Proximity

Project Report 3

Group 16: Ryan Wortmann (Leader), Song Vu Nguyen, Ryan Rottmann, Nathan Kulczak, John Oatey

Group Website: https://github.com/Rdubya54/Software_Engineering_Project

Individual Contributions Breakdown

******Everyone contributed equally

Table of Contents

Summary of Changes

- In part **3) System requirements**, we remove many of the user stories that are irrelevant to our project and update the traceability matrix in part **4) Functional Requirement Specification** so our use cases now cover every user stories
- We added part **5) Effort Estimation using Use Case Points** as part of the requirement for report 3
- We update the Unit Testing, Integration Testing in part **12) Design of Tests** so it match up with the current tests that we uses for our WebApp
- Update Domain diagram in part **6) Domain Analysis** to include the design patterns that used in our program
- Update part **7) Interaction Diagrams** by adding Design Patterns section to explain what design patterns that we used to improve our program
- Update the Plan of Work into History of Work in part **13) History of Work, Current Status and Future Work**
- Update Use case casual description in part **4) Functional Requirement Specification** to be more relevant to our current implementation
- In Report 1 part 2 we updated of Use Case Diagram with a more correct version
- Update Class Diagram and Data Type and Operating Signatures in part **8) Class Diagram and Interface Specification** to match with our current code.
- Added Object Constraint Language section to part **8) Class Diagram and Interface Specification** as part of requirement for report 3.
- Update screenshots of our application in On-Screen Appearance Requirements section of part **3) System Requirements (User Stories)** and in Preliminary Design section of part **11) User Interface Design and Implementation** to match with what our current design look like

1) Customer Statement of Requirements

(a)Problem Statement

There are many different social media platforms, most of which implement some form of networking, messaging, and posting. These platforms let people talk to each other, share life events, share personal information, find communities for the like-minded or for those they share interests with, and many other features. However, while these platforms let us talk with friends and family online, they do not encourage face to face interactions. These types of platforms do not offer a realistic view of the lives of users to those that they have connected with. Whether this is through user's curating their content to control their public image, like Instagram, or the platform using uniquely tailored algorithms to decide what content they deem most relevant, like Facebook or Twitter.

Our project is to create a new type of social media platform that will address these issues and provide its users with a better representation of the local community. In our social media platform the focus will be moved away from forming networks with other users and moved towards geographically based communities. Posts will be flagged internally with approximate locations so that users can be shown posts from other users near them. This will have many benefits over traditional social media. It will show its users a representative view of their local communities than they could otherwise never see. This will in turn help foster a more tight knit online community as its users will be able to connect with others with many shared life experiences. Connecting locals online can also help people grow closer at already existing communities, such as college campuses.

There are many issues that need to be addressed when developers are making a social media platform. All users expect a number of standard features to be implemented well and easy to use. These features include the ability to post media or text, commenting on other user's posts, direct messaging, and much more. Developers of the social media platform must ensure that these features behave as a user would expect them too from their experience with other platforms. Most of these features, and most others that will be implemented, will be enhanced by only showing users local content. A lot of content users on popular social media platforms are shown is largely irrelevant to their lives unless they have gone through the effort of tailoring the users they have networked in or the groups they have joined to be relevant to them. Posts made near a user have a much higher chance of being relevant information. For example, someone could post about a street being shut down or a good new restaurant in town. People visiting from out of town could get a good feel for the local atmosphere or if there's an expected influx of out-of-towners, such as during a football game, locals put post helpful advice.

The way to measure the overall success of a social media platform is, at least for a high level approach, quite simple. A social media platform is successful if it has an active user base. An active user base means that not only does the platform have a steady or increasing number of users, but the users that are signed up are posting content and interacting with each other.

There are other metrics that our platform will be judged by. The user client must be responsive with a well designed user interface. The backend server must implement an algorithm that decides which posts to serve to which users so that users find the posts interesting, helpful, or relevant in some way. The developers must keep all private data private by using industry standards such as using TLS to secure all communication between the client and the server. Stored confidential information must also be kept secure. For example, industry standard hashing algorithms for passwords or using an industry standard identity provider service such as Cognito through AWS or Active Directory B2C through Azure Cloud to handle user accounts.

2) Glossary of Terms

Technical Terms:

Progressive Web App - Web applications that load like regular web pages or websites but can offer the user functionality such as working offline, push notifications, and device hardware access traditionally available only to native mobile applications.

Database - A large table where user information will be stored.

Geolocation - The process or technique of identifying the geographical location of a person or device by means of digital information processed via the Internet.

Algorithm- A process or set of rules to be followed

Transport Layer Security (**TLS**) – Cryptographic protocol that provides communications security over a computer network.

Amazon **Cognito** - An Amazon Web Services (**AWS**) product that controls user authentication and access for mobile applications on internet-connected devices.

Non Technical Terms:

Posts - These are things that the users share with everyone on the app. These could be text, pictures, or location.

Social Media - Websites and applications that enable users to create and share content or to participate in social networking.

Users - The people who have user accounts with the application

Interaction - An exchange between two or more individuals.

Friend - A user who is connected with a given user in some way in the app. This usually means that they see each other’s posts.

News feed - A data format used for providing users with frequently updated content

3) System Requirements (User Stories)

(a)Functional Requirements

Identifier	User Story	Size Points
ST-1	As an user, I can create an account by providing a username, password and other necessary information	5
ST-2	As an user, I can login to my account by using my registered username and password	5
ST-3	As a logged user, I can share my thought by posting it online as text or media	5
ST-4	As a login user, I can view and comment on other people posts	4
ST-5	As a login user, I can choose which information I want to share to friend or the public	4
ST-6	As a logged in user, I can add people to be my friends	3
ST-7	As a logged in user, I can send messages to people that I added as friend	3
ST-8	As a user, I can create public or private groups for other users to join	2
ST-9	As a user, I can join public or private groups	3
ST-10	As a member of a group, I can post on the group’s page	3

ST-11	As a member of a group, I can view and comment on posts made by other group members on the group’s page	2
ST-12	As the creator of a group, I can remove posts from the groups page	2
ST-13	As the creator of a group, I can remove members of a group and prevent them from rejoining	2

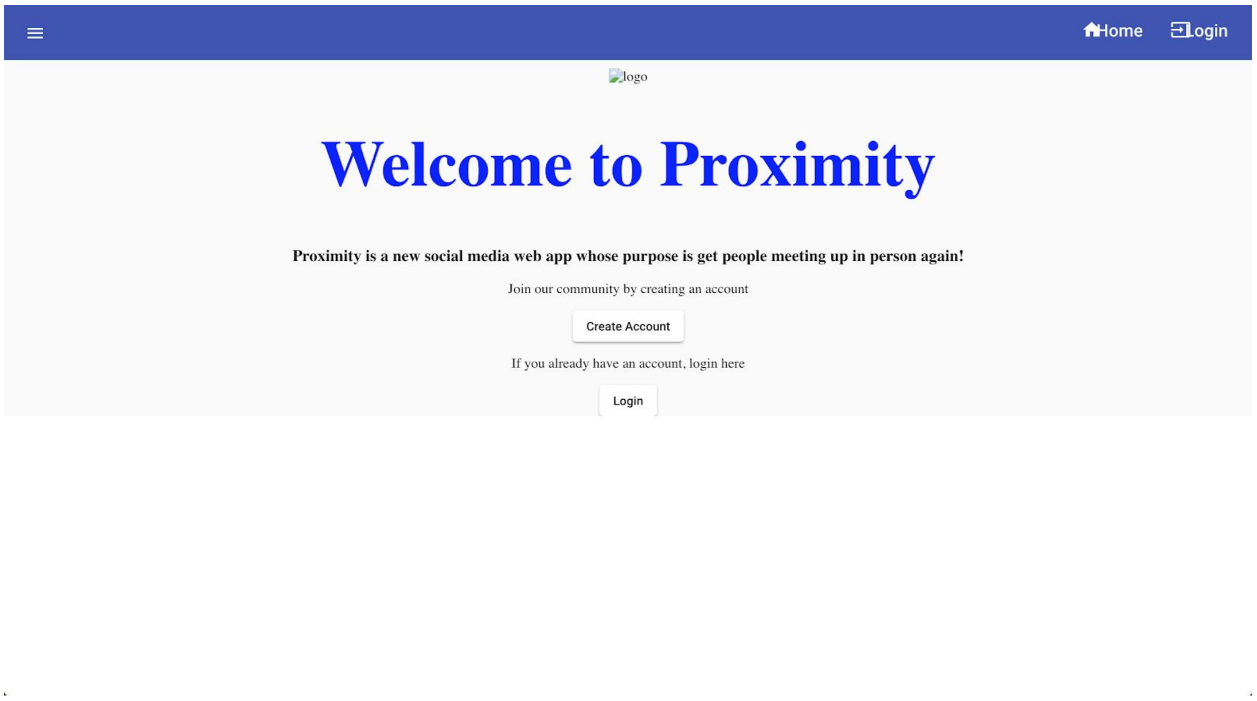
(b)Nonfunctional Requirements

Identifier	User Story	Size Points
ST- 14	As a user, I can easily navigate the web app and find what I want quickly	3
ST- 15	As a user, I can view a post as soon as it was posted by people	2
ST- 16	As a user, I can expect that my personal information are store privately and securely	5
ST- 17	As a user, I can expect the web app's map is giving me pertinent data about events that happen near me	5

(c)On-Screen Appearance Requirements

Identifier	User Story	Size Points
ST-18	As a user, the there should be a profile page that lets me customize some information about myself as well as have a profile picture of me	3
ST-19	As a user, the app should have a map where my general geolocation is presented as well as other information	2
ST-20	As a user, the app should have a page dedicated to messaging other users	3

Home Page



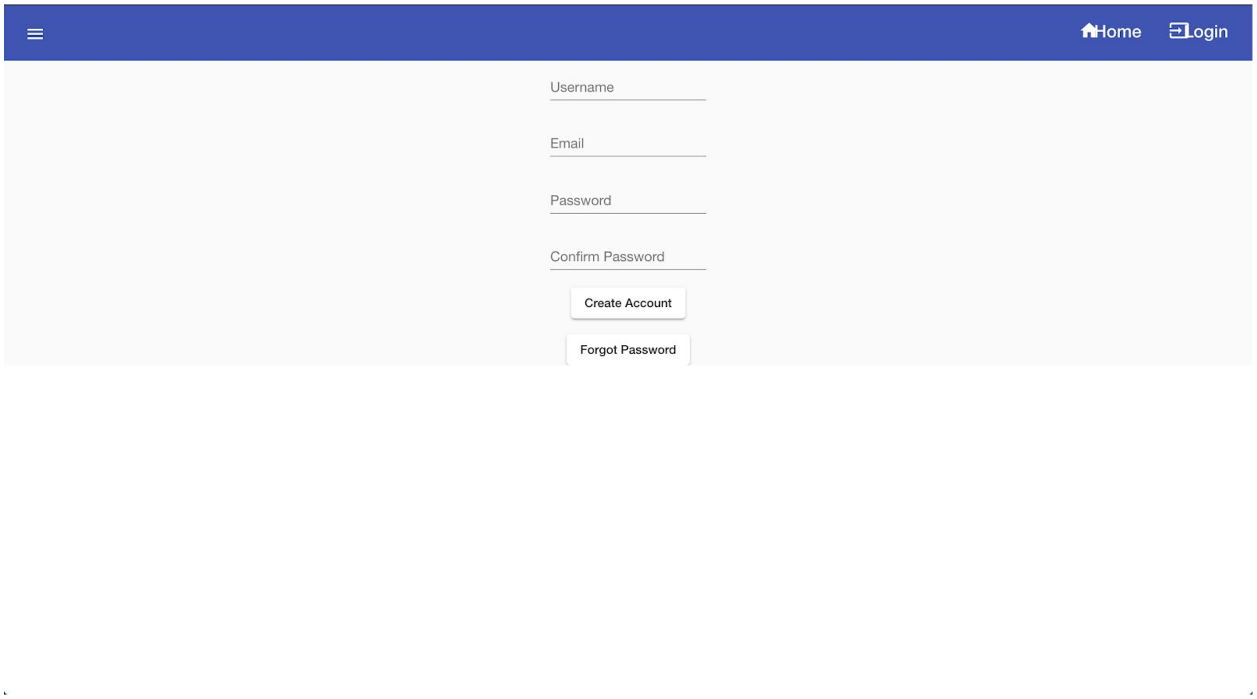
Welcome to Proximity

Proximity is a new social media web app whose purpose is get people meeting up in person again!

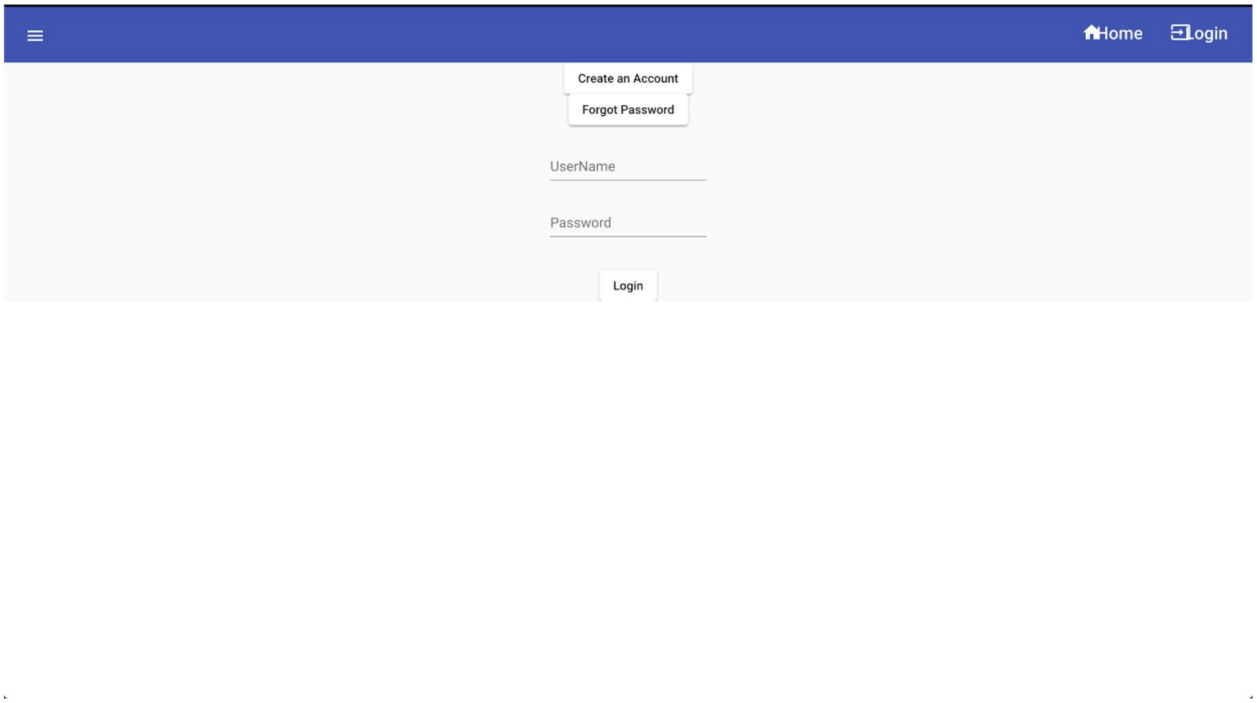
Join our community by creating an account

If you already have an account, login here

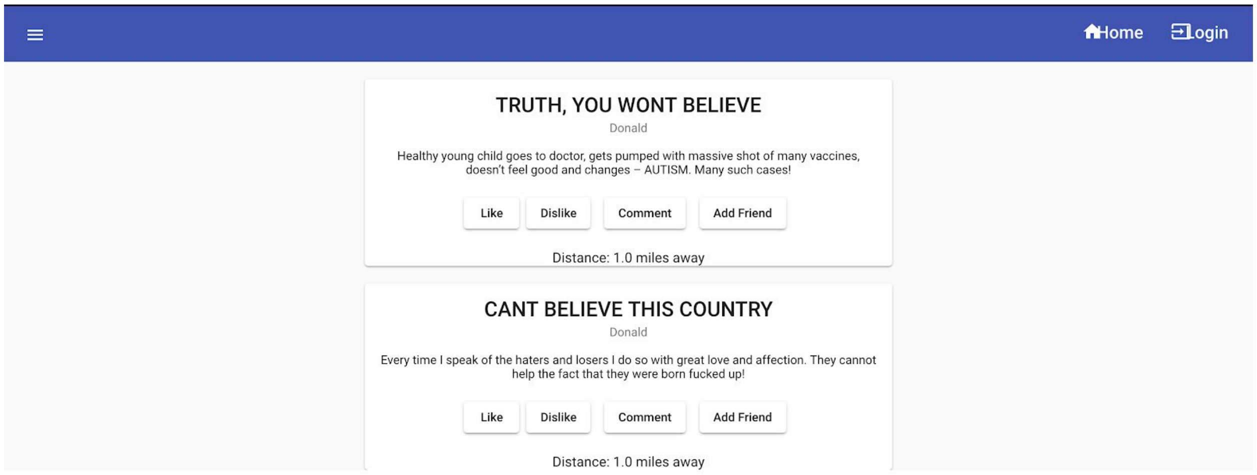
Create Account



Login Page



Feed Page


Home Login

TRUTH, YOU WONT BELIEVE

Donald

Healthy young child goes to doctor, gets pumped with massive shot of many vaccines, doesn't feel good and changes – AUTISM. Many such cases!

Like Dislike Comment Add Friend

Distance: 1.0 miles away

CANT BELIEVE THIS COUNTRY

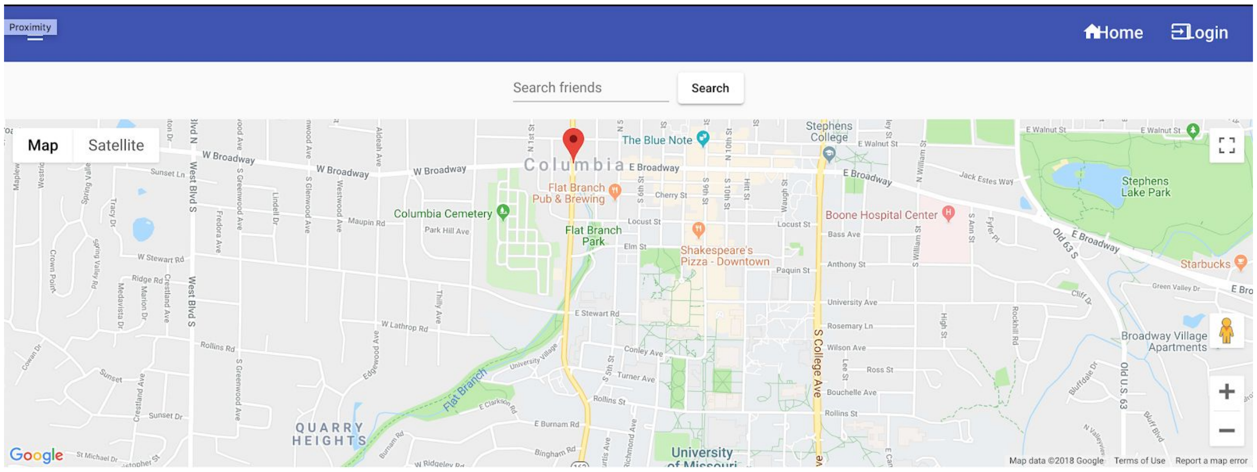
Donald

Every time I speak of the haters and losers I do so with great love and affection. They cannot help the fact that they were born fucked up!

Like Dislike Comment Add Friend

Distance: 1.0 miles away

Map Page



4) Functional Requirement Specification

(a)Stakeholders

Our stakeholders include anyone who is interested in using the social media platform. Social media platforms with an active user base have quite a few different parties who would be likely to use it.

These include:

- Individuals who want to make a user account and use the platform to connect with other people in their area, friends, and local groups of users.
- A business who wants to use the platform to advertise to local users. The nature of the platform means that businesses can make themselves visible to local users easily just by tagging their location for events they may be hosting.
- An organization who wants to reach out to potential new members. This could be any sort of organization such as a volunteer group, clubs, teams, or anything else.

(b)Actors and Goals

Initiating Actors

- **User**
    - Role - A human using the app.
    - Goal - Connect with friends via messaging, posts, event sharing, and location sharing.
- **Group**
    - Role - A private environment created by users to share information only within the bounds of a group.

    - Goal - Let the group connect with each other via messaging, posts, event sharing, and location sharing.

Participating Actors

- **Database**
    - Role - The object that stores information needed for the system.
    - Goal - Store and modify information related to users, groups, and events.

(c)Use Cases

Casual Description

Use Case 1: A user can create and sign in to a user account. Creating an account is requirement in order for user to access our webapp. User informations is store privately in a secure database. User stories cover: ST-1, ST-2, ST-14, ST-16, ST-18.

Use Case 2: A user can add other users as ‘friends’ to their account. User then can share their post privately only to friends and view their friends posts. User stories cover: ST-5, ST-6, ST-7, ST-14

Use Case 3: A user can contact other users. User can use either direct message for friends, or group message to people that in a group. User stories cover: ST-3, ST-7, ST-11, ST-14, ST-20

Use Case 4: A user can create and manage a group of users. User and other people in the group can do group activities. User stories cover: ST-8, ST-9, ST-10, ST-11, ST-12, ST-13, ST-14

Use Case 5: A user can join and interact with other users in the group. User can choose to join a group and communicate with other people in the group via messaging or posts. User stories cover: ST-9, ST-10, ST-11, ST-14

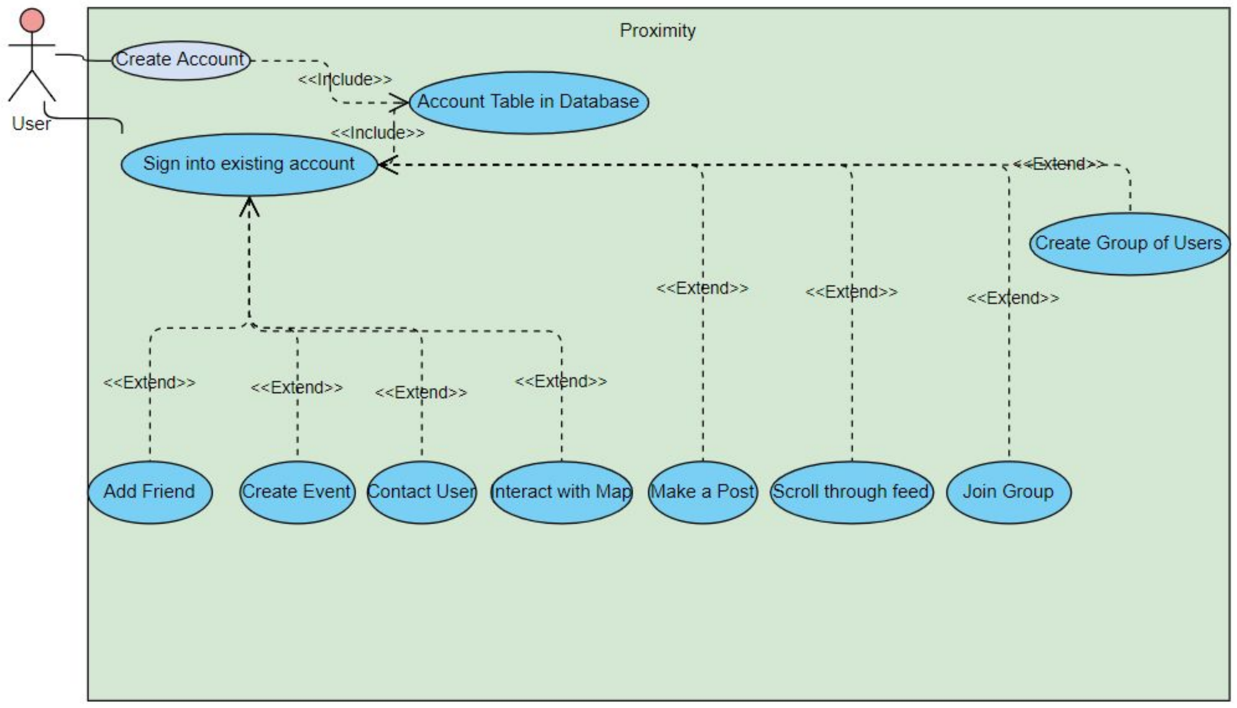
Use Case 6: A user can create and manage a post that will be viewable to others of their selection. User can have text, media in their posts and they can choose to share it with general public, their friends or group. User stories cover: ST-3, ST-5, ST-10, ST-11, ST-14

Use Case 7: Users can view and interact with other user's posts on a feed. User can like, dislike, and comment on the post. User stories cover: ST-4, ST-10, ST-11, ST-12, ST-14

Use Case 8: A user can create and manage an event that is visible on the map to other users. User can choose to host an event and posted its location on the map to share with other users nearby who interested. User stories cover: ST-14, ST-19

Use Case 9: A user can look at the map and interact with events that other users have created. User can view events events that are posted near their location. By clicking on that event on the map, it will give the user more detail on the event like what kind of event is this, who invited and and what time does it start etc. User stories cover: ST-4, ST-14, ST-15, ST-17, ST-19

<u>Use Case Diagram:</u>



Traceability Matrix

User Stories	Size Points	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
ST-1	5	**X**								
ST-2	5	**X**								
ST-3	5			**X**			**X**			
ST-4	4							**X**		**X**
ST-5	4		**X**				**X**			
ST-6	3		**X**							
ST-7	3		**X**	**X**						
ST-8	2				**X**					
ST-9	3				**X**	**X**				
ST-10	3				**X**	**X**	**X**	**X**		
ST-11	2			**X**	**X**	**X**	**X**	**X**		
ST-12	2				**X**			**X**		
ST-13	2				**X**					
ST-14	3	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
ST-15	2									**X**
ST-16	5	**X**								
ST-17	5									**X**
ST-18	3	**X**								
ST-19	2								**X**	**X**
ST-20	3			**X**						
Max Size Points		**5**	**4**	**5**	**3**	**3**	**5**	**4**	**3**	**5**
Total Size Points		**21**	**13**	**15**	**17**	**11**	**17**	**14**	**5**	**16**

Fully-Dressed Description

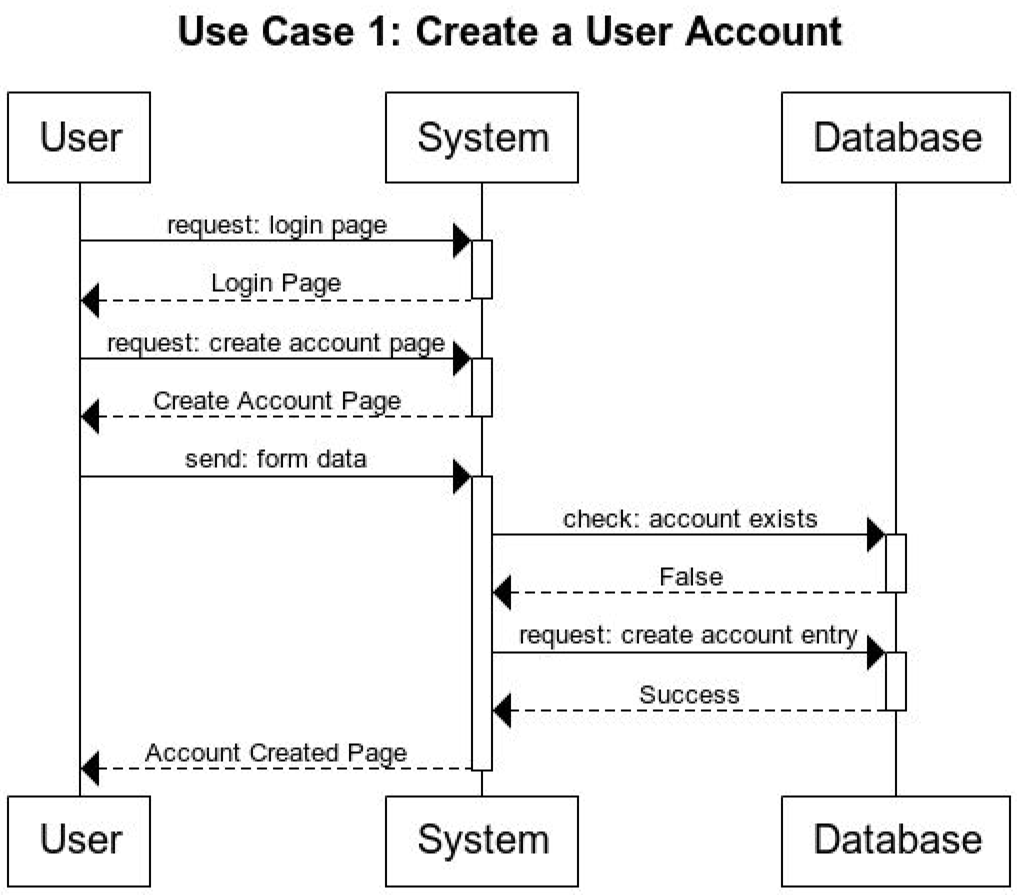
Use Case UC-1:		**User Account**
Related Requirements:		**Trace Matrix**
Initiating Actor:		General User
Actor's Goal		To have a permanent account in application
Participating Actors:		Database, WebApp System
Preconditions:		1. No database entry for user
Postconditions:		Compete database entry for user User can manage their data
Flow of Events for Main Success Scenario:		
→	**1)**	**General User** loads application to see login page
→	**2)**	**General User** clicks create an account
←	**3)**	**System** displays form for **General User** to fill out
→	**4a)**	**General User** fills out form and submits data
←	**5a)**	**System** returns a success for account create page as well as creating a database entry for new **General User**
Alternative flow of Events		
→	**4b)**	**General User** fills out form and submits email tied to another account
←	**5b)**	**System** returns a failure for account create page due to email being found in the database already

Use Case UC-4:	**Group Creation and Management**
Related Requirements:	**Trace Matrix**
Initiating Actor:	General User
Actor's Goal	To be able create and manage a group of other users

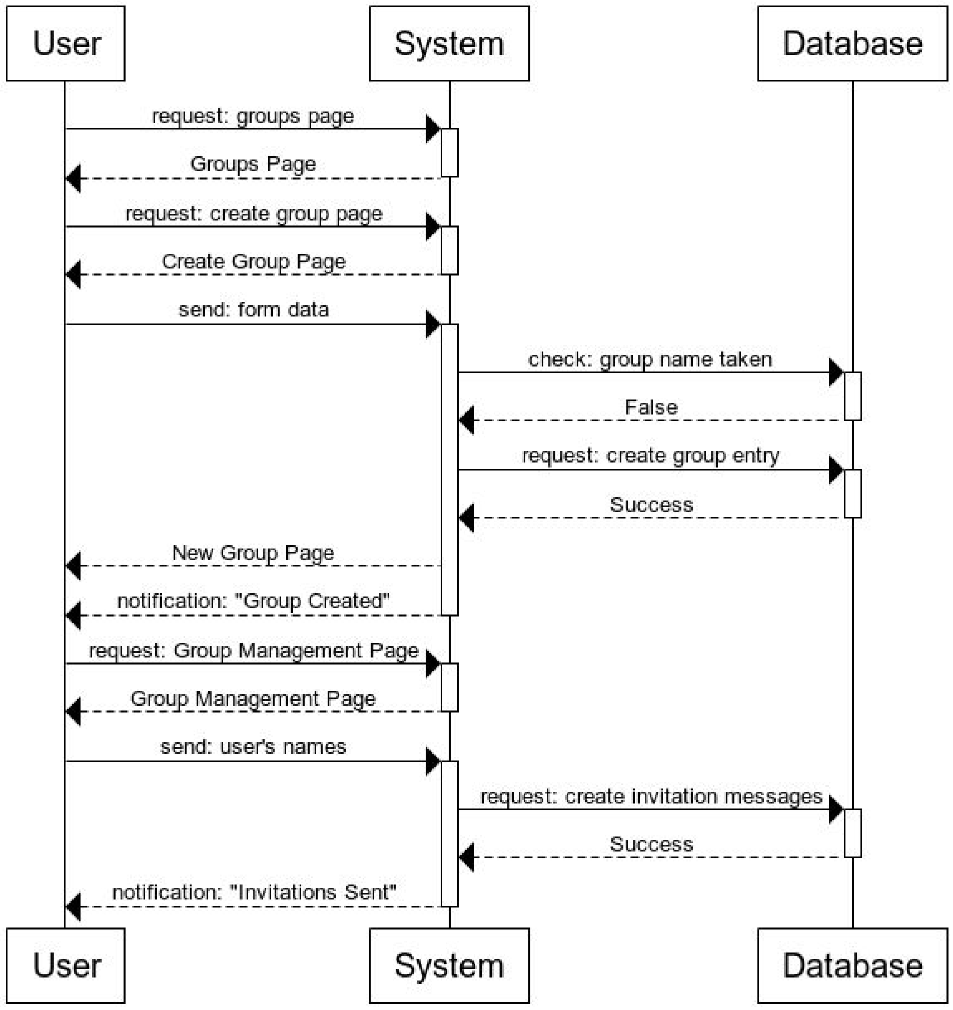
Participating Actors: **Preconditions:** **Postconditions:** **Flow of Events for Main Success Scenario:**		Webapp System, database 1. The user doesn't have any data about belonging to a group in the database 1. The user has data in the database that identify them with a group and link them with other users that belong to the same group 2. System enables user to have various actions to interact with their group
→	1)	**General User** select the group option on the web app menu
→	2)	**General User** clicks to create new group
←	3)	**System** displays Create Group form for **General User** to fill out.
→	4)	**General User** fills out form and submits data
←	5a)	**System** checks the database and verifies that no groups with this name exists. Then add this newly created group to the database
←	6a)	**System** displays the new group page and a notification alerting the **General User** that the group has been created.
→	7)	**General User** clicks the "Manage Group Members" button.
←	8)	**System** displays the Manage Group Members page.
→	9)	**General User** types in the names of other users to add to the group.
←	10)	**System** adds invitations for these users to the message database.
←	11)	**System** displays a notification alerting the **General User** that the invitations have been sent.
Alternative flow of Events		
←	5b)	**System** checks the database and sees that a group with this name already exists.
←	6b)	**System** displays an error alerting the **General User** that this group name has been taken.

Use Case UC-9:		**User Map Event**
Related Requirements:		**Trace Matrix**
Initiating Actor:		User
Actor's Goal		To interact with an event
Participating Actors:		Event Owner
Preconditions:		1. Event Owner has created an event
Postconditions:		1. User has interacted with event 2. User can view data about the event
Flow of Events for Main Success Scenario:		
→	**1)**	**User** loads up map page
←	**2)**	**System** displays map and events based on **Users** location
→	**2)**	**User** selects an event
←	**3)**	**System** displays data about event (time, location, who is going)
→	**4a)**	**User** decides they want to attend by selected "going"
←	**5a)**	**System** tells **Event Owner** that **User** is attending
Alternative flow of Events		
→	**4b)**	**User** picks not attending
←	**5b)**	**System** will not let **Event Owner** know as well as hiding event from **User**

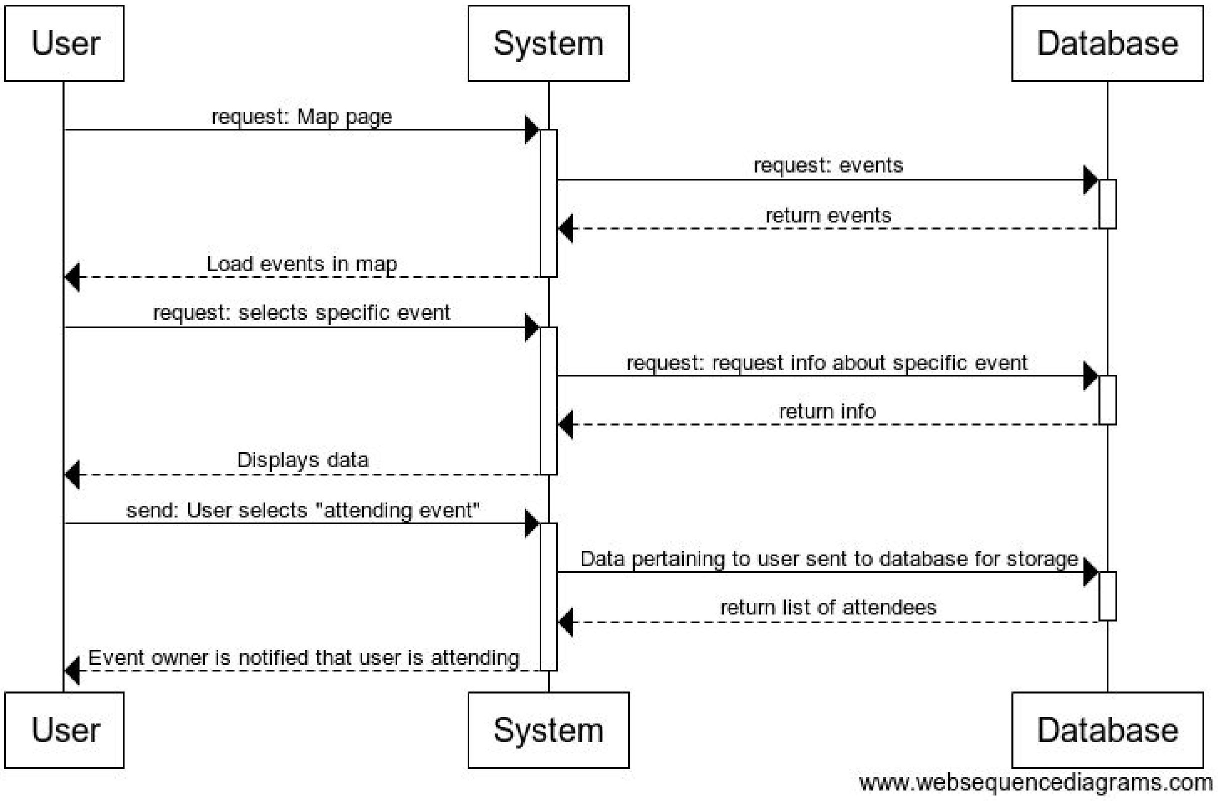
(d)System Sequence Diagrams



Use Case 4: Create and Manage a Group



Use Case 9: User can view map and interact with events that other users have created.


User
System
Database
request: Map page
request: events
return events
Load events in map
request: selects specific event
request: request info about specific event
return info
Displays data
send: User selects "attending event"
Data pertaining to user sent to database for storage
return list of attendees
Event owner is notified that user is attending
User
System
Database
www.websequencediagrams.com

5) Effort Estimation using Use Case Points

Unadjusted Actor Weight (UAW)

Actors-

- User- **Complex**, User interact with the Web App through a Graphical User Interface
- Group- **Complex**, same as a user, group interact with the Web App through a GUI
- Database- **Average**, Database interact with the Web App by passing data back and forth through Http protocol

UAW = 0 x Simple + 1 x Average + 2 x Complex
= 0 x 1 + 1 x 2 + 2 x 3
= 8

Unadjusted Use Case Weight (UUCW)

Use cases-

- UC1- User Register and Login - Moderate user interface. Two participating actors (database, webapp system). Five steps for the success scenario. **Average**
- UC2- Adding Friends- Simple user interface. One participating actors (database). Two step for the success scenario. **Simple**
- UC3- Messaging- Moderate user interface. Two participating actors (database, webapp system). Five steps for the success scenario. **Average**
- UC4- Group Creation and Management- Complex user interface. Two participating actors (database, webapp system). Eleven steps for the success scenario. **Complex**
- UC5- Group Joining and Interaction- Moderate user interface. Two participating actors (database, webapp system). Four steps for the success scenario. **Average**
- UC6- Post Creation and Management- Moderate user interface. Two participating actors (database, webapp system). Five steps for the success scenario. **Average**
- UC7- Post View and Interaction- Simple user interface. One participating actors (database). Two step for the success scenario. **Simple**

- UC8- Map Event Creation and Management- Complex user interface. Two participating actors (database, webapp system). Eleven steps for the success scenario. **Complex**
- UC9- Map Event View and Interaction- Moderate user interface. Two participating actors (database, webapp system). Five steps for the success scenario. **Average**

UUCW = 2 x Simple + 5 x Average + 2 x Complex
= 2 x 5 + 5 x 10 + 2 x 15
= 90

<u>Technical Complexity Factor (TCF)</u>
Nonfunctional Requirements-

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	Our WebApp can run on multiple machines (Computer, laptop, tablet, mobile etc.)	2	3	6
T2	Performance objective is important in our design	1	3	3
T3	Our WebApp guarantee end-user efficiency	1	3	3
T4	Not really complex internal processing	1	3	3
T5	Some reusable design and code	1	1	1
T6	Our WebApp is easy to install (you just need internet connection and a web browser)	0.5	3	1.5
T7	Our WebApp is easy to use	0.5	3	1.5
T8	Our WebApp is portable (can access anywhere with a device that have internet connection	2	3	6
T9	Our code is organize so it easy to change or update	1	3	3
T10	Our WebApp can be concurrently use by multiple users by creating multiple accounts	1	4	4
T11	Some implementation of security by having user data store in a secure database	1	3	3
T12	No direct access for third parties	1	0	0

T13	No special user training facilities are required	1	0	0
			Technical Factor Total	35

TCF = Constant-1 + (Constant-2 x Technical Factor Total)
= 0.6 + (0.01 x 35)
= 0.95

Environment Complexity Factor (ECF)
Experience of each team member-

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor
E1	Somewhat similar with the development process	1.5	1	1.5
E2	Average application problem experience	0.5	3	1.5
E3	Average familiar with the object-oriented approach	1	3	3
E4	Somewhat familiar with lead analyst process	0.5	1	0.5
E5	All team member are highly motivated	1	5	5
E6	Changing requirements is average	2	3	6
E7	No part-time staff involve	-1	0	0
E8	Average programming language will be used	-1	3	-3
			Environmental Factor Total	14.5

ECF = Constant-1 + (Constant-2 x Environmental Factor Total)
= 1.4 + (-0.03 x 14.5)
= 0.965

Productivity Factor (PF)
Person-hours needed per use case point-

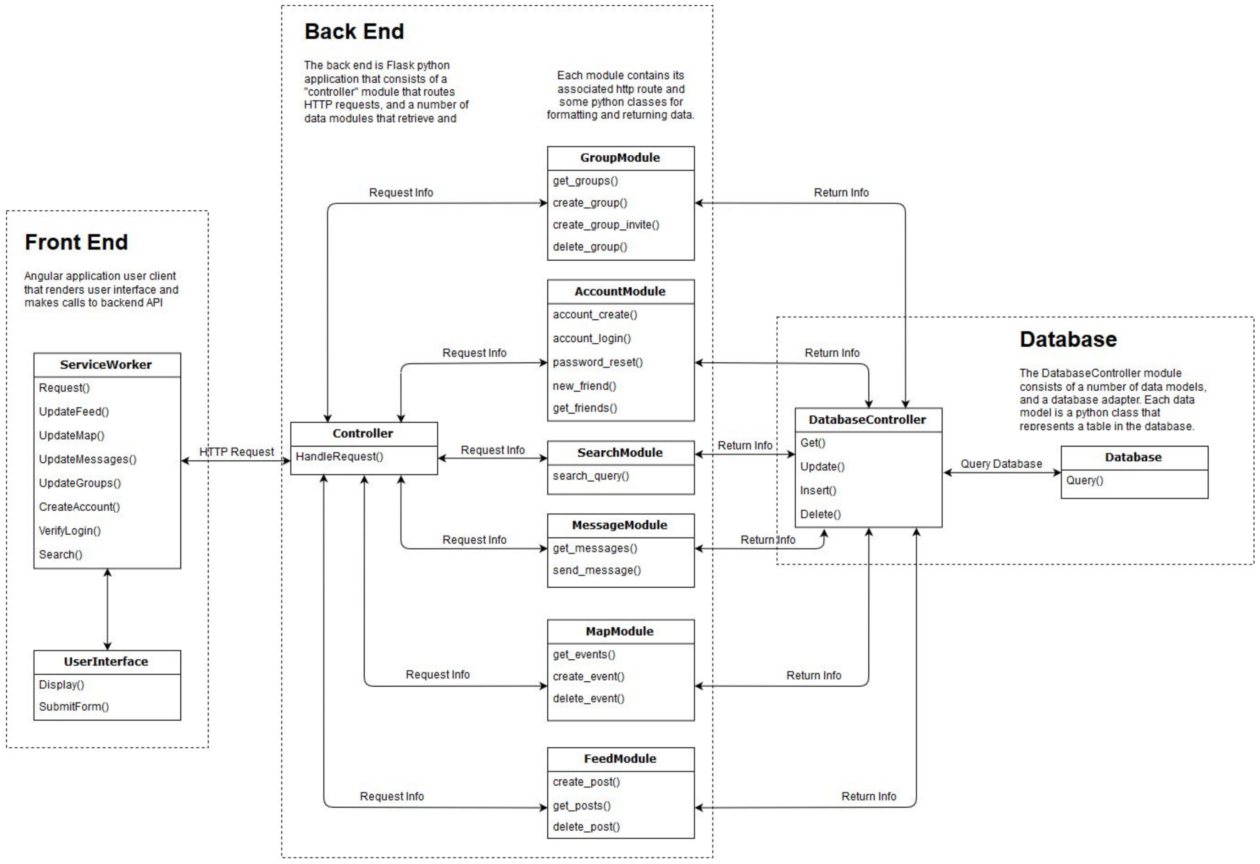
Team of all undergraduate students: **PF = 30**

UCP = UUCP x TCF x ECF
= (UAW + UUCW) x TCF x ECF
= (8 + 90) x 0.95 x 0.965
= 90 Use Case Points

Duration = UCP x PF
= 90 x 30 = 2700 person-hours

6) Domain Analysis

(a)Domain Model



In the diagram we can see the clear distinction between client (front end) and server (back end) in our application. This application follows the client-server design pattern.

Domain Model Diagram

- Concepts Definition

Responsibility	Type	Concept
R1: Accepts and handles request from the front end the application.	D	Controller
R2: Fetches new information from the back end	D	ServiceWorker

server and updates the user interface.		
R3: Formats data received from the ServiceWorker and displays it to the user. Accepts input from the user and transmits it to the ServiceWorker.	D	UserInterface
R4: Stores and provides access to data	K	Database
R6: Sits between database and the rest of the application and provides an easy interface.	D	DatabaseController
R7: Fetches data from the posts table using the DatabaseController and formats it to be sent to the user to display the user's home feed. Also uses the DatabaseController to add new post data to the database.	D	FeedModule
R8: Fetches a list of events to display on the map using the DatabaseController and formats it to be sent to the user to display the user's event map. Also uses the DatabaseController to add new event data to the database.	D	MapModule
R9: Fetches a list of messages to display using the DatabaseController and formats them to be sent to the user to display the user's messages. Also uses the DatabaseController to add new message data to the database.	D	MessagingModule
R10: Fetches group information from the DatabaseController and formats it to be sent to the user to display a group's page and new posts. Also uses the DatabaseController to add new group data to the database.	D	GroupModule
R11: Creates new account entries in the database and verifies sign ins for already existing accounts.	D	AccountModule
R12: Fetches information relating to a search query from the relevant databases using the DatabaseController and formats it to be sent to the user.	D	SearchModule

- Associate definitions

Concept pair	Association Description	Association Name

UserInterface<->ServiceWorker	The User Interface Displays information to the user that is provided by the ServiceWorker	Display
ServiceWorker<->Controller	The ServiceWorker communicates with the Controller request and send data back and forth	Request Communication
Controller<->GroupModule	The Controller handles requests from the GroupModule about updating group information	Request handling
GroupModule<->DatabaseController	The GroupModule Contacts the the DatabaseController to get or update group data in the Database	Data Interaction
Controller<->AccountModule	The Controller handles requests from the AccountModule about creating and verifying user accounts	Request handling
AccountModule<->DatabaseController	AccountModule sends create read, or delete requests to the DatabaseController	Data Interaction
Controller<->SearchModule	Controller handles requests from the SearchModule	Request handling
SearchModule<->DatabaseController	SearchModule sends create read, or delete requests to the DatabaseController	Data Interaction
Controller<->MessagingModule	Controller receives requests to send, read, and delete messages from the MessageModule	Request Handling
MessagingModule<->DatabaseController	MessageModule sends create read, or delete requests to the DatabaseController	Data Interaction
Controller<->MapModule	Controller handle request about map interaction by sending appropriate request through MapModule	Request Handling
MapModule<->DatabaseController	MapModule sends create, read, or delete requests to the DatabaseController	Data Interaction
Controller<->FeedModule	Controller handle request about feed interaction by sending appropriate request through FeedModule	Request Handling
FeedModule<->DatabaseController	FeedModule sends create, read, or delete requests to the DatabaseController	Data Interaction

DatabaseController<->Database	Database sends Queries to Update the database	Data handling

- Attribute definitions

Concept	Attribute	Attribute Description
Controller	Handle Request	Handle requests from the user by interact with appropriate module to update data in the database and update user interface
ServiceWorker	Request	Communicate user request from the interface to the controller
UserInterface	Display	Display information data to the user on screen
	SubmitForm	Send request base on user interaction on the screen through the ServiceWorker to the Controller for appropriate action
Database	Query	Modify data in the database, base on the request of the DatabaseController
DatabaseController	Get	Handle request to get and return data from the database
	Update	Handle request to update data in the database
	Insert	Handle request to insert new data into the database
	Delete	Handle request to delete data in the database
FeedModule	GetFeed	Get the post data that the user request from the database and format it to human readable format and send it back to interface to be display on screen
	CreatePost	Take user's newly created post and added it to the database
	DeletePost	Delete the post that the user request from the database

MapModule	GetEvent	Get data about a map event from the database and send it to be display on the interface
	CreateEvent	Added data about a map event that is created by user into the database
	DeleteEvent	Delete data about a map event from the database
MessagingModule	GetMessage	Get data about user request messages from the database
	CreateMessage	Added user newly send message to the database
	DeleteMessage	Delete message that user requested from the database
GroupModule	GetGroups	Get data information about a group from the database
	CreateGroup	Add group data that user entered into the database
	CreateGroupPost	Add post data from a group to the database
	CreateGroupInvite	Add user's send group invitations to the database
	DeleteGroup	Remove a group data from the database
	DeleteGroupPost	Remove post data from a group from the database
AccountModule	CreateAccount	Add a newly user created account information into the database
	VerifyLogin	Check user entered login information with user's data in the database and allow user to access that information if it matched
SearchModule	Search	Check if a specific data that that the user enter is in the database

- <u>Traceability matrix</u>

Use Case	Size Points	Domain Concepts										
		Controller	Service Worker	UserInterface	Database	DatabaseController	FeedModule	MapModule	MessagingModule	Group Module	AccountModule	Search Module

UC-1	21	X	X	X	X	X					X	
UC-2	13	X	X	X	X	x						X
UC-3	15	X	X	X	X	X			X			
UC-4	17	X	X	X	X	X				X		
UC-5	11	X	X	X	X	x				X		
UC-6	17	X	X	X	X	X	X					
UC-7	14	X	X	X	X	X	X					
UC-8	5	X	X	X	X	X		X				
UC-9	16	X	X	X	X	X		X				

(b)System Operation Contract

Use Case 1:

- Preconditions: The application's interface is displaying a login screen that will prompt the user to click a link if they do not have an account.
- Postconditions: The user will have a username and account in the database. The user will now be able to use the app.

Use Case 4:

- Precondition: The application's interface is displaying a "groups page", listing all of the groups the user belongs to. On the same page is a create group button.
- Postcondition: New group data is added to the database. The user now can interact directly with his group or create new groups.

Use Case 9:

- Preconditions: The application's interface is displaying a map with user created events on it.
- Postconditions: The event's information on the database was fetched and displayed on the screen when user click on a specific event. The user can now interact with the map to find events near them.

7) Interaction Diagrams

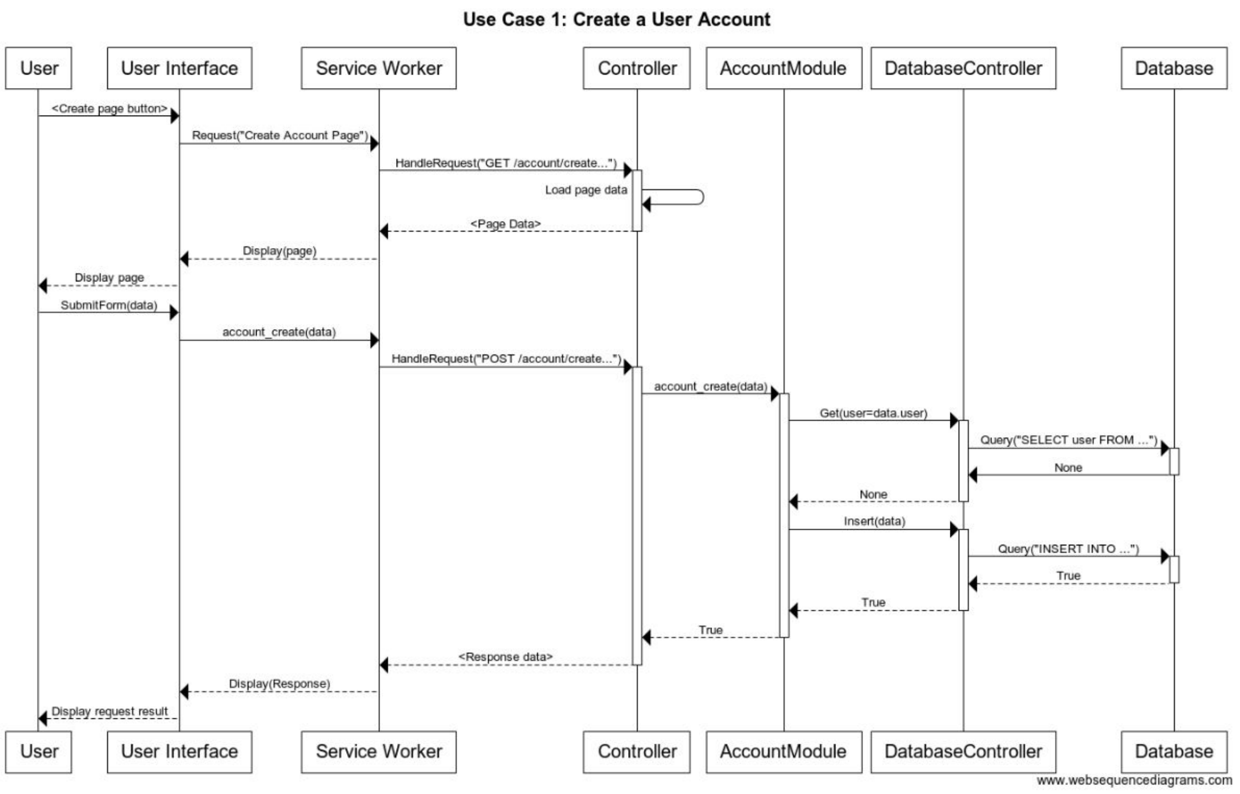
(a) Design Patterns

Our project is a web app that uses a Angular for frontend and Flask for the backend. It is architected using modern web design, where the frontend is deployed as a single javascript application and run on the client device. The backend consists of an Azure SQL database that is accessed by a python application. An API for the frontend to access is exposed using the Flask python web framework. The project uses the client-server design pattern. The angular application serves as the client, and is run on any modern day web browser by someone using the application. The server is the python application and it is run on an Azure virtual machine.

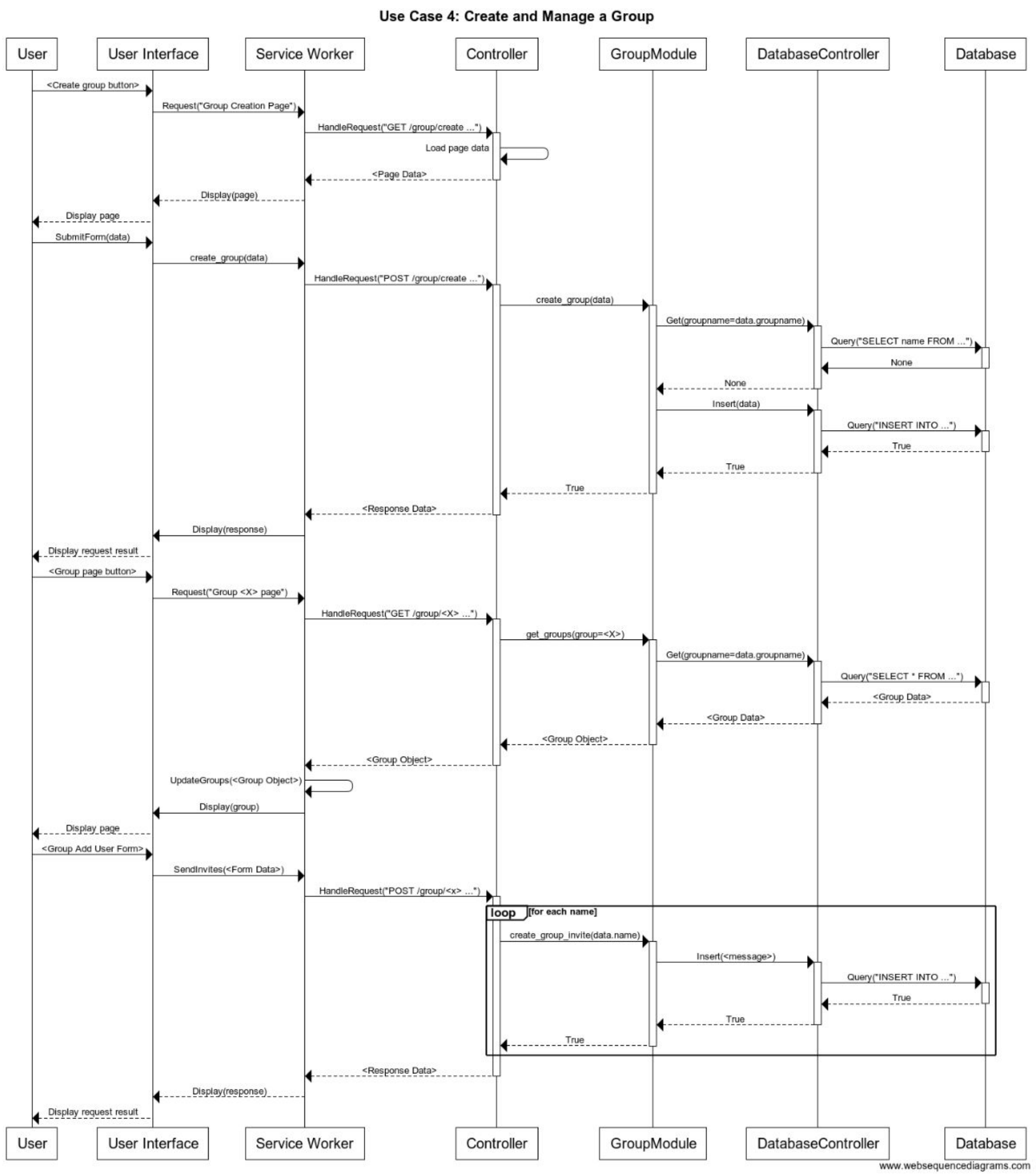
All of our interaction diagrams use the same client-server design pattern in the same way. The client is the angular application and is run on the user's web browser. The client creates a user interface to gather information and commands from the user. This is packaged and sent asynchronously to the server by a process in the angular app called a serviceworker/httpworker. This app does not use the old way of handling web traffic where there is one http request per page. Instead of fully separates the client application and the server application so information can travel in both directions asynchronously to provide a smoother user experience. By separating the client and the server we are also able to keep sensitive information and business logic completely encapsulated in our cloud while a thin client layer interacts with it only through the exposed API endpoints. This lets us keep the web app more secure as well as keeping the frontend application smaller making it faster to load and more accessible.

(b) Diagrams

Our design tries to minimize the number RDD types that each object has. For instance, the AccountModule is Type 2 because it has many methods that request data from other objects and return said data to the object calling it. It is also Type 3 because it calls methods from DatabaseController in its methods to get requested data. However, AccountModule is completely stateless. It maintains no information about user accounts other than how to request and transform user account data.
The database is strictly Type 1. It knows and can return information but it communicates with no other objects unless it is called upon and it does no data transformations on the data it contains.

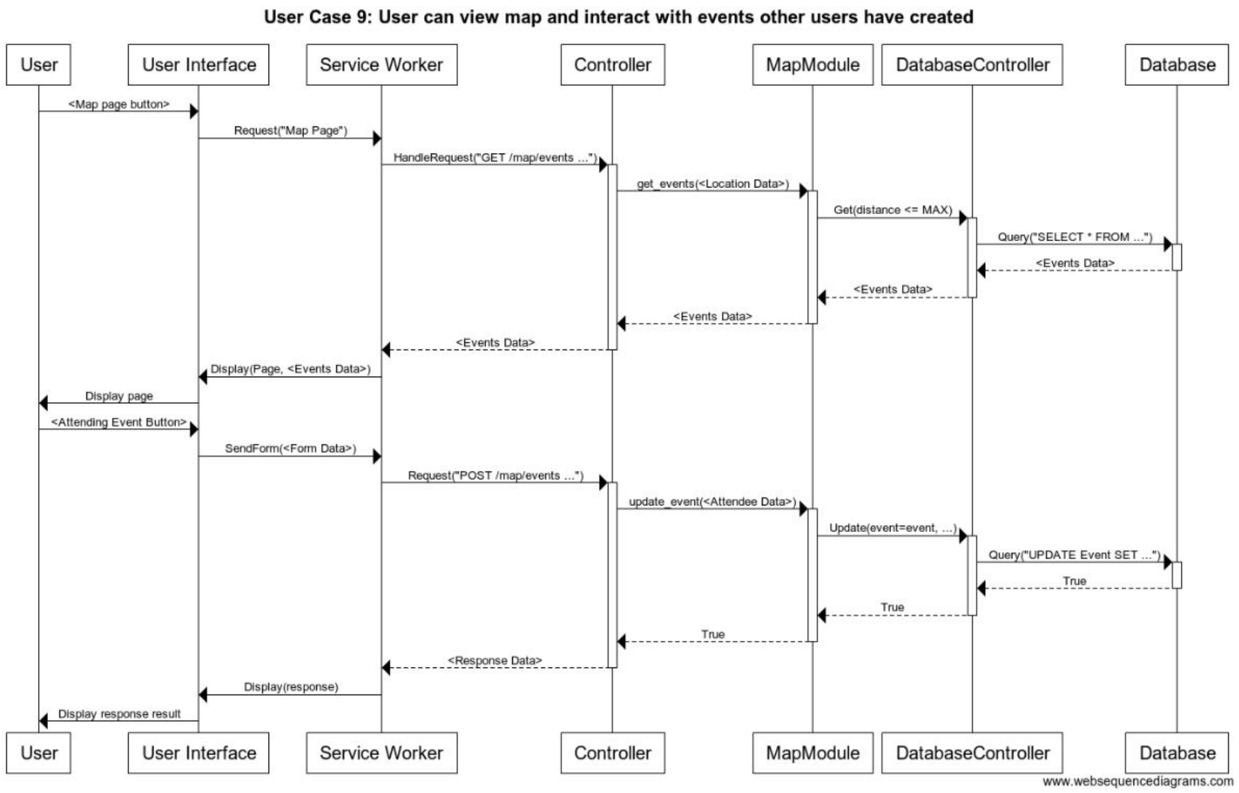


Interaction Diagram 1: All modules are designed with a balance of the High Cohesion and Low Coupling Principles in mind. The AccountModule does nothing but request data from the DatabaseController and return it to the requesting object. Since it only communicates with the DatabaseController it has low coupling, and since it only handles data about user accounts, it has high cohesion.



Interaction Diagram 2: The GroupModule is similar to the account module in that it operates only on GroupData. In this we can see that the ServiceWorker also has additional responsibilities because it must maintain state information as well as communicate with the controller.
While this means that the ServiceWorker is a Type 1, 2 and 3 object we felt this was necessary to maintain asynchronous data flow between the user interface and the backend objects.
The controller also does not follow the overall design principles closely. It is highly coupled because it must communicate with all modules that get data from the database. However, the

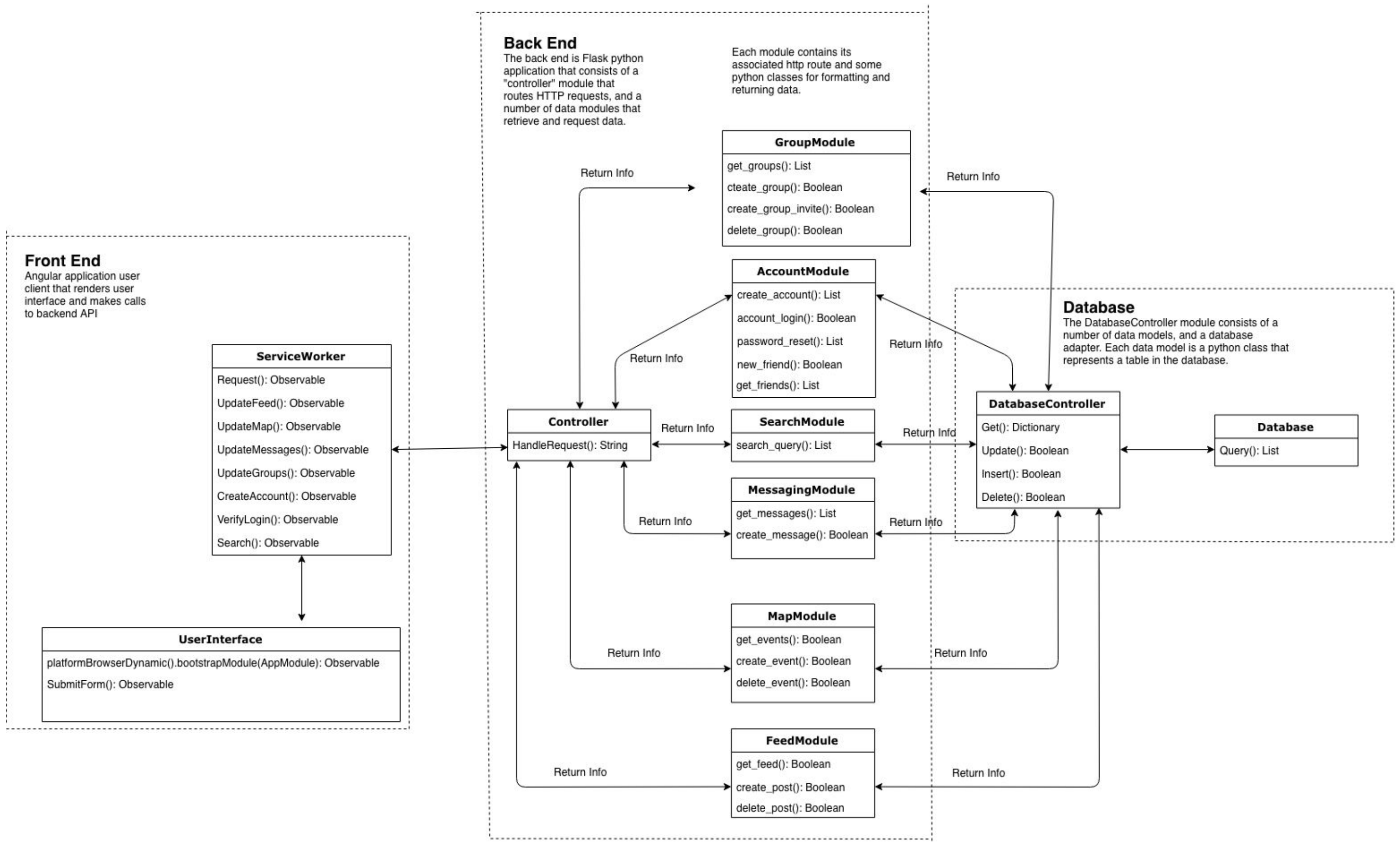
controller has no Type 1 or Type 2 responsibilities because it exists only to pass messages from the service worker to back end modules.



Interaction Diagram 3: The MapModule is similar to other modules in that it has low coupling and high cohesion.
The DatabaseController is another object that has very low coupling and very high cohesion. The DatabaseController communicates only with the database and does nothing other than request data from the database.
It exists to separate other modules in the backend from the database to reduce overall module coupling. This also aids in overall design because it means that only one object is interacting with the database directly. With all data from the database flowing through the DatabaseController it is easy to map the data flow throughout the backend making testing the objects much easier.

8) Class Diagram and Interface Specification

(a) Class Diagram


Back End
The back end is Flask python application that consists of a "controller" module that routes HTTP requests, and a number of data modules that retrieve and request data.
Each module contains its associated http route and some python classes for formatting and returning data.
Front End
Angular application user client that renders user interface and makes calls to backend API
Database
The DatabaseController module consists of a number of data models, and a database adapter. Each data model is a python class that represents a table in the database.
ServiceWorker
Request(): Observable
UpdateFeed(): Observable
UpdateMap(): Observable
UpdateMessages(): Observable
UpdateGroups(): Observable
CreateAccount(): Observable
VerifyLogin(): Observable
Search(): Observable
UserInterface
platformBrowserDynamic().bootstrapModule(AppModule): Observable
SubmitForm(): Observable
Controller
HandleRequest(): String
GroupModule
get_groups(): List
cteate_group(): Boolean
create_group_invite(): Boolean
delete_group(): Boolean
AccountModule
create_account(): List
account_login(): Boolean
password_reset(): List
new_friend(): Boolean
get_friends(): List
SearchModule
search_query(): List
MessagingModule
get_messages(): List
create_message(): Boolean
MapModule
get_events(): Boolean
create_event(): Boolean
delete_event(): Boolean
FeedModule
get_feed(): Boolean
create_post(): Boolean
delete_post(): Boolean
DatabaseController
Get(): Dictionary
Update(): Boolean
Insert(): Boolean
Delete(): Boolean
Database
Query(): List
Return Info

(b) Data Types and Operation Signatures

UserInterface

platformBrowserDynamic().bootstrapModule(AppModule): Boolean

- Boolean variable corresponds to if UI is displayed

SubmitForm(): Boolean

- Boolean variable corresponds to if the form submitted

ServiceWorker

Request(): Boolean

- Boolean variable corresponds to if the request was successful or not.

UpdateFeed(): Observable<String[]>

- Observable variable corresponds to a String array that will populate the feed as new data is received. The data is subscribed to

UpdateMessages(): Observable<String[]>

- Observable variable corresponds to a String array that will populate the messages for the user. The data is subscribed to

UpdateGroups(): Observable<String[]>

- Observable variable corresponds to a String array of Groups. The data is subscribed to to keep up to date.

CreateAccount(): Boolean

- Boolean variable corresponds to if the request to create an account was successful or not.

VerifyLogin(): Boolean

- Boolean variable corresponds to if the login was successful or not

Search(): Observable<String[]>

- Observable variable corresponds to a String array of Events and People nearby. The data is subscribed to

Controller

HandleRequest(): String

- String variable corresponds with the data that is returned from the modules

GroupModule

get_groups(): List

- List variable corresponds with the list of groups that are returned from the search

create_group_invite(): Boolean

- Boolean variable corresponds to if the group invite was successful or not

delete_group(): Boolean

- Boolean variable corresponds to if the group was able to deleted or not

delete_group_invite(): Boolean

- Boolean variable corresponds to if the group invite was able to deleted or not

AccountModule

create_account(): List

- List variable corresponds with the account information that is provided by the user

account_login(): Boolean

- Boolean variable corresponds to if the login was successful or not

password_reset(): List

- List variable corresponds to the new password that is sent to the back end of the application

new_friend(): Boolean

- Boolean variable corresponds to the respons to the new friend request response

get_friends(): List

- List variable corresponds to the returned list of friends

SearchModule

search_ query(): List

- List variable corresponds with the list that is returned as a result of the search

MessagingModule

get_messages(): List

- List variable corresponds to if the messages were properly retrieved

create_message(): Boolean

- Boolean variable corresponds to if the message was able to deleted or not

MapModule

get_events(): Boolean

- Boolean variable corresponds to if the events were properly retrieved

create_events(): Boolean

- Boolean variable corresponds to if the event was able to be created or not

delete_event(): Boolean

- Boolean variable corresponds to if the event was able to deleted or not

FeedModule

get_feed(): Boolean

- Boolean variable corresponds to if the feed was properly retrieved

create_post(): Boolean

- Boolean variable corresponds to if the post was able to be created or not

delete_ost(): Boolean

- Boolean variable corresponds to if the post was able to deleted or not

DatabaseController

Get(): Dictionary

- Dictionary variable corresponds with the data that is returned from the database to the controller

Update(): Boolean

- Boolean variable corresponds to if the database was able to delete or not

Insert(): Boolean

- Boolean variable corresponds to if the database was able to insert or not

Delete(): Boolean

- Boolean variable corresponds to if the database was able to delete or not

Database

Query(): List

- List variable corresponds with the returned information from the database

(c) Traceability Matrix

	Class

Domain Concepts	Search Module	Messaging Module	FeedModule	MapModule	Database Controller	Database
Controller					X	
View						
Data Storage						X
Communicator						
Data Manipulator	X	X	X	X		
Data Receiver						
Location Driven	X		X	X		

	Class				
Domain Concepts	User Interface	ServiceWorker	Controller	GroupModule	AccountModule
Controller			X		
View	X				
Data Storage					
Communicator		X			
Data Manipulator				X	X
Data Receiver	X				
Location Driven				X	

Due to the nature of the application, the domain concepts were fairly simple. The UserInterface receives data from the user and sends it to the backend through the service worker. There is then a module for each type of data we need. Each module (Group, Account, Feed, Map, and Messaging) knows how to manipulate its data that it receives from the database. The controller orchestrates the above. It is a very typical MVC (Model View Controller) design.

The main purpose of Proximity is to create a location driven social media application. The modules that implement this location driven idea are the Group, Feed, Map, and Search Modules.

(d) Object Constraint Language

Class name	Invariant	Precondition	Postcondition
Search Module	Class is successfully linked to Controller and DatabaseController.	User's login has been verified by AccountModule and starts a new session.	User is able to search the app for content that is returned as a list.
User Interface	Display() = True	User inputs information into form.	Form is sent to controller.
Controller	Class is successfully linked to ServiceWorkers.	Data is sent and received from ServiceWorkers.	Data is sent and received from requesting Objects.
GroupModule	Class is successfully linked to Controller and DatabaseController.	User's login has been verified by AccountModule and starts a new session.	User can create, delete, and invite groups.
AccountModule	Class is successfully linked to Controller and DatabaseController.	Data is requested from DatabaseController.	User can login to an existing account or create a new account.
MessagingModule	Class is successfully linked to Controller and DatabaseController.	User's login has been verified by AccountModule and starts a new session.	User can get, create, and delete messages.
FeedModule	Class is successfully linked to Controller and DatabaseController.	User's login has been verified by AccountModule and starts a new session.	User can get, create, and delete posts.
MapModule	Class is successfully linked to Controller and DatabaseController.	User's login has been verified by AccountModule and starts a new session.	User can get, create, and delete events.

DatabaseController	The database is readily accessible and contains proper records.	The user requests or creates an event, post,message, or group.	The request is sent to the database.
Database	The database is readily accessible and has sufficient storage capacity.	The user creates an event, post,message, or group and request is received from DatabaseController.	The information is stored or sent by the database.

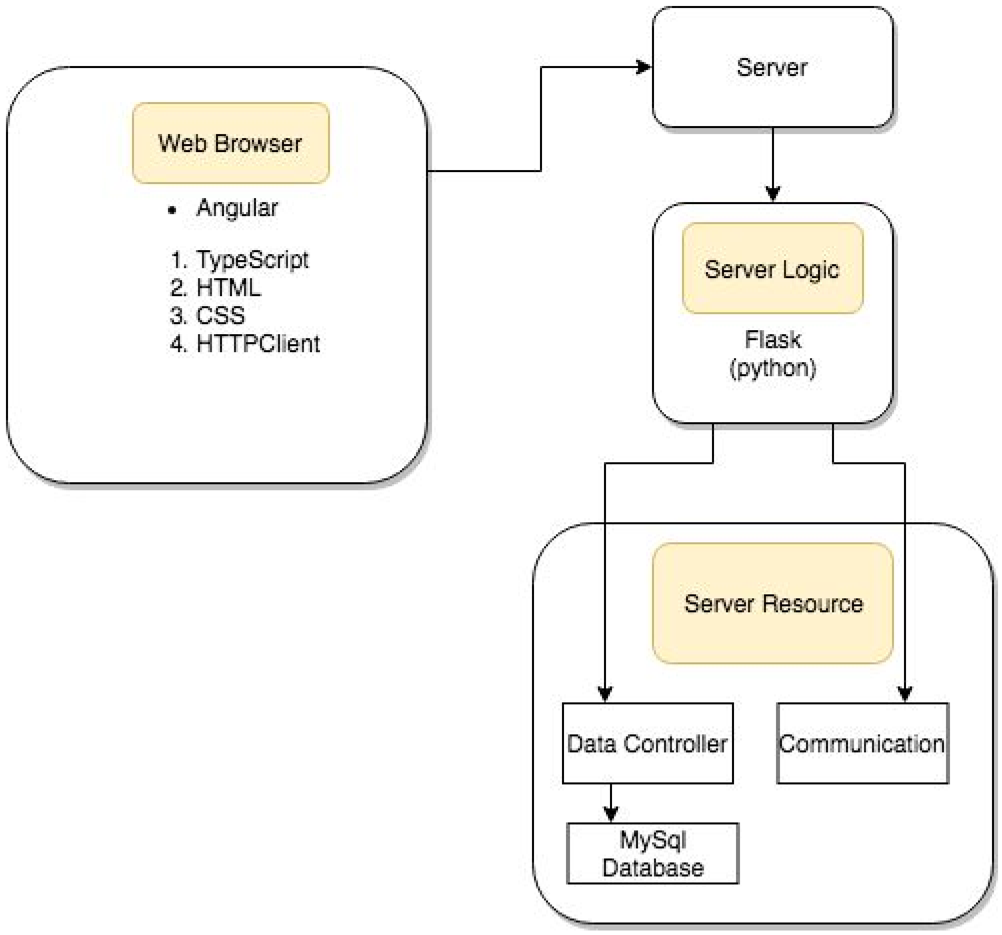
9) System Architecture and System Design

(a) Architectural Styles

Proximity uses a combination of a server-client architecture system and model-view-controller system. Using the Model View Controller architecture pattern, we have divided our application into three parts. The model handles storing data and CRUD operations and the view handles requests from the user. The model and the view are totally separate from each other, and do interact directly with each other in any way. The controller is used to handle the flow of requests and response between the two.

We also utilize a client-server architecture system because are application exists on a server and works by displaying information to clients. Our server exists on Amazon Web Services and the client is the web browser of the user.

(b) Identifying Subsystems



Subsystems Diagram

(c) Mapping Subsystems to Hardware

Proximity runs on a client-server architecture. The client exists on the user's computer or mobile device and the server exists on Amazon Web Services.

(d) Persistent Data Storage

Proximity uitizes MySQL, a relational database system, to store all the information needed about users, posts, events, and groups. Only the model part of the system is allowed to interact with this database. The database lives on Proximity's server.

(e) Network Protocol

Proximity uses the HTTP as the communication protocol. This was chosen due to HTTP being the standard client-server protocol. It also has the advantage of being stateless and platform independent.

(f) Global Control Flow

Execution Orderliness

The execution order is event-driven. The application provides a variety of events that the user can activate in an order they desire. For example, a user can create an event then message a friend, or the user can message a friend and then create an event. Aside from from first logging in, there’s is no required order for these events.

Time Dependency

Proximity is a real time system in the sense that it records the datetime of every post and every message. In addition events that are posted to the map expire after a certain amount of time.

(g) Hardware Requirements

- Screen Display
- An internet connection
- A web browser

10) Algorithms and Data Structures (If applicable)

(a) Algorithms

The main algorithm in use is a distance finding algorithm that takes two sets of location data, in (latitude, longitude) format, and returns the distance between the two points in miles. This algorithm is used in database queries to find nearby posts, events, and people.

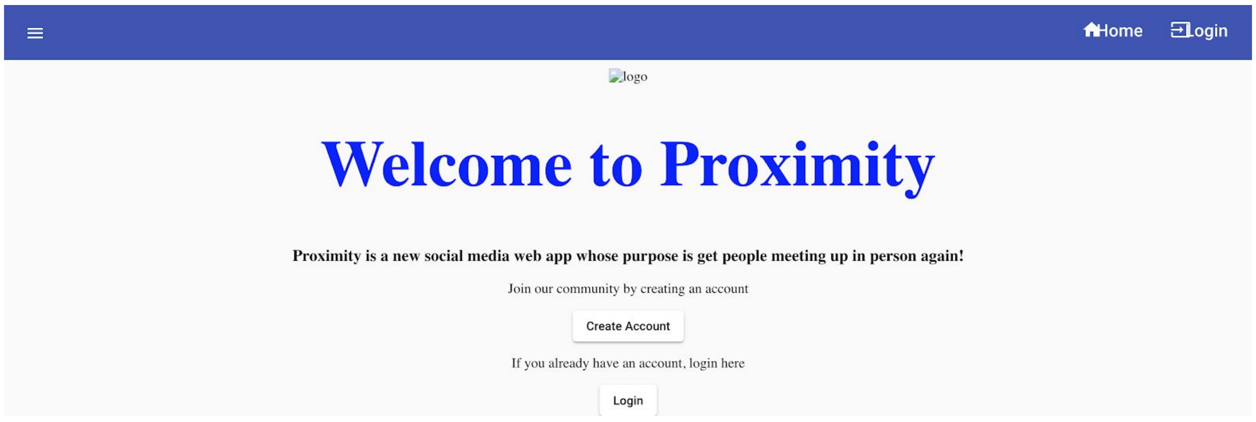
(b) Data Structures

Our WebApp is going to pass data back and forth between the database and controller to update the user interface. These data will have the format of associative arrays, which is easy to implement. Other than that, we are not going to use any other complex data structures in our webapp.

11) User Interface Design and Implementation

(a)Preliminary Design

Home Page



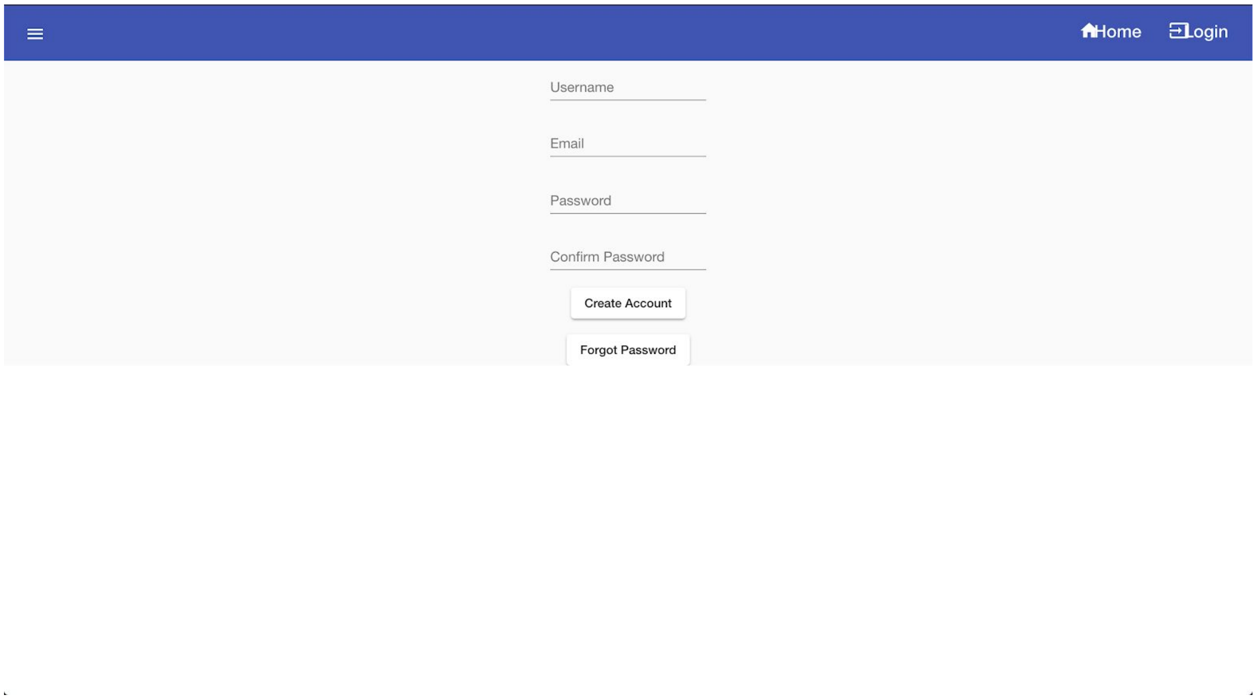
Welcome to Proximity

Proximity is a new social media web app whose purpose is get people meeting up in person again!

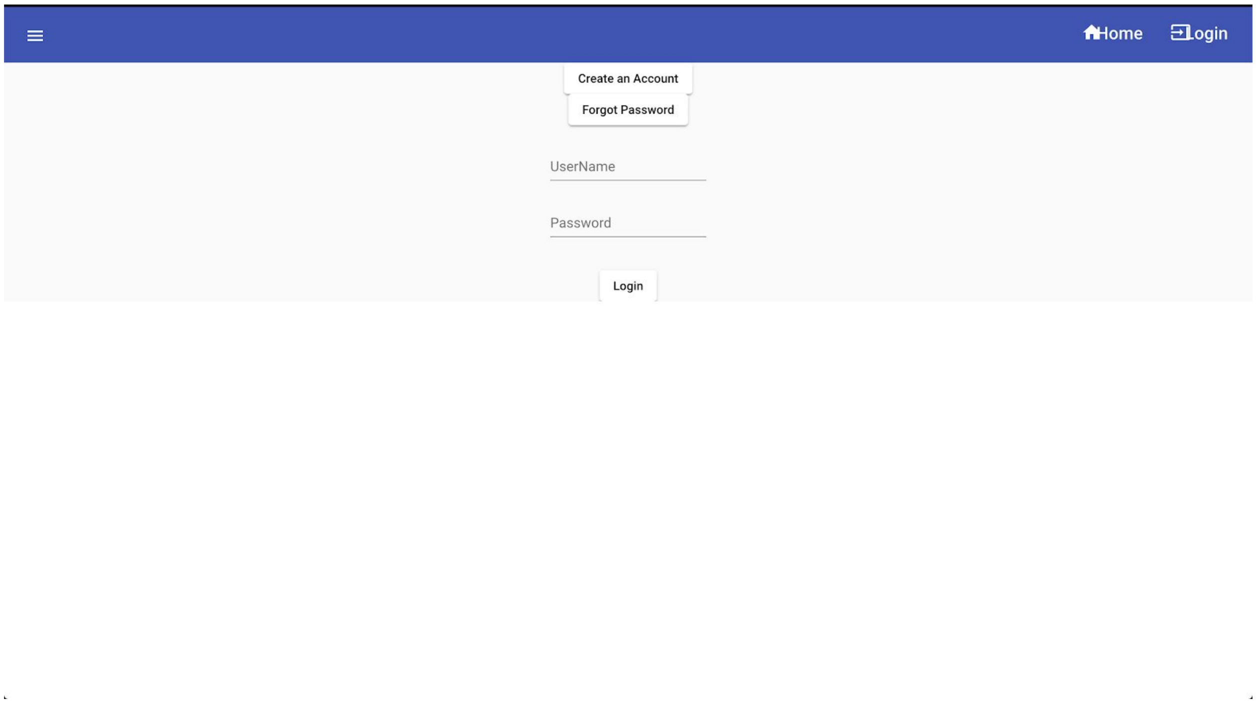
Join our community by creating an account

If you already have an account, login here

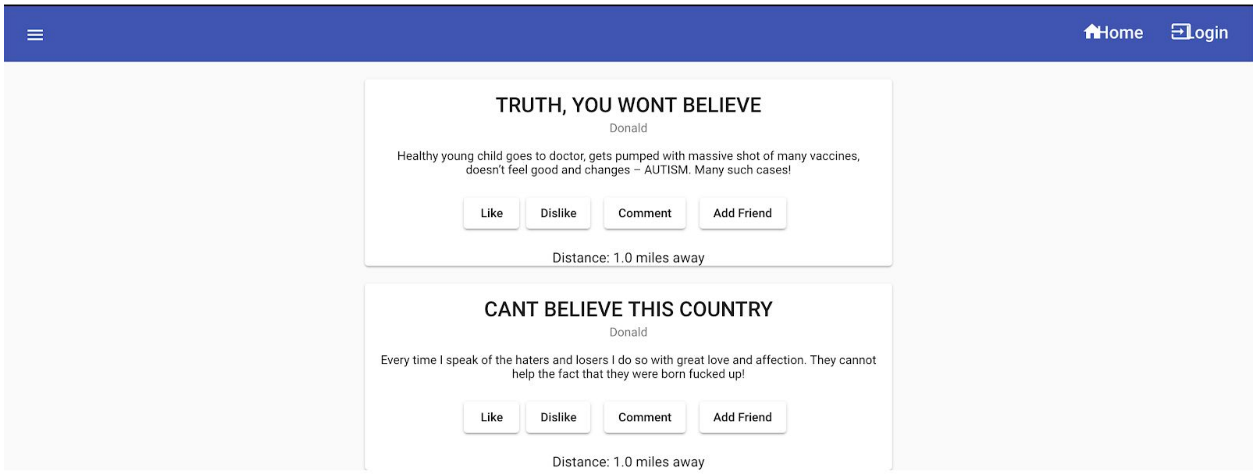
Create Account



Login Page



Feed Page



TRUTH, YOU WONT BELIEVE

Donald

Healthy young child goes to doctor, gets pumped with massive shot of many vaccines, doesn't feel good and changes – AUTISM. Many such cases!

Distance: 1.0 miles away

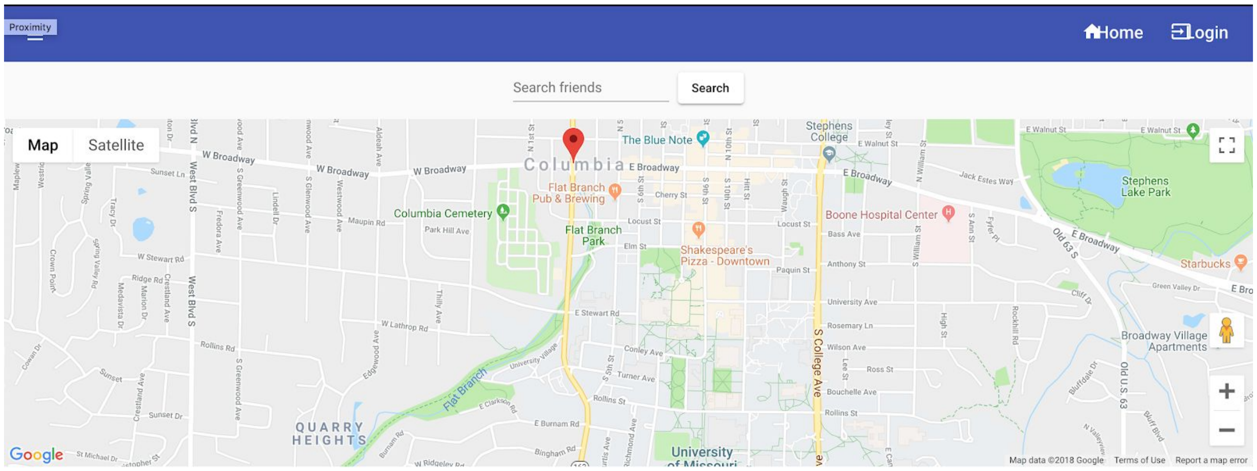
CANT BELIEVE THIS COUNTRY

Donald

Every time I speak of the haters and losers I do so with great love and affection. They cannot help the fact that they were born fucked up!

Distance: 1.0 miles away

Map Page



(b)User Effort Estimation

Use Case 1: A user can create and sign in to a user account. To sign in a user will need to fill in the username and password text boxes on the entry page to our app. This involves the user pressing into both of the text boxes, typing their username and password in, and pressing the login button. There are 3 button presses the user will need to make along with key presses to type their username and password.

Use Case 4: A user can create and manage a group of users. To create a group the user first selects a friend this can be done by searching their name in the messages page and then selecting their profile. Once selected there will be a button where the user will be able to click add user to group. This will open all of your current groups which you can add the user to or you will be able to create and name a new group. There mouse presses a user needs to do is open search bar, search individual, select individual, add to group, and then select the group to add to.

Use Case 7: Users can view and interact with other user's posts on a feed. A user can view other users posts by navigating to the feed page. Once on the page the user can simply scroll through and select a post for further information about the post. This takes only one click to retrieve this information with the addition of a user scrolling through the feed

Use Case 9: A user can look at the map and interact with events that other users have created. The user can view events events that are posted near their location. By clicking on that event on the map, it will give the user more detail on the event like what kind of event is this, who invited and and what time it starts. It only takes one click to open information about the events if it is already visible on the map. If the event is not visible on the map the user can zoom out and move the map to find the event. This will add additional clicks for the user.

12) Design of Tests

Unit test cases:

1) AccountModule tests:
   Account Register test:
   - User create a new account by entering valid information like username, email, password -> expect account information to be added to database and system return message that account was created successfully.
   - Password needed to be enter twice and they have to match. If the user entered a confirm password that doesn't match with the chosen password-> expect the WebApp to issue a warning to user that the passwords entered doesn't match

   Account Login test:
   - User entered username and password that matched with the username and password that they registered in the database -> expect user to be able to login and access the WebApp's contents.
   - The username and password that user entered is doesn't match any combination in the database -> expect the system to take user back to the login screen and said that you have a wrong username and/or password. That way it more secure.

2) MapModule tests:
   Map viewing test:
   - User access to the map by clicking on the map tab on the WebApp interface -> expect the map to be load correctly on the page and accurately display the detail of the created test event.
   - The event on the map will only be visible to people locally so only the people who nearby in the area within 10 miles radius will be able to view the event. User loaded map page 20 mile away from the test event -> expect the system to not show the event when user load up the map page.
   - User create a test event and posted on the map -> expect the map to display the event at the location that the event were created and not somewhere else.

4) GroupModule tests:
   Group Creation test:
   - User finishes filling out the the group page using valid information and hit the button “Create New Group” -> expect group data to be added to the database and system passes back message that the group is successfully created

- User finishes filling out the the group page using same name of some existing test group in the database and hit the button “Create New Group” -> expect group data to be redirect back to the user with the system message of “The group you try to create already exist. Please choose a different group name”.

5) <u>MessagingModule tests:</u>
Message sending test:

- User finishes typing their message to the chat box and hit send -> expect the message data to be send to the the test account that specify in the header and not to other test accounts.
- User finish typing their message to the group message chat box and hit send -> expect the message data to be send to the of test accounts that specify in the header

6) <u>FeedModule tests:</u>
Post on a feed test:

- User finishes writing a post and hit post -> expect post data to be successfully added to the database. If the post is public, expect to be able to view the post from any test account. If the post is a friend only post, expect to be able to access and view the post only from account that on the test account friend list.

*** During the testing process, if any of the test case return unexpected result, it will be considered failure and need to be debug. As the time that this report is written, we are still implementing unit tests for Demo 2, so some of the test cases (MapModule, GroupModule, MessagingModule) that listed above is just a design and maybe change when we actually implement it.

<u>Test Coverage</u>

Each test case coverage is the class that it written under (for example, Map viewing test cover MapModule class). Also, for each test case, although not mentioned explicitly, necessary classes like UserInterface, ServiceWorker, Controller, DatabaseController and Database were also involved.

<u>Integration Testing</u>

1. Test the home page -> expecting it to load correctly
2. Test the account creation page by going through unit tests for that page. In the end, expect to successfully create a new account with the username equal "test" and password equals "pass"
3. Test the login page by going through unit tests for that page. In the end, expect login successfully by using the username and password that created in step 2.
4. Test the map page by going through unit tests for that page. In the end, expect the map page to successfully loaded event data into the map markers

5. Test the feed page by going through unit tests for that page. In the end, expect the feed page to successfully loaded feed data from the database
6. Test the messaging page by going through unit tests for that page. In the end, expect the messaging page to successfully loaded messages data from the database
7. Test the logout button and expect it to signed out by removing current session and redirect you to the home screen

***As the time of this report, the integration test is still being update/written so the final implementation can be different from the design above

13) History of Work, Current Status, and Future Work

(a) History of Work and Current Status

As part of the first milestone for Demo 1, our group plan to complete the basic functions of our Web App. that including basic user interface, register and login function, map interaction and database. We were able to get all of them working by the time of Demo 1. Although most of our function are still very basic and our user interface still look really unprofessional.

For the second milestone for Demo 2, our group plan to wrap up the rest of the use cases as well as addresses any issues that we still have from Demo 1. We plan to working on it throughout Thanksgiving break. However, due to the lack of communication, we weren't able to accomplish as much as we hope. At the time that this report is turned in, we are still working on the WebApp. We got most of the functionality on the back end done and continuing to working on the front and will have it done by the time of our presentation for Demo 2.

For the final project report, we divided the works evenly between team members. A single document that included a cover page, table of contents, headers for each require sections were created and shared to all team members on Google Drive. This ensure a consistent format throughout the report. Each team members were assigned specific parts of the report to work on but since some parts are associate with each other, sometimes we need to wait for one part to finish before we can get working on the next part. This wasted lot of time but we were able to speed up the process by maintaining constant communication through our phone and were able to work together and finished the report on time.

(b) Key Accomplishments

- Learn how to use Angular to make a professional WebApp
- Learn how to work and solve problems as a team.
- Learn the software engineering design process and how to write documentation for our software
- Learn how to work under pressure and complete each milestone by the deadline

(c) Future Work

For our WebApp, there are endless possibilities of what we can do with it in the future. For starter, we can go back and implement any use cases that we didn’t finish on time (Like UC3-Messaging, we may put it up for future work if we cannot finish it before Demo 2 presentation). Then we can further extend it by adding more functionality that a social media app should have. This include the ability to share online article, customize your profile, etc. We can also host our WebApp on a public server for people to test and get feedbacks from them on how we could further improve our WebApp

14) References

- This Project Report 3 is made follow Lecture Slides and example Project Samples (included HeartRateAdjuster.pdf, HeathMonitoring.pdf, RestaurantAutomation.pdf) that posted on Canvas

- Part **5) Effort Estimation using Use Case Points** of the report made follow Lecture Slides 15 Measurement-Estimation.pdf

- Part **6) Domain Analysis** follow example of Project Samples: HealthMonitoring.pdf that posted on Canvas

- This Project Report 3 document made using tools from Google Drive
  https://drive.google.com/

- “Software Engineering” book by Ivan Marsic
  http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

- System Sequence Diagrams and Interaction Diagrams made using tool from:
  https://www.websequencediagrams.com/

- Class Diagrams and Subsystems Diagram made using tool from:
  https://www.draw.io/

- Use Case Diagram made using tool from:
  https://www.smartdraw.com/

- Complementary Color Generator
  https://coolors.co/c1edcc-b0c0bc-a7a7a9-797270-453f3c