SOFTWARE REQUIREMENTS SPECIFICATION

For CS 4770 Group B

Prepared by:

Osede Onodenalore, Samuel Ash, Jeff Conway, Maria Aloysius.

March 17, 2017

Contents

Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 5

1.4 Reference 5

1.5 Overview 5

Overall Description 6

2.1 Product Perspective 6

2.2 Product Functions 6

2.3 User Characteristics 6

2.4 Design and Implementation Constraint 6

2.5 Assumptions and Dependencies 7

2.6 Operating Environment 7

Specific Requirements 7

3.1 User Interfaces 7

3.2 Hardware Interfaces 7

3.3 Communications Interfaces 8

3.4 Software Interfaces 8

Functional Requirements 8

4.1 User Creates a Profile 8

4.2 User Signs In 9

4.3 User Accesses Another User’s Profile 9

4.4 User Adds a Friend 10

4.5 User Creates a Post 10

4.6 User Edits the Visibility of a Post 10

4.7 User Makes a Comment on a Post 11

4.8 User Creates a Group 11

4.9 User Edits a Group 12

4.10 User Creates a Schedule 13

4.11 User Uploads a Resumé 13

4.12 User Adds Item to Lost and Found 13

4.13 User Removes an Item from Lost and Found 14

4.14 User Creates a Poll 15

4.15 User Votes on a Poll 15

Non-Functional Requirements 15

5.1 Performance 16

5.2 Scalability 16

5.3 Capacity 16

5.4 Availability 16

5.5 Reliability 16

Use Cases 17

6.1 Use Case Diagrams 17

6.2 Use Case Outlines 18

Modularization of the System 44

7.1 Why Modularization? 44

7.2 Module Diagram 45

System Modules as Black & White Box Expressions 45

8.1 Expressions by Module 45

UML DIAGRAM 66

# Introduction

## 1.1 Purpose

The intended users for this software are students at Memorial University of Newfoundland. Its purpose is for students within the university to be able to interact with each other through the system and share opinions and content. Initially, this document is to be read by our professor/client Dr. Saeed Samet and his teaching assistant Navid Shekoufa. However, we recommend that any individual who has knowledge of programming and web development may read this document.

## 1.2 Scope

This project will be written in Node.js and will use MongoDB as its database software.

Objective:

Node.js is an event driven architecture that is scalable in Web Applications with many input/output operations. One advantage of using Node.js is that the functions are designed to be non-blocking. Commands do not have be sequential and wait for the preceding command to finish.

MongoDB is an open-source cross platform environment that uses JavaScript server side execution for queries. It can be used to store files over multiple machines and provides function to manipulate files.

Benefit:

Node.js works very well with the document-oriented database program MongoDB. This program is also written in JavaScript like Node.js.

Goals:

Since Node.js is primarily used to build network web programs such as web servers, it is a better environment for this software. And we will be using MongoDB to store any data relating to the user and the associated functionalities and privileges. When necessary we will retrieve any data to use for the output of the webpages.

## 1.3 Definitions, Acronyms, and Abbreviations

**Node.js:** a JavaScript runtime environment used to develop many server applications

**MongoDB:** an open-source, cross platform database program.

**MUN:** Stands for Memorial University of Newfoundland.

**HTML:** Hypertext markup language.

**GIT:** Open source environment used to handle projects in a fast and efficient way.

**D2L:** Desire2Learn is an educational technology used at the university by professors and students.

**UC:** User Classes.

**The System:** refers to the software product this document is being specified for. Since it is a network which is being developed, it is easier to refer to the product as **the system**.

**The User:** refers to the primary user of the system at any given time. In some situations, where privileges are important, for example when discussing editing a post or group, **the user** is used to refer to the creator of the post or the owner of the group.

**The Other User:** refers to another user of the system with whom the user is interacting.

**Black box:** involves the demonstration of expected input and output while hiding, or simply not putting emphasis on, the internal workings of a use case or module.

**White box:** situation involves taking a closer look at the internal functionality for each case.

## 1.4 Reference

Resources used to create this document have been taken from the templates found on D2L for the CS 4770 course and an example of an SRS document found online from the URL <http://www.cise.ufl.edu/~dgoldste/se/SRS.doc> and <http://www.slideshare.net/indrisrozas/example-requirements-specification>.

## 1.5 Overview

This document gives a description of the software, the product, the intended users, the software and hardware constraints, assumptions, dependencies. It also gives a general idea of the product functions and it’s use cases. The use cases show the flow of control and it is portrayed using use case diagrams. Additionally, the document describes the functional and the non-functional requirements for the software.

# Overall Description

## 2.1 Product Perspective

This product is a self-contained product. Within this system, a student is able to register as long as they have a student ID and an email just like the MUN self-service application. Like many social media applications, this product enables the user to add friends, communicate with them and much more. This product assumes that the student is a verified user at the university with an email suffix “@mun.ca”, making it indirectly dependent to the MUN database.

## 2.2 Product Functions

The major functions the product must perform is:

i) create a user profile

ii) sign in a registered user

iii) display another user's profile

iv) send friend requests, add friends when accepted

v) create a post, allow user to edit own posts and comment on any post they can see.

vi) create a group, allow group owner to edit a group, allow users to join groups

vii) create a course schedule for any user

viii) allow users to upload a resume

ix) maintain lost and found section, allow users to create posts within it

x) create a poll, allow owner to delete polls, allow users to vote on polls

## 2.3 User Characteristics

The User characteristics of the intended users is novice - advanced education level, provided the user is a registered student at University. Any type of user will be able to access the software provided they are a registered student at MUN or have administrative privileges.

## 2.4 Design and Implementation Constraint

Any regular user will be able to perform any basic functions like signing in, creating a profile, adding friends, creating/editing a post, joining a group, creating and voting a poll. However, each user has to have a valid email address that ends with mun.ca, which is the way the system assumes that the user is a MUN user. Every user has privacy restrictions; the profile has limited visibility for non-friends/the public. However, friends of the user have complete access to the user's profile (with the exception of posts with the “only me” restriction) and vice-versa.

## 2.5 Assumptions and Dependencies

Any software-related dependencies will be listed in a readme file on the GIT repository. We are assuming that whoever reads the documents has already viewed this file and has an understanding.

Almost all of the capabilities of the system require that the user already has a valid account on the website.

## 2.6 Operating Environment

The Project is web-based and will be hosted on the university server. It is intended to work on any browser in any operating system. It is currently assumed that it is only viewed on a personal computer and not on any mobile environment.

# Specific Requirements

## 3.1 User Interfaces

The first interface will be our login screen. In this interface the user will have to input their username and password to gain access to their account and profile.

We intend to have an interface for creating posts and for all of the associated functionalities such as commenting, uploading images and publishing content.

As well, there will be an interface for shared interaction between users such as the study groups and lost and found.

## 3.2 Hardware Interfaces

* The system must be connected to the internet.
* The system will be deployed on a MUN server that will connected using a port number.
* Network interaction will use https

## 3.3 Communications Interfaces

The system will interact with the database to retrieve any necessary data related to the user, friends, groups and content. This will be connected via internet.

## 3.4 Software Interfaces

Along with an internet connection. The system will make indirect use of an internet browser. As mentioned before, this application is written in Node JS(v4.6.1) with use of MongoDB (v3.2.11) as the database. Other than that, the system does not tell any software what to do.

# Functional Requirements

For each of the following functional requirements, each requirement will be discussed in the following fashion:

1. Explanation: a brief description of the requirement
2. Inputs: the input required to achieve the expected output
3. Processing: what happens when the input is received by the system
4. Expected Output: an explanation of what is expected from the system for this requirement.

## 4.1 User Creates a Profile

4.1.1 The user should be able to create a profile, with an @mun.ca username required, and receive a confirmation email with a link. When the link is clicked, the user should return to the index page and be notified that their profile is ready.

4.1.2 The user must input a number of required fields, including their real name, a chosen username (which must end in @mun.ca), a password, etc.

4.1.3 The system must verify that these fields are appropriate, including ensuring that the input username includes @mun.ca. If any field is not as desired, the user will be told to change it. If all required fields are filled and meet the desired criteria, the system must create a new profile associated with the user in the database and send a confirmation email to the provided username/email.

4.1.4 Once the system has sent the confirmation email to the user, the user should be able to log in to the system at any time after the confirmation link has been accessed.

## 4.2 User Signs In

4.2.1 The user should click "sign in" and be presented with fields to enter their username and password, and then click "sign in" to be taken to their homepage.

4.2.2 The user must input their username and password correctly in order to access their profile.

4.2.3 The system should present the user with a username and (hidden characters) password field when they click "sign in”. The system must verify that the user attempting to log in exists within the database. If not, the system will inform the user that the username or password is incorrect. If so, the system must ensure that the password is correct for the associated username before logging in the user. If not, the system will inform the user that the username or password is incorrect. If so, the system should start a session and allow the user to use the system.

4.2.4 The system should present the user with their own personal homepage after logging them in.

## 4.3 User Accesses Another User’s Profile

4.3.1 The user should be able to access the profiles of other members a number of different ways, including via a search, through a received friend request, through a group, etc. The user should click that member's name to be taken to their profile.

4.3.2 The user is able to access another user’s profile a number of ways. The user must click the profile of the other user, which can be displayed to the user via a search, through a received friend request, or through a group.

4.3.3 The system must acknowledge the request of the user to view the other user’s profile page and redirect the user there. When a user navigates to another member's profile, the visibility of each post should be checked and only those posts that the user has permission to view should be displayed.

4.3.4 The user should find themselves on the profile page of the other user.

## 4.4 User Adds a Friend

4.4.1 The user should be able to send a friend request to another user by clicking a link to do so once they are on the other user’s profile page.

4.4.2 While on the profile page of the other user, the user should click “Add Friend” to begin the process of adding the other user as a friend.

4.4.3 When the user attempts to add the other user as a friend, the system must send a notification shortly thereafter to the other user. The other user may view this notification from their homepage and may choose, at their discretion, accept or decline. If the other user chooses to decline, the user will not be notified of this, but they will not be added to each other’s friends lists. If the other user chooses accept, the system must place a reference to each user in the other’s friends list, likely their username, so that it can refer to each user as being confirmed as friends with the other in the future.

4.4.4 As mentioned above, after the user attempts to add a friend, the notification of this request must be sent to the other user.

## 4.5 User Creates a Post

4.5.1 The user should be able to click "create post" while on their homepage or on the page of a group they belong to, and have the ability to post content.

4.5.2 The user must input at least once character into a text field and/or an image via a file upload field before clicking “post”.

4.5.3 The system must present the user with field(s) though which they can post content when requested. When a user creates a new post the system should update the database, display the content of the post in the appropriate place (on the user’s profile or in a group) within 3 seconds, and refresh the page.

4.5.4 The content of the post should be viewable shortly after it is posted in the appropriate location (on the user’s profile or in a group).

## 4.6 User Edits the Visibility of a Post

4.6.1 User should be able to edit the visibility of any post they have made by clicking on "edit post" and changing the visibility.

4.6.2 The user must request to edit their post, and must provide whatever changes desired (changes to text, a replacement photo, or removing the text or photo of a post so long as one of those remains), changing the visibility options for a post (public, friends only, selected list of friends or only me (the user)) or possibly deleting the post entirely.

4.6.3 The system should present the user with options for visibility, along with actual editing ability, when the user clicks "edit post”. When the user edits the visibility options of a particular post, the system should update the database with the new permission rules. The new permission rules and any changes should take effect within 3 seconds.

4.6.4 The post will be edited in any way the user desires, or removed if desired, and any changes should take effect within 3 seconds.

## 4.7 User Makes a Comment on a Post

4.7.1 The user should have the ability to click "reply" on any post they can see (on their dashboard or in a group) and should be able to post content underneath a post.

4.7.2 Once the user clicks “reply”, they must enter at least one character in the presented text field before being able to click “post”.

4.7.3 System should present the user with field(s) for posting content under a post when requested. When a user attempts to reply to a post, the system should check if that user has permission to reply based on the visibility rules of the post.

If they have permission, the system should prompt the user to enter a reply. Once submitted (and verified; ensure the text field is not empty, etc.) the database should be updated, the reply should be displayed within 3 seconds, and the user’s page should be refreshed. The system should send the owner of original post a notification that informs them someone has added a comment to their post.

If they do not have permission, the system should display a message explaining why the reply cannot be made.

4.7.4 The user’s comment should be displayed underneath the original post within 3 seconds.

## 4.8 User Creates a Group

4.8.1 The user should have access to a groups page, accessible from their homepage, on which they can create a group, or edit any owned group.

4.8.2 The user must fill out a number of fields (including group name, group purpose/description, etc.) and choose the group permission (public, private) when creating a group.

4.8.3 When the user requests to create a new group, the system should ensure the group details are acceptable (required fields are filled out, permission is set, etc.) Once accepted, the system should update the database to include the new group, add the user who created it to the group and set them as the owner. The system is then responsible for ensuring the visibility of the group (set by the owner at creation and can be changed at any time).

4.8.4 Once the user has created a group, it should become visible to all users who can see it (based upon the permission chosen by the user). The user then has control of the group to add or remove any members, create or delete any posts, and update the group details at any time.

## 4.9 User Edits a Group

4.9.1 The user should be able to edit any group they own to change the attributes of the group, add or remove members, remove posts, etc.

4.9.2 When the user requests to edit one of the groups they own, they must perform any changes they desire (add or remove members, change group name, group description, permission) before being able to click “confirm changes”.

4.9.3 The system must allow the user to edit some, if not all, attributes of the group, including possible deletion. The system is responsible for updating any edits in the database within 5 seconds.

When the user invites a user to the group, the system should send a notification to the other user asking them if they would like to join the group. If accepted, the system should update the group in the database to include the other user as a member.

When the user changes another group member's invite privileges, the system should update the other user’s privileges in the database and send a notification of their new privileges to the other user.

When the owner removes a group member from the group, the system should update the group in the database by removing the other user, and should send a notification to the other user informing them of their being removed from the group.

When the owner deletes the group, the system should update the database by removing the group, and should send a notification to all group members of its deletion.

4.9.4 After editing a group, the user should be informed their changes have been saved, and the page should be refreshed.

## 4.10 User Creates a Schedule

4.10.1 The user should be able to create a (one) schedule (from their profile page) of their courses for their friends to view.

4.10.2 When the user wishes to create a schedule of their courses, they must do so from their own profile page. The user must fill out necessary fields including course name/number, days and times of each class before they can publish their schedule.

4.10.3 The system must allow the user to input their course schedule, however, the user will be limited to a set number of courses (6) to put into their schedule. The system must store the data from the schedule and associate it with the user, and redirect the user back to their own profile page.

4.10.4 After creation of the user’s schedule, the schedule should be visible to any of the user’s friends from the user’s profile page.

## 4.11 User Uploads a Resumé

4.11.1 The user should be able to upload a (one) resumé (from their profile page) for their friends to view.

4.11.2 To display their resumé from their profile, the user must upload their resumé in an appropriate file format.

4.11.3 The system must ensure that the file format the user provides for upload can be displayed properly on the user’s profile. If it cannot be displayed properly, the user must be notified by the system to try to upload their resumé in an appropriate file format, and the system should provide a list of formats which work for the user to consider. The system is responsible for maintaining the file for the user’s resumé in the database and associating it with the user. The system should redirect the user back to their profile page

4.11.4 After the user uploads their resumé, it should be visible to any of the user’s friends from the user’s profile page.

## 4.12 User Adds Item to Lost and Found

4.12.1 The user should be able to create a post in the lost and found section (available from dashboard) about an item they have found.

4.12.2 To add an item to the lost and found, the user must include an approximate location and a short description of the item they have found, along with a method of contact (and possibly a photo).

4.12.3 The system should present user with fields to create a post about an item they have found, when requested. The user should be able to include a short description, location, and a photo of the item, so system must present fields appropriate for this.

When the user attempts to add an item to the lost and found, the system should first confirm that they are signed in. If not, give the user an error. If so, the user should be redirected to the appropriate page to enter information about the item.

When a user submits information about an item, the system should confirm the required fields are filled out (description, location found, etc.). If the fields are not filled, the system should give the user an error. If the fields are filled, the system should update the database to include the new item.

4.12.4 After adding an item to the lost and found, the item should appear near the top of the lost and found section (most recent adds to the lost and found should appear at the top of the page).

## 4.13 User Removes an Item from Lost and Found

4.13.1 Once the other user (in this case the owner of an item) has been in contact with the user (the person who found their lost item), the user should be able to remove the post for that item.

4.13.2 Upon being contacted by the other user outside of the system, the user should be able to return to the lost and found section and click “remove post” on the post that the user made corresponding to the item owned by the other user.

4.13.3 The system must ensure that any post in the lost and found section can be easily removed by the user who posted it. When attempting to remove an item from the lost and found, the user should be asked if they are sure of this action. Upon removing the item, the system should update the database to reflect this change and update the site within 3 seconds.

4.13.4 Upon removal of an item, the lost and found section should appear the same, with the exception of the item removed being no longer visible.

## 4.14 User Creates a Poll

4.14.1 The user should be able to create a poll from within a group they are in, and be able to include the question/proposal as well as multiple choices for other users to answer, up to a limit of 7 possibilities.

4.14.2 To create a poll, the user must access the page of the group they wish to create a poll in, and must provide the question which they are asking, as well as up to 7 options for other users to choose as a response.

4.14.3 The system is responsible for storing the poll and all answers to it in the database, and make it visible to all members of the group it was created in. After a set amount of time (selected by the user), the poll should close and the answers to it should be visible for 48 hours.

4.14.4 Upon creation of the poll, it should be visible to all members of the group in which it was created.

## 4.15 User Votes on a Poll

4.15.1 The user should be able to see polls in any group they're in, and answer any polls they wish to.

4.15.2 To vote on a poll, the user must simply select whichever answer they wish, and click “submit”.

4.15.3 The system must ensure that all members of a group can see and vote on a poll created within that group. The system must check to make sure that the user has not already answered the poll. If not, the system must give the other user the option to choose a response. If so, the other user should not be able to select a response.

4.15.4 After voting on a poll, the user should no longer be able to vote on that poll, and their answer should be stored in the database.

# Non-Functional Requirements

In the maintenance of this social media network, the system is responsible for many aspects of operation - it must constantly maintain and update the database, complete page requests and constantly check for permissions for visibility of posts, groups, etc. given to certain users.

# 5.1 Performance

The system is responsible for loading all pages, updating the database and completing all requests within only a few seconds. This is the basic expectation of most websites of its kind.

## 5.2 Scalability

The system must be able to maintain a database of increasing or decreasing size. At launch, there will not be many pages to maintain, but with users interacting and creating posts, groups, etc., the system must be able to steadily maintain a growing database of pages and content.

## 5.3 Capacity

With multiple users accessing the system simultaneously, the system is responsible for maintaining multiple sessions and keeping them separate. What one user can see or do may have little to no relation to what another user can see or do.

## 5.4 Availability

At all hours of the day, the system must be available to any users who possess a valid @mun.ca email account. Anyone with a valid @mun.ca email address may register and use the system to their liking.

## 5.5 Reliability

Throughout its widespread use, the system must reliable enough that it can be expected to have no errors. It is also imperative that each user’s session and access to each user’s profile be restricted to that user only. The system must ensure at all times that sessions do not cross over, and each account must be protected so that it can only ever be used by the owner of that account.

# Use Cases

## 6.1 Use Case Diagrams

## 



## 6.2 Use Case Outlines

UC1 Create a Profile

UC2 Sign In

UC3 View Profile

UC4 Accessing Another User's Profile

UC5 Add Friend

UC6 Comment on a Post

UC7 Create a Post

UC8 Edit the Visibility of a Post

UC9 Edit a Post

UC10 View Groups

UC11 Create a Group

UC12 Join a Group

UC13 View Schedule

UC14 Create Schedule

UC15 Edit Schedule

UC16 View Resume

UC17 Upload Resume

UC18 Check Lost and Found

UC19 View My Lost and Found Items

UC20 Add Item to Lost and Found

UC21 View Polls

UC22 Create Poll

UC23 Vote a Poll

6.2.1 UC1 - Flow of Events for the *Create a Profile* Use Case

UC1.1 Preconditions:

None.

UC1.2 Main Flow:

The user clicks the “Sign-up” button from the main page and is presented with a form in which they must enter their username, password, email [E1], student ID, campus, and gender [E2]. The user will then be required to access their email and click the confirmation link from the confirmation email.

UC1.3 Subflows:

None.

1.4 Alternative Flows:

[E1] The email that is entered must be a valid @mun.ca email address. If the email does not end with @mun.ca an error is given.

[E2] All the fields must be filled out. If any fields are empty, an error is given.

UC1.png

6.2.2 UC2 - Flow of Events for the *Sign In* Use Case

UC2.1 Preconditions:

None.

UC2.2 Main Flow:

After clicking the “Sign-In” button, the user is prompted to enter their account information. Once the user enters their account name and password and the “Submit” button is clicked [E1] [E2], the user will be brought to their profile.

UC2.3 Subflows:

None.

UC2.4 Alternative Flows:

[E1] All fields must be filled out. If any fields are left blank, an error is given.

[E2] Must be a valid username/password combination. If they information is not valid, an error is given.

UC2.png

6.2.3 UC3 - Flow of Events for the *View Profile* Use Case

UC3.1 Preconditions:

1. The user is signed in [UC2].

UC3.2 Main Flow:

Upon creating an account [UC1] or signing in [UC2], the user is brought to their profile. The user may also click their name/profile picture from the top of the page to be brought to their profile [E1].

UC3.3 Subflows:

None.

UC3.4 Alternative Flows:

[E1] A user is automatically signed out after 20 minutes of inactivity. If the user attempts to view their profile after being logged out, they will be prompted to sign-in [UC1].

UC3.png

6.2.4 UC4 - Flow of Events for the *Accessing Another User's Profile* Use Case

UC4.1 Preconditions:

The user is signed in [UC2].

UC4.2 Main Flow:

The user clicks another user's name to view their profile [S1][S2][S3][S4][S5].

UC4.3 Subflows:

[S1] The user enters a name in the search bar and clicks “Search” which brings up a list of users based on the name they entered.

[S2] After accessing their own profile [UC3], the user clicks on a friend from their suggested friend's list and is brought to their profile.

[S3] After accessing their own profile [UC3], the user clicks on a user in a friend's friend list and they are brought to the users profile.

[S4] The user clicks a friend request from their notification area and can click their name to view their profile.

[S5] After accessing their own profile [UC3] or another user's profile [UC4], the user clicks on the name of someone who has posted on their newsfeed and is brought to their profile.

UC4.4 Alternative Flows:

None.

UC4.1.png

UC4.2.png

6.2.5 UC5 - Flow of Events for the *Add Friend* Use Case

UC5.1 Preconditions:

The user is signed in [UC2].

UC5.2 Main Flow:

After the user accesses another user's profile [UC4], they click the “Add Friend” button to send that user a friend request, or they can accept a friend request from their notification area [S1].

UC5.3 Subflows:

[S1] The user clicks on a friend request notification and clicks the “Accept” button.

UC5.4 Alternative Flows:

None.

UC5.png

6.2.6 UC6 - Flow of Events for the *Comment on a Post* Use Case

UC6.1 Preconditions:

1. The user is signed in [UC2]

2. The user is viewing another user's profile [UC4]

UC6.2 Main Flow:

After the user accesses another user's profile [UC4], they click on the “Add Comment” button under a post on the other user's timeline[E1]. The user is prompted to enter text [E2], pictures [E3], hyperlinks, or other text-based information.

UC6.3 Subflows:

None.

UC6.4 Alternative Flows:

[E1] The post cannot be empty. If the post is empty an error given.

[E2] The post cannot be longer than 1000 characters. If the post exceeds the character limit an error is given.

[E3] Only .jpeg, .png, .gif files can be uploaded up to the size of 50MB. If the image is an invalid type or too large an error is given.

UC6.png

6.2.7 UC7 - Flow of Events for the *Create Post* Use Case

UC7.1 Preconditions:

1. The user is signed in [UC2].

2. The user is viewing their profile [UC3].

UC7.2 Main Flow:

After clicking “Create Post” on their newsfeed, the user is prompted to enter text [E2], pictures [E3], hyperlinks, or other text-based information. The user may select the visibility settings of their post [S1]. The user then clicks “Post” and their post is displayed on their newsfeed [E1].

UC7.3 Subflows:

[S1] The user can choose to set the visibility of their post to something other than their default setting. If they choose to select a list of friends, their friends list is displayed and they can check a box next to the friend that they wish to allow visibility.

UC7.4 Alternative Flows:

[E1] The post cannot be empty. If the post is empty an error given.

[E2] The post cannot be longer than 1000 characters. If the post exceeds the character limit an error is given.

[E3] Only .jpeg, .png, .gif files can be uploaded up to the size of 50MB. If the image is an invalid type or too large an error is given.

UC7.png

6.2.8 UC8 - Flow of Events for the *Edit Visibility of a Post* Use Case

UC8.1 Preconditions:

1. The user is signed in [UC2].

2. The user is viewing their own profile [UC3].

UC8.2 Main Flow:

After finding a post they wish to edit the visibility of, the user clicks the “Edit Visibility” button and select how they want to change it [S1][S2][S3][S4].

UC8.3 Subflows:

[S1] The user selects “Everyone,” allowing anyone to see the post.

[S2] The user selects “Friends Only,” allowing only users on their friends list to view the post.

[S3] The user selects “User Only,” allowing only themselves to view the post.

[S4] The user selects “Only Certain Friends,” and is prompted to select friends from a list to allow to view the post.

UC8.4 Alternative Flows:

None.

UC8.png

6.2.9 UC9 - Flow of Events for the *Edit a Post* Use Case

UC9.1 Preconditions:

1. The user is signed in [UC2].

2. The user is viewing their own profile [UC3].

UC9.2 Main Flow:

After finding a post they wish to edit, they click the “Edit Post” button and select how they want to change it [E1][E2].

UC9.3 Subflows:

None.

UC9.4 Alternative Flows:

[E1] The post must meet the requirements for a post [UC7]. If it does not an error is given.

[E2] The user may cancel the edit and leave the post the same.

UC9.png

6.2.10 UC10 - Flow of Events for the *View Groups* Use Case

UC10.1 Preconditions:

1. The user is signed in [UC2].

UC10.2 Main Flow:

The user clicks the “Groups” button from the top of the page and is taken to the list of groups.

UC10.3 Subflows:

None.

UC10.4 Alternative Flows:

None.

UC10.png

6.2.11 UC11 - Flow of Events for the *Create a Group* Use Case

UC11.1 Preconditions:

1. The user is signed in [UC2].

2. The user is on the Groups page [UC10].

UC11.2 Main Flow:

After navigating to the groups page, the user clicks “New Group.” The user enters the name and description of the group, sets the privacy settings, and clicks “Create Group.” [E1] The user then has the option to invite friends.

UC11.3 Subflows:

UC11.4 Alternative Flows:

[E1] The group must have a name and description. If either fields are blank, an error is given.

UC11.png

6.2.12 UC12 - Flow of Events for the *Join a Group* Use Case

UC12.1 Preconditions:

1. The user is signed in [UC2].

UC12.2 Main Flow:

After finding a group they wish to join, they click “Join Group.” [S1][S2][S3]

UC12.3 Subflows:

[S1] The user finds a public group from the groups page [UC10][E1].

[S2] The user recieves an invite from a friend in their notifications tab and clicks “Accept Invite.”

[S3] The user requests an invite to a private group and a message is set to the group owner. Once approved by the owner, the user is placed in the group.

UC12.4 Alternative Flows:

[E1] If the user tries to join a private group without an invite, an error is given.

UC12.png

6.2.13 UC13 - Flow of Events for the *View Schedule* Use Case

UC13.1 Preconditions:

1. The user is signed in [UC2].

UC13.2 Main Flow:

The user clicks “Schedule” on the top of the page and is brought to their schedule page.

UC13.3 Subflows:

None.

UC13.4 Alternative Flows:

None.

UC13.png

6.2.14 UC14 - Flow of Events for the *Create Schedule* Use Case

UC14.1 Preconditions:

1. The user is signed in [UC2].

2. The user is on the Schedule page [UC13].

UC14.2 Main Flow:

After navigating to the Schedule page, the user clicks “Create Schedule.” [E1] The user is brought to a new screen which prompts them to add courses and set the times [E2].

UC14.3 Subflows:

None.

UC14.4 Alternative Flows:

[E1] Users can only have one schedule. If a schedule already exists, the “Create Schedule” button is replaced with the “Edit Schedule” button.

[E2] The user may not have two courses at the same time. If a conflict exists an error is given.

UC14.png

6.2.15 UC15 - Flow of Events for the *Edit Schedule* Use Case

UC15.1 Preconditions:

1. The user is signed in [UC2].

2. The user is on the Schedule page [UC13].

UC15.2 Main Flow:

After navigating to the Schedule page, the user clicks “Edit Schedule.” [E1] The user is brought to a new screen which prompts them to add or remove courses and set the times and clicks “Create Schedule” [E2].

UC15.3 Subflows:

None.

UC15.4 Alternative Flows:

[E1] If the user does not have a schedule, the “Edit Schedule” button is replaced with the “Create Schedule” button.

[E2] The user may not have two courses at the same time. If a conflict exists, an error is given.

UC15.png

6.2.16 UC16 - Flow of Events for the *View Resume* Use Case

UC16.1 Preconditions:

1. The user is signed in [UC2].

2. The user is viewing their profile [UC3].

UC16.2 Main Flow:

The user clicks “View Resume” from their profile page.

UC16.3 Subflows:

None.

UC16.4 Alternative Flows:

None.

UC16.png

6.2.17 UC17 - Flow of Events for the *Upload Resume* Use Case

UC17.1 Preconditions:

1. The user is signed in [UC2].

2. The user is viewing their resume page [UC16].

UC17.2 Main Flow:

From their profile, the user clicks the “Upload Resume” button and is prompted to select a file to upload [E1].

UC17.3 Subflows:

None.

UC17.4 Alternative Flows:

[E1] Only files of the type .pdf, .docx, and .txt can be upload. If another file type is selected an error is given.

UC17.png

6.2.18 UC18 - Flow of Events for the *Check Lost and Found* Use Case

UC18.1 Preconditions:

None.

UC18.2 Main Flow:

The user clicks “Lost and Found” from the top of the page.

UC18.3 Subflows:

None.

UC18.4 Alternative Flows:

None.

UC18.png

6.2.19 UC19 - Flow of Events for the *View My Lost and Found Items* Use Case

UC19.1 Preconditions:

1. The user is signed in [UC2].

2. The user is viewing the lost and found [UC18].

UC19.2 Main Flow:

The user clicks the “My Items” button on the top of the lost and found page.

UC19.3 Subflows:

None.

UC19.4 Alternative Flows:

None.

UC19.png

6.2.20 UC20 - Flow of Events for the *Add Item to Lost and Found* Use Case

UC20.1 Preconditions:

1. The user is signed in [UC2].

2. The user is viewing their lost and found items [UC18].

UC20.2 Main Flow:

The user clicks the “Add item” button and enters a description of the object, the area where it was found, and a photo (optional) [E1].

UC20.3 Subflows:

None.

UC20.4 Alternative Flows:

[E1] A description and location are required. If either field is blank, an error is given.

UC20.png

6.2.21 UC21 - Flow of Events for the *View Polls* Use Case

UC21.1 Preconditions:

1. The user is signed in [UC2].

UC21.2 Main Flow:

The user clicks “Polls” from the sidebar and is brought to the list of polls.

UC21.3 Subflows:

None.

UC21.4 Alternative Flows:

None.

UC21.png

6.2.22 UC22 - Flow of Events for the *Create Poll* Use Case

UC22.1 Preconditions:

1. The user is signed in [UC2].

2. The user is on the polls page [UC23].

UC22.2 Main Flow:

The user clicks “Create Poll” from the top of the page and enters the poll information (name, voting options, course the poll is about, and who is allowed to vote) [E1][E2].

UC22.3 Subflows:

None.

UC22.4 Alternative Flows:

[E1] The poll must have a name and at least one voting field. If field are blank, an error is given.

[E2] The user must have the course the poll is about on their schedule. If it is not an error is given.

UC22.png

6.2.23 UC23 - Flow of Events for the *Vote on a Poll* Use Case

UC23.1 Preconditions:

1. The user is signed in [UC2].

2. The user is on the polls page [UC21].

UC23.2 Main Flow:

The user clicks the “Vote” button under the poll they wish to vote on [E2]. The user is brought to a new page that prompts them to enter answers to the poll questions [E2].

UC23.3 Subflows:

None.

UC23.4 Alternative Flows:

[E1] The user must have the course the poll is about on their schedule. If it is not, an error is given.

[E2] The user must fill out all the fields of the poll. If any are left blank, an error is given.

UC23.png

# Modularization of the System

## 7.1 Why Modularization?

To ease in the design process, we have modularized our system. Decomposing the system into modules has been done to:

* shorten development time by dividing these modules amongst our group members
* decouple these parts of our system so that they can work together in unison without completely relying upon each other
* present our system in a way which makes it easier to study and understand

## 7.2 Module Diagram

# System Modules as Black & White Box Expressions

Input

Output

Each use case can also be expressed as a black box and a white box situation. See the **definitions, acronyms, and abbreviations section (1.3)** for more information on what this means.

Detailed Process

Input

Output

Figure 8.01: Example of a black box and white box.

## 8.1 Expressions by Module

**Module 1 - User Model**

8.1.1 M1 - Create a Profile

Input:

<username>: {type: String}, <password>: {type: String}, <email>: {type: String}, <name>: {type: String}, <student\_id>:{type: Number}, <gender>: {type: String}, <campus>: {type: String}.

Additional input (not part of the model): <confirm\_password>.

Output:

The system creates unique user object in the database associated with the user which can be accessed via login with the corresponding username and password, and redirects the user to the login page.

Detailed Process:

1. Is <email> unique?
   1. YES > continue.
   2. NO > alert the user, redirect back to signup page with all fields unchanged.
2. Is <username> unique?
   1. YES > continue.
   2. NO > alert the user, redirect back to signup page with all fields unchanged.
3. Is <student\_id> unique?
   1. YES > continue.
   2. NO > alert the user, redirect back to signup page with all fields unchanged.

4. Does <email> end with “@mun.ca”?

* 1. YES > continue.
  2. NO > alert the user, redirect back to signup page with all fields unchanged.

5. Is <password> == <confirm\_password>?

* 1. YES > continue.
  2. NO > alert the user, redirect back to signup page with all fields unchanged.

4. Generate confirmation link valid for 24 hours.

5. Email confirmation link to email specified in <username/email>.

* 1. Did the user click link?
     1. YES > continue
     2. NO > wait

6. Output.

8.1.2 M1 - Sign In

Input:

<username>, <password>.

Output:  
The system starts a session and redirects the user to their dashboard.

Detailed Process:

1. Is <username> == username associated with a user profile in the database? (i.e. is the user registered?)
   1. YES > continue
   2. NO > alert the user, redirect back to login page with all fields unchanged.
2. Is <password> == password associated with the user profile in the database associated with <username>?
   1. YES > continue
   2. NO > alert the user, redirect back to login page with all fields unchanged.
3. Output.

**Module 2 - Profile**

8.1.3 M1 - View Profile

Input:

The user clicks <My Profile> from the dropdown menu, or their <name> in a comment or post.

Output:

The user is redirected to their profile page, from which they can view their friends list, friend requests, schedule and resumé.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.

2. Is the profile page being loaded corresponding to the current user?

* 1. YES > continue
  2. NO > see 8.1.4 step 3 onward. **Note** that clicking “My Profile” should never cause this.

3. Output.

8.1.4 M1 - View Another Profile

Input:

The user clicks <name> of another user.

Output:

The user is redirected to the other user’s profile page.

Detailed Process:

1. Is session expired?

* 1. NO > continue
  2. YES > log the user out and redirect the user to login page with all fields empty.

2. Is the profile page being loaded corresponding to the current user?

* 1. NO > continue
  2. YES > see 8.1.3 step 3 onward. **Note** that the current user can access their own profile by clicking their own username on a post or comment they have made.

1. Check the privileges for all posts on the other user’s profile page. Prepare to display only posts which the user is eligible to view.
2. Output.

8.1.5 M1 - Add Friend

Input:

The user clicks <+Add Friend>. **Note** that this option will not be available if the user and the other user are already friends, or if the user has already sent the other user a friend request.

Output:

The other user has been sent a friend request from the user. The other user must then click <+Add Friend> here in order to become friends with the user.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Display the user’s name under “Friend Requests” on the other user’s profile page, which is only visible to the other user when he or she is logged in.
3. Output.

**Module 3 - Comments**

8.1.6 M2 - Comment on a Post

Input:

<postId>: {type: String}, <userId>: {type: String}, <text>: {type: String}, <date>: {type: String}, <author>: {type: String}, <isEdited>: {type: Boolean}.

Output:

The user’s comment is posted publicly underneath the other user’s post. **Note** that commenting on a post which the user does not have permission to view is not possible.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is <text> empty or whitespace?
   1. NO > continue
   2. YES > alert the user, redirect back to previous page.

3. Output.

**Module 4 - Posts**

8.1.7 M2 - Create a Post

Input:

userId: {type: String}, author: {type: String}, text: {type: String}, date: {type: String}, image: {type: Number}, visible: {type: Number}.

Output:

The user’s post is posted, according to the visibility selected for the post (which is defaulted to the user’s preferred default), either in the group in which they are posting or on the dashboard.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is <text> empty or whitespace?
   1. NO > continue
   2. YES > alert the user, redirect back to previous page.
3. Output.

8.1.8 M8 – Edit a Post

Input:

The user clicks <edit post>.

Output:

The user is redirected to the edited post.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is input valid? (more than just whitespace)
   1. YES > continue
   2. NO > alert the user, redirect back to the post with all fields unchanged.
3. Output.

8.1.9 M9 – Edit the Visibility of a Post

Input:

The user clicks <visible to>.

Output:

The user is now shown a few toggle icons to change the visibility of the post.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is input valid? (no selection made)
   1. YES > continue
   2. NO > alert the user to make a selection
3. Confirms the user to make the changes, and prepares to make the changes to the visibility of the post.
4. Output.

**Module 5 - Groups**

8.1.10 M3 - View Groups

Input:

The user clicks <groups>.

Output:

The user is redirected to the groups page and may view any groups which they have permission to.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Prepare to display only the groups which the user has permission to view.
3. Output.

8.1.11 M3 - Create a Group

Input:

userId: {type: String}, owner: {type: String}, name: {type: String}, description: {type: String}, privacy: {type: String}.

Output:

A group is created, with the user as the owner, with the given input.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is input valid? (<name> and <description> are not empty or whitespace)
   1. YES > continue
   2. NO > alert the user, redirect back to create group page with all fields unchanged.

3. Output.

8.1.12 M3 - Join a Group

Input:

The user clicks <join group>. Note that this assumes the user is already viewing the groups page or accepting an invite. In the case of an invite to a group, step 2 of the detailed process can be skipped.

Output:

The user is added to the list of users in the group and can now perform actions within it.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. If the group is request to join, the owner of the group must first accept the user’s request. Has request been accepted?
   1. YES > continue
   2. NO > wait
3. Output.

**Module 6 - Schedule**

8.1.13 M4 - View Schedule

Input:

User clicks <My Schedule> from the dropdown menu or while on their profile page.

Output:

The user can view their schedule, if they have made it.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Does the user have a schedule created?
   1. YES > display it
   2. NO > display a page which says they have not created their schedule yet.

3. Output.

8.1.14 M4 - Create Schedule

Input:

<slot <day>, <time>>, <course <name>, <slot>, <slot>, <slot>, <slot>>

Output:  
The user’s schedule for a semester is created.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is input valid? (more than just whitespace)
   1. YES > continue
   2. NO > alert the user, redirect back to create schedule page with all fields unchanged.
3. Does every <course> have at least one <slot> which isn’t blank?
   1. YES > continue
   2. NO > alert the user, redirect back to create schedule page with all fields unchanged.
4. Is every <course <name>> in the format “X0000” or “XX0000”? (eg. CS4770, EN1020, M1000)
   1. YES > continue
   2. NO > alert the user, redirect back to create schedule page with all fields unchanged.
5. Is every <slot <day>> a valid day between Monday and Friday?
   1. YES > continue
   2. NO > alert the user, redirect back to create schedule page with all fields unchanged.
6. Is every <slot <time>> a valid time, no more than 3 hours in length, between 0:00 and 23:59?
   1. YES > continue
   2. NO > alert the user, redirect back to create schedule page with all fields unchanged.
7. Output.

8.1.15 M4 - Edit Schedule

Input:

The user alters their schedule and clicks <submit>.

Output:

The user’s schedule is altered.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is input valid? (more than just whitespace)
   1. YES > continue
   2. NO > alert the user, redirect back to edit schedule page with all fields
3. Perform steps 3-6 from 8.1.14.
4. Output.

**Module 7 - Resumé**

8.1.16 M5 - View Resumé

Input:

The user clicks <view resume> while on their profile page.

Output:

The user’s resumé is displayed, if it has been uploaded.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Does the user have a resumé uploaded?
   1. YES > display it
   2. NO > display a page which says they have not uploaded their resumé yet.

3. Output.

8.1.17 M5 - Create Resumé

Input:

The user clicks “upload resumé” while on their profile page. The user will be inputting <file>, which is their resumé uploaded via file upload, and clicks “submit”.

Output:

The user’s resumé is uploaded and is visible to their friends from the user’s profile.

Detailed Processes:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is <file> in a valid format? (One of the formats requested by the system before the user uploads)
   1. YES > continue
   2. NO > alert the user, redirect back to upload resumé page without uploading resumé.
3. Output.

**Module 8 - Lost and Found**

8.1.18 M6 - View Lost and Found

Input:

The user clicks <Lost and Found> in the dropdown menu.

Output:

The user is redirected to the lost and found page and may view any items posted.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Prepare to display all posts, with most recent at the top.
3. Output.

8.1.19 M6 - View My Lost and Found Items

Input:

The user clicks <view my items> to see the items they have posted in the lost and found section.

Output:

The user is taken to a page with a list of the items they have uploaded to the lost and found.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Does the user have any posts in the lost and found?
   1. YES > display them
   2. NO > display a page which says they have not made any posts in the lost and found yet.
3. Output.

8.1.20 M6 - Add Item to Lost and Found

Input:

description: {type: String}, location: {type: String}, posterId: {type: String}, poster: {type: String}.

Output:

A found item is added to the lost and found by the user in case another user owns it.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is input valid? (more than just whitespace)
   1. YES > continue
   2. NO > alert the user, redirect back to add item page with all fields unchanged.
3. Output.

**Module 9 - Polls**

8.1.21 M7 - View Polls

Input:

User clicks <view polls>. **Note** that for this, the user can be viewing polls either for a group or public polls.

Output:

User is redirected to the polls page.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Prepare to display the appropriate polls, with most recent at the top.
3. Output.

8.1.22 M7 - Create a Poll

Input:

<question>, <option 1>, <option 2>, <option 3>, <option 4>, <duration>

Output:

A poll is created and is available for all users who can see it to vote on it.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Is input valid? (more than just whitespace, at least for option 1 and 2)
   1. YES > continue
   2. NO > alert the user, redirect back to create poll page with all fields unchanged.
3. Is duration a valid period of time? (0-72 hours)
   1. YES > continue
   2. NO > alert the user, redirect back to create poll page with all fields unchanged.
4. Output.

8.1.23 M7 - Vote on Poll

Input:

User clicks whichever option they desire for the poll.

Output:

The user’s vote is counted and reflects upon the current results.

Detailed Process:

1. Is session expired?
   1. NO > continue
   2. YES > log the user out and redirect the user to login page with all fields empty.
2. Prepare to update the current results based upon the user’s vote.
3. Output.

**Modules**

**User**

Inputs: username: {type: String},

password: {type: String},

email: {type: String},

name: {type: String},

student\_id:{type: Number},

gender: {type: String},

campus: {type: String}.

Output: These inputs are stored in the database. These inputs then become accessible to the user.

Detail Processes: doesConfirmPassMatchPass()

isPlaceholderString()

isPlaceholderInteger()

doesEmailAlreadyExist()

doesEmailEndInMun()

isPlaceholderEmpty()

**Post**

Inputs: userId: {type: String},

author: {type: String},

text: {type: String},

date: {type: String},

image: {type: Number},

visible: {type: Number}.

Output: Post.visible->setting

Detail Processes: changeVisiblilty(setting)

Input->setting(boolean), the intended setting to change to. If true, the post is visible to all. If false, the post is visible to just the user.

getComments()

Input->id, this is used to match the post id with the post id of the comment model

Output: Comment

editText(newText)

Input->newText(String), the new text to replace what is current in the post.

Output post.text->newText

removeImage()

Output: removes image from post

getId()

Output:post.\_id

Delete(id)

Input->post.\_id(number), the id of the post to be deleted from the database.

Output: post deleted from the database

**Comment**

Inputs: postId: {type: String},

userId: {type: String},

text: {type: String},

date: {type: String},

author: {type: String},

isEdited: {type: Boolean}.

Detail Processes: addComment(userId, postId, text)

Input->userId(number), the id of the associated user, or the author of the comment

->postId(number), the id of the post from the comment

->text(String), the text of the comment

Output: new comment created in the database

editComment(id, newText)

Input->id(number), the id of the comment to be edited

->newText(String), the text to be replace the existing comment text

Output: comment.Text->newText

deleteComment(id)

Input->id(number), the id of the comment to be deleted

Output: the comment is deleted from the database

**Group**

Inputs: userId: {type: String},

owner: {type: String},

name: {type: String},

description: {type: String},

privacy: {type: String}.

Detail Processes:

createGroup(userId, name, description)

Input-> userId, the user that creates the group will be set to admin by default

->name(String), the name of the group

->description(String), the description for what the group is for

Output: new group created in the database

addUser(id, user.\_id)

Input: id(number), the id of the group

user.\_id, the intended user and the id to be added

Output: user added to group

Join(id, user.\_id)

Input: id(number), the id of the group

user.\_id, the intended user and the id requesting to join

Output: request is added to group

handleRequest(id, user.\_id, option, request)

Input: id(number), the id of the group

user.\_id, the intended user and the id requesting to join

option(boolean), this will be determined by the admin. If true, the request is accepted, else the request is rejected

request, the pending request

Output: if option is true, user is added to group, else, request is removed

deleteGroup(id)

Inputs: id(number), the id of the group

Output: the group is deleted from the database

editName(newName)

Inputs: newName(String), the name to replace the existing name of the group

Output: group.name->newName

editDescription(newDesc)

Inputs: newDesc(String), the new description to replace the existing description of the group

Output: group.Description->newDesc

**Schedule**

Inputs: slot: {type: Number},

days: {type: String},

time: {type: String},

name: {type: String},

userId: {type: String}.

Detail Processes:

addSchedule(slot, Days, Time, courseName, scheduleName)

Inputs: ->slot(number), the slot number of the course

->Days (String array), the days for each course

->Time (String, array), the times for each course (Array has to correspond with Days)

->courseName(String), the name of the course

->scheduleName(String), the name of the schedule.

Output: new schedule created in the database

editCourseName(courseName, newCourseName)

Input: ->courseName(String), the current course Name

-> newCourseName(String), the new course name to replace the current one

Output: Schedule.courseName->newCourseName

editScheduleName(scheduleName, newScheduleName)

Input: ->scheduleName(String), the current schedule Name

-> newscheduleName(String), the new schedule name to replace the current one

Output: Schedule. scheduleName->newScheduleName

deleteSchedule(id)

Input: id(number), the id of the schedule to be deleted

Output: schedule is deleted from the database

**Resume**

Inputs: userId: {type: String},

file: {type: File}.

Detail Processes

addResume(userId, file)

Inputs: ->userId(number), id of the user uploading the resume

->file(file), the resume file

Output: new resume is created in the database

deleteResume(id)

Inputs: id(number), the id of the resume

Output: the resume is deleted from the database

**Lost and Found**

Inputs: description: {type: String},

location: {type: String},

posterId: {type: String},

poster: {type: String}.

Detail Processes:

addObject(user.id, location, description, title)

Inputs: ->user.id(number), the id of the user adding to the list of lost and found objects

->location(String), the location of the object  
->description(String), the description of the object

Output: new lost and found object added to the database

deleteObject(id)

Inputs: id(number), id of the lost and found object

Output: Lost and found object deleted from the database

**Poll**

Inputs: group.id, number

User.id, number

Rating, number (array)

Title, String

newPoll(group.id, user.id, title)

Inputs: ->group.id(number), the id of the corresponding group

->user.id(number), the id of the user creating the poll, which is the author

->Title(String), the title of the poll

Output: new poll is created in the database

addRating(ratingNumber, user.id)

Inputs: -> ratingNumber(number), rating out of 5 for the poll

->user.id(number), id of the corresponding user adding the poll

Output: ratingNumber and user.id are added to the rating array

GetAverrageRating(rating)

Output: Average of all the ratings.

**Interfaces**

The User Interfaces (UI) will be use JavaScript, HTML5 and CSS code

**Controller**

The controller class objects will be used as intermediaries between the views and the mongo DB database.

# UML DIAGRAM

https://raw.githubusercontent.com/samuelash94/teamproject/e19ce71079648b87ba6b750c8aeed14becad8c1b/milestone%202/UML.png