termination the system will display a message of completion.

Post condition

The system waits for a User to perform an action.

Exograph Technical Report

2.3 Non-Functional Requirements

2.3.1 Performance Requirement

The system shall perform to a minimum set threshold measured in response time. It is important that any system conforms to certain performance requirements. Performance is an issue that required addressing as the system relies heavily on third parties as a source of data. As a result, it is imperative that the system responds to a User within a maximum of 5 seconds. This should be achieved by adopting multi-threading to parallelise the process of requesting data. Caching of responses from third parties should be employed to avoid requesting the same data multiple times.

2.3.2 Recovery Requirement

The system shall meet a set of recovery requirements. Recovery refers to the ability to recover the system to a prior state after an incident. The ability to restore a system and its data to a previous state ties in with good source code management (SCM). A contingency plan is in place for disaster recovery in the event of an incident. The key to achieving this requirement is redundancy. Regular backups must be made of User oriented data so that there is minimal disruption. This requirement is measurable by down time. In the event of an incident, the system should attempt to recognise such an event and respond appropriately.

2.3.3 Scalability Requirement

The system shall be scalable as the User base broadens. Scalability refers to the ability of the system to appropriately manage increasing or decreasing workloads. The system shall scale horizontally according to the load it is experiencing. Scalability and Performance requirements go hand in hand. In order for the system to perform efficiently it must scale laterally. As the load increases more processing power is required. Therefore, the system should recognise an increasing or decreasing workload and respond by incrementing or decrementing the number of instances that the application is currently running on different servers. Scalability is measurable by quantifiable testing in a production environment.

Exograph Technical Report

2.3.4 Reliability Requirement

The system shall be reliable in the event of increased workload or other unplanned events. Reliability refers to the systems competency to perform to a minimum set of requirements such as operational behaviour. Reliability is closely related to scalability and is measurable by the time span of operational behaviours which the system can perform. To achieve true reliability, a set of automated system tests should be used to verify that the system can continue to meet the requirement as it evolves and grows.

2.3.5 Security Requirement

The system shall be secure to ensure Users are protected. Security refers to the system's ability to protect User's data through three main means Privacy, Physical, and Access. The privacy requirement should be met by encrypting data in transit. This protects Users and the system from potential attacks such as man in the middle attacks. The physical security requirement shall be fulfilled by using a Platform as a Service vendor. For the purpose of achieving accessibility, the system should offer accounts that are only accessible to Users who know the credentials associated with a particular account.

2.3.6 Maintainability Requirement

The system shall be maintainable as the size of the system grows and matures. Maintainability is tightly related to code quality and cleanliness. If code becomes messy and tangled, it become harder to maintain while costing more to upkeep. The system shall be modularised so components are reusable. The main aim of creating a maintainable system is to have high cohesion and low coupling. This requirement can be measured by the number of anti-patterns and the cyclomatic complexity of the code.

2.3.7 Usability Requirement

The system shall conform to a set of usability requirements such as Learnability, Efficiency, and high User Satisfaction. The system should be easy to use in terms of learnability. An end-user should be familiar with the layout of the application before using it. This is because certain interface design patterns are utilised that can be seen in other consumer applications. A training manual should not be required to use the system, it should be self-intuitive and easily learnable. The system should also be efficient in terms of the number of actions required to complete a task. User satisfaction can be measured as the percentage of customers who are satisfied vs those who are not during end-user testing and production usage.

Exograph Technical Report

2.3.8 Reusability Requirement

The system shall utilize reusable components in order to compliment maintainability. Reusability refers the ability to reuse system components of a similar composition to reduce and conclusively avoid duplication and lower code complexity. The system should use frameworks to reduced duplication and improve reusability. Reusability can be measured as a percentage of reused requirements, design elements and tests.

2.3.9 Interoperability Requirement

The system shall be interoperable with other systems to exchange and make use of information. Given the vast volume of information available externally to the system it is vital to make use of such information. The system should be interoperable with other systems such as Twitter and LinkedIn for the purposes of exchanging information in a seamless and transparent manner such that end-users are unaware of what is happening behind the scenes. Interoperability can be measured by the number of computer systems Exograph interacts with.

Exograph Technical Report

2.4 Design and Architecture

2.4.1 Design Approach and Standards Overview

2.4.1.1 Bottom up Approach

The bottom up design approach begins with the fundamental components. Lower level components are used to compose higher level components. The process is repeated until the desired system is achieved. The system may not evolve as one single monolith. With each higher level component, the level of abstraction is increased, hiding complexities. Using “both top-down analysis and bottom-up design simultaneously is likely to lead to the most robust software systems.” (Henderson-Sellers B. and Edwards J.M. et. al 1990). The bottom up approach was utilized alongside top down analysis in the project as a means of composing decoupled higher level components, creating a more robust system.

2.4.1.2 Service Encapsulation

In an architecture where components are distributed, it is necessary to ensure that each service encapsulates its own data. Each service maintains an independent data store that is exposed through well-defined interfaces. This reduces the dependencies among the services and minimises the risk of breaking changes to other services while allowing the service to exist autonomously.

2.4.1.3 Single Responsibility

The Single Responsibility Principle (SRP) states that a unit of code should have a single responsibility over the entire system. Each unit or class encapsulates its data to reduce unwanted exposure to external units which could cause unwanted dependencies therefore making it harder and more expensive to make a change to a class without impacting the consumers using it. The SRP is used extensively through the system and compliments the non-functional requirement of maintainability and reusability.

Exograph Technical Report

2.4.1.4 Data Representation Standards

A common predefined messaging format is used for the exchange of information between components. The common standard used is JavaScript Object Notation (JSON) which is a human readable format that can be easily transmitted over a network with little overhead due to its reduced verbosity compared to other data representation standards such as XML. JSON can be easily serialised and de-serialised.

2.4.1.5 Communication Standards

All components in the system communicate using the Hyper Text Transfer Protocol (HTTP) over a TCP/IP link to ensure a reliable connection. Each service exposes a RESTful interface that makes use of the methods provided by HTTP to govern the type of interactions between services.

Exograph Technical Report

2.4.2 Architecture Design

2.4.2.1 Class Diagram

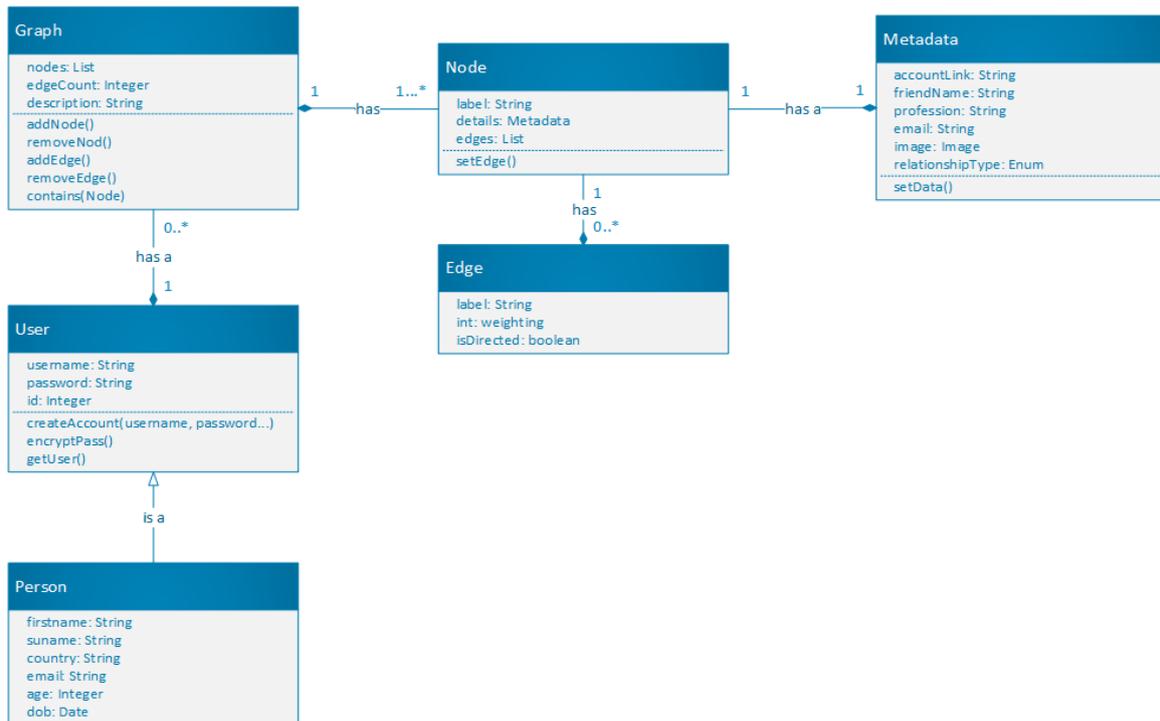


FIGURE 7: SYSTEM CLASS DIAGRAM

Figure 7 details a high level view of the relationships within the Exograph system. A User is a Person and therefore inherits their attributes and behaviours from the Person class. A User can have 0 to many Graphs. A Graph is composed of Nodes and Edges. One Graph can have 1 to many Nodes while a Node can have 0 to many Edges. A Node has a Metadata object which describes the data associated with a Node. Since a social network in real life is a directed graph. The Class Diagram assumes all are direct weighted graphs.

2.4.2.2 Sequence Diagram

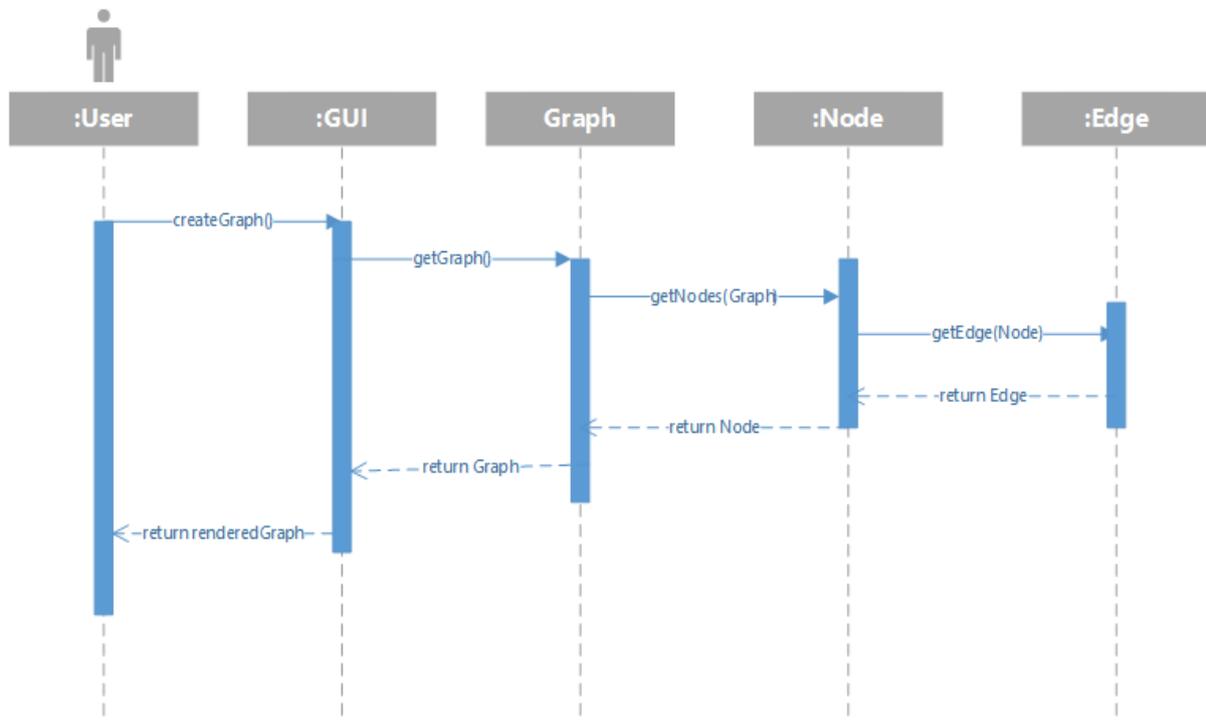


FIGURE 8: SYSTEM SEQUENCE DIAGRAM

The system sequence diagram in Figure 8 outlines the interactions a User has with the system when generating a graph over a time series. The diagram presumes that a User has already been authenticated with the system prior to the flow beginning. The flow begins when a User requests to create a graph that is already stored in the system. The GUI interacts with the graph object and makes a call to fetch it. The graph object then makes a subsequent call to request the Nodes associated with that Graph. The Node object makes a request to the edge object which returns all the edges for a given node. The node object then returns all the nodes and edges to the graph object which is rendered and displayed to a User.

Exograph Technical Report

2.4.2.3 Hardware Architecture

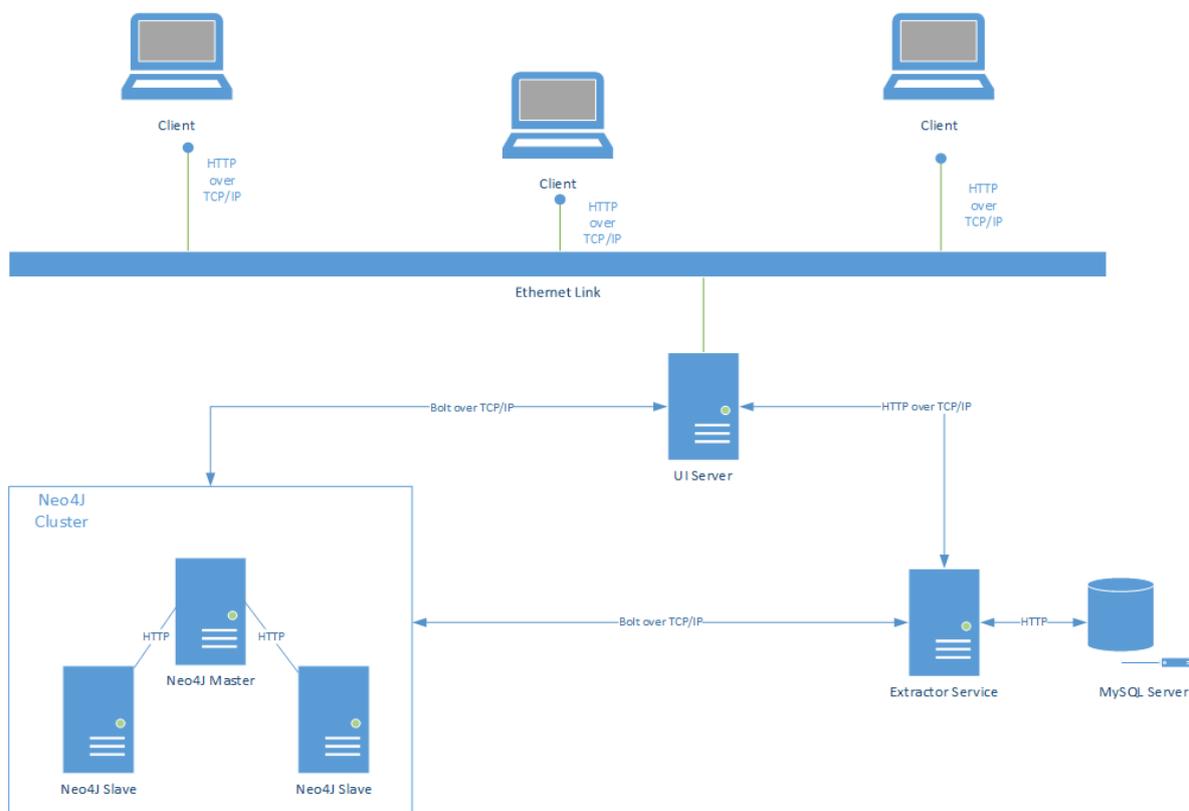


FIGURE 9: SYSTEM HARDWARE ARCHITECTURE

Figure 9 is an illustration of the Exograph hardware components hosted by the Infrastructure as a Service (IaaS) vendor, Amazon Web Services (AWS). The core component in the hardware stack is the UI Server running NodeJS that serves clients over a network. The UI Server communicates directly with the Neo4j cluster using the bolt protocol. The Extractor service is hosted on the same Virtual Machine (VM) and encapsulates job oriented data in an SQL database hosted on the same VM. The Extractor service interacts directly with the Neo4j cluster when bulk loading newly imported graphs.

Exograph Technical Report

2.4.2.4 Software Architecture

2.4.2.4.1 UI Server

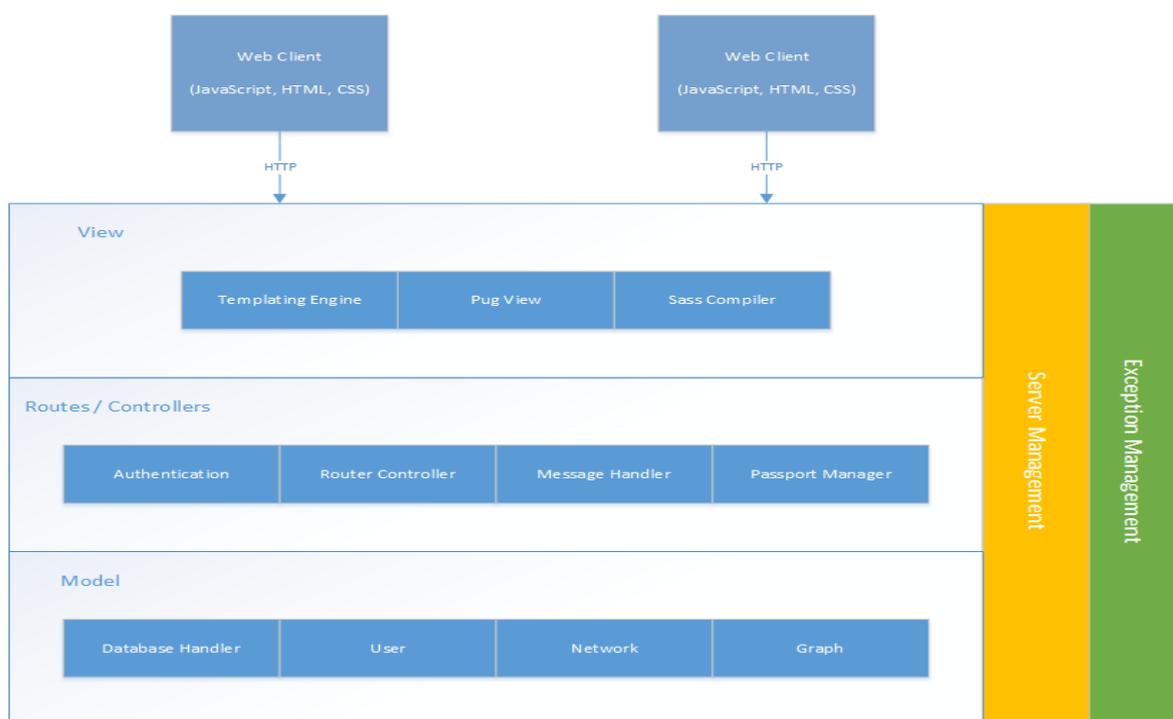


FIGURE 10: UI SERVER SOFTWARE ARCHITECTURE

Figure 10 details a high level overview of the UI Server, acting as the orchestrator in the stack. The UI Server is written in JavaScript and runs on the NodeJS Google V8 engine. In order to achieve an appropriate level of abstraction between each layer in the UI Server, the Model, View Controller (MVC) design pattern was used. The views consist of Jade based templates which are rendered on the server side for client side performance reasons as rendering the page on the client side can degrade the User Experience (UX) by increasing page load time and causing choppiness when painting is being performed by the browser. The routes are the entry point into the system, they use the Express framework to reduce redundant boiler plate code. The appropriate controllers are called based on the incoming requests, they handle interacting with the models to create, read, update and delete entities. When all operations have completed on the models the controller renders the template and responds to the client with the result of the execution. Security is handled at the route level by the passport module performing session management. The UI Server also communicates directly with the Extractor service to launch import jobs and query it for data related to the jobs triggered by Users.

Exograph Technical Report

2.4.2.4.2 Extractor Service

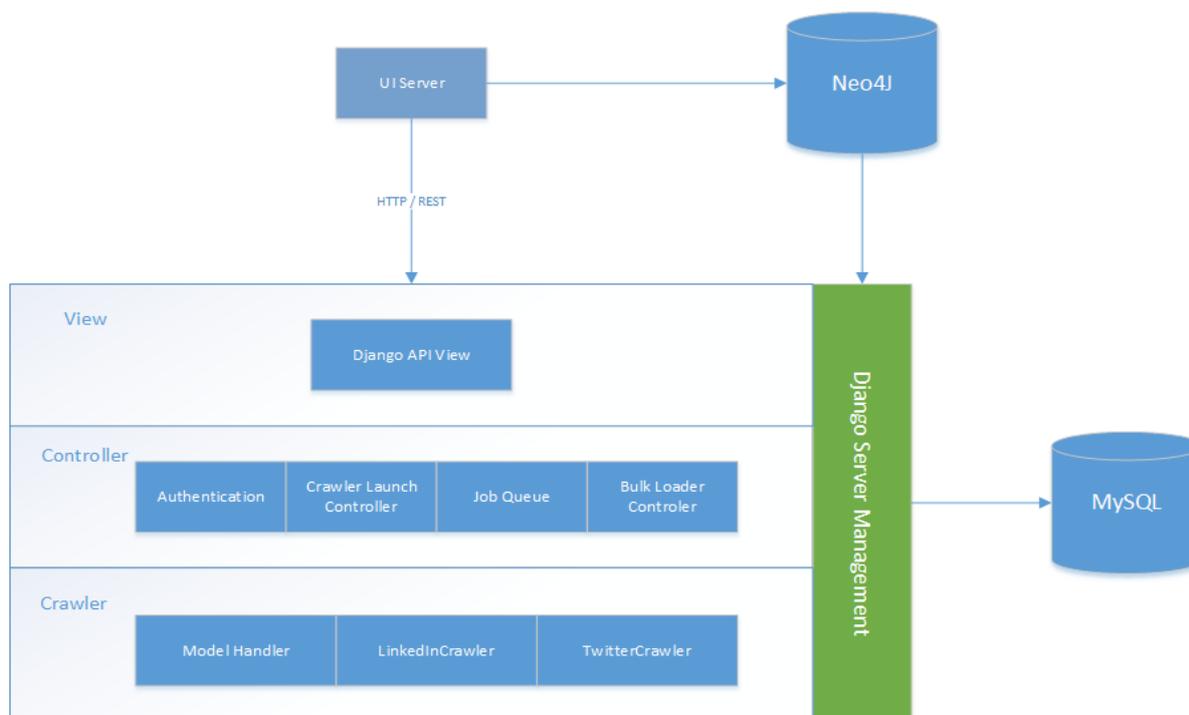


FIGURE 11: EXTRACTOR SERVICE ARCHITECTURE

The Extractor service is the most complex component in the stack. Its role is to provide a service that handles the interactions with third party APIs such as LinkedIn and Twitter. The service is written in Python and exists independently of the UI Server albeit it is tightly coupled to the Neo4j graph database. The service is composed of a number of different layers with minimal dependencies on one another.

The bottom layer of the Extractor consists of the crawlers and models. The crawlers perform the task of requesting data from the third party APIs, maintaining a cache of responses and building up a graph data structure. There are two possible ways of performing the task of building a graph data structure. The first way is to pull back all the data and build the data structure by traversing nodes in the cache when all the data has been gathered to generate a list of edges. The second way involves creating the graph data structure as the crawler is executing. The first approach works well for the Twitter crawler while the second approach was suited to the LinkedIn crawler.

The middle layer in the architecture are the controllers, they handle launching the crawlers asynchronously and managing a job queue of concurrent threads that are running. The Bulk Loader uses the resulting graph data structure from a crawler which is an object of two arrays, one containing nodes and their attributes and the other for storing edges and their

Exograph Technical Report

attributes to generate a dynamic Cypher query which can be executed on Neo4j to load the elements into the database.

The entry points to the service are provided by a Django API. Incoming requests call the bound view which invokes the appropriate controller before interacting with the models mapped to a SQL database using an Object Relational Mapper (ORM) framework provided by Django. Controllers trigger the threads that the crawlers are executed on. When a crawler is triggered, a JSON response is subsequently returned to the UI Server detailing the information relating to the triggered job. The UI Server processes the response and sends it to the client side to be rendered dynamically.

As mentioned previously, the aim was to achieve an autonomous service. The Extractor is tightly coupled to the Neo4j database. A possible solution was to allow the UI Server to retrieve a JSON representation of a graph and then delegate the loading of the graph into Neo4j to the UI Server. On further investigation it soon became evident this would not work well as NodeJS by nature is single threaded and makes extensive use of call-backs to perform tasks asynchronously meaning it is not good at handling computationally expensive operations, especially on large datasets. The solution implemented allows the Extractor service to perform the loading using the Bulk Loader.

2.4.2.4.3 Security Architecture

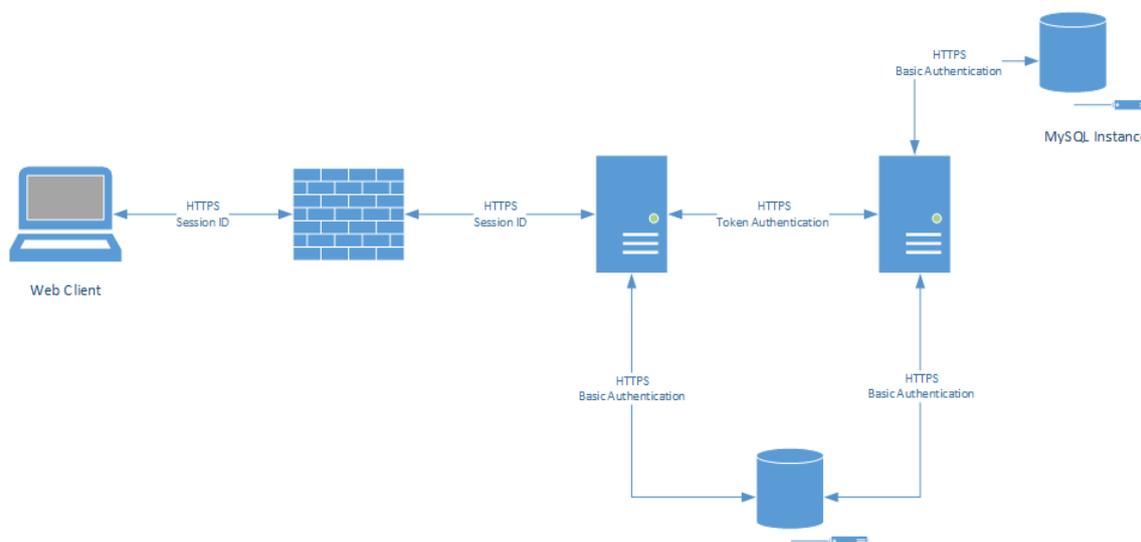


FIGURE 12: EXOGRAPH SECURITY QUESTION

Security within any system is an important non-functional requirement. It ensures the integrity and protection of User's data. Figure 12 is an overview of the security mechanisms used to ensure the system is secure. All communication over HTTP uses the Secure Socket Layer (SSL) version of HTTP, HTTPS. This ensures data is encrypted while in transit to avoid man in the middle attacks. At rest the data is not encrypted but important User data such as passwords are encrypted using the BCrypt hashing algorithm.

When a User authenticates with their credentials, a session is started on the server side and a cookie containing the session id is stored by the client, this is required to be passed back in subsequent requests to allow a session to be retrieved from memory and be verified by the Passport session management module. The session based approach does not scale well but fits the requirements of the project. An alternative solution would be to use token based authentication which would mean the server side would not need to store a session in memory. The use of OAuth was investigated for system side authentication, due to time limitation its implementation was decided against. This would have required developing a separate authorization service adding unnecessary complexity.

The Extractor service uses token based authentication to authorize incoming requests. The UI Server requests a token after authenticating and stores it in memory to be passed back in each request. When a token has expired, a request is made for a new one. Both the SQL and Neo4J databases are secured using basic username and password authentication.

2.4.2.4.4 Communication Architecture

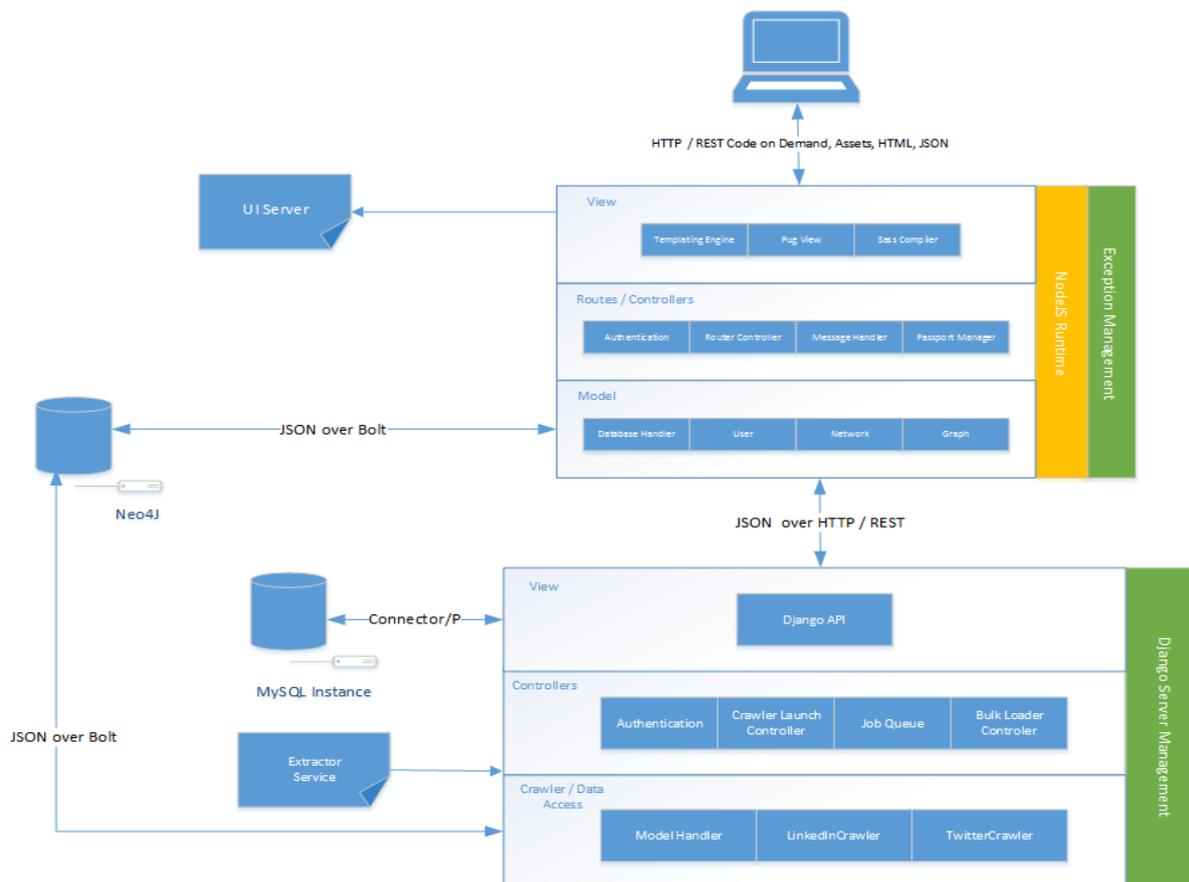


FIGURE 13: SYSTEM COMMUNICATION ARCHITECTURE

In any software system, standardised communication mechanisms and formats are required to ensure consistency throughout the system. The main format the system uses is JSON, which is a lightweight interchange format that is easy to parse when compared with other formats such as XML. The UI Server API communicates with the client using a mix of HTML and JSON using Representational State Transfer (REST).

The Extractor service exposes an interface that requires JSON to be passed to it on POST requests. It responds with a JSON structure that is consistent across all views. All communication exchanges with the Neo4j database use the bolt protocol, this is handled by the underlying libraries for connecting to it.

Exograph Technical Report

2.4.3 Database Design

The system contains two databases. A Neo4j Graph Database is accessible system wide by both the UI Server and the Extractor service. Information relating to Users such as social graphs along with their application oriented data is persisted in Neo4j. A MySQL relational database is used by the Extractor service to store Job information relating to current and previous crawler processes.

2.4.3.1 Neo4J Schema

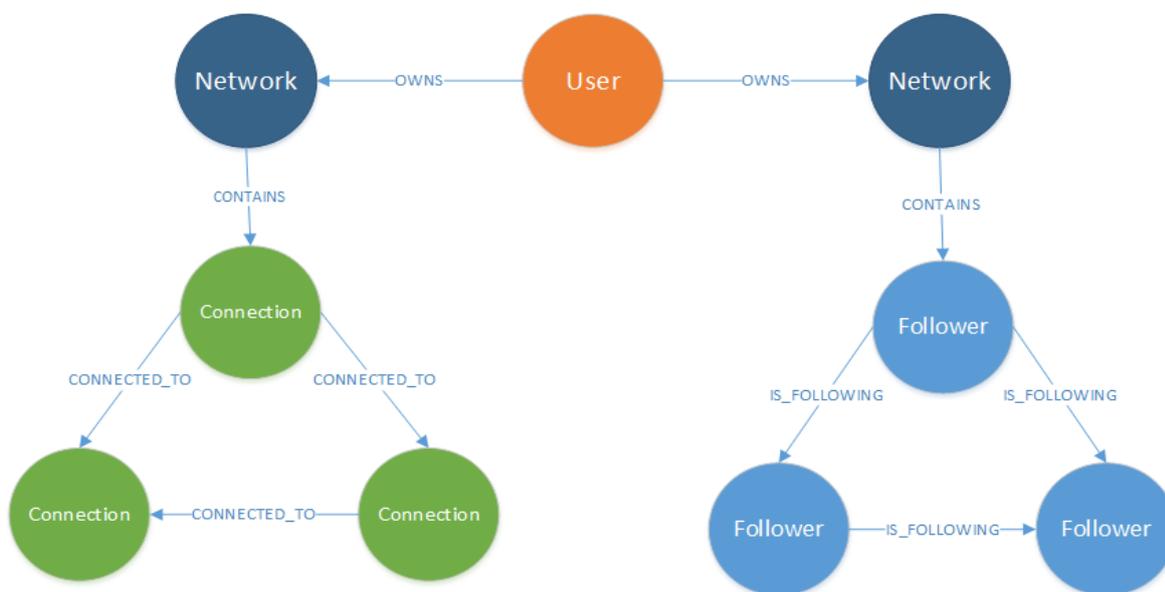


FIGURE 14: NEO4J DATABASE SCHEMA

Figure 14 provides an overview of the entities (Nodes) along with the relationships (Edges) that exist in the database. One User can have many Networks. Each Network contains a sub graph that was imported into the system via the Extractor service. A Network is connected to the root element in the sub graph of Connections (LinkedIn) or Followers (Twitter). A Network is connected to the root node by the CONTAINS relationship. In a LinkedIn graph a friend is represented by the 'Connection' label while for a Twitter graph a Follower is represented by the 'Follower' label. The relationship between Connections is the CONNECTED_TO relationship type while in a Twitter graph the relationship type between followers is the IS_FOLLOWING type. Each Node has a set of key-value properties stored on them. Edges can have properties associated with them, although this is not required.

Exograph Technical Report

2.4.3.2 MySQL Extractor Service Schema



FIGURE 15: EXTRACTOR SERVICE DATABASE SCHEMA

The Extractor service uses a simple relational schema. Two entities exist within the database. The Job table stores data relating to the jobs executed. While the Job_Type table is a lookup table containing data relating the types of jobs that can be ran. A job type is associated with many Jobs.

Exograph Technical Report

2.5 Implementation

2.5.1 Twitter Crawler

The Twitter crawler is part of the Extractor and is written in Python. There was a vast amount of challenges encountered throughout the implementation phase. These challenges centred around initially finding the appropriate API endpoints to use and developing an algorithm which could recursively gather the data in an efficient manner. The first implementation was inefficient and slow due to request limitations imposed by Twitter.

```
# Retrieve Friends
if not user.has_discovered_friends():
    print('No cached friends for "%s"' % user.screen_name)

    print('Retrieving friends for user "%s" (%s)' % (user.name, user.screen_name))

    # page over friends
    c = tweepy.Cursor(api.friends, id=user.id).items()

    while True:
        try:
            friend = User.create(c.next())
            if not User.exists(friend.id):
                friend.persist()
            user.add_friend(friend.id)
            if len(user.friends_ids) >= max_friends:
                print('Reached max no. of friends for "%s".' % user.screen_name)
                break
        except tweepy.TweepError as error:
            if isinstance(error, tweepy.RateLimitError):
                print('Rate limited. Sleeping for 15 minutes.')
                time.sleep(15 * 60 + 15)
                continue
            else:
                print('Error: ' + str(error))
                continue
        except StopIteration:
            break
    user.persist()
```

FIGURE 16: GATHERING A USER'S TWITTER FRIENDS

Figure 16 demonstrates the implementation for retrieving a person's friends. A limitation restricts the number of friends gathered because performance issues were encountered due to the number of API requests required to gather friends of friends growing exponentially. Caching is implemented in order to not make requests for a Follower if they already exist in the cache. This addition greatly improves the total import time.

Exograph Technical Report

```
# get friends of friends
cd = current_depth
if cd + 1 < max_depth:
    for fid in user.friends_ids[:max_friends_of_friends]:
        visited_list = get_friends(fid, max_depth=max_depth,
                                   current_depth=cd + 1, visited_list=visited_list)
```

FIGURE 17: RETRIEVING TWITTER FRIENDS OF FRIENDS

After the process of first retrieving a User and then getting their friends ids, recursion is used to retrieve friends of friends using a depth first traversal until the max depth is reached. Originally, it was intended to allow a User to set the maximum depth when launching an import, but this was subsequently changed and the maximum depth was restricted to 3. After all the data has been gathered and cleansed, the next stage in the flow involves processing the cached data based on a seed node then processing their friends recursively to generate a list of edges from which a graph data structure can be created. The graph data structure is converted into a Neo4j Cypher query that can be loaded into Neo4j.

```
def process_friends(seed_id, edges=None, depth=0, max_depth=3):
    if edges is None:
        edges = []

    if User.exists(seed_id):
        user = User.load(seed_id)
    else:
        return edges

    for friend_id in user.friends_ids:
        if not User.exists(friend_id):
            continue

        edges.append([seed_id, friend_id])

        if depth + 1 < max_depth:
            process_friends(friend_id, edges, depth + 1, max_depth)

    return edges
```

FIGURE 18: CREATING A LIST OF EDGES

Figure 18 illustrates how a list of edges is generated. The seed id passed into the function is used as a starting point, on subsequent recursive calls it allows friends of friends to be processed. The resulting list is a mapping between follower to followee, creating a directed relationship between two nodes. The Bulk Loader is used to convert the graph data structure into a single Cypher statement.

Exograph Technical Report

2.5.2 LinkedIn Crawler

The LinkedIn crawler was implemented prior to the Twitter crawler as it was the most difficult to development since access to the LinkedIn API is hard to acquire. Access was not granted which meant a web scraper had to be developed to gather data. The first challenge was to login via the web client and store a session cookie locally to maintain account access for the duration of a crawl.

```
def configure_opener(self):
    if os.access(cookie_filename, os.F_OK):
        self.cookie_jar.load()

    self.opener = urllib.request.build_opener(
        urllib.request.HTTPRedirectHandler(),
        urllib.request.HTTPHandler(debuglevel=0),
        urllib.request.HTTPSHandler(debuglevel=0),
        urllib.request.HTTPCookieProcessor(self.cookie_jar)
    )

    self.opener.addheaders = [
        ('User-agent', 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR 1.1.4322)')
    ]
```

FIGURE 19: CONFIGURING LOCAL COOKIE OPENER

Figure 19 demonstrates configuring the opener sent with each request. To work successfully, both the urllib and cookielib are required to provide basic HTTP support. The aim is to trick the LinkedIn backend into thinking the crawler was a web browser. This allowed the JSON used to dynamically render the LinkedIn client to be requested. The urllib allows data of any format to be requested seamlessly. This reduced the amount boilerplate code needed meaning more time could be spent working on the crawler.

Exograph Technical Report

```
def login(self):
    soup = self.load_soup("https://www.linkedin.com/")
    csrf = soup.find(id="loginCsrfParam-login")['value']
    login_data = urllib.parse.urlencode({
        'session_key': self.username,
        'session_password': self.password,
        'loginCsrfParam': csrf,
    }).encode('utf8')

    response = self.request("https://www.linkedin.com/uas/login-submit", login_data)

    if response.url != config['urls']['home_url']:
        raise LoginException('Login Failed')

    self.cookie_jar.save()
```

FIGURE 20: AUTOMATED LINKEDIN LOGIN

Figure 20 demonstrates the login code. The LinkedIn homepage is first requested and parsed using the BeautifulSoup library. The Cross Site Request Forgery (CSRF) token is parsed. A URL encoded string is created with the User's authentication credentials and the CSRF token. A POST request is made to submit the login data, upon successful login the session cookie is stored in memory and the crawler can begin requesting a User's connections. The crawler first requests the homepage to gather the root node's information then it requests the raw JSON for a User's connections list, paging the data until all connections are gathered.

```
def load_connections(self, member_id):
    count = 10
    offset = 0

    # Make initial request for 10 items
    data = self.request_json(shared_connections_url % (member_id, offset, count))

    if data is not None and self.has_shared_connections(data):
        self.parse_shared_connections(data)
        total_fetched = len(self.network[0].connections[self.pos].connections)

        num_shared = data['content']['connections']['numShared']

        if num_shared > 10:
            while offset < num_shared and total_fetched < num_shared:
                offset += 10
                data = self.request_json(shared_connections_url % (member_id, offset, count))

                if data is not None and self.has_shared_connections(data):
                    self.parse_shared_connections(data)
                    total_fetched = len(self.network[0].connections[self.pos].connections)
```

FIGURE 21: REQUESTING A USER'S LINKEDIN CONNECTIONS

Exograph Technical Report

The load connections function is used to load the connections for a User. The endpoint exposed by LinkedIn uses paging and only returns a maximum of 10 items per requests. The JSON response is parsed and the nodes are added into a graph data structure. The volume of requests required to fetch every connection for a person is large. This is one factor that has an impact on performance. When the crawler has finished executing, the graph data structure is passed to the Bulk Loader which generates a Cypher statement. As of April 1st 2017, the LinkedIn crawler no longer works as LinkedIn released a new UI that is more difficult to crawl.

2.5.3 Bulk Loader

The Bulk Loader enables a graph data structure to be converted into a Cypher query and loaded efficiently into Neo4j. There are currently no open source Python libraries that can support the loading of a large graph into Neo4j efficiently. Attempts were made to use an Object Graph Mapping (OGM) tool but performance issues were encountered as there was a high volume of interaction between the Extractor and Neo4j when loading nodes and edges one at a time. After some considerable thought to devise a new solution, a blog post by a Neo4j engineer was found detailing the best solution to perform the loading. The result was an algorithm that could generate a bulk insert statement for nodes and edges in a graph while preserving their attributes.

```
def generate_node_statements(self):
    # Partially generate the node representations
    for node in self._graph.nodes(data=True):
        # Generate a node identifier for Cypher
        identifier = self.generate_tag(5) + self.generate_tag(6, dct=num_dct)

        # Append the node's ID attribute so that the node-ID information used by Networkx is preserved.
        node_items = node[1].items()

        # Create the key-value representation of the node's attributes taking care to add quotes when the value is
        node_attributes = "[%s]" % ",".join(
            map(lambda x: "%s:%s" % (x[0], x[1]) if not type(x[1]) == str else "%s:'%s'" % (x[0], x[1]),
                node_items))

        # Store it to a dictionary indexed by the node-id.
        self._node_statements[node[0]] = [identifier, "(%s:%s %s)" % (identifier, self._label, node_attributes)]
```

FIGURE 22: BULK LOADER GENERATING NODE INSERT STATEMENTS

The generate node statements function iterates over all the nodes in a graph and creates a key-value representation of a node's attributes and id set on a node. The statement is indexed by a unique generated id which maps to a nodes label and attributes. Each node statement is then added to a dictionary.

Exograph Technical Report

```
def generate_edge_statements(self):
    # Generate the relationship representations
    for edge in self._graph.edges(data=True):
        edge_items = edge[2].items()

        edge_attributes = ""
        if len(edge_items) > 0:
            edge_attributes = "[%s]" % ", ".join(
                map(lambda x: "%s:%s" % (x[0], x[1]) if not type(x[1]) == str else '%s:"%s"' % (x[0], x[1]),
                    edge_items))

        # Declare the links by their Cypher node-identifier rather than their Networkx node identifier
        self._edge_statements.append("(%s)-[:%s %s]->(%)" % (
            self._node_statements[edge[0]][0], self._rel, edge_attributes, self._node_statements[edge[1]][0]))
```

FIGURE 23: BULK LOADER GENERATING EDGE STATEMENTS

The generate edge statements function iterates over each edge in a graph and formats a string containing an edges attributes, the final statement links two nodes with an edge in the form (node)-[relationship]->(node). The edge defines the type of relationship that exists between two nodes. The resulting statement is appended to a list of edge statements. When all node and edge statements have been generated, a lambda is used to combine the lists of node and edge statements into a single CREATE statement that can be executed on Neo4j. The Bulk Loader was designed to work on any graph data structure and can therefore be used when generating Twitter or LinkedIn bulk insert statements without the need for different implementations.

Exograph Technical Report

2.5.4 UI Server Generating a D3 Graph Model.

The UI Client requires a JSON representation of a graph containing a list of nodes and edges. The D3 visualisation framework uses this model to render the visualisation dynamically.

```
function parseGraph(results, callback) {
  var nodes = [], links = [];

  results.forEach(function (row) {
    row.nodes.forEach(function (n) {
      if (idIndex(nodes, n._id) == null)
        nodes.push({
          id: n._id,
          type: 'circle',
          size: 60,
          score: 1,
          label: n.labels[0],
          name: n.properties.name,
          endpoint: '/' + n.labels[0].toLowerCase() + '/' + n._id
        });
    });

    var r = row.relationships;
    links = links.concat(
      {source: idIndex(nodes, r._fromId), target: idIndex(nodes, r._toId), type: r.type}
    );
  });

  return callback({
    nodes: nodes,
    links: links
  });
}
```

FIGURE 24: UI SERVER, GENERATING A D3 GRAPH REPRESENTATION

A custom Neo4j procedure written in Java performs a Breadth First Search (BFS) traversal of a sub graph and retrieve all nodes and edges. The result is passed to the parse graph function as seen in Figure 24. The function iterates overall nodes in the result set and adds nodes based on their id to an array, the relationships a node has are added to an array of edges. The returned model is composed of a list of nodes with their id, label and attributes and a second list containing edges represented by id, relationship and attributes. D3 requires that each node is unique in the list and that the relationships refer to the array index, not the id of the node. The domain specific terminology D3 uses is nodes and links where links map to edges.

2.6 Graphical User Interface

2.6.1 Import Graph

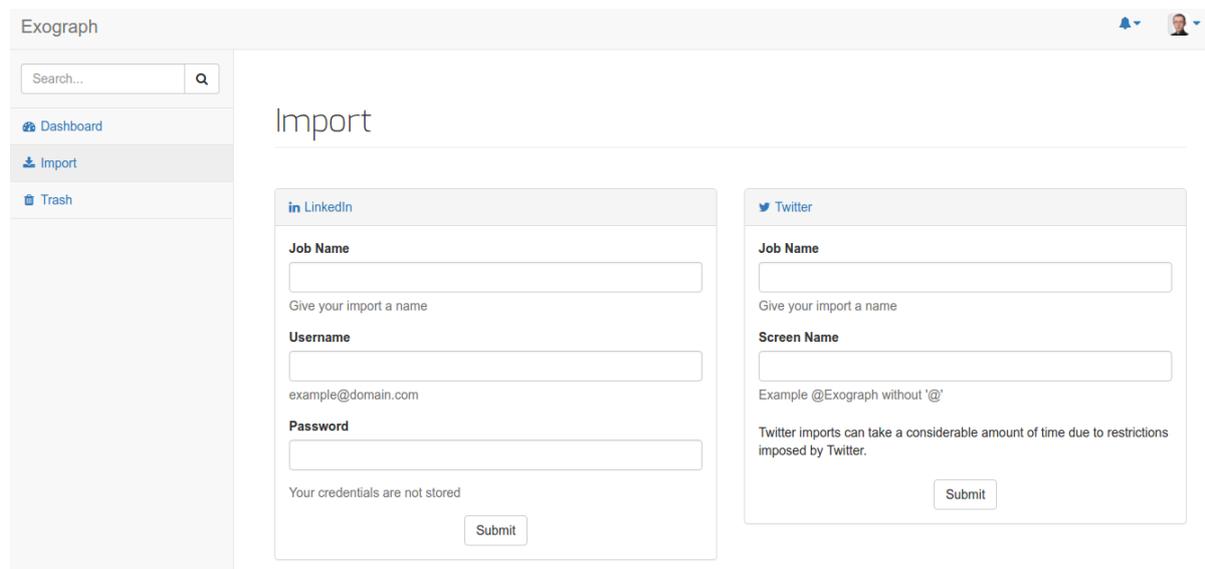


FIGURE 25: IMPORT GRAPH PAGE

Importing a graph is made simple and easy. The page provides two separate import options, LinkedIn and Twitter. To launch a LinkedIn job, the User must first enter a job name along with their LinkedIn username and password. It is made clear to the User that their credentials are not stored. The second option is launching a Twitter job, a User must enter a job name and a screen name that corresponds to the Twitter account to be crawled. The job name is used later to distinguish between multiple graphs. When they submit the form, a modal appears detailing the information associated with the triggered job. The progress of the import can be tracked by selecting the alert icon in the horizontal navigation bar, this displays dropdown of the five most recent jobs ran, a link is displayed which redirects to a page displaying all jobs executed in a coherent way. The overall layout of the page is consistent with the standardised components used throughout the system. This makes the system more learnable and compliments ease of use.

2.6.2 Graph Dashboard

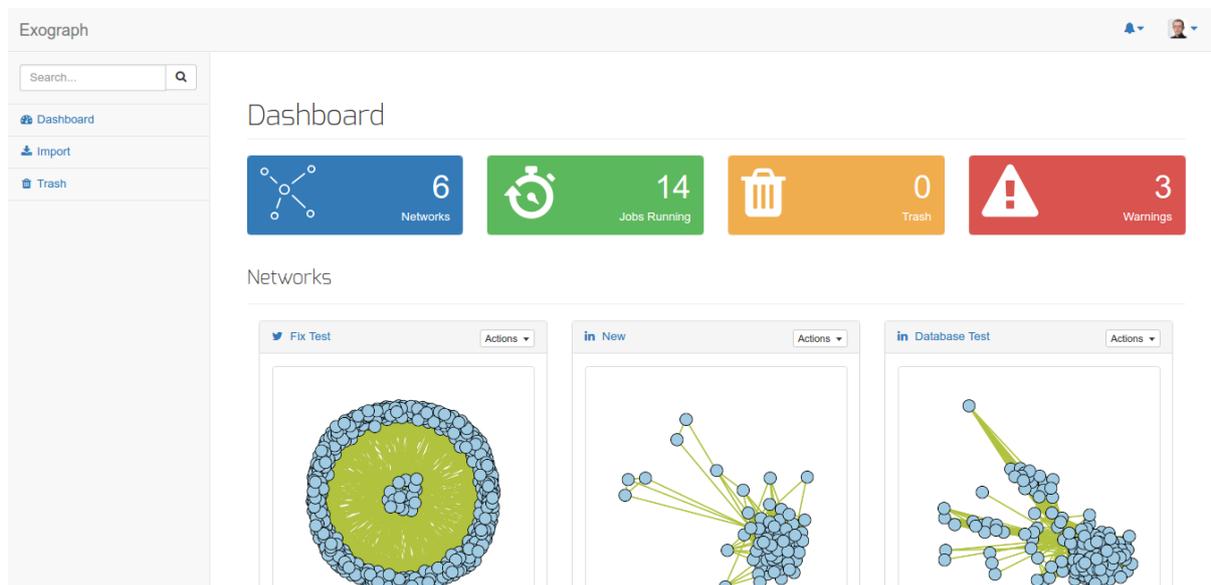


FIGURE 26: DASHBOARD PAGE

The Dashboard is the focal point of the system, from here a User can view all their imported graphs. The top container contains four coloured rectangles that provide information such as total networks, number of current jobs running, how many items are in the trash bin, and the number of system generated warnings due to import failures. The section located below the alert boxes contains a listing of all a User's imported graphs that are not located in the trash bin or have been deleted. When a graph has been successfully imported an image is generated for displaying in the UI. Each item contains an action button located in the right side of the panel heading bar. When clicked, a dropdown menu is displayed, the options include, view, analytics, send to trash and view import details. The action button was a key design consideration that was added after many prototyping iterations allowing all the actions that can be performed on a graph to be contained in an easy to find place, reducing the number of steps required to otherwise perform such actions.

2.6.3 Graph View

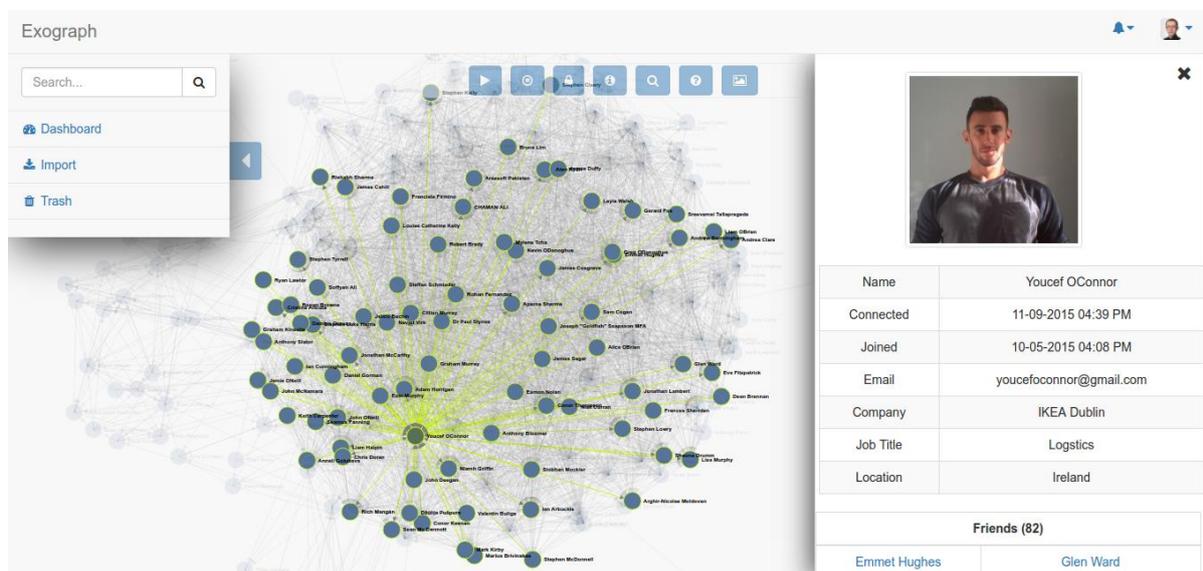


FIGURE 27: VIEWING A GRAPH

The graph view page is accessible by selecting a graph on the Dashboard and is one of the most complex pages within the UI. When the page is initially loaded the data for a graph first needs to be retrieved and the visualisation rendered. While this is happening a loading spinning is displayed to inform the User that their request is being processed. After loading and rendering the page is displayed to the User. A force layout is used to render the graph in an aesthetically pleasing way. D3 only provides the tools for loading a graph into the Document Object Model (DOM) and event handlers to setup events based on different User actions. How the visualisation works is up to the developer, this meant a lot of work was involved in making the visualisation interactive. When a User zooms with the mouse wheel more of the graph is visible. Click and drag is supported, this allows different areas of the visualisation to be brought into focus. The toolbar located at the top of the page provides an easy way to perform actions on a graph. When a node is hovered over, its neighbouring nodes are highlighted to allow their connections to be easily identified. If User clicks and holds on a node, its first degree neighbours are brought into the foreground and all other nodes and edges are brought into the background and their opacity is reduced, all while highlighting a ring around the focused nodes and edges. If a node is brought into focus by hovering over it, if the enter key is pressed the node information container is loaded which displays the attributes associated with that node. The blue button located to the left of the vertical navigation bar provides a way to expand and collapse the navigation bar to avoid blocking the visualisation.

2.6.4 Graph Analytics

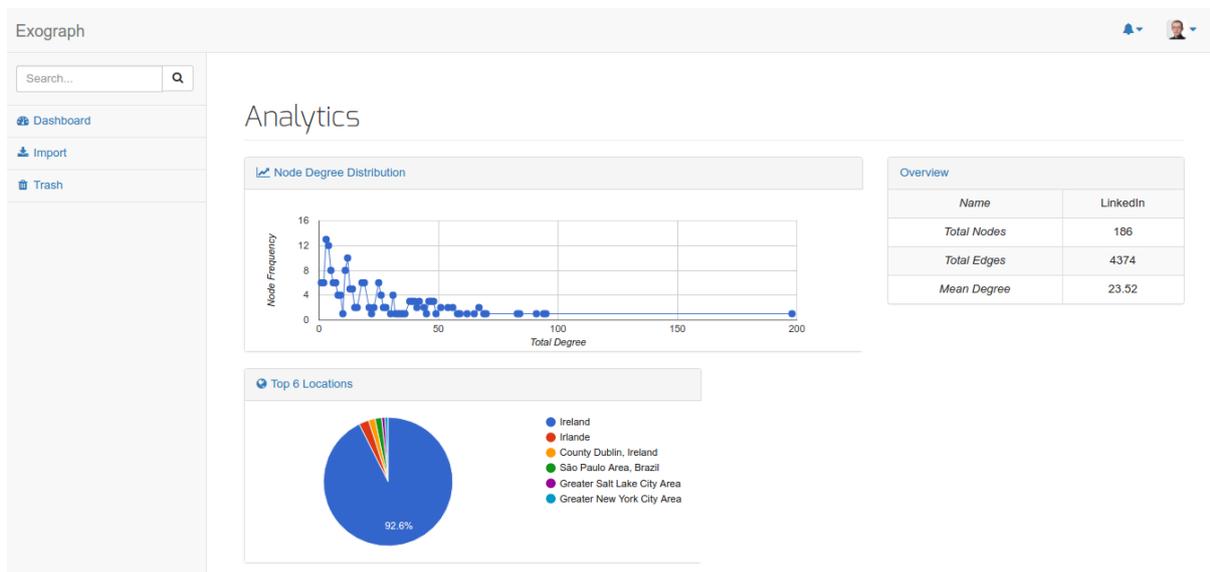


FIGURE 28: GRAPH ANALYTICS PAGE

The Analytics page provides insights into both LinkedIn and Twitter graphs. The page is accessible by selecting the analytics option in the actions dropdown of each graph on the Dashboard. The overview panel shows key information such as total nodes and edge and the mean degree centrality. The degree frequency distribution shows the total degree count aggregated by the number of nodes. A pie chart located below the degree distribution shows the top six node locations by their percentage. Each chart is interactive, when a User hovers over key parts such as a point in the scatter plot, hidden information is displayed showing the number of nodes that have a particular degree count. Similarly, when a sector in the pie chart is hovered on, the percentage and location are shown in a popup.

2.6.5 Graph Trash

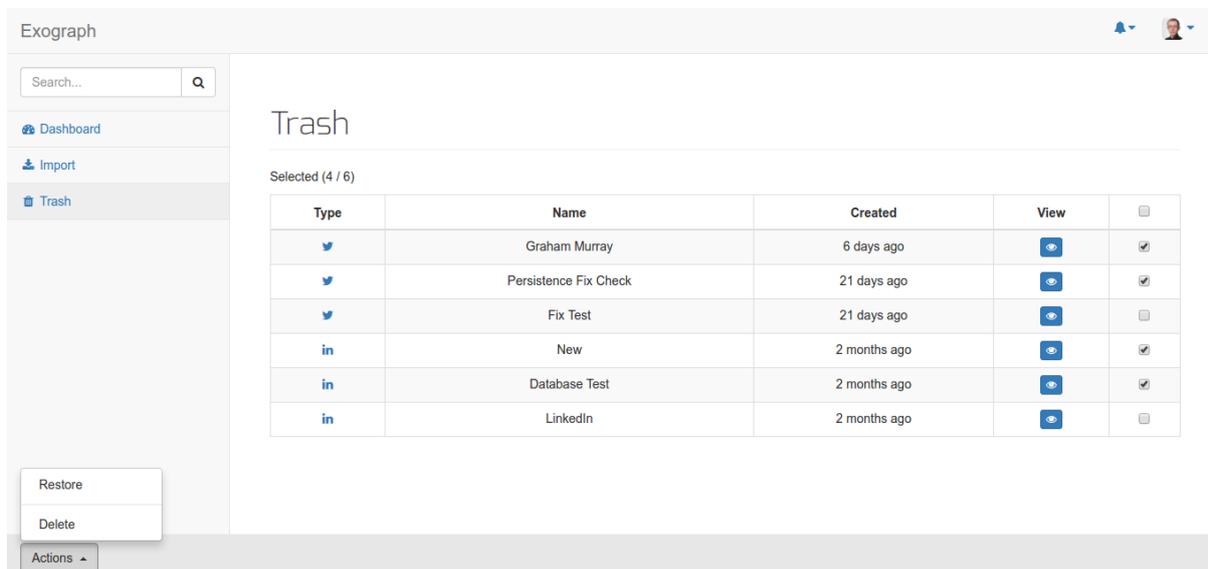


FIGURE 29: TRASH PAGE, DELETE AND RESTORE

The trash bin provides a listing of all graphs in a staging area prior to either deleting or restoring them. The table is interactive and provides three User friendly ways of selecting an item, if a row is clicked the item will be selected or the checkboxes can be checked by directly clicking them. In the table header there is a select all checkbox which selects all the listed items. Above the table is a count to inform the User how many items have been selected. An actions button contained in the footer becomes enabled when one or more items are selected, two options are contained within, one to restore and another to delete the items. There are many benefits to the User by incorporating the multi action table. The desired action can be applied to many items at once instead of individually. This is particularly useful when there is a large amount of items in the trash bin.

2.6.6 User Profile



FIGURE 30: PROFILE PAGE WITH PROFILE IMAGE UPLOAD

The Profile page is accessible by clicking the far right dropdown button located in the header of any page. When the “Profile” option is selected, the Profile page will appear providing an overview of the current User’s account details while providing an option to update the data on the Account page where a User’s password and can be changed along with other account settings. Located on the left hand side of the Profile page is an option to upload a profile image. This can be triggered by selecting the “Change Image” button. When clicked, a modal appears containing a form to upload a new image and provide a preview of the selected image before being uploaded. Only PNG format images are accepted as they are resized by the backend before being stored. In the event of an error, the User is informed.

Exograph Technical Report

2.7 Testing

In Industry today, Software Testing including manual and automated Quality Assurance consumes a considerable portion of the development lifecycle. Testing is vital to ensuring a system performs and behaves as expected.

2.7.1 Unit Testing

A Unit Testing methodology was employed for testing small components in isolation of factors such as dependencies and the environment that the code will be executed in. Each module includes a suite of unit tests that are executed each time a build is ran to ensure there were no undesirable changes since the previous successful build. Data required by each unit is mocked while external dependencies are stubbed.

Test Driven Development (TDD) and Behaviour Driven Development (BDD) are two primary unit testing paradigms, each have advantages and disadvantages over one another. TDD was selected as the paradigm to use. The key idea behind TDD is to develop test cases alongside the software. Each time a change is made all the tests should be executed to validate each unit of code behaves as expected. Both the UI Server and Extractor have a vast amount of code, therefore providing a high percentage of code test coverage was priority. The testing frameworks used differ as the UI Server is written in JavaScript while the Extractor is implemented in Python, although the core idea behind the process does not change. The use of mocking and stubbing is important. The two are different but easily confused. Mocking allows dependencies to be removed. They are pre-programmed with expectations that form a specification of calls expected to be received. To follow a true unit testing methodology, each unit of code is tested in isolation as units often depend on other units. Stubs provided set answers to method calls during a test, instead of executing a method the predetermined result is returned (Venners B, et.al 2002). Travis is used to provide a Continuous Integration (CI) build on each repository. Each time a pull request is made a build runs on the feature branch and another on the merge to master. In the event of a failure, a notification email will be sent and the branch highlighted as failing. The build logs can be examined to determine the cause of the failure before fixing the issue.

Exograph Technical Report

2.7.1.1 Extractor

The Extractor uses standard Python libraries for Unit Testing, unittest and mock. The unittest library provides a framework for setting up tests, executing the test cases and cleaning up after execution is complete. The unittest.TestCase class provides functionality for assertions to validate the result of a test and assert whether expected results match the actual returned results. The mock framework is used in tandem with the unittest framework to provide dependency mocking and stubbing. The Extractor has a code coverage percentage of 68%.

```
@mock.patch('http.client.HTTPResponse')
@mock.patch.object(LinkedInCrawler, 'request')
def test_request_json(self, mock_request, mock_response):
    reference = LinkedInCrawler('', '')

    mock_request.return_value = mock_response
    mock_response.read.return_value = b'{"key": 1024}'
    mock_response.info().get_content_charset.return_value = 'utf-8'

    json = reference.request_json('some url', 'some data')

    mock_request.assert_called_with('some url', 'some data')
    self.assertTrue(mock_response.read.called)
    self.assertDictEqual(json, {'key': 1024})
```

FIGURE 31: EXTRACTOR PYTHON UNIT TEST

The unit test provided in the above example tests that a JSON response is returned when the request JSON method is called on the reference object. Prior to the test being executed, the HTTPResponse object from the http library is mocked along with the request field on the LinkedInCrawler class. The two mocks are injected as arguments to the test before the method being tested can be called. Methods are stubbed with predetermined return values to avoid the crawler making real requests to LinkedIn for data. After the function call is made on the reference, assertions are made to verify the expected behaviour.

Exograph Technical Report

2.7.1.2 UI Server

The same Unit Testing methodology is employed for testing units of code within the UI Server, the only difference compared to the Extractor is the frameworks used. Writing JavaScript unit tests was confusing at first. The UI Server uses a Model View Controller (MVC) style approach to provide separation of concerns. This was a strategic design decision from a testability perspective, components would be difficult to test without using an architectural design pattern. Three testing tools were used, Mocha, Sinon, and Chai. Mocha is an asynchronous testing framework for running tests, Sinon is a mocking and stubbing framework while Chai is an assertion library.

```
describe("testing getNetwork()", function () {
  it('should resolve with a network', function () {
    var subject = new model(getMockNode());
    var results = {network: 'a network'};

    mockNetwork.get = sinon.stub();
    mockNetwork.get.callsArgWith(2, null, results);

    subject.getNetwork(1)
      .then(res => {
        expect(res).to.have.property('network');
        expect(res.network).to.equal(results.network);
      });
  });

  it('should reject with an error', function () {
    var subject = new model(getMockNode());
    var error = {message: 'a message'};

    mockNetwork.get = sinon.stub();
    mockNetwork.get.callsArgWith(2, error, null);

    subject.getNetwork(1)
      .catch(err => {
        expect(err).to.have.property('message');
        expect(err.message).to.equal(error.message);
      });
  });
});
```

FIGURE 32: UI SERVER USER MODEL UNIT TEST

Figure 32 provides an example of a UI Server test. The tests verify the `getNetwork()` function associated with a User. The first test uses Sinon to stub the `get()` method on the network model dependency. The callback passed to `get()` has the desired mock results injected as parameters so the path executed can be controlled to verify a valid model is returned when the Promise resolves and an error is returned if it fails and is rejected. Chai is used to assert the expect property values in the results match the mocked property values. In total 50 Unit Tests were written for the UI Server with a code coverage percentage of 78%.

Exograph Technical Report

2.7.2 Integration Testing

A further equally important testing methodology is Integrated Testing (I&T). I&T occurs after Unit Testing and combines modules into a complete form to validate behaviour, guaranteeing modules work together. This process can be done in a sandboxed environment using a range of frameworks. Selenium is a browser automation framework that allows the system to be tested from the perspective of a User. Selenium can automate the actions and scenarios performed by a User therefore, validating the system as a whole works as expected. The final solution includes a suite of Selenium integration tests that are ran in a sandboxed environment prior to the system being deployed. The RESTful Extractor API uses the Django testing framework to test the API views as if real requests were being made.

2.7.2.1 Extractor

```
def test_run_linkedin_post(self):
    payload = '{"name": "a name", "username": "someusername", "password": "somepassword", "user_id": 1}'

    response = self.client.post('/api/v1/linkedin', payload, content_type='text/plain;charset=UTF-8', **self.header)

    self.assertEqual(200, response.status_code)

    # Test job was created in db
    response = self.client.get('/api/v1/job/1', **self.header)
    self.assertEqual(200, response.status_code)
    self.assertTrue(isinstance(response, JsonResponse))
```

FIGURE 33: EXTRACTOR INTEGRATION TEST

Figure 33 provides an Extractor integration test using the Django I&T framework. Before each test in the suite executes a test database is created and populated with mock data. The Django client sends a request to the specified endpoint along with other optional data. The response received from the API is subsequently validated to ensure the expect behaviour of the API. In total 15 API integration tests were written.

Exograph Technical Report

2.7.2.2 UI Server

The UI Server utilised the WebDriverIO library which provides NodeJS bindings for the W3C WebDriver protocol. This allows for the standalone selenium server written in Java to be used. Mocha was employed as a test runner while Chai was used as the assertion library.

```
describe("login tests", function () {  
  it('should login successfully', function () {  
    LoginPage.login("test@test.ie", "12345", function () {  
      assert(browser.isVisible('.panel-primary'));  
      assert(!LoginPage.message_panel.isVisible());  
    });  
  });  
  
  it('should login failure with incorrect password', function () {  
    ComplexHeader.profile_dropdown_toggle.get().click();  
    ComplexHeader.items.logout.get().click();  
  
    LoginPage.form.waitForExist(5000);  
    LoginPage.login("test@test.ie", "Pd!", function () {  
      assert(!browser.isVisible('.panel-primary'));  
      assert(LoginPage.message_panel.isVisible());  
    });  
  });  
  
  it('should login failure with incorrect email', function () {  
    LoginPage.form.waitForExist(5000);  
    LoginPage.login("test@test.i", "12345", function () {  
      assert(!browser.isVisible('.panel-primary'));  
      assert(LoginPage.message_panel.isVisible());  
    });  
  });  
});
```

FIGURE 34: UI SERVER SELENIUM LOGIN INTEGRATION TESTING

Pages were represented as Objects. This allowed for the functionality each page possessed to be encapsulated in one place. Page components such as Widgets were further decomposed into component Objects allowing pages to be composed of different components and reduce code duplication. The tests demonstrated in Figure 34 checks the login functionality works as expected. The first test logs in with the correct credentials and verifies that the page displayed is the Dashboard. The second test uses an incorrect password and asserts that an error message display is displayed after submitting the form. The same process occurs for the third test, instead using an incorrect username.

Exograph Technical Report

2.7.3 Customer Testing

User Acceptance Testing (UAT) is considered to be one of the final phases in the development lifecycle. It is vital to validating that the system fulfils all the requirements from an end User's perspective. For the purpose of UAT, a number of testing scenarios were created. They were used to verify the system by means of manual testing. A number of Recruitment Consultants were contacted to test the system in March. This was prior to when LinkedIn changed their UI which broke the LinkedIn crawler. It is important to bear in mind that during the time of testing the system was still an early prototype. Out of 8 Recruiters contacted, 6 responded and agreed to trial the system. Each Recruiter was provided with the aims and requirements of the system allowing them to decide if they felt the system conformed to the outlined requirements and goals. Three different tasks were developed to test key parts of the system. They include;

1. Import your LinkedIn network.
2. When the import has completed, view the newly import graph and interact with the visualisation by clicking and dragging people.
3. Send a graph to the trash bin and use the grid to select items before choosing either the delete or restore actions.

The system was tested over a period of three days. This gave vital feedback and insights into usability issues never thought of during the design and prototyping phase. The results of their findings are documents below in a summarised format. When the results were received from each Recruiter, gratitude and gratefulness was show as the time they spent evaluating the system led to major improvements. Unfortunately, the intended testing period for the LinkedIn crawler and visualisation was cut short. After three days of testing, LinkedIn rolled out their new UI, but the feedback gained over the first three-day period was enough to gather information to improve the system.

Exograph Technical Report

2.7.3.1 Import your LinkedIn Network.

	What is your opinion of the import page?	Did you encounter any issues importing your network?	What elements did you most like?	What elements did you dislike?	What changes would you recommend if any?	Out of 100. What is your level of satisfaction?
Recruiter A	The design is clean and easy to use. Navigating to the page initially is easy.	Restrictions caused by the job name field.	The hints to import a graph on the dashboard page provided great guidance.	None	Possibly add icons to each import box to show the type of import.	96
Recruiter B	The page is easy to understand but lacks attractiveness.	None	The job import summary that is displayed after launch.	The page title font doesn't fit in with the page.	Overall, the page works very well. Change the title font	93
Recruiter C	Easy to use, a very responsive	It took a long time to import my network.	The simplistic design.	There's lack of consistency with fonts	Make the fonts consistent across the system.	89
Recruiter D	Excellent design and very usable	None	The popup after starting an import	Having to handover my LinkedIn credentials.	None	98
Recruiter E	Well-designed which makes the experience.	After clicking submit the system froze	There was clear visibility of system status.	None	Remove restrictions on the import name	88
Recruiter F	Very nice layout and easy to use	None	Any errors were clearly displayed	None	None	96

TABLE 1: CUSTOMER TESTING LINKEDIN IMPORT RESULTS

The results of importing a LinkedIn graph showed that overall the Recruiters were happy with the design. One suggested that icons be added to the header of each form panel while another suggested that the fonts be made consistent. These suggested changes were made.

Exograph Technical Report

2.7.3.2 View the Newly Imported Graph

	What is your opinion of the graph view page?	Did you encounter any issues?	What elements did you most like?	What elements did you dislike?	What changes would you recommend if any?	Out of 100. What is your level of satisfaction ?
Recruiter A	The concept is good but improvements need to be made	The graph took a long time to settle down and was very slow.	The zoom and drag functionality .	The choppiness and slowness. My graph had 1500 people in it.	Improve performance and add a way to navigate	80
Recruiter B	The dashboard is really cool. The visualisation was really good.	Trying to see what people are connected to each other because there were so many lines.	The full page view with nothing blocking the network.	It was very hard to see who was connected to who	Add instructions on the page and highlight peoples connections somehow.	89
Recruiter C	It was a little hard to use because there was no instructions.	Mainly the lack of instructions for using the keyboard.	The side panel for showing each information on people	No way to easily see who is in what groups of people.	Add instructions and having a way of focusing on different groups of people	92
Recruiter D	Great effort. I really like it.	Minor issues trying to get back to the dashboard	The interactivity with the network and zoom and pan.	There's no way to lock people in place.	Add a way to lock people in position.	79
Recruiter E	It is still primitive but has potential.	My network was large which meant it was slow and froze my PC	The fact there was no obstructions when viewing.	No way to get back to the dashboard	Try to make it faster.	82

Exograph Technical Report

Recruiter F	Really good idea but not easy to use	The network kept flying around the page initially	The dragging of people.	The network seemed to excited when clicking people.	Stop the network going crazy at first.	91
-------------	--------------------------------------	---------------------------------------------------	-------------------------	-----------------------------------------------------	----------------------------------------	----

TABLE 2: CUSTOMER TESTING VIEW GRAPH RESULTS

The results of viewing the graph import demonstrated that there was still work that needed to be conducted to improve the main part of the system. The testing took place at a stage when the visualisation was not fully complete, planned functionality was still to be added but some issues highlighted were never considered prior to the testing being conducted. The number one issue always priority was performance of the page loading and graph rendering. This was a known issue and was brought to attention. A number of changes took place to enhance performance in a number of areas. A custom Neo4j procedure was written in Java to perform a Breath First Search traversal when retrieving a sub graph as the default way Neo4j handled queries of this kind was inefficient. The choppiness experienced in large networks was attributed to rendering a graph as a Scalable Vector Graphic (SVG), the solution implemented to solve this issue was reducing the number of DOM manipulations and frequency of node and edge position calculations as the browser had to keep repainting sectors of the page each time an elements coordinates were altered. Another problem encountered was the initial positioning of elements in the graph on the same coordinates causing the visualisation to become erratic and fly around the page. This was fixed by randomly assigning nodes with an initial position. A number of Recruiters detailed an issue with not being able to easily see a node’s connections due to the high density of edges. The implemented solution now highlights a node’s 1st degree neighbours when a node is hovered over with a mouse. A similar issue to this problem was not being able to focus on different clusters in a graph. The now implemented solution changes the opacity of a node and its neighbours when clicked. To release the opacity a restore the visualisation to normal, the C key can be used. The most significant issue emphasised centred around the lack of instructions, as the graph became more advanced in functionality used to keyboard to perform actions on the visualisation the need for a help popup became apparent. A help icon is now provided which details all the operations that can be performed on the visualisation. A toolbar was added to the visualisation to expose the functionality using buttons.

Exograph Technical Report

2.7.3.3 Send a Graph to Trash

	What is your opinion of the trash bin?	Did you encounter any issues?	What elements did you most like?	What elements did you dislike?	What changes would you recommend if any?	Out of 100. What is your level of satisfaction?
Recruiter A	Nice piece of functionality and easy to use.	No issues found	Being able to group networks before deciding an action.	Having to select items one by one	Add a checkbox for selecting all items	90
Recruiter B	The simple design makes it easy to learn	Finding the option to trash a graph took a while	Being able to view a graph from trash.	No way of selecting items by click on a row in the table	Add functionality to select items by clicking on them	95
Recruiter C	It works really well and is not hard to use.	Sending items to trash one by one is slow.	Being able to restore items.	There was no count to show how many items are selected.	Display the number of selected items	80
Recruiter D	The page is nice but lacks more advanced features.	No issues found	All the actions possible are contained in one place	There is no way to delete or restore multiple items in one go.	Add a feature to delete or restore multiple items in one go.	87
Recruiter E	Cool feature that adds real value to the project.	No issues found	All	The page jumping straight to trash each time a graph is selected.	Stop the dashboard jumping to trash when a graph is trashed.	90

TABLE 3: CUSTOMER TESTING SENDING A GRAPH TO TRASH RESULTS

The results of the trash functionality testing yielded further vital information. Overall, participants felt there was a need to improve performing multiple actions on trash items. The implemented changes include showing a selected items count, selection by clicking a row and the addition of a checkbox in the grid header to select all items. These changes have greatly improved the UX of the trash bin.

Exograph Technical Report

2.7.3.4 Final Customer Testing

A final round of Customer Testing was conducted with 10 people from various backgrounds in May aimed to see where the system needed further improvements. The same scenarios were used from the first round of testing along with new ones for new parts of the system that were not presented in March. The scenarios asked were;

1. Import your LinkedIn network.
2. When the import has completed, view the newly import graph and interact with the visualisation through the provided toolbar.
3. Send a graph to the trash bin and use the grid to select items before choosing either the delete or restore actions.
4. Upload a profile image.
5. View all jobs that you have ran.
6. Update your profile information.

The results gathered were a big improvement from previous. Overall, there were no issues found with the graph visualisation. All participants felt that it performed beyond their expectations. There were minor issues and bugs highlighted that were yet to be fixed but in all the participants were satisfied with the system. The average satisfaction rate recorded in March was 85%, this rose 8% to 93% User satisfaction.

Exograph Technical Report

2.8 Evaluation

System evaluation is an important process for any system. It allows the system to be assessed in a number of areas such as Performance and User Satisfaction. The Exograph system was evaluated on a number of different levels. The first level focused on performance and stress testing on a component level and subsequently on components when they are combined to form the complete system. The second level of evaluation aimed to see what level of satisfaction Users got and to what degree they felt they system conformed to the outlined goals and requirements. The evaluation results were gathered and stored so they could be compared to ascertain whether changes to the system have positively or negatively impacted previous result sets.

2.8.1 Performance Testing

Performance testing was conducted on both the LinkedIn and Twitter crawlers to assess the total time required to gather the data to produce a graph model and load it into Neo4j. The evaluations were carried out after the Bulk Loader was implemented which greatly enhanced loading performance. The tests of both crawlers was conducted based on the size of a network.

2.8.1.1 LinkedIn Crawler Import without Caching

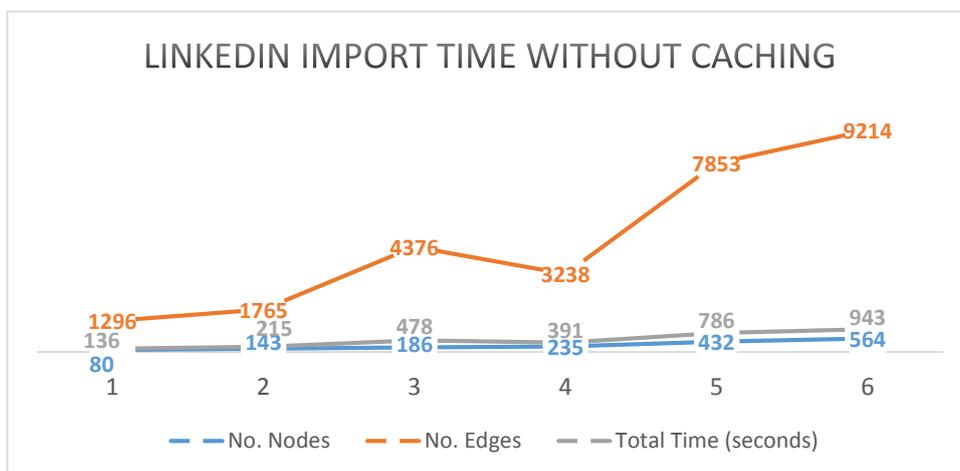


TABLE 4: TOTAL LINKEDIN NETWORK IMPORT TIME NO CACHING

Based on the results presented in Table 4, the LinkedIn crawler has a mean import time of 491 seconds for a graph containing 473 nodes and 4263 edges. The mean increase in time observed was 135 seconds for an average increase of 80 nodes and 1320 edges. The results suggest a correlation between the number of edges and import time instead of the number nodes. The LinkedIn crawler did not have caching fully implemented prior to LinkedIn changing their UI.

Exograph Technical Report

2.8.1.2 Twitter Crawler Import without Caching

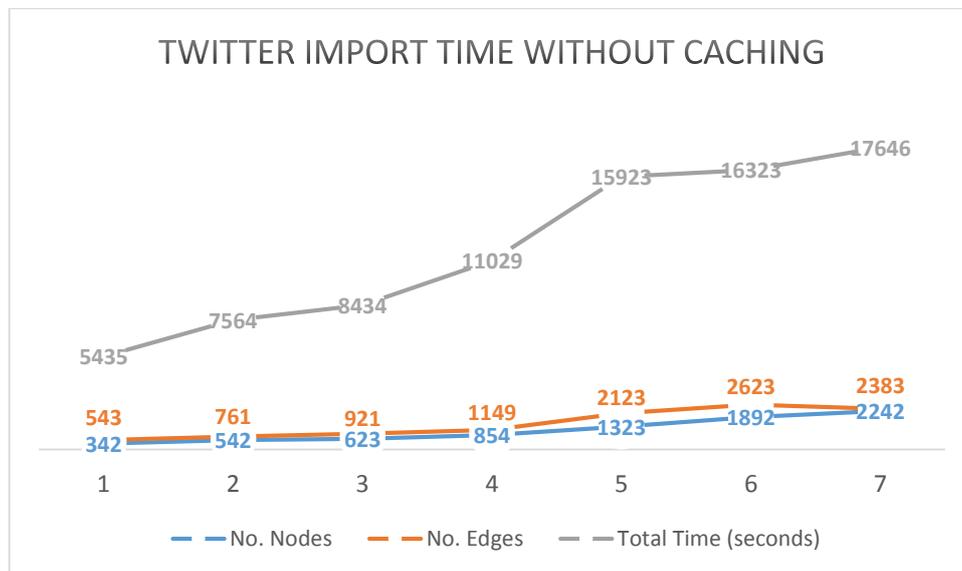


TABLE 5: TOTAL TWITTER NETWORK IMPORT TIME NO CACHING

Table 5 provides metrics for the total Twitter network import time when caching is disabled. The minimum observed import time was 5435 seconds (1 hour 30 minutes) for a graph with 342 nodes and 543 edges while the maximum observed time was 17646 seconds (4 hours 54 minutes) for a graph with 2242 nodes and 2383 edges. The mean time difference 2035 seconds (33 minutes) for a consistent average increase of 317 nodes and 307 edges. It was clear early on that there would be considerable issues in a production environment if API response caching was not implemented. There is a direct correlation between the number of nodes and total import time as opposed to time and edges in the case of a LinkedIn network. The difference can be associated with the implementation differences of each crawler. There is a considerable difference in total import time when compared to the LinkedIn crawler. This is as a result of Twitter imposing usage limitations meaning only 15 requests can be made on an API endpoint in a 15-minute window.

2.8.1.3 Twitter Crawler Import with Caching

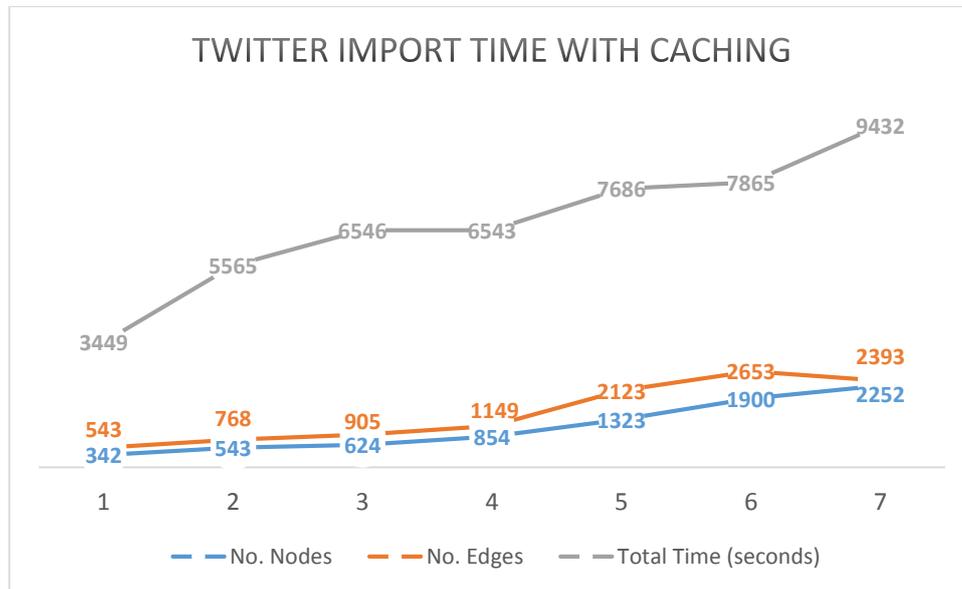


TABLE 6: TOTAL TWITTER NETWORK IMPORT TIME WITH CACHING

Table 6 illustrates the total Twitter network import time when caching is enabled. From the chart, it can be seen that there was a considerable reduction in the time taken to import a graph when caching is enabled as opposed to not. The same 7 Twitter accounts were used for both datasets as a control. There is a minor difference in node and edge count values as the data was gathered one week after the dataset when caching was not enabled. The minimum observed time was 3449 seconds (34 minutes) for a graph with 342 nodes and 543 edges while the maximum time was 9432 seconds (2 hours 37 minutes) for a graph with 2252 edges and 2393 edges. There was a mean consistent increase of 997 seconds (16 minutes) for an average increment of 307 nodes and 317 edges. Overtime with increasing in the cache size, graph import time will be reduced further as a result of effective caching.

2.8.1.4 Twitter Import Caching Comparison

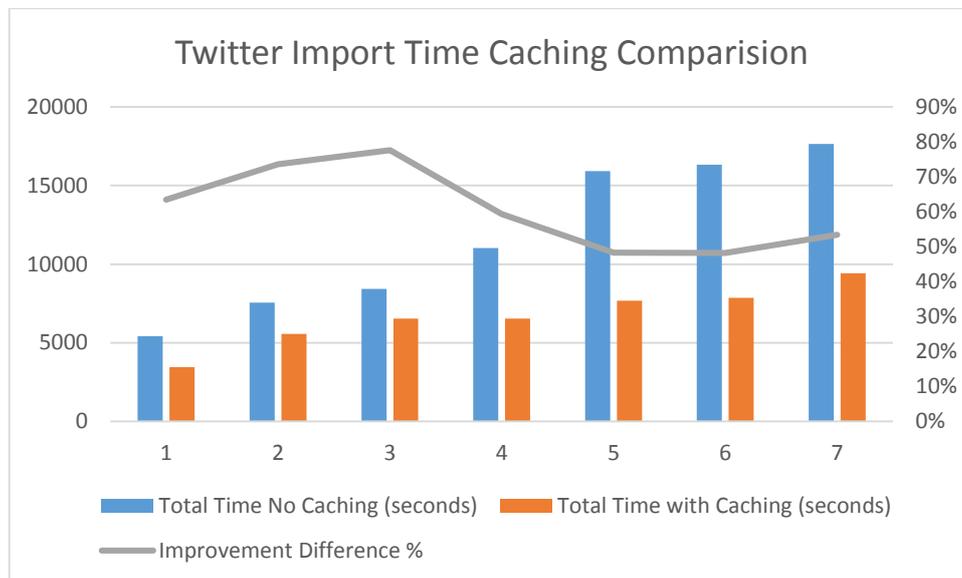


TABLE 7: TWITTER IMPORT TIME CACHING PERFORMANCE IMPROVEMENT

Table 7 provides a metrical comparison between improved import performance time when caching is not enabled and when caching is enabled. The table shows the total import times as bars while the time reduction metric is represented by the grey line in percentage. The blue bar represents the total time in seconds when caching is not in use, while an orange bar shows the total time when caching is enabled. The datasets used for the comparison are based on the results presented in section 2.7.1.2 and 2.7.1.3. The results show that there is an improvement in import time across all 7 imported Twitter networks. The lowest observed improvement difference was a 48% reduction on import time for graphs 5 and 6 while the highest observed reduction was 78% for graph number 3. The average reduction in time across the board was 61%. After 7000 seconds there was drop in the improvement of import time. Upon investigation, it was concluded that drop is related to the disproportionate difference in times compared to when there is a continued increase in improvement. Based on the results presented, it can be concluded that caching was a big success in term of system wide advancement in ensuring that the system meets the non-functional requirement of performance.

2.8.1.5 Neo4j Query Performance

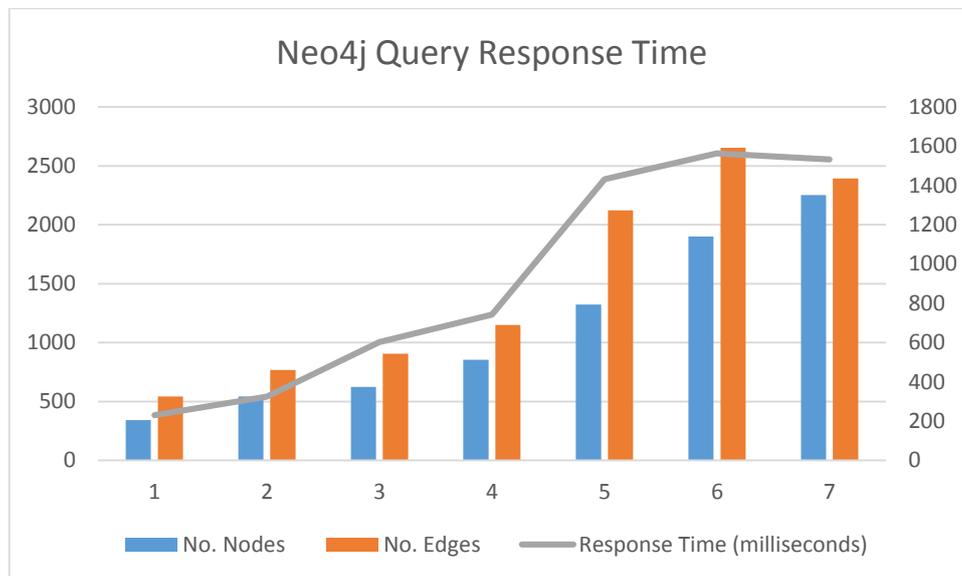


TABLE 8: NEO4J SUB GRAPH RETRIEVAL RESPONSE TIME

Table 8 provides an insight into Neo4j’s query performance when retrieving a sub graph. The results were recorded after the custom APOC procedure was implemented as it was not possible to compare the current results without the custom procedure as small to medium size graphs caused Neo4j to hang when using a standard Cypher query. From the results presented in Table 8, it can be seen that the query response time is fast as the mean response time was just shy of 1 second. The lowest observed time was 221 milliseconds for a sub graph with 342 nodes and 543 edges, while the highest observed time was 1564 milliseconds for a sub graph with 1900 nodes and 2653 edges. The response time increases in tandem with the increase in the number of nodes and edges until observation number 6, but decreases slightly in observation 7. This was attributed to the reduction in the number of edges contained in observation 7 compared to observation 6. The dataset gathered is small and therefore does not show any intriguing patterns. It would be more interesting to see the same visualisation on a larger dataset to find trends. The reason Neo4j was chosen as the primary database was because of the performance it has over other offerings.

Exograph Technical Report

2.8.2 User Satisfaction March

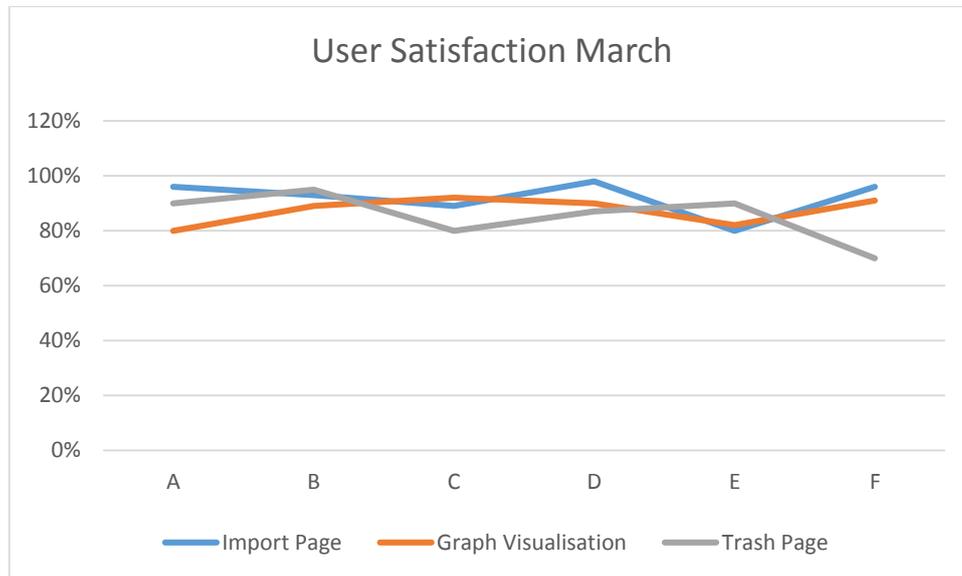


TABLE 9: USER SATISFACTION RATINGS ON KEY PAGES IN MARCH

Table 9 presents the findings of total combined User Satisfaction. The data was gathered from the six Recruitment Consultants who tested the application in March. Users were asked when testing the application to provide a rating between 1 and 100 based on whether they felt the application met their needs based on the requirements. The data was gathered on three key pages, Import page, Graph Visualisation page, and the Trash page. The results yielded a good insight into what pages Users felt were better than others. The blue line represents the Import page, while the orange and grey lines represent the Graph Visualisation and Trash pages. The results show that Users were most satisfied with the Import page as the mean rating percentage was 92%. The lowest rated page was the trash page with a mean percentage of 85%. The graph visualisation which is the highlight of the system and was the median rated page with an average rating of 88%.

2.8.3 User Satisfaction May

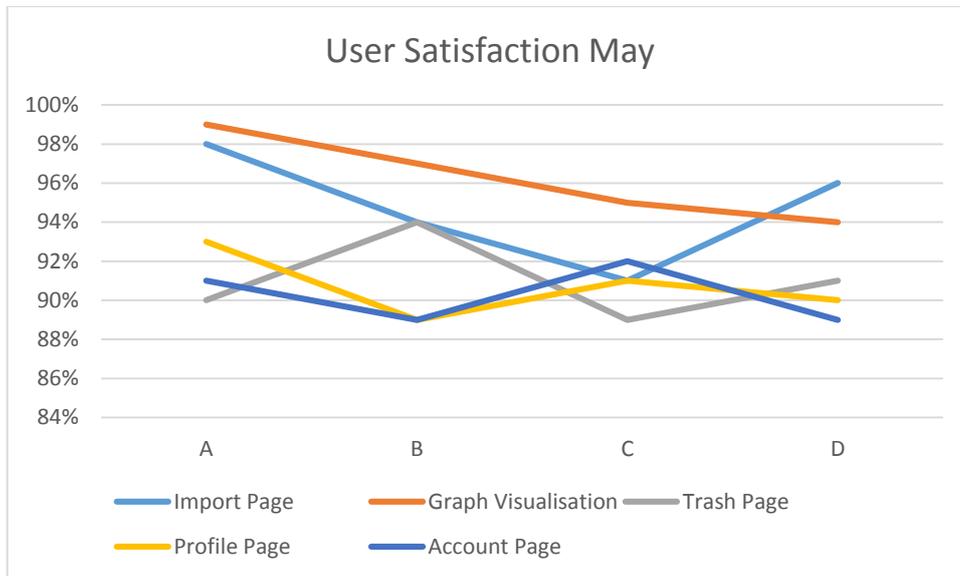


TABLE 10: USER SATISFACTION RATINGS IN MAY

User Satisfaction ratings were recorded in May during a second round of Customer Testing. Table 10 provides the results obtained from the four participants. The results contain two additional pages that were not present during testing conducted in March. The results show the highest rated page was Graph Visualisation with an average rating of 96%, this is an increase of 8% compare to March. The lowest rated pages were the Trash and Profile pages with an average rating of 91%. The Visualisation page had the highest increase. This can be attributed to the vast amount of work that went into the page after the first round of Customer feedback. No page had a decrease in rating.

Exograph Technical Report

2.9 Deployment

Amazon Web Services (AWS) was selected as the chosen Cloud Platform to deploy the project. AWS was chosen over other vendors due to the lower costs associated with running an application. Prior to deploying the application there were a number of different areas work needed to be conducted. Research was conducted into the instance types provided by AWS and what capabilities they have. The system required an instance with a minimum of 8 gigabytes of RAM and a 2GHz processor. The instance type that met these needs was the m4.xlarge from the m4 family of multipurpose instances. The m4.xlarge has 16GB of RAM and a 2.4GHz Intel Xeon processor which is enough to host the application with a low to medium level of traffic. The m4.large instance costs \$0.20 an hour to run. One hundred dollars of AWS credit was acquired on a Student development account this meant there was enough credit to run the application for 20 days in total. CentOS version 7.3 was chosen as the Operating System (OS). The goal of deployment was to automate the process as much as possible which would reduce the time required to deploy changes into production. A series of shell scripts were written for each repository that install all the required dependencies and set the environment up prior to launching both the UI Server and Extractor service. The scripts took a number of days to write as issues arose when trying to test the scripts which took time to rectify. The scripts pull the latest code from each of the repositories master branches and begin the setup. For the UI Server, all the modules dependencies are installed, then the server is started to begin listening on port 80. A similar process occurs for the Extractor service but instead, Pip which is the Python package manager is used to install a list of requirements before starting the server on port 3000. The application is only deployed once all unit and integration tests have passed on each repository's Travis Continuous Integration (CI) builds.

3 Conclusions

In this report, the aims of project which were to design and develop a social network analysis system that facilitates Recruitment Consultants and others wishing to visualise their social networks were discussed. The background to the project was outlined along with the reasons why there is a need for this project which was primarily due to the fact there is no commercially available product that provides the same services as the discussed solution. The technologies that were used to develop the implemented solution were investigated in depth along with their competitors to ascertain the best overall set of core technologies that could meet the needs of the project. Section 2 defines the System, it included the requirements of the system that were compiled and documented based on the needs and wants of Users and gaps in the current market for such a product. During the requirements gathering process, a lot of requirements were gathered but due time limitations not all could be included and are therefore later documented in Future Developments section. When the requirements gathering process was complete, the system architecture was designed along with prototypes of the User facing elements of the system. An iterative methodology was employed to strive for continuous improvement of the overall system that lead to an implementation that meets the needs of Users. Sections 2.5 presented specifics about how the system was implemented and explains some other algorithms used in the system. The implementation details discussed were some of the more difficult aspects of the system to implement due to the level of complexity.

Building great software requires dedication to testing at all stages throughout the lifecycle of a project. In Section 2.7, there were three different levels of testing discussed, they included low level Unit Testing, Integration Testing, and Customer Testing. The specifics of how they were conducted along with the results from Customer Testing were presented. They showed that out of 10 customers who evaluated the system, 94% were satisfied the system meets the proposed goals and provides a User Experience that is easy to use and learn. Performance issues were always high on the priority list, a range of different approaches used to enhance the performance of the system were discussed, the metrics based on the observed improvements were provided. They showed that the total time in importing a graph was reduced by up to 78% by implementing caching. Graph visualisation loading times were greatly reduced by the outlined performance enhancements. Throughout the development of the project there were a number of different issues encountered in gaining access to the LinkedIn API which lead to the use of web scraping to gather the required data. Unfortunately, as of April 1st 2017, LinkedIn released a new User Interface which made crawling the site more difficult than it previously was. These changes mean the LinkedIn graph import functionality no long functions as outlined.

4 Acknowledgements

I would first like to thank my project supervisor Mr Vikas Sahni of the School of Computing at the National College of Ireland. The door to Mr Sahni's office was always open whenever I ran into a trouble spot or had a question about my project or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to thank my family members especially my mother Karen who spent a vast amount of time proof reading this paper and other documentation.

Finally, I would like to thank all the members of faculty at the School of Computing who have given advice throughout my project.

Exograph Technical Report

5 Further Development

As of today, the system is continuing to grow and evolve. There is ample opportunity to further add advanced functionality with more resources. Since there is a lack of commercially available solutions, the project has the potential to become a feasible start up with adequate funding.

5.1 Advanced Analytics

As the User base grows, so does the volume of data within the system. Functionality could be included to allow multiple Users to combine their social graphs to visualise their network and see how and whom they have mutual relationships with. This would be a very powerful tool to a recruiter, potentially increasing their chances of finding a suitable person for a role.

5.2 Micro Services

Micro services are an architecture that allow complex systems of a service oriented field to evolve from being large monoliths to smaller services that focus on doing one thing well. Micro services are highly scalable but complex. They bring problems to the surface that would not be encountered in a monolith architecture such as concurrency issues.

5.3 Graph Merging

As the system evolves, functionality could potentially be added for allowing related graphs from different data sources to be merged into one graph to allow for more advanced visualisations. A User may have a graph from Twitter and another from LinkedIn. It may not be obvious to them at the time that there are hidden relationships which would not become evident until they are combined.

5.4 Custom Graphs

As the system continues to advance, a useful feature could enable Users to create graphs from scratch. This would have a lot of application areas such as profiling people and would further compliment the system.

6 **References**

- [1] Roesslein, J., 2009. *tweepy Documentation*.
[Online] <http://tweepy.readthedocs.io/en/v3>, 5.
[Accessed 10 December 2016].

- [2] Holzschuher, F. and Peinl, R., 2013, March. Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j. *In Proceedings of the Joint EDBT/ICDT 2013 Workshops (pp. 195-204)*. ACM.

- [3] Henderson-Sellers, B. and Edwards, J.M., 1990. The object-oriented systems life cycle. *Communications of the ACM*, 33(9), pp.142-159.

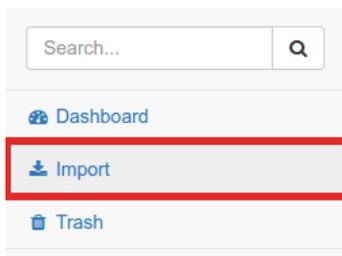
- [4] Venners, B., 2002. Test-driven development. *A Conversation with Martin Fowler, Part V (Retrieved 2017)*, <http://www.artima.com/intv/testdrivenP.html>.

7 Appendix

7.1 User Manual

7.1.1 Importing a Graph

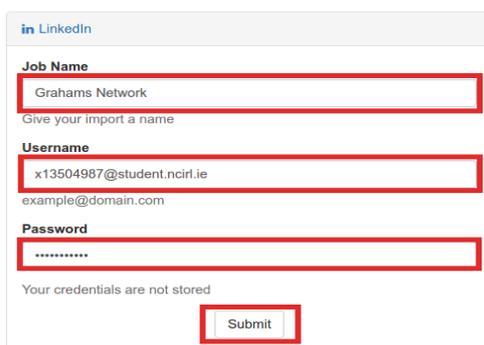
Step 1



Navigate to the import page by selecting the “Import” tab in the left vertical navigation bar.

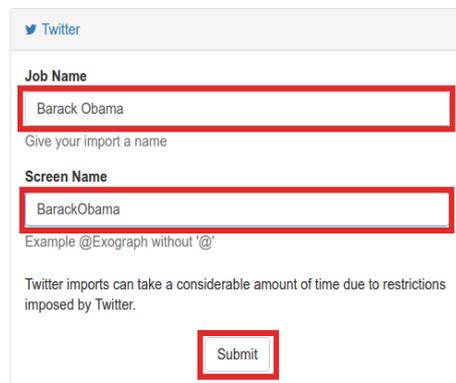
Step 2 (LinkedIn Import)

Import

A screenshot of the LinkedIn import form. The form has a header with the LinkedIn logo and the text "LinkedIn". Below the header are three text input fields: "Job Name" (containing "Grahams Network"), "Username" (containing "x13504987@student.ncirl.ie"), and "Password" (containing "*****"). Each of these three fields is highlighted with a red rectangular border. Below the password field is a "Submit" button, also highlighted with a red rectangular border. A note below the password field reads "Your credentials are not stored".

Complete the three text fields as can be seen highlighted in red. The username and password fields are your LinkedIn credentials not Exograph, on input complete click the “Submit” button located at the bottom.

Step 2 (Twitter Import)

A screenshot of the Twitter import form. The form has a header with the Twitter logo and the text "Twitter". Below the header are two text input fields: "Job Name" (containing "Barack Obama") and "Screen Name" (containing "BarackObama"). Each of these two fields is highlighted with a red rectangular border. Below the screen name field is a "Submit" button, also highlighted with a red rectangular border. A note below the screen name field reads "Example @Exograph without '@'". A larger note below the submit button reads "Twitter imports can take a considerable amount of time due to restrictions imposed by Twitter."

To launch a Twitter job, complete the job name and screen name fields. Note the screen name should not contain “@” at the beginning, on field competition select the “Submit” button located at the bottom of the form.

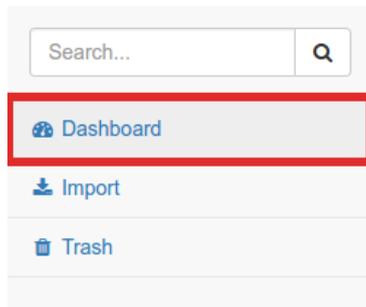
Step 3

A screenshot of a "Import Launched" popup window. The window has a title bar with the text "Import Launched" and a close button (an 'x' icon). The main content area displays the following information: "Job ID: 202", "Name: Grahams Network", "Status: running", "Type: LINKEDIN", "Complete: false", and "Start Date: 04-28-2017 02:31 PM". At the bottom right of the popup is a "Close" button.

After selecting “Submit”, if the job was successfully launched a popup will appear containing summary information, in the event of a failure an error will be displayed.

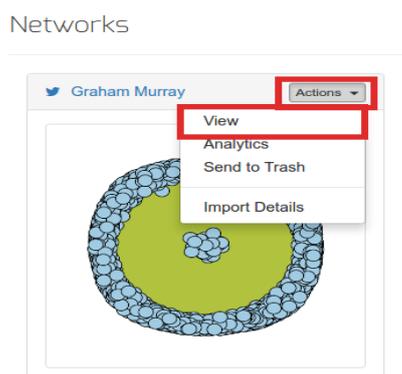
7.1.2 Viewing an Imported Graph

Step 1



Navigate to the dashboard page by selecting the “Dashboard” tab in the left vertical navigation bar.

Step 2



After the page loads, the networks area will become visible. Choose a network, select the actions button located in the top right corner of each network and select view in the dropdown menu. A loading spinner will appear and the graph will be displayed once loaded.

Step 3



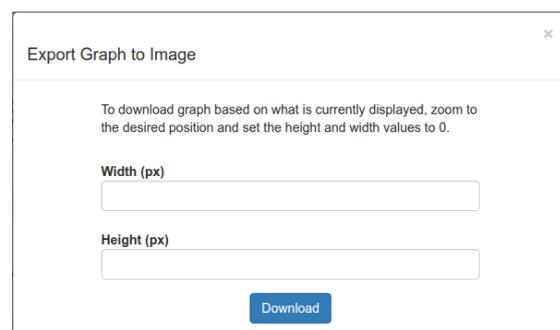
To view the vertical navigation bar, select the blue button with a right point caret which will enable the menu.

Step 4



To export a graph to an SVG image, select the blue button with an image icon located in the centre of the screen below the horizontal navigation bar.

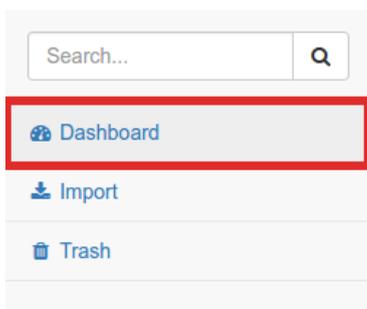
Step 5



The “Export Graph to Image” dialog will now appear. Complete the “height” and “width” fields and select “Download”.

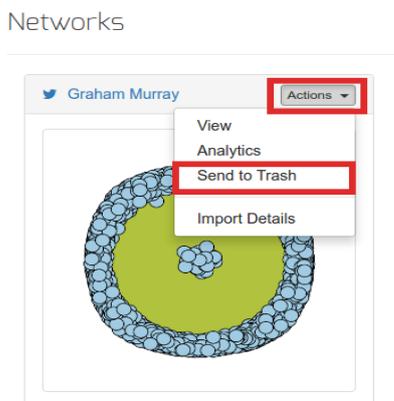
7.1.3 Deleting a Graph

Step 1



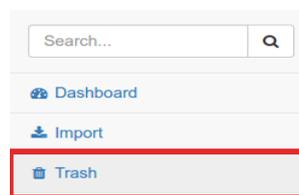
Navigate to the Dashboard page by selecting the “Dashboard” tab in the left vertical navigation bar.

Step 2



After the page loads, the networks area will become visible. Choose a network, select the actions button located in the top right corner of each network and select “Send to Trash” in the dropdown menu. The trash count located on the top of the Dashboard will now be updated with the current number of items marked as trash.

Step 3



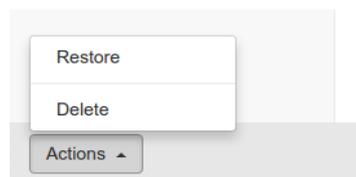
To view the trash bin, select the “Trash” tab in the vertical navigation menu.

Step 4



The Trash page will contain a table with all items marked as trash. To delete or restore items, select the desired items by clicking the checkboxes located in the last row of the grid. Alternatively, items may be selected by clicking a row. To select all items, click the checkbox located in the grid header.

Step 5

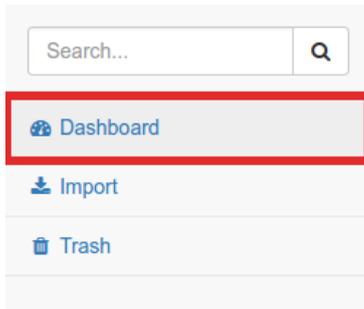


When one or more items have been select, the actions button located in the footer toolbar will become enabled. Click the “Actions” button and select either “Delete” or “Restore”.

Exograph Technical Report

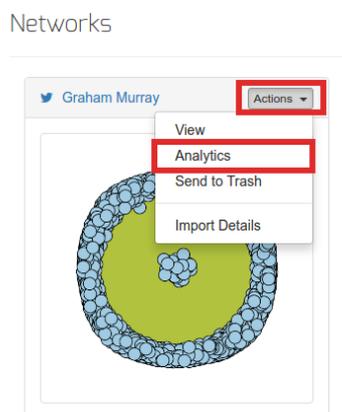
7.1.4 Accessing Graph Analytics

Step 1



Navigate to the Dashboard page by selecting the “Dashboard” tab in the left vertical navigation bar.

Step 2



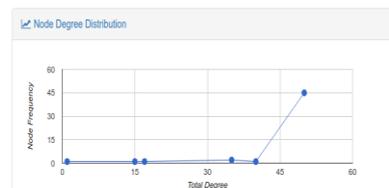
After the page loads, the networks area will become visible. Choose a network, select the actions button located in the top right corner of each network. Select “Analytics” in the dropdown menu and wait for the page to load.

Step 3

Overview	
Name	Graham Murray
Total Nodes	2252
Total Edges	2393
Mean Degree	1.06

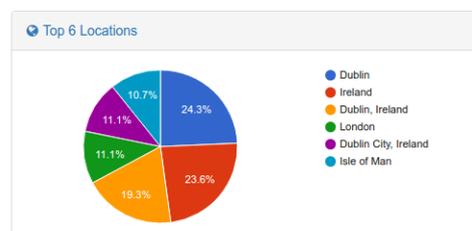
The “Overview” table contains information such as the total count of nodes and edges in a graph, along with displaying the mean node degree.

Step 4



The node degree distribution represents the total number of outgoing and incoming edges a node has. The x-axis of the chart contains the aggregated total degree while the y-axis represents the number of nodes.

Step 6

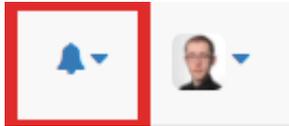


The pie chart located at the bottom of the page provides an analysis of the “Top 6 Locations”.

Exograph Technical Report

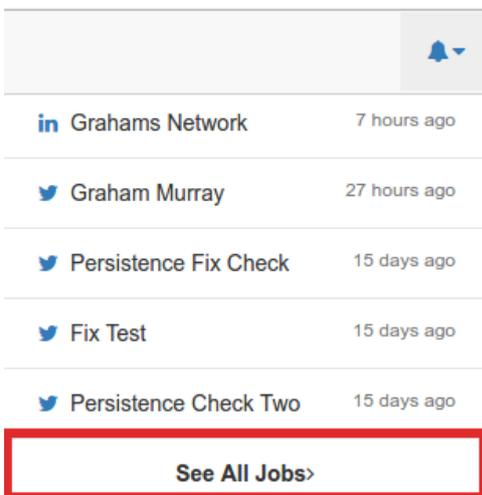
7.1.5 View Executed Jobs

Step 1



Navigate to the top horizontal navigation bar located in the header of all pages and select the bell icon as seen highlighted in red.

Step 2



After clicking the bell icon, a dropdown menu will appear. This contains a listing of the last five jobs ran in ascending order. Click the “See All Jobs” button located at the bottom of dropdown menu as seen highlighted in red.

Step 3

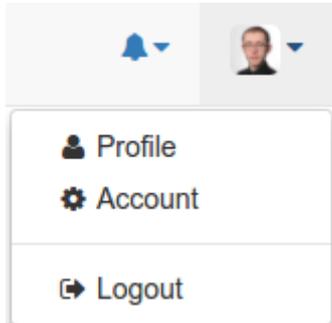
Jobs

Type	Name	Started	Total Time	Status	Success	Complete
in	LinkedIn	59 days ago	489 seconds	Successfully imported	Yes	Yes
in	Database Test	59 days ago	478 seconds	Successfully imported	Yes	Yes
in	New	59 days ago	136 seconds	Successfully imported	Yes	Yes

The Jobs page includes a listing of all executed jobs order by the date ran in ascending order. The progress of launched jobs can be tracked from here.

7.1.6 Updating Account Information

Step 1



Navigate to the profile image button located in the horizontal header navigation bar of each page and select “Account”

Step 2A – Update Details

Account Settings

A screenshot of the 'Update Details' form. The form has a title 'Update Details' and a blue 'Update' button at the bottom. The form contains several input fields, each with a red border: 'Firstname' (Graham), 'Surname' (Murray), 'Company' (NCI), 'Country' (Ireland), and 'Email' (graham.murr@yahoo.ie).

To update your existing details, locate the “Update Details” form which will be populated with your existing data. Make the required changes and click the blue “Update” button.

Step 2B – Change Preferences

A screenshot of the 'Change Preferences' form. The form has a title 'Change Preferences' and a blue 'Change Settings' button at the bottom. There is a checkbox labeled 'Receive email notification of import completion' which is checked. The checkbox and the 'Change Settings' button are highlighted with red borders.

To change your preferences, change the “Receive email notification of import completion” checkbox and click the “Change Settings” button to submit the form.

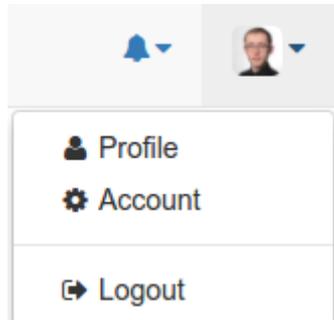
Step 2C – Change Password

A screenshot of the 'Password Reset' form. The form has a title 'Password Reset' and a blue 'Reset' button at the bottom. There are three input fields, each with a red border: 'Old Password', 'New Password', and 'New Password Confirm'.

To change your password, enter your current password in the top field, along with your new password in the remaining two fields. Click the “Reset” button to persist your new password.

7.1.7 Changing a Profile Image

Step 1



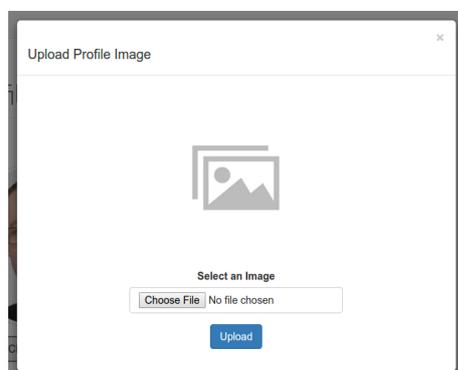
Locate the profile image and click the image in the top navigation bar, a dropdown menu will appear. Select the "Profile" option and wait for the Profile page to load.

Step 2



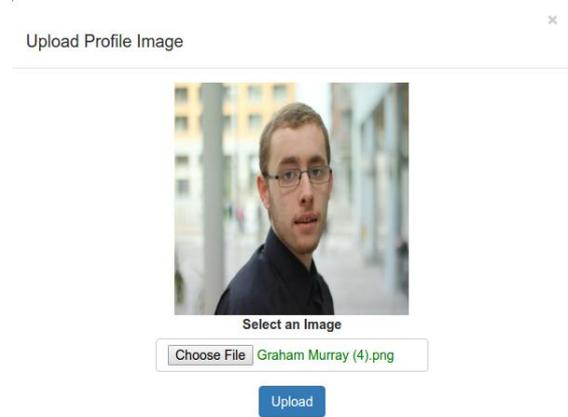
Select the "Change Image" button.

Step 3



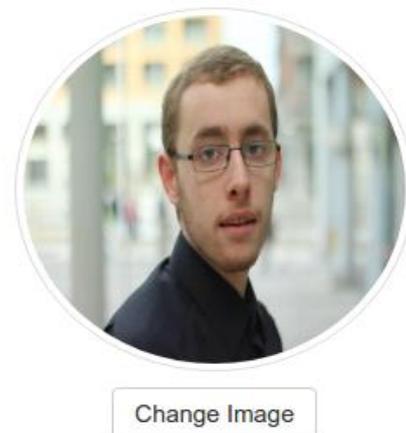
Choose browse files and select a PNG format file, all other formats will not be accepted.

Step 4



After selecting the image, click the "Upload" button.

Step 5



The page will reload and your new image will be available system wide.

7.2 Project Proposal

Objectives

The main objective of the Project is to design and develop an application that will enable people to visually analyse social connections. The solution will enable recruiters to understand how they connect with other people. From the perspective of a Recruiter, the solution will facilitate them to find people who are commonly connected to either themselves or others in order to locate a suitable person for a job they may be hiring for. The solution will use multiple channels such as LinkedIn, Twitter and Facebook as data sources. The project will demonstrate multiple Computer Science concepts such as Graph Theory, Parsing, Distributed Computing, and Data Visualisation. The process of gathering and performing computations on the data will be transparent to the end user. From the time they set a target to when the graph is rendered they will not know anything about the process. The application will be based around a Software as a Service model in a cloud environment. Data that is imported and modified by Users will be stored and retrievable from any location worldwide. Data privacy and ethics surrounding the project in the real world environment could be problem. Therefore, such factors will be taken into account appropriately.

Time management will play a vital role in meeting the required deadlines to ensure that the project stays on track and that all requirements are fulfilled. The scope of the initial solution will focus on getting one social media source working. Following on from that, the application can then be extended to provide functionality for integrating with other data sources.

Exograph Technical Report

Background

Social Network theory has been in existence for a long time. People have been analysing social networks since medieval times. With the rise of the digital age, the capabilities of such analytics have become more accessible. The application of the area is used extensively by historians, sociologists, and economists alike.

In today's world millions of people connect with each other using social network mediums such as Facebook, Twitter, LinkedIn and Instagram. Social Networks can be described as a graph of people who come in contact with each other through such a medium. Everyday there are massive amounts of User generated data as a result of such relationships. People constantly rely on social mediums to keep in contact with each other and meet new people, forging new relationships. From a Computer Science aspect this opens a world of possibilities to exploit such data. Graph theory is a very powerful mathematical concept which can be applied to solving such problems as representing relationships between groups of people.

LinkedIn currently have 450 million Users worldwide. Their domain area is connecting professionals together. From the point of view of a Recruiter looking to hire someone it can be a very tedious process manually attempting to see who they are connected to and finding people who are connected with the same profession to find a suitable person for a role. A Recruiter would be a prime example of a person who would be connected with people in a cohesive way. As the old saying goes, good people know other good people.

Facebook's domain area is connecting people for the purpose of personal relationships, while Twitter is more oriented toward voicing opinions and following people based on ones' interests rather than personal relationships. Twitter has more loosely cohesive relationships compared to Facebook and LinkedIn. Combing such relationships over multiple social network mediums could prove to be very effective as it allows more than one comprehensive view to be taken on a dataset.

Data visualisation is the graphical representation of data in all-inclusive way. It enables us to perceive things that would not be otherwise possible. By combing such social network graphs and visually representing them, we can get new insights from data about relationships that we didn't even know existed.

Exograph Technical Report

Technical Approach

To date there are a vast number of research papers that discuss the topic of graphing and analysing social distributions for the sociology purposes and investigating how groups of people interact. From a technological perspective there are other solutions accessible but not is a Software as a Service (SaaS) model in a cloud environment. The first stage of the project deals with researching available data sources and how their Application Programming Interfaces (APIs) work to ascertain what is possible and what is not. The initial stage of extracting the data via a service's API will involve a functional language that is fast at processing data and capable of multi-threading.

From my findings, the best approach to use lean towards the Knowledge Discovery from Data (KDD) technique used widely in the area of Machine Learning to build advanced systems for Data Mining which has proven to be both robust and scalable. When all the data is gathered it will need to be cleansed into a form that is representative of the domain. Graph theory is a concept which I have come across briefly during my studies and I therefore will need to conduct further research and learning surrounding the area before jumping into designing and developing the application.

A further area research is how to store complex graph based models in a database that fits the problem being solved. These findings are discussed more in the 'Technical Details' section. The frontend will require investigation around various different open source data visualisation graph render frameworks that will integrate with the back end to visually display the complex graph data structure models that can be mutated by end-users. The key concept behind the application is that it should be a metadata driven application. A graph can be represented in any data form. These models will be modified by a User and then persisted in a graph oriented database. The metadata approach is one that has proven to be scalable.

Syndio Social is an enterprise level application that already exists on the market. It allows employers to visualise how their employees are connected using different data sources, and includes integrated Data Visualisation tools to aid in the process.

Exograph Technical Report

Special Resources Required

There are numerous books and online resources required that are available at the National College of Ireland's Library. Research papers focusing on the area for the purpose of social studies were identified, Algorithmic Problem Solving by Roland C is available at the library. The book focuses on areas that are applicable to the project. Discrete Mathematics by Johnsonbaugh R. is another resource required as it describes the core mathematical theories behind graphs and will be a valuable asset with regards to further increasing understanding of the discipline. "Analyzing Social Media networks with NodeXL" is a research paper that was conducted at the University of Maryland alongside Microsoft in 2009. After reading the paper, it was clear it would be an excellent project resource as it provides guidelines based on past experiences.

The final application will be deployed to Amazon Web Services (AWS). Therefore, access to an AWS student account will be needed. AWS provide a number of different instance types based on the required needs of a system although it won't be known exactly the type of instance required until certain performance metrics are gathered.

Exograph Technical Report

Technical Details

The first part of the project will focus on developing a script that will use the APIs exposed by the Social Media platforms. This piece will require a functional language. Python is a good fit as it is fast and efficient at process large volumes of data and has numerous libraries that support web scraping such as BeautifulSoup and others for integrating with RESTful APIs. Most of the data gathered will be JSON, Python supports the parsing of JSON and XML so the options available are broad depending on the type of data being extracted from the data sources. Python also has support for running computations over multiple processes or even distributed over a network. The pre-processing of the data can also be implemented using Python which has great backing for data manipulation.

MongoDB is a document oriented NoSQL database which is capable of storing large JSON data structures. It has support for sharding and replication, therefore it would work well in a distributed environment. Another storage mechanism that would work well is Neo4J. It's a graph oriented database it doesn't work well with storing JSON structures, but works well at storing groups of objects.

For the server side component of the web backend NodeJS fits the requirements well. It has a diverse set of libraries available such as ones for working with MongoDB and Neo4J. There are other libraries for spawning background processes for triggering the Python services to fetch, parse and load the data into the database. Node has its own package manager for installing application dependent libraries and takes the hassle out of managing such dependencies manually. For the frontend piece a templating engine such as Jade or Handlebars will be used alongside Bootstrap for the responsive layout and styling.

There are multiple data visualisation rendering engines available for graphs. One such solution to the problem is VisJS which is a dynamic JavaScript browser based visualisation library that can handle large volumes of data at any given time. It also allows for the manipulation of the rendered graph which is a requirement of the project.

Exograph Technical Report

7.3 Monthly Journals

7.3.1 September

Achievements

This month the main achievement was to finalise the project idea and to conduct research in the chosen areas domain. The project I have decided on is a Social Profiling Cloud application which will use graph algorithms to represent people's relationships and friendships. This type of application will have many uses. It could be used by government agencies such as police forces, airport security, private security to conduct checks on people to see who they're connected to using data visualisation. The architecture will be cloud based, therefore any previously generated graphs can be re-rendered, viewed and edited. I also researched the possibility of use artificial intelligence to flag any suspicious people, something that could be extremely useful considering the heightened public security threat after terrorist attacks such as the 2015 Paris bombings which took place near the Stade de France. I spent a significant amount of time investigating the possible technologies at my disposal which I haven't used before as I want to challenge myself to learn a new programming language for the purpose of the Software Project. The most prominent ones being NodeJS and Java and Python. I have no real experience with NodeJS and think it would fit really well into the problem being solve. It's very useful for getting projects up and running quickly and takes a lot of time out of writing boiler plate code while also being extremely fast while on the other hand I have a vast amount of experience using Java and feel competent using it.

Reflection

I feel that this month I achieved quite a lot in terms of deciding an idea. I'm very happy with how the concept is evolving and am looking forward to starting to work on the Project Proposal and Requirements documents. I feel that my project approval presentation went very well, all three judges were very interested in the idea which gave me confidence in knowing that it's achievable. Other than that, I'm overall looking forward to what's to come.

Intended Changes

During the presentation the judges suggested possibly using a static dataset to get the initial project off the ground in case there are technical issues encountered. This is a very valid point and is something that will be investigated.

Exograph Technical Report

Supervisor Meetings

For this month I haven't had a meeting with a supervisor as I am yet to find one. I emailed Michael Bradford in mid-September to discuss the possibility of him being my supervisor but am yet to receive a reply. I have also thought about asking Ralf Beirig about being my mentor and will email him once my project has been approved as I feel his area of expertise applies to the problem.

Exograph Technical Report

7.3.2 October

Achievements

This month I achieved a lot in terms of research and completing the required documentation. My focus for the beginning of the month was to find the appropriate academic supervisor who has the skill set that the project requires. I looked through the list of approved project ideas submitted by lectures and noticed the Vikas Sahni had an idea that was similar to mine. I decided to contact him and see if he was available to further discuss the idea and understand what the differences were between mine and his idea. This was the real initial starting point in beginning to research what technologies fit the problem and seeing what has already been developed in the field of social network theory. During September I had already conducted some preliminary research to gain a footing in the area. It was clear from the start a functional programming language such as Python or R would be required for the data extraction process. I started learning Python in the Data Application Development module which has given me a good refresher as it has been a while since I've used it. I continued to research different storage mechanisms looking at a document oriented NoSQL database such as MongoDB but it doesn't really fit in well with the project requirements. I concluded that a graph oriented database would work well in terms of performance. Based on what I have seen from other projects of a similar nature I have decided to use Neo4j. I looked into how it works and the advantages and disadvantages of using it.

As the month progressed onwards, I began to work on the Project Proposal. I decided to do a brainstorming session to ascertain some of the answers that would be required in the proposal. This proved to very successful as it helped me to further gain an understanding into what it is I'm trying to achieve. The proposal took me some time to complete as I still had some unanswered questions that required further investigation. After I reflected on what I felt I was doing wrong I took a different approach and looked at the questions from a different perspective. I completed the project proposal a few days early and decided to bring it to Vikas for a review before I submitted it. I wanted to get the proposal completed early so I could start on the Requirements Specification as I was heading away on the weekend of reading week and knew I would have a lot of new work from other modules to complete over reading week.

For the Requirements Specification I completed the first two sections over the weekend of the 22nd of September this ensured I wouldn't be rushing to complete it as the deadline approached. I made good progress on the requirements document and almost have it completed one week early. A lot of time of time was spent further brainstorming what functional requirements the project needed. It is almost a year since I've done any requirements gathering as I was on work placement for six months. Eamon done a requirements gathering seminar during one of the lectures. I found this to be of great help.

Exograph Technical Report

Task Log

Friday 14 th
Research best data sources
Look into best functional programming
Start Project Proposal
Monday 17 th
Search for available data visualisation frameworks
Research suitable backend technologies
Finalise Draft Project Proposal
Tuesday 18 th
Meeting with Vikas at 1:30
Wednesday 19 th
Polish and upload proposal documents
Research hosting platforms (Azure and AWS)
Thursday 20 th
Uploaded project proposal
Saturday 22 nd
Further research data sources and graph oriented databases
Monday 24 th
Started on the requirements specification
Wednesday 26 th
Applied to LinkedIn for API access
Complete first section of the requirements spec
Thursday 27 th
Completed User Requirements Definition
Started October Reflective Journal
Conducted Research on Neo4J and Cytoscape
Friday 28 th
Brainstorm Functional Requirements
Started Functional Requirements
Monday 31 st
Completed Functional and Non Functional Requirements

Exograph Technical Report

Reflection

Overall I'm very happy with the progress being made. The deadlines I have set in the project plan are being met ahead of time. As a result, the pressure of the workload has reduced as I am completing it ahead of schedule. I feel that I could begin to work on getting some of the basic functionality of the project setup such as a login system and begin to start testing the technologies that were researched.

I am also very happy with my academic supervisor Vikas. He has been of great help in guiding and pointing me towards different technologies that he's familiar with and thinks could work well.

Intended Changes

Before meeting with Vikas, my initial idea was to build a social network analysis tool that would allow people to view other people's social graph as well as their own. The APIs provided by Facebook and LinkedIn don't allow this unless you have permission from the person who you want to conduct the analysis on. I decided to go with Vikas's idea of allowing a person such as a Recruiter analyse their own personal social graph.

One Data Visualisation framework researched was VisJS. At first it seemed like it had the required functionality I needed but after further investigation it doesn't. VisJS isn't a graph specific framework and therefore doesn't have full support for allowing graphs to be manipulated. I've decided that Cytoscape fits in well with the requirements instead of VisJS.

Exograph Technical Report

Supervisor Meetings

I emailed Vikas on the 11th of October to ask if he could be my academic supervisor. He agreed so we organised our first meeting for Wednesday 12th of October. During this meeting we discussed what was required for the project. Below are the minutes from the meeting.

- Research best data source (LinkedIn, Facebook, Twitter, Instagram)
- Decide what functional language would suit the solution best (Python, R)
- Look into a data visualization rendering library. One that I came across was VisJS
- Research web app back-end and front-end technologies, possibly NodeJS, Jade, and ReactJS.
- Next meeting Tuesday 18th Oct to review project proposal (Meeting Room 3)

The next follow up meeting we had was on Tuesday 18th of October. I had completed the Project Proposal after our previous meeting. I met with Vikas at 1:30 to review the draft project proposal. This was a very helpful meeting as I got to see some of the things I had overlooked in the proposal that I would've missed if Vikas didn't reviewed it.

The next meeting we had was on the 28th of October. During it we discussed how to do a good Requirements Specification and what would be my plan for the coming week. Below are the minutes from the meeting.

- LinkedIn require an application to be submitted in order to gain access to their related-connections API. I have made an application to gain access which takes 15 days to be approved.
- I made the suggested changes to the Project Proposal and uploaded it to Moodle.
- I look into graph databases. The best solution that I feel fits the requirements is Neo4J.
- Started to work on the Requirements Specification.
- I found a better data visualisation library that is graph specific. It's called Cytoscape.js and has all the required functionality that is be detailed in the Requirements Spec.

Exograph Technical Report

7.3.3 November

Achievements

This month was a difficult month due to the workload from other modules. At the beginning of the month I took another look at the project plan and reevaluated it to ensure that it took into account my commitment to other modules. I set personal goals to be achieved by the end of the month. The first goal was to make an application to gain access to the LinkedIn API. As of recently, LinkedIn have tightly restricted access to their API. In order to gain access to an end point an application needs to be submitted describing the purpose for which access is required along with the uses cases. I spent some time putting this application together with the hope that access would be grant for educational purposes. Unfortunately, the application was unsuccessful. This posed a major problem as the primary use case for the project focuses on professional networks for the purposes of helping recruitment consultants find suitable candidates for roles they're hiring. The problem was discussed with Vikas and we decided that he would take over on resubmitting a new application. Afterwards, the next goal was to take a deeper look into the Twitter and Facebook APIs to make sure the project was still viable. The Facebook API also proved to be a problem. A friends list can be accessed for the current User using authentication through the API but not using App Authentication unless the person who the friends list is being requested for gives the App permission to access the information.

As the time continued to pass, the next item on the agenda was to start working on the prototype and technical report document. I always aim to have documents and projects finished approximately a week early so time can be spent ensuring they're of high quality when submitting. The aim was to start doing an hour of work each day on the technical report so it would be completed early. This approach worked well and the document was compiled and on its first draft a week before the deadline. By this time, I had to temporarily put it aside so I could focus on other projects and revisit it four days before the deadline. Progress is going extremely well on my other projects. I was in a position where I could dedicate two days of work on the prototype when the technical report was submitted. At the beginning of the semester I had done some work in my spare time to get a NodeJS instance up and running along with a basic User Interface so it would form the foundations for the prototype.

Exograph Technical Report

Task Log

Monday 7 th
Meeting with Vikas 3pm
Gather use cases for LinkedIn application
Research the API requirements
Thursday 10 th
Finish LinkedIn application and submit
Saturday 12 th
Setup NodeJS Server
Monday 14 th
Finish configuring server
Technical report requirements section
Thursday 17 th
Meeting with Vikas 1pm
Further research Twitter API
Saturday 19 th
Play around with Facebook API
Monday 21 st
Meeting with Vikas 10:15 am
Work on technical report
Thursday 24 th
Work on technical report
Fix problems with NodeJS server
Saturday 26 th
Finish technical report
Monday 28 th
Write script to pull data from twitter
Tuesday 29 th
Continue working on twitter script
Thursday Dec 1st
Meeting with Vikas 1:15pm

Exograph Technical Report

Reflection

Overall I feel this month was very stressful due to the workload and the high number of hours I have had to put in to stay on top of all the work. The main thing that had me worried for some time was trying to schedule in time for focusing on the prototype. I have a minimal prototype already but only have a number of days to left to do more work on it for the midpoint presentation.

Intended Changes

As of now, there are no intended changes but next month there could be if the second application to gain access to the LinkedIn API is unsuccessful. This would possibly mean changing either the targeted end-user or web scraping LinkedIn connections

Supervisor Meetings

The first meeting of November took place on Monday 7th of November. During this meeting we reviewed the Requirements Specification document. Below are the minutes of the meeting

- Make recommended changes to the Requirements Specification
- Discussed the application for LinkedIn access

The next follow up meeting took place on the 17th of December. During this time, we discussed the response to the application made to LinkedIn and how to next approach the project.

- Vikas took over dealing with the LinkedIn application
- Do more exploration on the Twitter and Facebook API and see if LinkedIn connections could be scraped instead.

On the 21st of November we had another meeting. The findings from the further exploration were discussed along with how to approach the technical report

- Start working on the technical report.
- Begin writing the Python scripts to gather the social network data.
- Continue to work on the prototype.

Exograph Technical Report

7.3.4 December

Goals & Achievements

This month the main goals and achievements were to finalise the prototype for the midpoint presentation. At the beginning of the month I focused on cleaning up the User interface to look well. Since LinkedIn wouldn't give access to their API I found a way around the problem by accessing the API using User Authentication instead of OAuth. This proved to work well but means that there is more work required around cleansing the data. I implemented the crawler in Python which took some time along with trial and error but eventually got it working for the presentation.

The next goal was to prepare for the presentation I put together a set of provisional slides and reviewed them with Vikas to see if anything else could be added. I started to practice my presentation so I knew the contents of the slides and wouldn't need to keep looking at them in the presentation. This meant I could engage with the audience.

After the presentation on the 19th of December we put the project aside until the end of semester exams are over.

Reflection

Overall I feel this month was very stressful due to the workload and the high number of hours I have had to put in to stay on top of all the work. I had to manage finishing all my projects along with working on the prototype. Time management was the key to this being so successful for me. I'm extremely happy with how the midpoint presentation went and feel that the hard work paid off with a grade of 99%.

Intended Changes

As of now, there are no intended changes as a way around the LinkedIn API was found.

Supervisor Meetings

At the beginning of the month I had a meeting with Vikas to talk about the presentation and what to expect and prepare for.

- Finalise Prototype
- Understand the grading rubix.

For the remained on time before the presentation Vikas was in China so we kept in contact via email.

Exograph Technical Report

7.3.5 January

Goals & Achievements

For the first two weeks of January, the project was put aside as my primary focus after Christmas was to study for my end of semester examinations which took place the start of January. The exam finished on the 14th of January. After having a well-deserved three-day break afterwards, my attention was fully focused on hitting the ground running with development. I had 10 days of free time before the final semester began. I wanted to make the most of this time as there would be no distractions.

The next major hurdle of the project was to get the Extractor Service complete as this was blocking me from developing the UI Server. I had anticipated that this would take approximately 12 days of working for 9 hours a day. In the end I got a beta version of the LinkedIn crawler complete in 4 days. Next I focused on exposing the crawler through a RESTful API written in Python using the Django framework. I first began with modelling the database since the Extractor has its own database separate from the primary Neo4J database. The Extractor database is used to store job details so they can be efficiently managed without polluting the main graph database. It was also a design decision as a service should encapsulate its own data and only expose what is needed.

The Extractor API and LinkedIn crawler were completed in 8 days. I spent a further 2 days writing a small unit test suite which is executed on a continuous integration build to ensure there have been no undesirable changes since the last successful build. I wasn't familiar with unit testing using the Django framework. This proved to be problematic as conventional mocking and stubbing wouldn't work because Django needed to be running before any tests could be executed. In the end I refactored the code to improve separations of concerns and make the code more testable. I discovered that there is a Test Framework available to specifically test Django APIs. I used this in the end to test the API views, controllers and models. The month of January was almost nearing an end, which meant it was time to begin my final semester. I continued to work on the UI Server as I was no longer being blocked by the Extractor Service and spent the last few days of January writing the logic to consume the Extractor Service. For February my plan of action is to continue working on development which also completing the Design and Analysis documentation.

Reflection

I'm very pleased with the progress being made. I revisited the project plan which is now three weeks ahead of schedule. The outlook of the project was looking bleak at the beginning of December because of the problems with LinkedIn not providing access to their API. I'm relieved now that I found a solution to the problem.

Exograph Technical Report

Intended Changes

As of January there have been no major intended changes. I have decided to use Tableau and D3.js as the visualisation frameworks instead of Cytoscape as they are more customisable.

Task Log

Saturday 14 th Jan
Write Middleware for PassportJS Login Handler
Configure Routes to use Passport
Write Unit Tests
Sunday 15 th Jan
Convert CSS to Sass
Configure Travis CI Build on Pull Requests
Monday 16 th Jan
Add NetworkX Graph Data structure to Extractor
Wednesday 18 th Jan
Unit Test LinkedIn Crawler
Write Neo4J Bulk Loader
Thursday 19 th Jan
Fix bugs in Extractor
Setup Django
Friday 20 th Jan
Write Django Models
Saturday 21 st Jan
Write the API controllers
Write API Views
Sunday 22 nd Jan
Configure Token based authentication using JWT
Unit and Integration Test API
Monday 23 rd Jan
Write logic to run crawlers concurrently
Write image generator to create image of graph
Tuesday 24 th Jan
Add more Unit Tests for the UI Server
Thursday 26 th Jan
Write UI server middleware to consume Extractor
Meeting with Vikas 1pm
Friday 27 th Jan

Exograph Technical Report

Unit Test UI Server
Add Jade Views to Launch LinkedIn job
Write reusable client side form validation logic
Saturday 28th Jan
Add dashboard views, routes and models
Unit Test new routes and models

Supervisor Meetings

The first meeting of January took place on the 23rd as I had exams. Below is a summary of what was discussed.

- Plan how the project will be deployed
- Focus on wiring up all of the components so a beta version can go live in March for User Acceptance Testing.
- Get Azure and Amazon credits.
- Keep the flame burning until the semester is complete.
- Look into the maximum number of API requests for a professional LinkedIn account.

Exograph Technical Report

7.3.6 February

Goals & Achievements

For the month of February, my primary goals and objectives were to continue working on implementation along with completing the Project Design and Analysis document. I began working on the Design and Analysis document in the beginning of February. My aim was to spend one hour per day working on it until it was complete. It took approximately 14 hours to complete and was complete one week earlier than it was due. Although it wasn't a mandatory submission I was eager to get it completed by February as it forms part of the Technical Report and would have added further pressure later in the semester.

My next focus on the project was to implement the trash functionality along with improving the visualisation piece and other functionality. This took time to do as there was a lot of working involved in it. I aimed to dedicate two hours per day to project as based on my project plan this is what will be required to complete it on time. I encountered performance problems when reading a graph from Neo4j this was mainly attributed to inefficient cypher queries. The solution which I devised was to write a custom procedure in Java that performed a traversal using a breadth first search. This greatly improved performance and reduced the original query time by 20 seconds. All the work I intended to complete was finished on time.

Reflection

I'm very happy with the progress being made with the project and feel that everything is continuing to go to plan.

Intended Changes

There are some minor changes to be made. An existing use case was to allow Users to edit and manipulate a graph but the new visualisation library being used doesn't support that functionality. I was originally using Cytoscape which supported graph manipulation but doesn't fully fit the needs to the project therefore I have had to modify this requirement.

Exograph Technical Report

Task Log

Saturday 4 th Feb
Fix login bug
Write cypher queries for retrieving graph
Sunday 5 th Feb
Write Java A* Neo4J procedure
Monday 6 th Feb
Client side visualisation
Wednesday 8 th Feb
UI server piece for converting graph to D3 model
Write Neo4J Bulk Loader
Friday 10 th Feb
Fix dashboard bugs
Setup controllers for handling D3 model
Saturday 11 th Feb
Write client side visualisation
Sunday 12 th Feb
Write client side visualisation
Monday 13 th
Write client side visualisation
Wednesday 15 th Feb
Write logic to run crawlers concurrently
Write image generator to create image of graph
Friday 17 th Feb
Add more Unit Tests for the UI Server
Saturday 18 th Feb
Start working sending item to trash bin
Sunday 19 th Feb
Continue trash bin work
Monday 20 th Feb
Add delete functionality from trash
Wednesday 24 th Feb
Write client side multi action trash
Fix toolbar issues

Exograph Technical Report

Supervisor Meetings

The first meeting of February took place on the 10th of February. The goal of the meeting was to discuss the project aims and objectives for the month.

- Look into deployment platforms more and getting credit.
- Start working on automating deployment
- Work on improving performance
- Start working on design and analysis document

The second meeting of February took place on the 24th of February. The goal of the meeting was to discuss the issues with query performance.

- Look into algorithms to improve performance
- Continue working on trash and visualisation
- Start working on the manual
- Continue on documentation

Exograph Technical Report

7.3.7 March

Goals & Achievements

For the month of March my primary aims and objectives were to improve client side graph visualisation, complete the trash functionality with a multi action grid that would make it easier for Users to bulk delete and restore graphs, and deploy all the latest changes at the end of March so vital feedback could be gained from Recruitment Consultants actively working in the field. At beginning of the month, I started on the improving the graph visualisation. This entailed adding more functionality such as:

- Panning and zooming
- Highlighting neighbouring nodes when a node is hovered on
- Reducing the opacity of non-neighbouring nodes when a node is clicked
- Mobile responsiveness
- Improving rendering performance.

This work took approximately one week to complete and was finished on the 7th of March. From the second week of March onwards I began to work on the multi action trash functionality. This took 6 days to complete and issues were encountered deleting relationships in Neo4J this slowed up the process as it took time to research possible solutions to the problem. I became ill mid-March as I had an allergic reaction to an antibiotic I was prescribed which unfortunately meant I was out of action for two weeks. When I got back to health I had to put the project work on hold as I had to work on projects for other modules. I caught back up quickly as I was way ahead before I became sick.

The final week of March I focused on deploying the project to Amazon Web Services. Issues were encountered as I was automating deployment using shell scripts and it took time to fine tune the scripts. The project was successfully deployed on 29th of March I emailed previous colleagues from a Recruitment Agency where I used to work as a Network Administrator asking if they could test the project. One the first weekend of April, 6 people began to test it and all was running smoothly. On Monday 3rd of April I received an email from a tester informing that the LinkedIn import functionality no longer was working. I spent that morning investigating the problem. It soon became apparent that the new LinkedIn User Interface (UI) was only supporting newer browsers. The crawler's user agent was set to an older version of Firefox which meant it continued to receive the existing UI but as of the 3rd of April LinkedIn rolled out changes that had a dramatic impact on the project as they have now made it even harder to use web scraping on their site. The endpoints the crawler previously exploited JSON data no longer exist. I contacted LinkedIn the first week of April explaining that I really needed access to the data and as of the 07/04/17 I am still awaiting a response. I have a backup of graphs which I previously imported so there won't be a problem with demonstrating the main application functionality.

Exograph Technical Report

Reflection

Overall I'm disappointed with the changes LinkedIn have made as the crawler took a lot of effort and time to build, approximately 3 to 4 weeks in fact. I am happy with the continuing progress that I am making and feel that the effort made is beginning to show. From a general perspective the LinkedIn changes were out of my control. I hope that they do give me access to the API as this is now the third time requesting access.

Intended Changes

If LinkedIn do grant access to the API the crawler will need to be rewritten to work with the API instead of scraping html and JSON. Currently I intend to complete the Twitter crawler which is about 50% complete as I left that part until now as the majority of the application is complete and that is one of the final things to finish.

Exograph Technical Report

Task Log

Wednesday 1 st March
Add panning functionality
Add zooming functionality
Thursday 2 nd March
Improve mobile graph mobile responsiveness
Friday 3 rd March
Reduce opacity on neighbour nodes when node is clicked
Saturday 4 th March
Reduce opacity on neighbour nodes when node is clicked
Sunday 5 th March
Reduce opacity on neighbour nodes when node is clicked
Monday 6 th March
Highlighting neighbouring nodes on hover
Tuesday 7 th March
Improve rendering choppiness
Wednesday 8 th March
Start on multi action trash restore and delete
Thursday 8 th March
Multi action trash restore and delete
Friday 9 th March
Multi action trash restore and delete
Saturday 10 th March
Multi action trash restore and delete
Monday 20 th March
Write deployment scripts
Tuesday 21 st March
Write deployment scripts
Wednesday 22 nd March
Start on view all Jobs
Friday 24 th March
Work on view all jobs

Exograph Technical Report

Supervisor Meetings

The first meeting of March took place on the 3rd of March. The goal of the meeting was to discuss the project aims and objectives for the month.

- Begin on working on the User Manual
- Continue working on graph rendering
- Next to complete is the advanced trash functionality

The second meeting of March took place on the 10th of March.

- Add user account page
- Work on improving the visualisation UX
- Finalise Twitter crawler.

The third meeting of March took place on the 28th of March.

- Get elastic IP working
- Get Users using the application
- Work on Job summary page