ISAD157 Coursework

REQUIREMENTS REPORT

David Wood

Contents

1 Introduction

The following is documentation on the coursework assignment for the ISAD157 module. This will contain the requirements, design process and a description of the final implementation. It will also contain a link to the remote git repository that houses the files for the project.

If you want to understand the final implementation and don't want to read through the full design process then skip to the implementation section (section 3, otherwise carry on reading.

Note that I will be referring to the system user as "System User" and a user that is stored in the database as "User".

2 Design

2.1 Requirements

Using the coursework specification document I have identified that the system is required to do the following:

- Store user information
- Store friend connections between users
- Display information of a particular user

And allow the System User to do the following:

- Read information of a particular user
- Update information of a particular user
- Delete information of a particular user
- Delete friend connections of a particular user
- Create a user
- Create User friend connections

I have used the terms Create, Read, Update and Delete to relate to CRUD so it is easier to identify which functions require which SQL statements further down the design route. Also note that information relates to any data that has an identifier relating to the user in question – this is described later in the document.

2.2 UML Diagrams

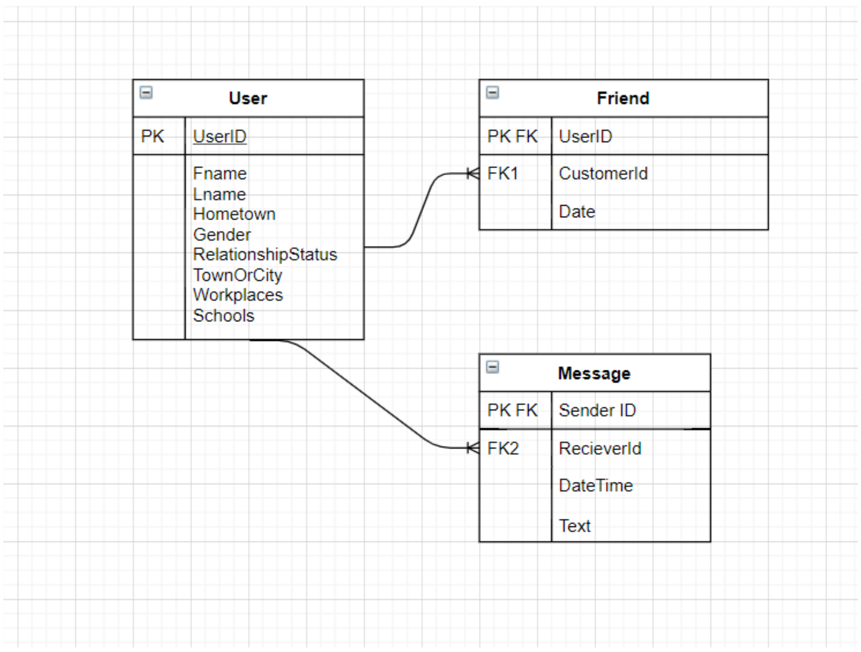
2.2.1 Entity Relationship Diagrams

The entities within the database are as follows:

- User
- Friend
- Message

2.2.1.1 Initial Relationship Diagram

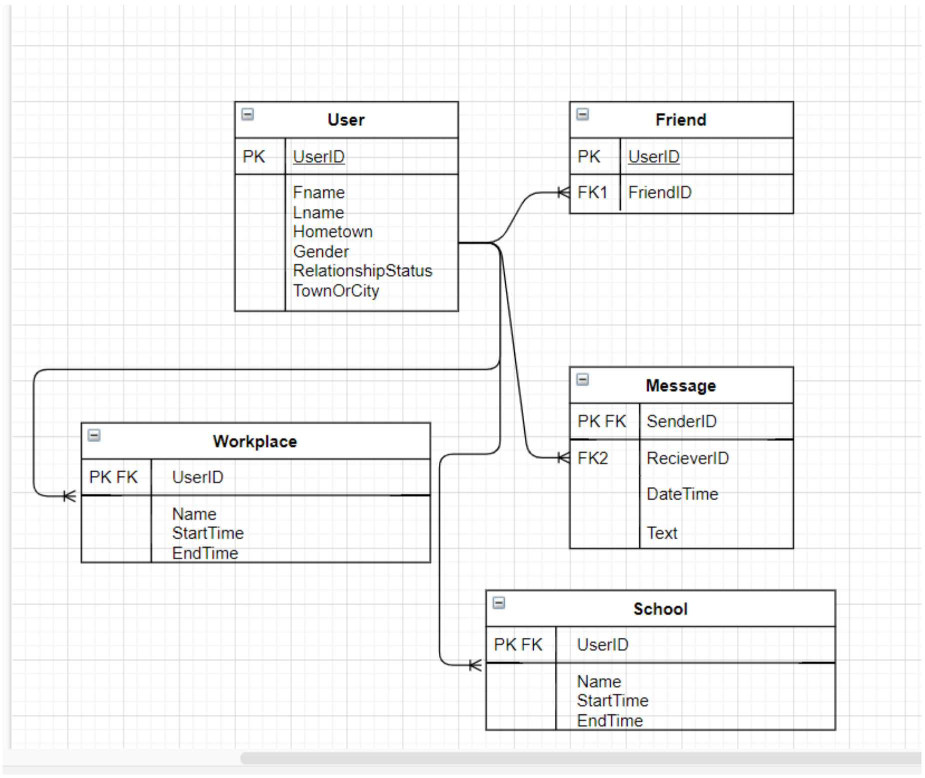
The relationship between them is such that a user can have many messages and many friends.



The issue with this diagram is that a user may have many workplaces and many universities/schools. This creates a problem when designing the database as it leads to repetition in the system.

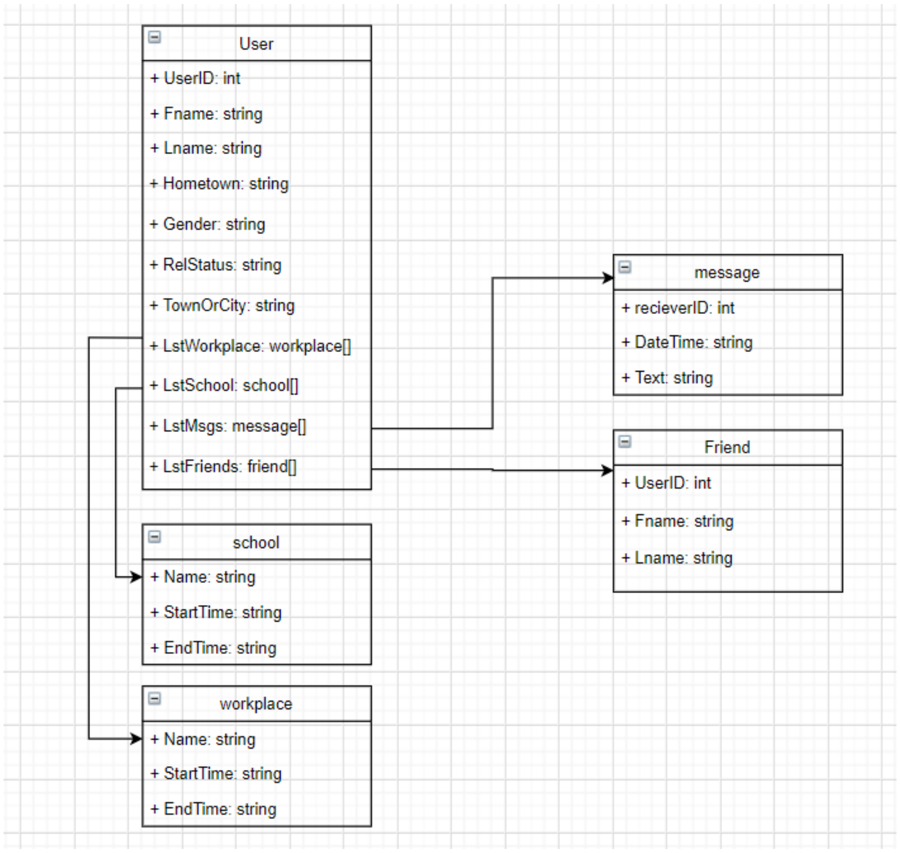
2.2.1.2 Final Relationship Diagram

To correct the issue of multiple data in one field I have separated the data into tables as shown below.



This resolves the issue of having multiple data in one field and abstracts that to separate tables with the userID being used as the Primary Key as Foreign Key. This will help to find the data in the prototype implementation a lot easier than having to loop through and segment data in one field. You will notice that the one-to-any relationships between the entities have been maintained such that the new tables don't impact the other entities of the databases.

2.2.2 Class Diagrams



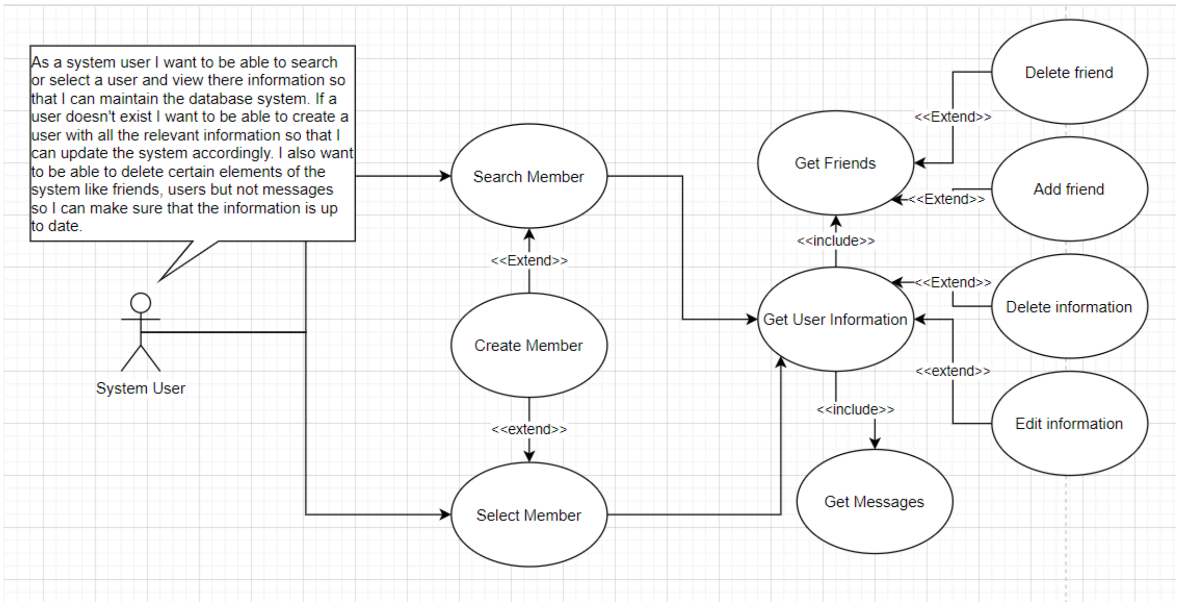
Above is an image displaying the class diagrams with the arrows dictating that the main class “User” is using the other classes as datatypes for certain attributes of “User” class. For example, the lstWorkplace attribute of User class is an array of object “workplace”. The arrow dictation along with the prefixes on the attribute names will allow me to build the class pseudocode more accurately than I would be able to if neither was included.

2.2.3 Use Cases

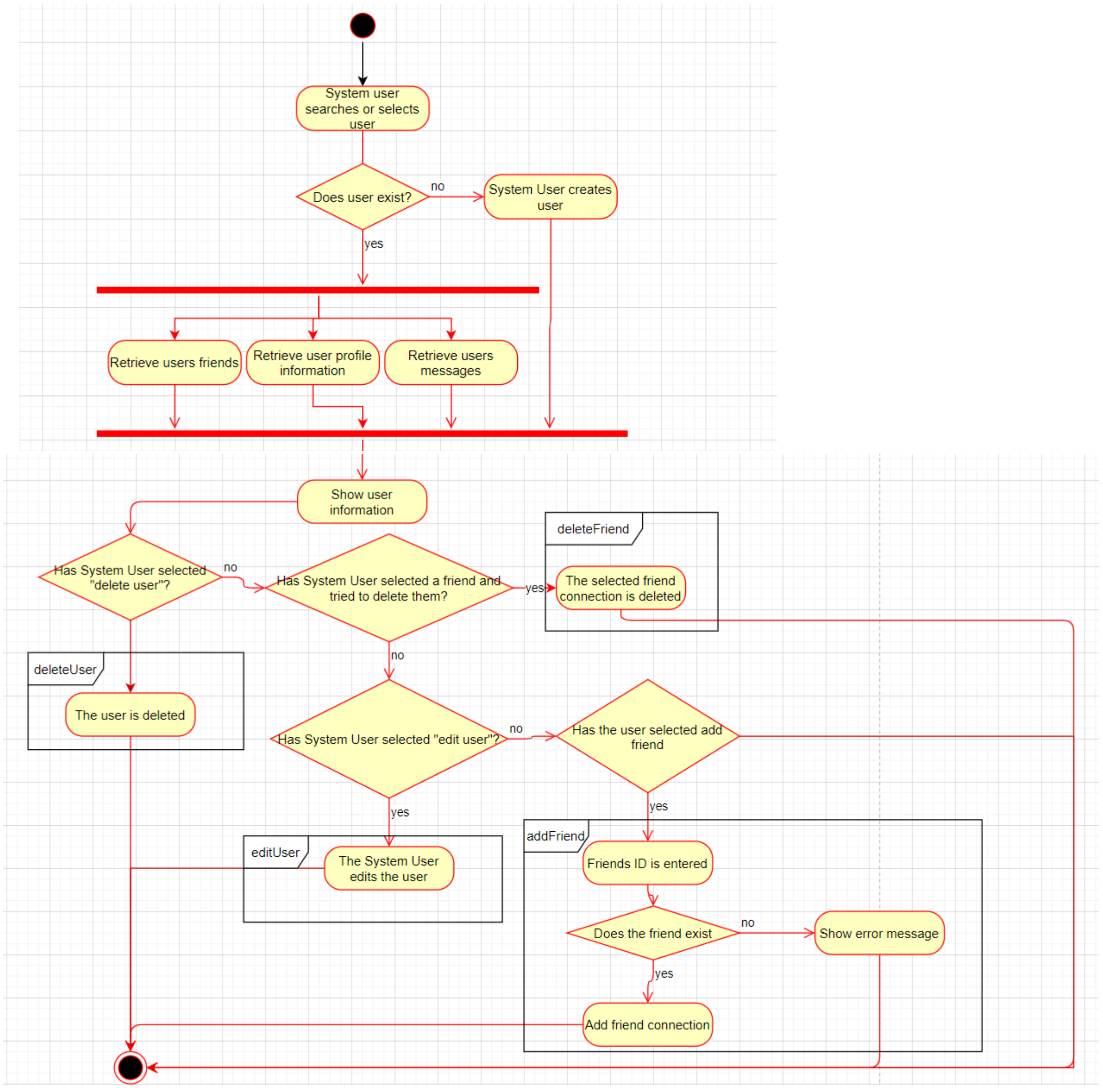
2.2.3.1 Use Case Description

As a system user I want to be able to search or select a user and view there information so that I can maintain the database system. If a user doesn't exist I want to be able to create a user with all the relevant information so that I can update the system accordingly. I also want to be able to delete certain elements of the system like friends, users but not messages so I can make sure that the information is up to date.

2.2.3.2 Use Case Diagram


As a system user I want to be able to search or select a user and view there information so that I can maintain the database system. If a user doesn't exist I want to be able to create a user with all the relevant information so that I can update the system accordingly. I also want to be able to delete certain elements of the system like friends, users but not messages so I can make sure that the information is up to date.
System User
Search Member
<<Extend>>
Create Member
<<extend>>
Select Member
Get Friends
<<include>>
Get User Information
<<include>>
Get Messages
Delete friend
<<Extend>>
Add friend
<<Extend>>
Delete information
<<Extend>>
Edit information
<<extend>>

2.2.4 Activity Diagrams



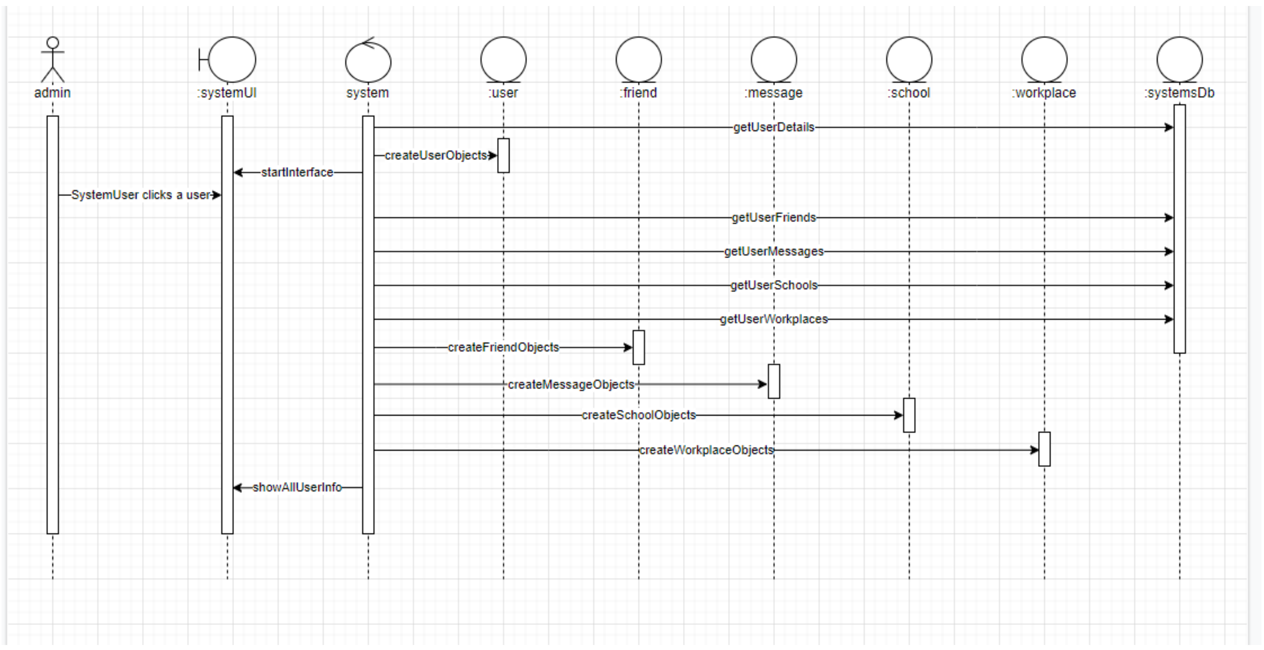
System user searches or selects user
Does user exist?
no
System User creates user
yes
Retrieve users friends
Retrieve user profile information
Retrieve users messages
Show user information
Has System User selected "delete user"?
no
Has System User selected a friend and tried to delete them?
yes
deleteFriend
The selected friend connection is deleted
deleteUser
The user is deleted
no
Has System User selected "edit user"?
no
Has the user selected add friend
yes
yes
editUser
The System User edits the user
addFriend
Friends ID is entered
Does the friend exist
no
Show error message
yes
Add friend connection

Above is the activity diagram for the system. It highlights the workflow of a System User interacting with the system. I have used frames to highlight certain functions that need to be included in the final implementation. For example, the addFriend function includes a check for whether the friend exists which is effectively what is done earlier in the program when the system checks if the user that the System User has searched for exists in the database. Using the above diagram I can identify workflow, important elements to include in the prototype and also functions that can be shared by various elements of the system.

2.2.5 Sequence Diagrams

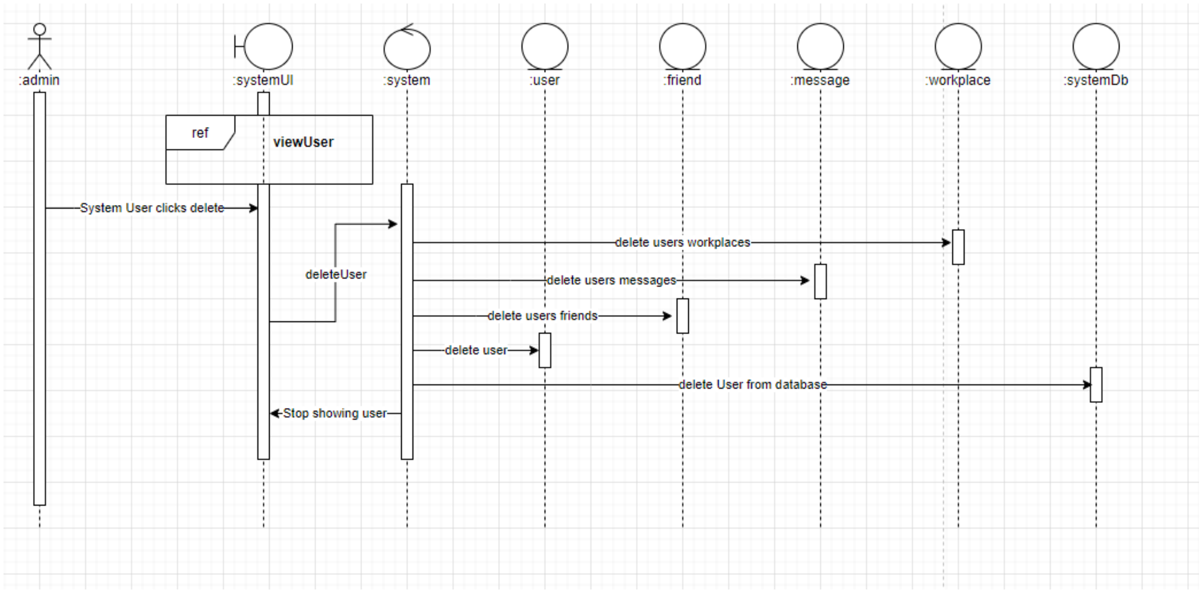
2.2.5.1 View User Sequence Diagram

Below shows a sequence diagram that explains the interaction between the system user and components of the system. Notice that most of the process happen in the backend of the system and is not shown to the system user. This is to prevent a confusing process while using the system.



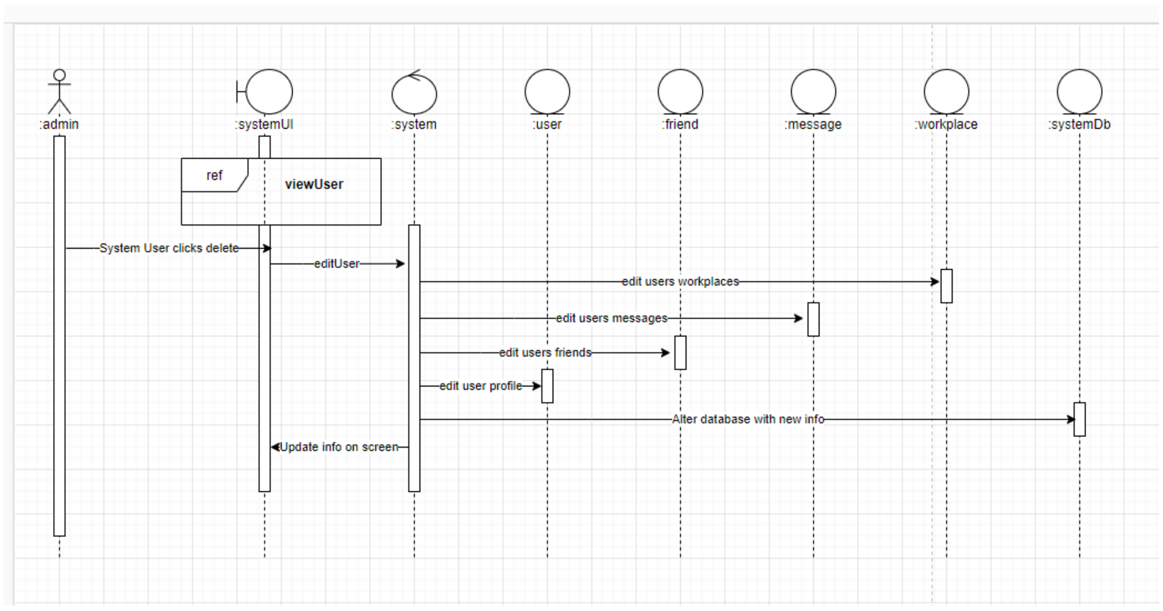
2.2.5.2 Delete User Sequence Diagram

The beginning of this process is the same as viewing a user so I reference that sequence diagram in the beginning of this sequence diagram and much like view user much of the process occurs in the backend and is not viewable by the user.



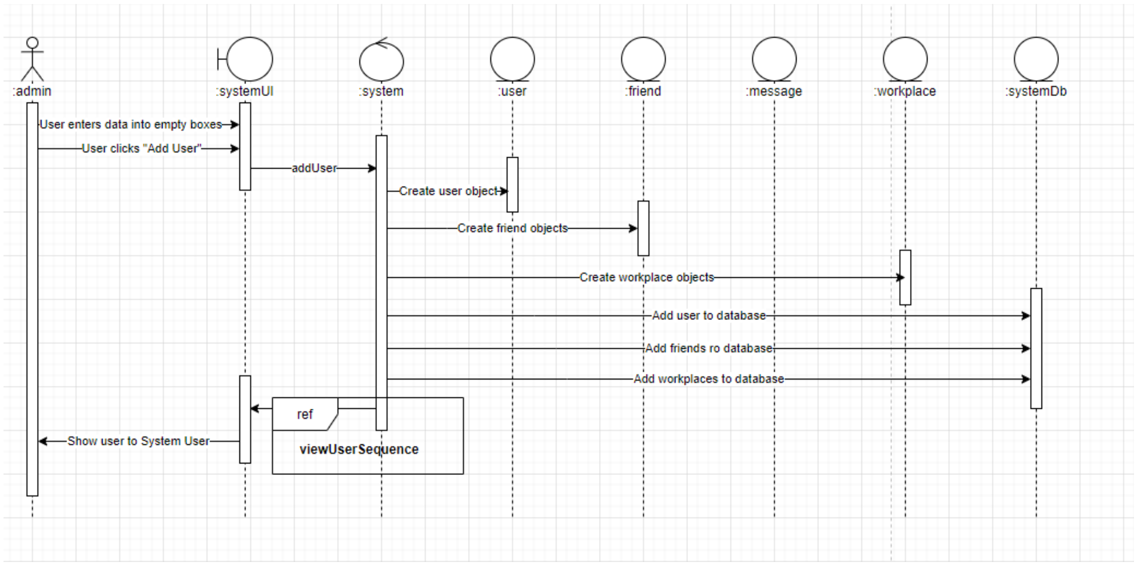
2.2.5.3 Edit User Sequence Diagram

This is the same process as deleting a user only with a few changes.

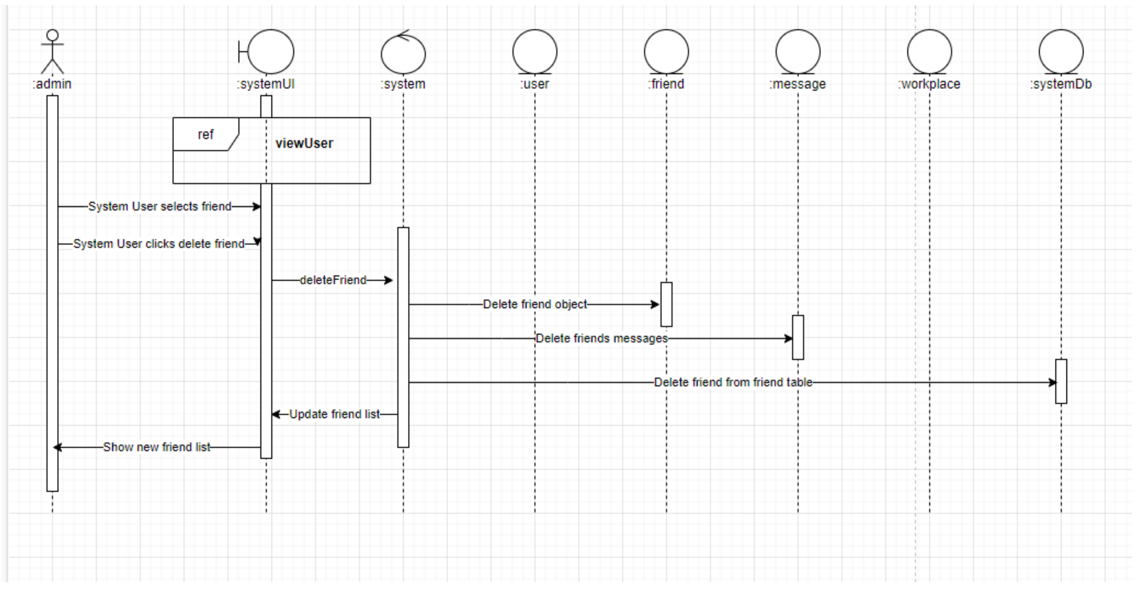


2.2.5.4 Add User Sequence Diagram

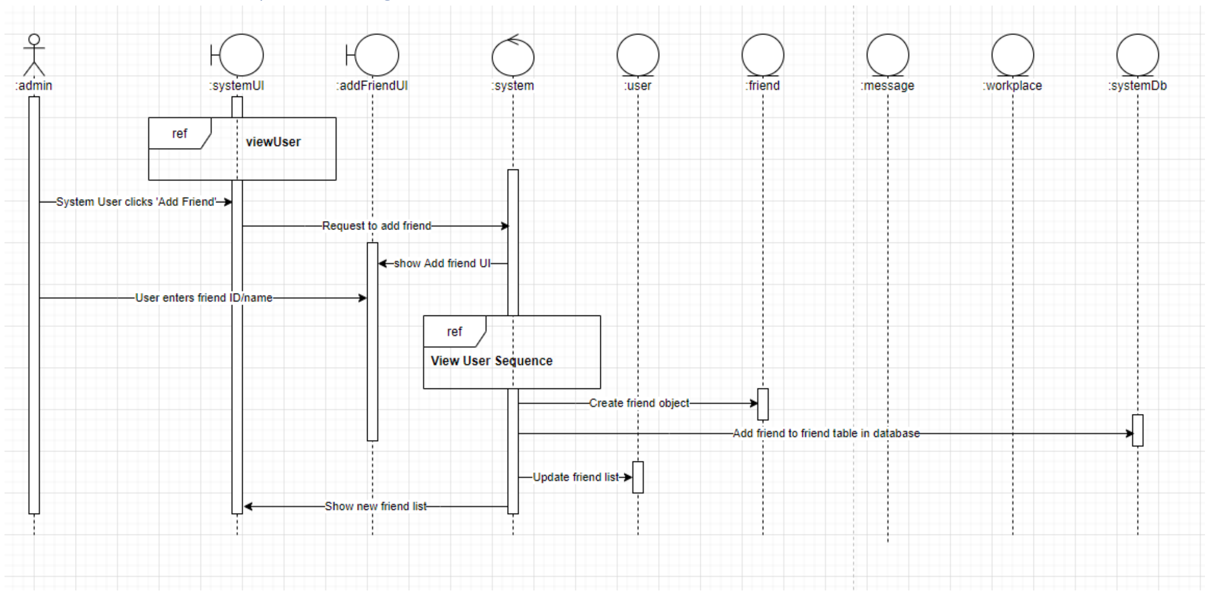
Same as before this sequence main operations happen in the backend and it also references the view user sequence to show the user to the System User.



2.2.5.5 Delete Friend Sequence Diagram



2.2.5.6 Add Friend Sequence Diagram



2.3 SQL Statements

2.3.1 Create the Tables

2.3.1.1 Create User Table

```
CREATE TABLE users(
        user_id INT NOT NULL AUTO_INCREMENT,
        f_name VARCHAR(40) NOT NULL,
        l_name VARCHAR(40) NOT NULL,
        hometown VARCHAR(40),
        gender VARCHAR(40) NOT NULL,
        rel_status VARCHAR(40) NOT NULL,
        town_or_city VARCHAR(40)
        PRIMARY KEY ( user_id)
);
```

2.3.1.2 Create Friend Table

```
CREATE TABLE friends(
        user_id INT NOT NULL,
        friend_id INT NOT NULL,
        PRIMARY KEY (user_id, friend_id)
        FOREIGN KEY  (user_id) REFERENCES users(user_id)
        FOREIGN KEY (friend_id) REFERENCES users(user_id)
);
```

2.3.1.3 Create Message Table

```
CREATE TABLE messages(
        sender_id INT NOT NULL,
        receiver_id INT NOT NULL,
        date_time VARCHAR(40),
        message VARCHAR(100) NOT NULL,
        PRIMARY KEY (sender_id, receiver_id),
        FOREIGN KEY (sender_id) REFERENCES users(user_id),
        FOREIGN KEY (receiver_id) REFERENCES users(user_id)
);
```

2.3.1.4 Create Workplace Table

```
CREATE TABLE workplaces(
        user_id INT NOT NULL,
        placename VARCHAR(40) NOT NULL,
        start_time VARCHAR(40) NOT NULL,
        end_time VARCHAR(40) NOT NULL,
        PRIMARY KEY (user_id, placename),
        FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

2.3.1.5 Create Schools Table

```
CREATE TABLE schools(
        user_id INT NOT NULL,
        schoolname VARCHAR(40) NOT NULL,
        start_time VARCHAR(40) NOT NULL,
        end_time VARCHAR(40) NOT NULL,
        PRIMARY KEY (user_id, schoolname),
        FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

2.3.2 Retrieve Data from Tables

2.3.2.1 Select user from Users Table

If the user has entered a user id then this will be run:

SELECT * FROM users WHERE user_id = user_input

If the user has entered a users name then this will run:

SELECT * FROM users WHERE f_name = user_input

2.3.2.2 Select Users Friends from Friends Table

SELECT * FROM friends WHERE user_id = user_id_selected

2.3.2.3 Select Users Messages from Messages Table

SELECT * FROM messages WHERE sender_id = user_id _selected
SELECT * FROM messages WHERE receiver_id = user_id_selected

2.3.2.4 Select Users Workplaces from Workplaces Table

SELECT * FROM workplaces WHERE user_id = user_id_selected

2.3.2.5 Select Users Schools from Schools Table

SELECT * FROM schools WHERE user_id = user_id_selected

2.3.3 Add Data to Tables

2.3.3.1 Add User to Users Table

INSERT INTO users
VALUES (FNameVar, LNameVar, HometownVar, GenderVar, RelStatusVar, TownOrCityVar)

2.3.3.2 Add Friend to Friends Table

INSERT INTO friends
VALUES (user_id_selected, friend_id_entered)

2.3.4 Delete Data from Tables

2.3.4.1 Delete Friend from Friends Table

DELETE FROM friends WHERE user_id = user_id_selected AND friend_id = friend_id_selected

2.3.4.2 Delete User from Users Table

DELETE FROM users WHERE user_id = user_id_selected

2.3.5 Edit Data in Tables

2.3.5.1 Edit User Information

UPDATE users SET f_name = FNameVar, l_name = LNameVar, hometown = HometownVar, gender = GenderVar, rel_status = RelStatusVar, town_or_city = TOrCVar WHERE user_id = user_id_selected

2.4 List of High Level Functions

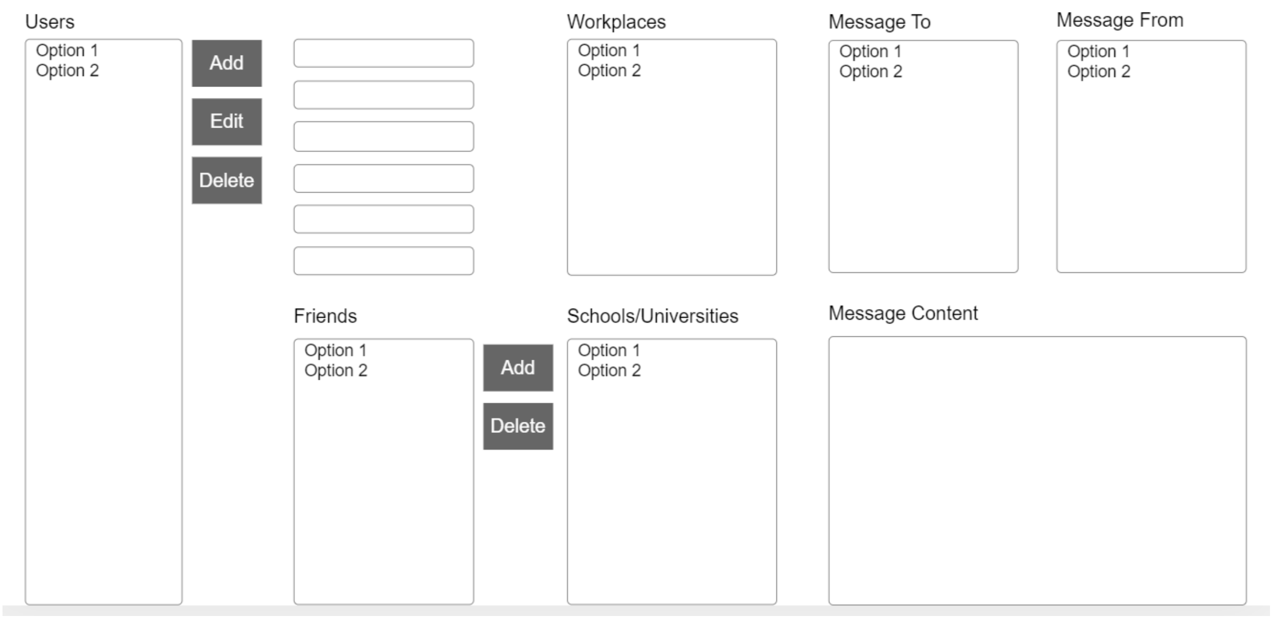
This is a list of all the functions that the final implementation of the system needs to have.

- Search User
- View User Information
- View Users Friends
- View Users Messages
- Add User
- Edit User
- Delete User
- Add Friend
- Delete Friend

2.5 Interface Designs

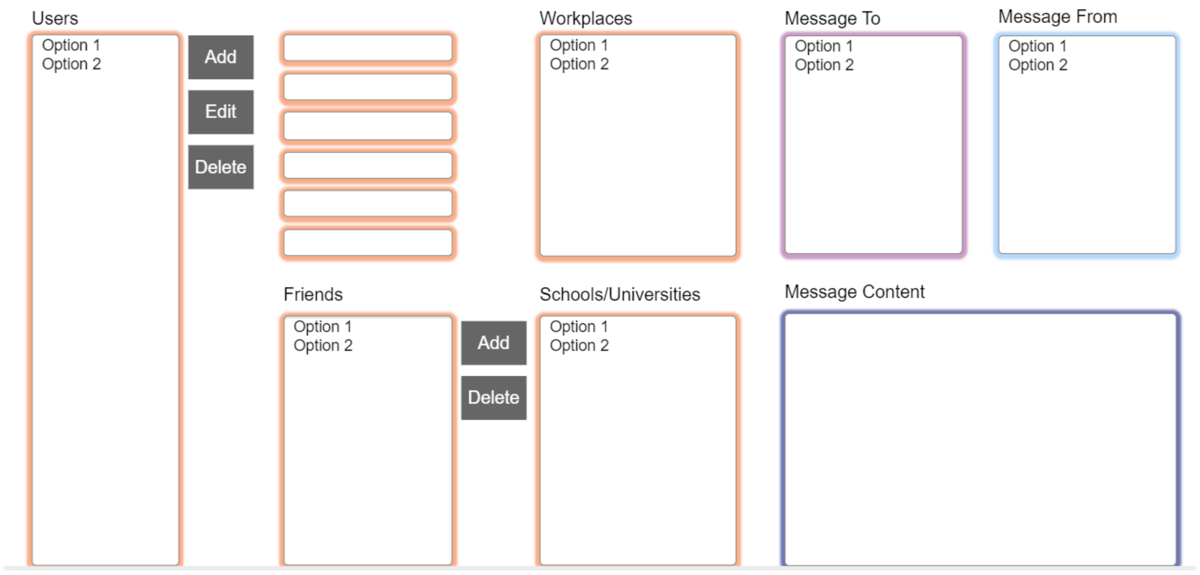
I used an online interface design tool to generate the following design ideas including the finalised interface design. I kept in mind good UX criteria as well as the user requirements and what I know is possible for me to achieve in a reasonable amount of time.

2.5.1 Design 1



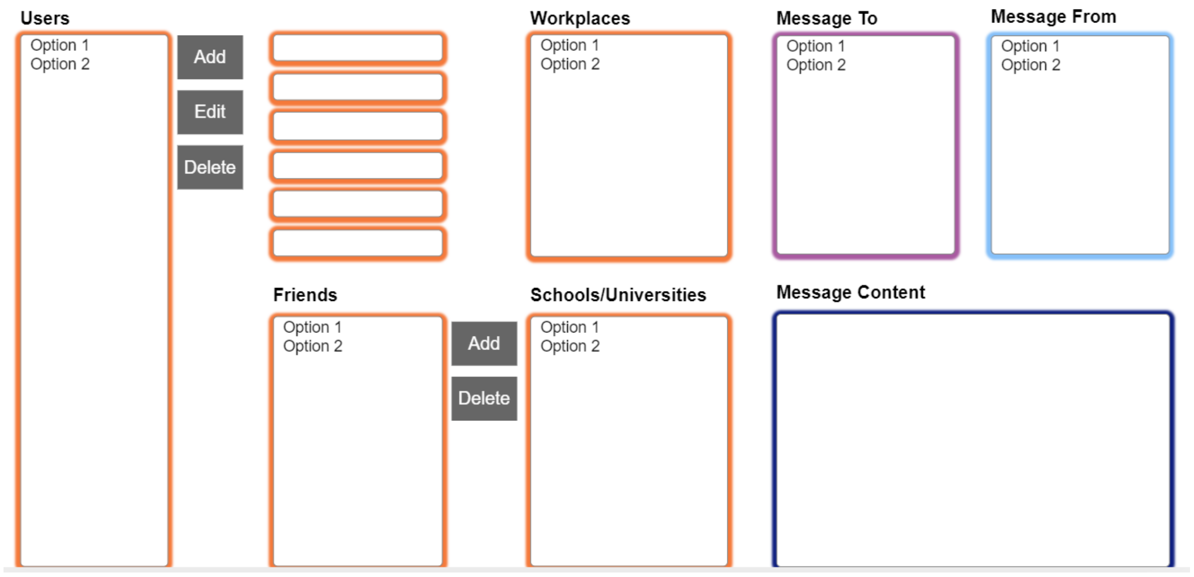
All elements are visible and very little interaction in that there are not multiple pages to cycle through to have the functionality required. One issue is that the System User will need to read the subtitles to know what each box is. If I can group, the box by a faint colour then this will be easier to notice for non-colour-blind users. I will make it easier to see for colour-blind users by using high contrasting colours so it doesn't affect usability.

2.5.2 Design 2



In this design I am focusing on keeping the layout of design 1 but adding coloured shadows while keeping in mind the affects of deuteranopia, protanopia and tritanopia (the three types of colour-blindness). I used a colour palette that is visibly distinguishable regardless of colour-blind variant. This colour-blind sensitive palette in hex is as follows; #F5793A, #A95AA1, #85C0F9, #0F2080. The four regions are User Information, Message To, Message From and Message Content.

2.5.3 Design 3

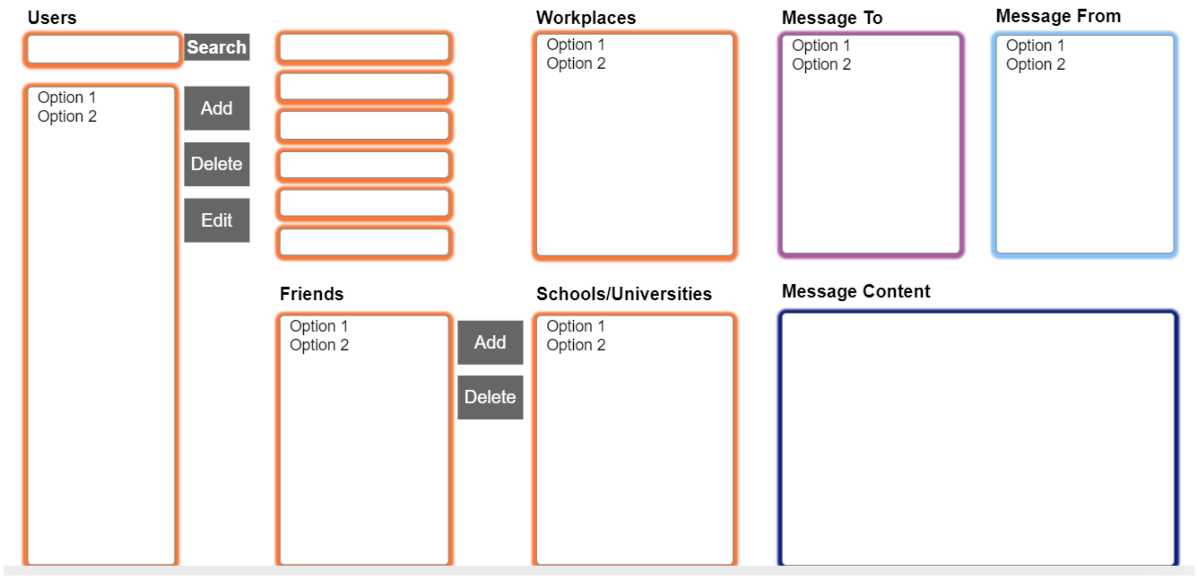


In this design I am going to focus on increasing accessibility for users such as making certain text bold and increasing contrast between elements of the interface.

2.5.4 Success Criteria for final design

Same design as design 3 but this time add ability to search for a user instead of scrolling through all of the users.

2.5.5 Final Design



2.4 Link to Git Account

https://github.com/DavidWood2001/ISAD157

3 Implementation

3.1 Included Functions

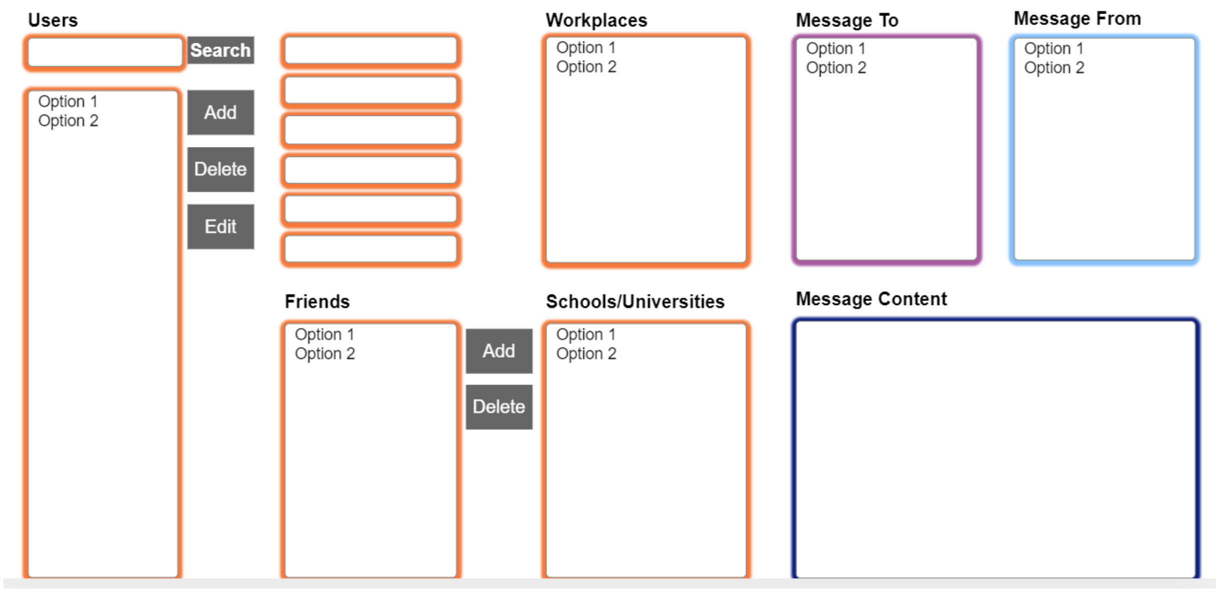
This is a list of the functions that the prototype is going to implement, these are taken from the high level functions list in this document.

- Search User
- View User Information
- View Users Friends
- View Users Messages
- Add User
- Edit User

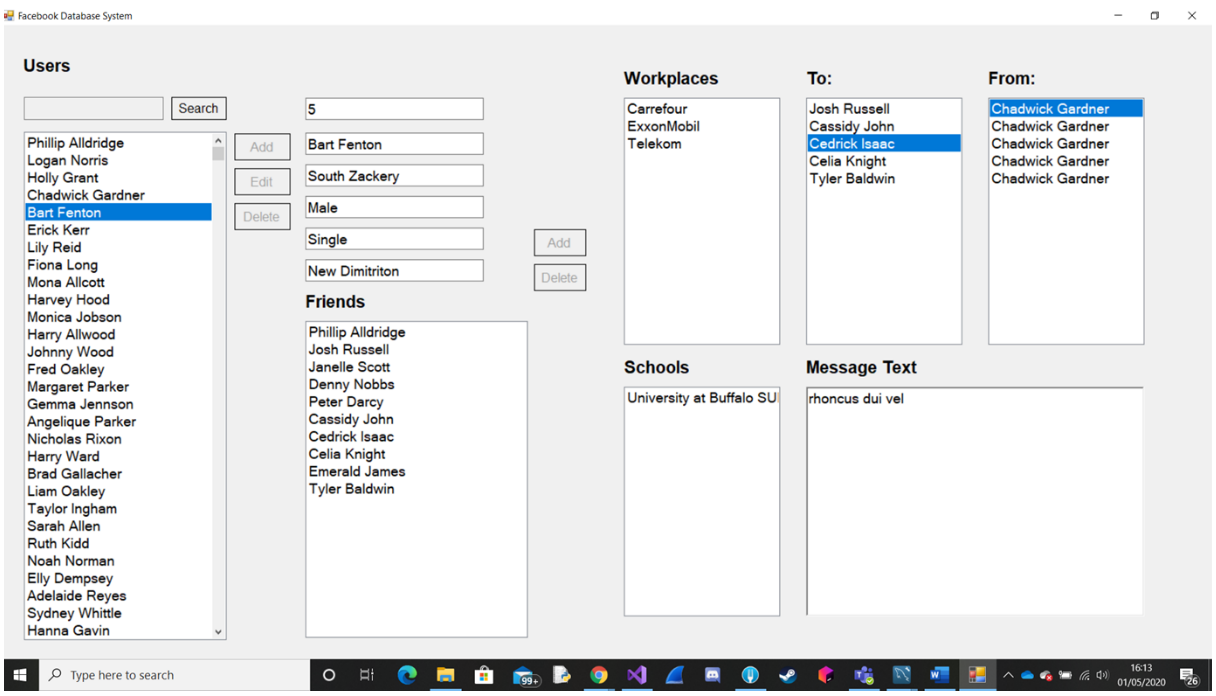
This means I won’t be implementing the ability to delete a user, add a friend or delete a friend. This is because these were extra capabilities that aren’t essential to demonstrate the systems potential.

3.2 Interface Design

The intended design is below.



This is the actual implementation.


Facebook Database System

Users

Search

Phillip Alldridge
Logan Norris
Holly Grant
Chadwick Gardner
Bart Fenton
Erick Kerr
Lily Reid
Fiona Long
Mona Allcott
Harvey Hood
Monica Jobson
Harry Allwood
Johnny Wood
Fred Oakley
Margaret Parker
Gemma Jennson
Angelique Parker
Nicholas Rixon
Harry Ward
Brad Gallacher
Liam Oakley
Taylor Ingham
Sarah Allen
Ruth Kidd
Noah Norman
Elly Dempsey
Adelaide Reyes
Sydney Whittle
Hanna Gavin

Add
Edit
Delete

5
Bart Fenton
South Zackery
Male
Single
New Dimitriton

Add
Delete

Friends

Phillip Alldridge
Josh Russell
Janelle Scott
Denny Nobbs
Peter Darcy
Cassidy John
Cedrick Isaac
Celia Knight
Emerald James
Tyler Baldwin

Workplaces

Carrefour
ExxonMobil
Telekom

To:

Josh Russell
Cassidy John
Cedrick Isaac
Celia Knight
Tyler Baldwin

From:

Chadwick Gardner
Chadwick Gardner
Chadwick Gardner
Chadwick Gardner
Chadwick Gardner

Schools

University at Buffalo SU

Message Text

rhoncus dui vel

Type here to search

16:13
01/05/2020

With more time I would create each element as it's own control which would mean I could enforce a paint event and a border colour onto each element which would have made it more closely resemble the intended design. Overall I stayed true to the layout of the design.

3.3 Program Structure

The general structure will be a frontend that uses the different methods that each class has as well as a general custom library that handles the functions that is used by a lot of different methods such as searching the database and retrieving database records.

4 Conclusion

This prototype is by no means a final implementation of the system. It serves as initially intended which is a representation of the final system. This prototype system is meant to provide the ability to search and view user information including friends, workplaces, schools and messages. It does this consistently as long as there is the right data in the database. The database is also structured exactly as shown in the final entity relationship diagram (See section 2.2.1.2). The prototype isn't an exact copy of the design material but it was never meant to be it is a basic implementation with the same layout as the final interface design in order to convey what the system can currently do and therefore demonstrate what it will be able to do when it is fully implemented. I have left the buttons to add, edit and delete users and friends as evidence to the potential progression of the system into the next stage of implementation.

5 Appendix

SQL Syntax Checker - https://www.eversql.com/sql-syntax-check-validator/