Sit With Us

Software Design Specifications

Will Crain, David Glenn, Ryan Mitchell, Hazem Tashkandi

October 31st, 2017

CHANGE HISTORY

Date	Version	Description	Author
September 19, 2017	0.01	Initial list of Functional and Nonfunctional Requirements	Team
September 21, 2017	0.02	Initial Use Case Diagrams published	Team
September 26, 2017	0.03	Revised Use Case Diagrams, start of Activity Diagrams	Team
September 28, 2017	0.04	Added more Activity Diagrams, Use Case Diagrams revised	Team
October 1, 2017	0.05	Finished draft of Activity Diagrams, began write up of requirements, including Table of Contents, Purpose, Glossary, and Project Description	Team
October 2, 2017	0.06	Continued work on Requirements document, Activity Diagrams revised, Class Diagram published. Sorting of essential parts of program finished.	Will, Ryan
October 3, 2017	0.1	Revising of Activity Diagrams and Use Case Diagrams, writing of activity diagram descriptions, writing of use case descriptions. Finalization of formatting.	Team
October 10, 2017	0.11	Beginning of Design Document	Team
October 17, 2017	0.12	Initial Draft of detailed Class Diagram. Added new terms to glossary.	Team
October 19, 2017	0.13	Further work on Class Diagram.	Team
October 23, 2017	0.14	Further work on Class Diagram, draft of Sequence Diagrams.	David, Ryan, Will
October 28, 2017	0.16	Addition of Sequence Diagrams	Ryan
October 30, 2017	0.17	Addition of Sequence Diagrams	Team
October 31, 2017	0.2	Refining Sequence Diagrams and Class Diagram, fixing of Activity and Use Case diagrams to match new program layout.	David, Ryan, Will

TABLE OF CONTENTS

CHANGE HISTORY 1

TABLE OF CONTENTS 2

I. INTRODUCTION 5
- Glossary 5
- Scope 6
- Goals 6

II. PROJECT DESCRIPTION 7
- Profile Creation 7
- Profile Management 7
- Meetup Creation 7
- Meetup Search 7
- Meetup Management 8
- Meetup History 8
- View Profile 8
- Settings Management 8
- Developer Management 9

III. REQUIREMENTS 10
- Functional Requirements 10
  - Profile Creation and Management 10
  - Group Search and Management 10
  - Settings and Miscellaneous 10
  - Developer Management 11
- Non-Functional Requirements 11

IV. CLASS DIAGRAM 12
- App Class Diagram 12
- Server Class Diagram 14

V. USE CASE DIAGRAMS 16
- Log-In Use Case Diagram 16
- Main Screen Use Case Diagram 17
- Account Management Use Case Diagram 18
- Meetups Main Use Case Diagram 19
- Meetup Matching Use Case Diagram 20
- In A Meetup Use Case Diagram 21

VI. ACTIVITY DIAGRAMS 22
- Account Management Activity Diagram 22

Block List Management Activity Diagram 23
Contact Developers Activity Diagram 23
Create Profile Activity Diagram 24
Edit Profile Activity Diagram 25
Meetup Management Activity Diagram 26
Login Activity Diagram 27
Meetup Search Activity Diagram 28
Main Screen Activity Diagram 29
Report User Activity Diagram 30
Server Meetup Search Activity Diagram 30
View Meetup History Activity Diagram 31
View Profile Activity Diagram 32

VII. Sequence Diagrams 33
Create User Diagram (App) 33
Handle Create User Diagram (Server) 34
Handle Verify User Diagram (Server) 35
Login User Diagram (App) 36
Handle Login User Diagram (Server) 37
Handle Verify User Login Diagram (Server) 38
Handle Check Login Verified Diagram (Server) 39
View Profile Diagram (App) 40
Edit Profile Diagram (App) 41
Handle View Profile Diagram (Server) 42
Handle Edit Profile Diagram (Server) 43
View Blocklist Diagram (App) 44
Handle View Blocklist Diagram (Server) 45
Handle Edit Blocklist Diagram (Server) 45
Handle Toggle Friend Status (Server) 46
Handle Get Friends Diagram (Server) 47
Meetup Search Start/Stop/Update (App) 48
Handle Meetup Search Start/Stop (Server) 49
Handle Meetup Search Update (Server) 50
Start Meetup (App) 51
Handle Start/Stop Meetup (Server) 52
Leave Meetup (App) 53
View Meetup History (App) 54
Handle Contact Developers Diagram (Server) 55
Contact Developers Diagram (App) 55

Afterword 56

I. INTRODUCTION

Purpose

Sit With Us is an Android social networking app that will allow users to spontaneously meet new people at their current location. The purpose of this app is to make finding new people to develop temporary or long-lasting friendships with easier. The app will allow people to form spontaneous meetups and search for other nearby people and meetups to join using GPS.

Glossary

Account: A user's method of logging in. Account refers to everything the user owns as a result of our program, including Profile, Friend List, and Block List. Users use their Account to sign into the application.

Block: An action a user invokes on another user that hides the user's profile and activity from the other user.

Block list: A list of other users that a particular user has blocked.

Description: A freeform text field designed to allow users to enter a brief description of themselves. This will not be used for matching algorithms, but will be displayed to other users during the matching process to shore up any gaps left by interests.

Developer: A software developer who works on creating the app.

Friend List: The list of contacts in the user's device that have accounts in this app associated with their contact number.

Interest: A textual tag corresponding to a hobby, used to match with other users who share interests.

Meetup: Two or more users currently matched by the app, either through the search function or through the friends list. Can be closed or open to additional users.

Profile: A page owned by a user containing (a) photo(s), interests, and a text field filled out by the user.

Server: The device that runs software used to connect users and manage user data.

Status: Text set before the user searches for a meetup. This text conveys why a user wants to meet up, what a user wants to talk about, or what is on the user's' mind.

User: A person who downloads the app to find other users. User1 refers to the user performing the action, while User2 refers to a user who receives the result of that action without initiating the action themselves.

Required: A feature that we deem is necessary to the application's functionality.

Possible: A feature that we deem is beneficial to the overall structure and functionality of the application, but not necessary to provide an end product.

Future: A feature that we deem unnecessary, but would be beneficial to the user experience.

Scope

The app will run on the Android platform. The app will utilize the Android framework to develop and easy-to-use user interface that facilitates meeting new people spontaneously. The app will use GPS to determine the location of the user and will access the user's contact list to make it easy to include their friends in meetups as well as make it easy to save the contact information of people a user meets.

The project will require a backend server to store user information and to connect users to each other. The server will have to use user feedback from previous meetups to determine which pair of users would mostly produce a meetup that goes well.

Goals

The final goal of the application is to have everything mentioned in this requirements document functioning precisely as intended, with each feature being implemented in an efficient, easy-to-use way. We hope users will be able to use our application to form lasting friendships, by finding users similar to themselves.

Our goals with the project are, in descending order of priority, having profiles implemented and working with ability to view another profile, matching users with profiles, giving the user the ability to manage their profile after creation, implementing meetups (both the meetup itself and the ability to provide feedback as a result of the meetup), having a meetup history users can view, implementing contact list integration, allowing a user to invite contacts to meetups, implementing GPS integration to allow users to more easily find each other after matching, implementing a block list, and finally, optimization. Optimization will ideally be performed throughout the course of the application's development, rather than at the end of the development cycle.

One stretch goal is to have a working machine-learning based algorithm for matching users in a more effective way than simply comparing similarity of tags. Since time is limited, it is likely there will not be enough time to fully teach the algorithm, the goal is to have at the very least a

framework in place that learns behind the scenes and uses user feedback from matching to train itself. The implementation of this goal is dependent on time and difficulty of the implementation itself. Another is the implementation of sponsored locations, which are currently on the roadmap but not a priority feature.

II. PROJECT DESCRIPTION

Profile Creation

Once the app has been downloaded, the user is prompted to create an account or log into an existing account on the initial screen.

Once the user has chosen their name, username, email, and phone number (which will most likely be the device they are currently on), the account will be created and the user will be logged in, assuming that the email and username are both currently available.

Profile Management

The user can edit their account information like their description, profile picture, phone number, and their interests. The user can also choose to deactivate their account so that it no longer appears on friend's lists. It will be reactivated the next time the user logs in.

By default, a user's "friend's list" will be their phone contacts. There will be a separate block list available for people to block users.

Meetup Creation

A possible feature, this would involve a user being able to create a meetup without ever searching for one, by inviting friends to a meetup beforehand. This would form a meetup that would then be able to operate as any other meetup, but skips the matching step.

Meetup Search

As a possible feature, users can set their status indicating why they would like to meet up with someone or what they want to talk about before searching for people to meet.

Users and existing meetups can browse for other users and/or currently active meetups who are also looking for people to join. A user or meetup will receive a list of other users and/or currently

active meetups who are also looking for people to join. From there the user can view the profiles of the those users and view the current status of those users. A user can toggle all in case they are willing to match with anyone. The user can also toggle which users the user is willing to join in a meetup. This toggle occurs when a user swipes right on a profile they would like to match with.

Once a combination of two user/meetups both toggle that they would like to join the other user/meetup, the two both receive a notification that the other is willing to join them and asking them if they would like to stop looking for matches to meet up with the other user/meetup. If both sides agree to meet up, they are removed from the search pool and form one combined meetup.

Meetup Management

After creating a meetup, a user can send invites to other users in their friends list to join the meetup.

When in a meetup, users can send messages to other users in their current meetup. Users in a meetup can also search for other users/meetups to join. A user can also choose to leave their current meetup.

Meetup History

The user can view a list of their previous meetups. The user can view the profiles of each user in that participated in meetup. A potential feature involves the ability to provide feedback on the meetups or the individual people to aid in finding desireable people from that user to meet.

View Profile

The user can view the profiles of other users. This allows a user to screen potential matches before toggling if they are willing to meet up with that person. The profile will contain information including a profile picture, the user's name, the description of the user, and their interests.

A user can also choose to add the user of the profile to their block list to prevent meeting up with this user again. Additionally, the user can toggle a box that states whether it is okay to share phone number information with the user or not. If both users agree to share phone number information, the phone number information is displayed on the profile.

Settings Management

The settings menu allows the user to perform several miscellaneous functions. The user can deactivate, but not delete, their account. The account would be reactivated if the user attempts to access the app again. The user can also contact the developers through the settings menu.

Developer Management

The developer screen will be inaccessible to normal users. It will allow users with administrative privilege to view feedback sent to developers by users in the settings menu.

III. REQUIREMENTS

Functional Requirements

Profile Creation and Management

- A user needs to be able to create a profile - **Required**
- A user needs to be able to manage and modify his or her profile, including preferences, tags, photos, and interests - **Required**
- A user needs to be able to suspend his or her account - **Possible**
- A user needs to be able to disassociate his or her phone number from his or her account - **Possible**
- A user needs to be able to manage a Friends List - **Required**
- A user needs to be able to manage a Block List - **Possible**

Group Search and Management

- A user needs to be able to search for groups - **Required**
- A user needs to be able to cancel a search - **Required**
- The application needs a way to grab the user's GPS location - **Required**
- A user needs to be able to select to utilize his or her current location, or a nearby sponsored location - **Future**
- A user needs to be able to match groups with a high degree of success - **Required**
- A user needs an option to provide group feedback upon leaving a group - **Future**
- A user needs an option to share contact info **- Possible**
- A user needs a way to view group history **- Possible**
- The application needs a method to track a user's group history - **Possible**
- A user needs a way to invite friends to an already existing meetup - **Possible**
- A user needs a way to invite friends to start a meetup - **Future**
- A user needs a way to promote the matching of certain interests - **Future**
- A user needs a way to edit his or her status, to provide additional information at the group searching screen - **Future**
- A meetup needs the ability to proactively search for new users, rather than simply taking requests as they come - **Possible**

Settings and Miscellaneous

- A user needs a method of contacting the developers to provide feedback - **Required**
- A user needs a way to report other users - **Possible**

- The application needs fake reporting options, such as “I didn’t like this user”, to reduce abuse of the report function - **Possible**
- The application needs to appropriately filter reports to prevent overloading developers with reports - **Possible**
- The application needs to properly punish offending users - **Possible**
- The application needs a database that can hold information about users and meetups - **Required**
- The application needs a database local to a phone that stores information about past meetups - **Required**
- A user needs a way to change his or her email address or phone number - **Possible**

Developer Management

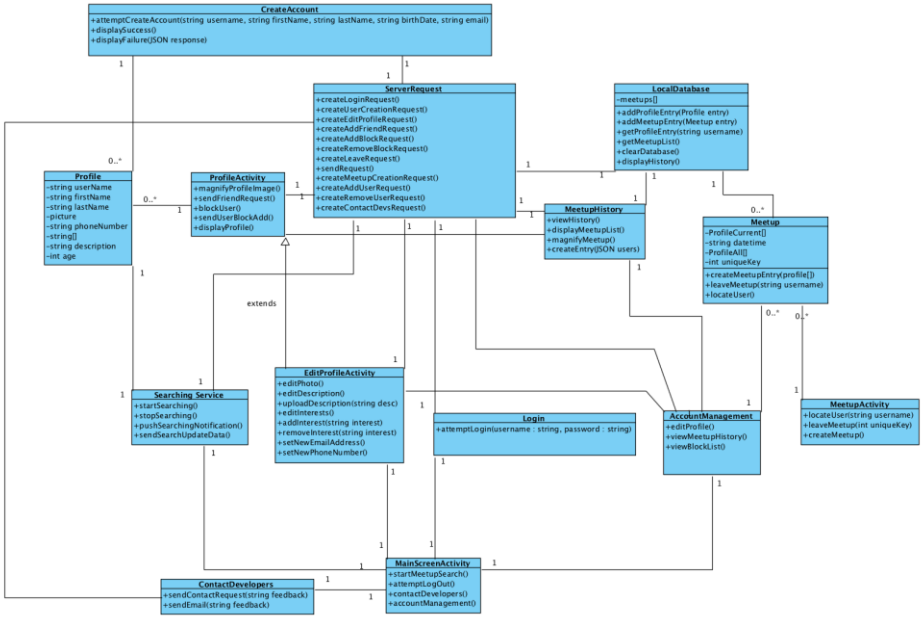
- The developers need a way to view report feedback - **Required**

Non-Functional Requirements

- The application must remain functional on older/lower-end Android phones - **Possible**
- The application must not unnecessarily drain battery - **Possible**
- The application must perform tasks (such as group searching, matching) in a reasonable amount of time - **Possible**
- Meetups and searching should have timeouts - **Required**
- Friends should be displayed above all other users during matching process - **Required**
- Log-in links should only remain active for a limited time - **Required**
- Log-in links should be device-locked - **Required**
- Log-in emails should contain device identifying information - **Required**
- Interaction with the server should only occur when necessary - **Possible**
- The application’s implementation of meetup history should only show users that the user was in a group with at some point in the meetup’s existence - **Possible**

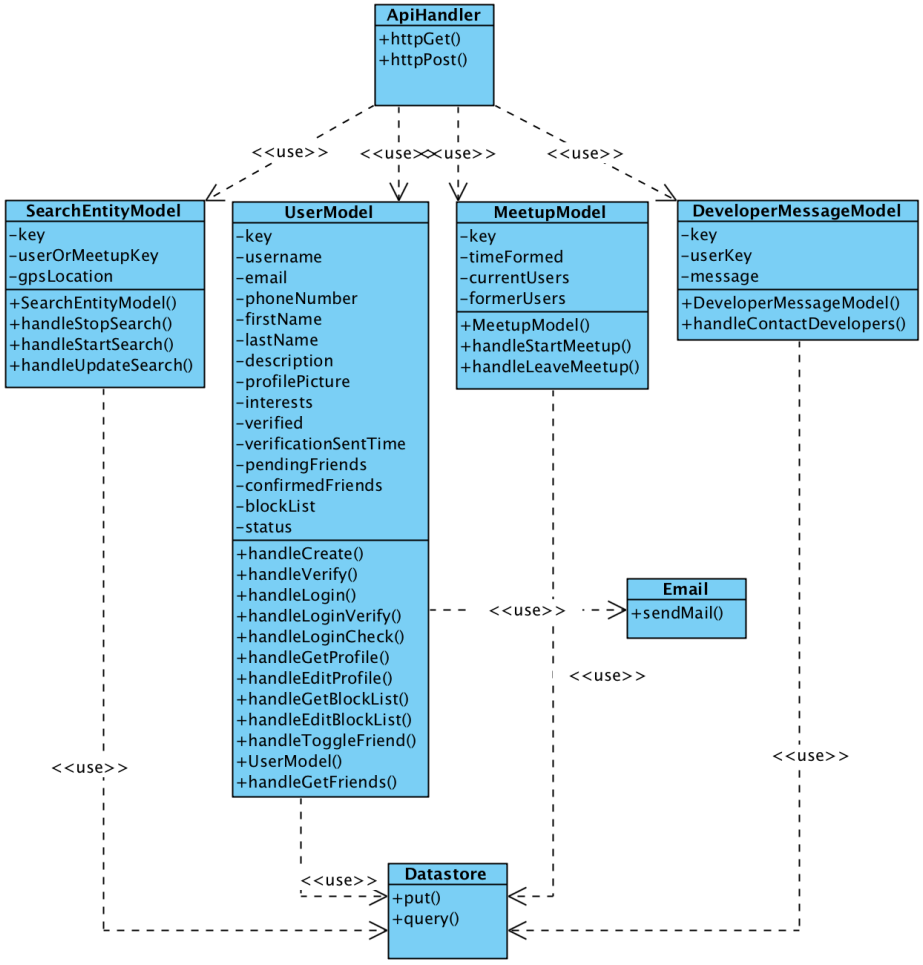
IV. CLASS DIAGRAM

App Class Diagram



The diagram shows the interaction, application-side, of our classes, with methods and attributes. The bridge between the two diagrams is the ServerRequest class, which provides methods for the various classes to communicate with the server. The CreateAccount class will contain the methods for a user to create an account, and will utilize ServerRequest to place a valid account on the server. Profile contains the information about a user saved in their profiles. Each user will have a single profile, which is linked to, but separate from, accounts. When viewing another user's profile, a user will be able to magnify the image, send friend requests, and block the user. This will utilize server requests to ensure the database is properly updated. When viewing a user's own profile, they will be provided with the options outlined in the EditProfileActivity class. The Searching Service runs in the background when a user begins searching. It interacts with ServerRequest by making a request every set amount of time (currently 30 seconds), and is responsible for keeping the user provided with up to date information about who is searching, as well as pushing information to the server about users who are currently searching and who have stopped searching. The ContactDevelopers class communicates with the Server to provide feedback to the developers. It places strings in a database that the developers can look at whenever they choose. MeetupHistory communicates

with the LocalDatabase and the ServerRequest class to place information about a user's previous meetups into a database local to the user. It uses ServerRequest to pull information about a user's profile when a user is looking at locally stored meetups. A meetup has information about the user's inside of it, and is placed on the server along with some other information about the meetup using the ServerRequest class. The AccountManagement class provides gateways to edit your profile, as well as to edit your blocklist and view your meetup history. As mentioned, the majority of these classes rely heavily on server communication, so it is essential that we design the ServerRequest class in a way that makes it easy to communicate with the database.

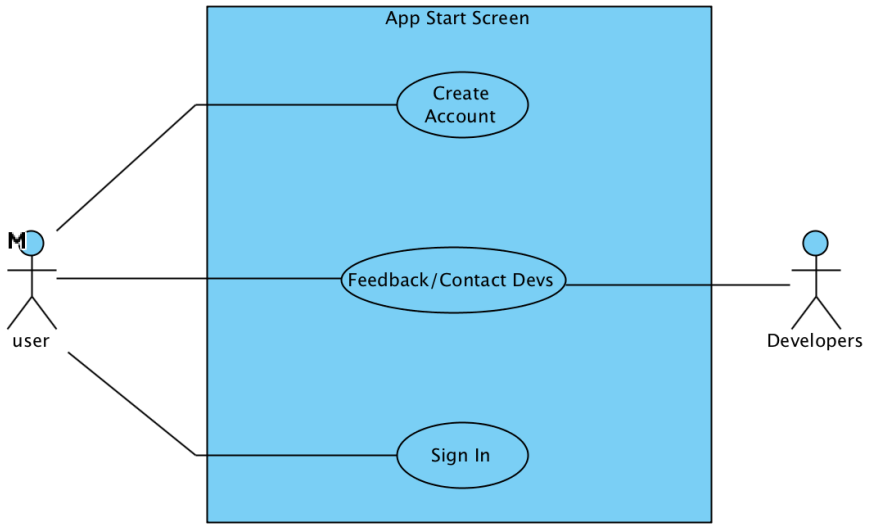


This diagram shows the core components of the backend server of the app. The ApiHandler class listens to requests from the app over HTTP GET and POST connections. Depending on the directory of the website, the handler will call different methods that will handle the request and return a JSON response to the app. The UserModel represents an account of a user. This class handles account creation, logging in, profile editing, profile retrieval, block list editing, sharing of phone numbers, and retrieval or friends. This class communicates to the email class when it sends verification emails to users to confirm their credentials. The MeetupModel represents a meetup of users. It stores a record of current users and former users and handles users starting and leaving meetups. The SearchEntityModel class represents a user or meetup

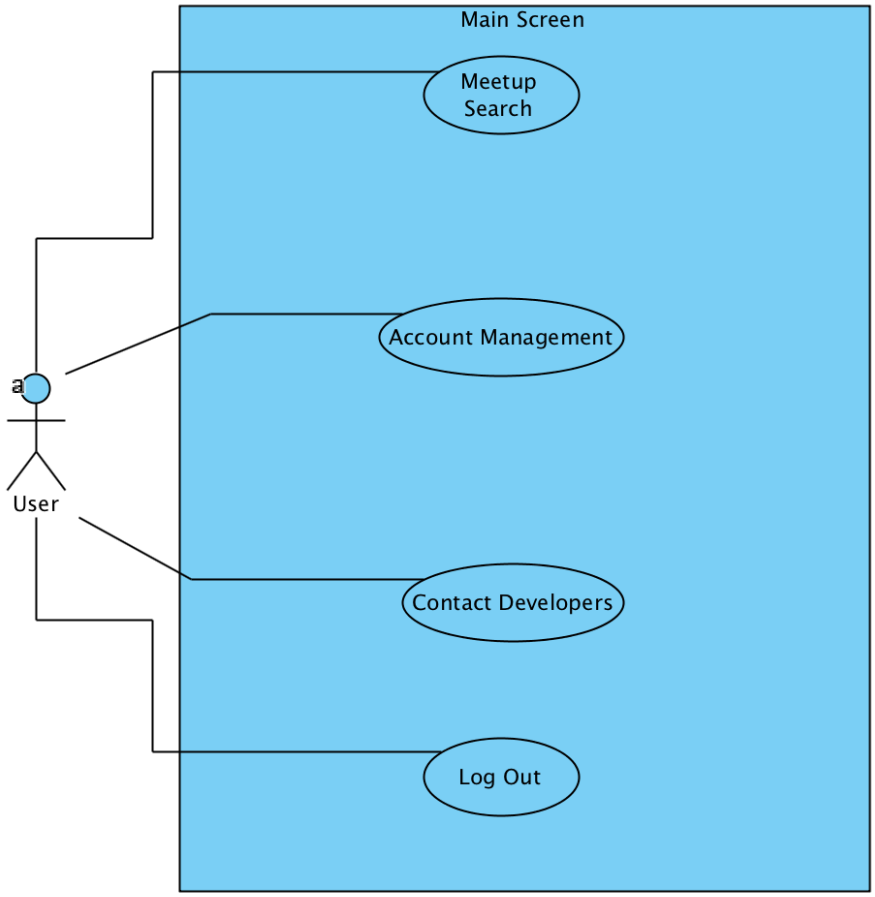
that is currently looking for other users or meetups to match with. All SearchEntityModels that are currently in the datastore are meetups and users currently searching. This class handles starting and stopping of searching as well as handling sending updates to the app regarding matches. All the models communicate with the database which can be query, inserted into, and removed from.

V. USE CASE DIAGRAMS

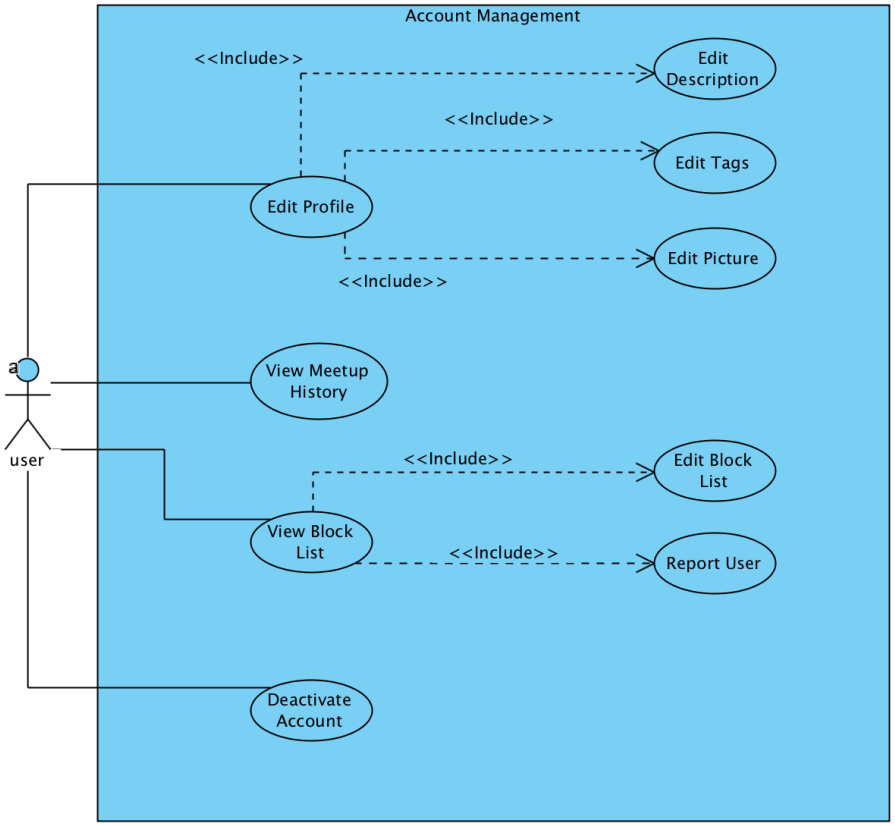
Log-In Use Case Diagram



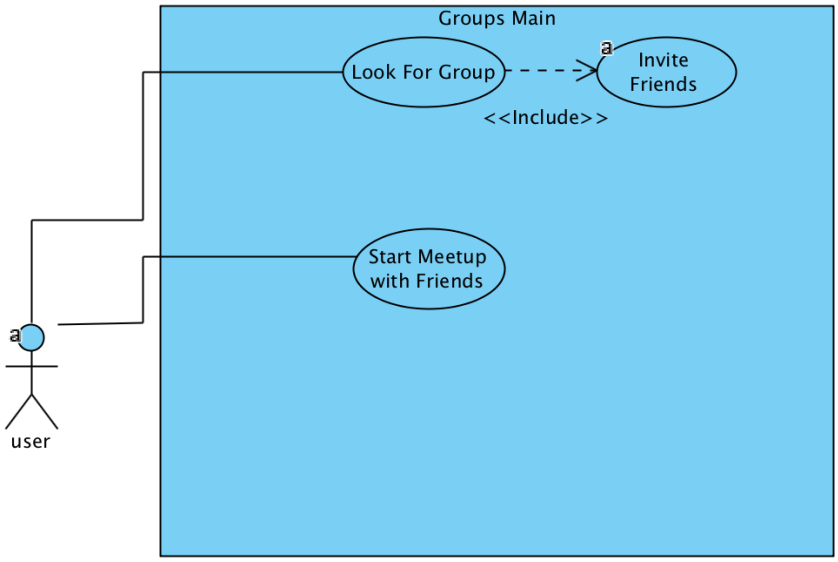
The Log-In Use Case encompasses activities a user can perform before logging into the application or creating an account. These are limited in scope to creating accounts and contacting the developers with feedback, as well as signing in. Recovering accounts will be unnecessary with our method of logging in, and reactivating accounts, if implemented, will occur automatically upon sign in.



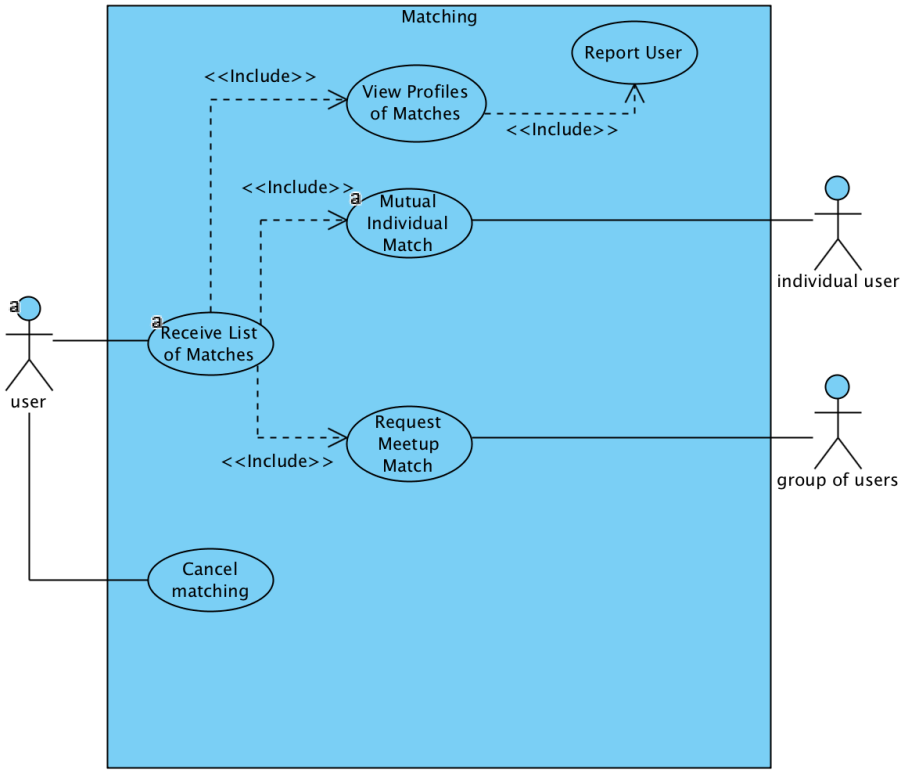
The **Main Screen** use case outlines what a user will be doing when on the primary screen of the application. These cases include searching for a meetup, managing one's account, contacting the developers with feedback, and logging out of the application. If time permits, the ability to edit a user's status will be available in this use case as well.



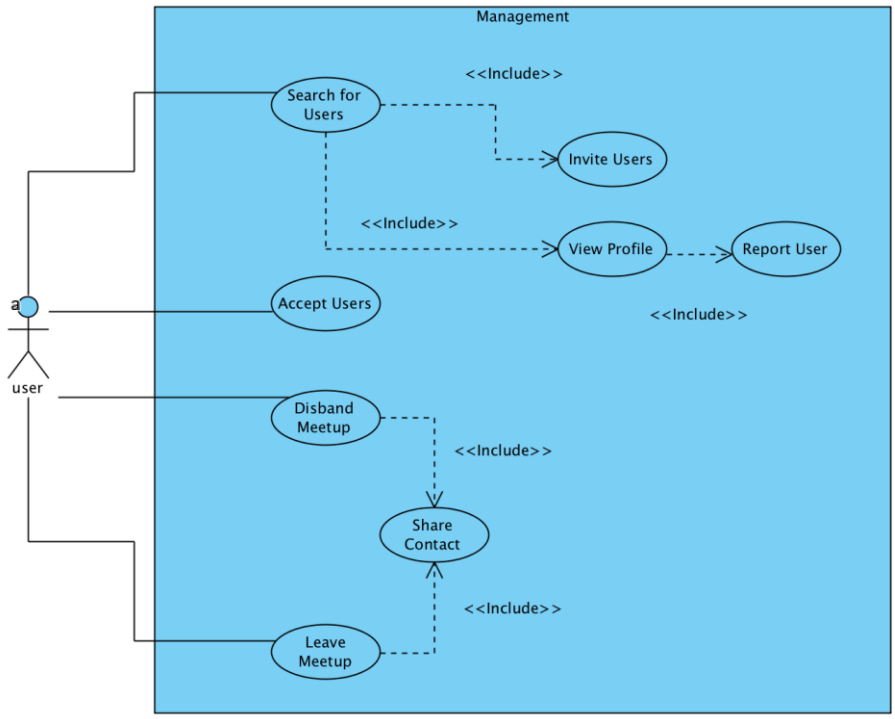
The **Account Management** use case encompasses what a user can do when managing his or her account. Options include editing one's profile, editing one's block list, viewing one's meetup history, and deactivating one's account. Reactivation occurs when a user logs back into their account, and deactivated accounts are not visible in a friend's list.



The **Meetups Main** use case encompasses the group/meetup related activities that a user can perform before or after searching for a meetup, without actually requiring the user to search for a group or be a part of a group. These are spread across different screens, but encompass the same general idea. Starting a Meetup with Friends is in this Use Case diagram, but is only a possible feature, not a required one.



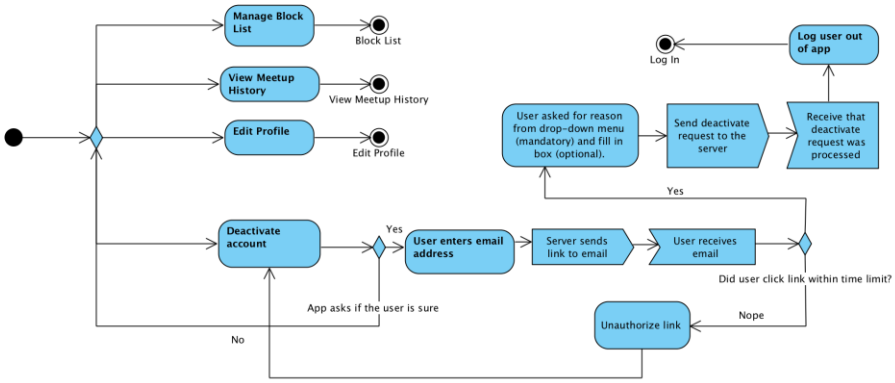
The **Meetup Matching** use case encompasses the activities a user can do during a search for a meetup. These are limited to cancelling the searching/matching service, viewing profile of potential matches, as well as toggling willingness to match with a user or meetup.



The **In A Meetup** Use Case encompasses activities a user can perform whilst in a group. These are limited to accepting new users, viewing users currently in one's group, locating these users, leaving meetups, and inviting friends to the meetup. We hope to implement the ability for meetups to search for individual users or other meetups, but this is currently a possible feature, not a required one.

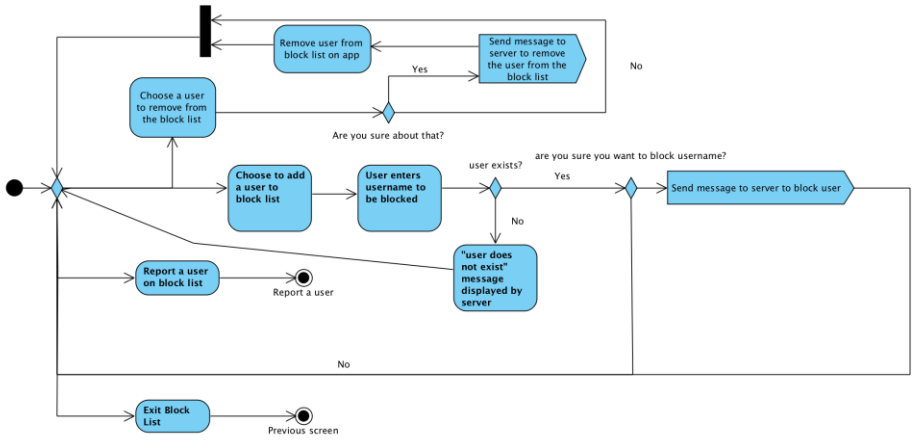
VI. ACTIVITY DIAGRAMS

Account Management Activity Diagram



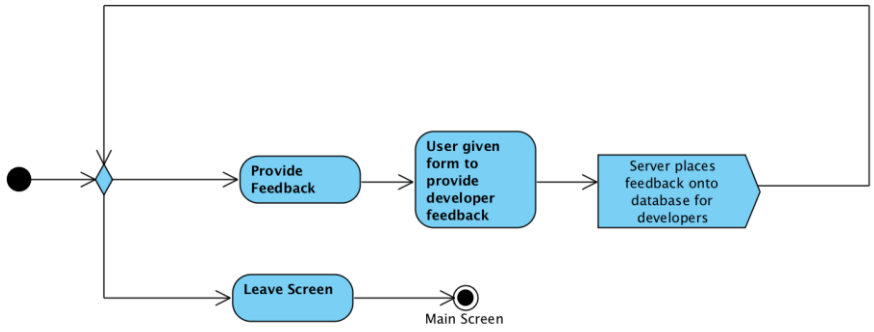
The user can choose to manage their list of blocked users, view their meetup history, edit their profile information, deactivate their account.

If the user chooses to deactivate their account, a dialog box asks for confirmation of whether the user is sure they desire to deactivate their account. The user must then type in their password and the server verifies if that password is correct. If the password is not correct, the user is prompted again to type in their password. Once the user types in their password correctly, they are prompted to optionally fill out a reason for deactivation. Next, a deactivation request is sent to the server, the server processes the deactivation request, and the app receives that the server processed the request. The user is then logged out of the app.

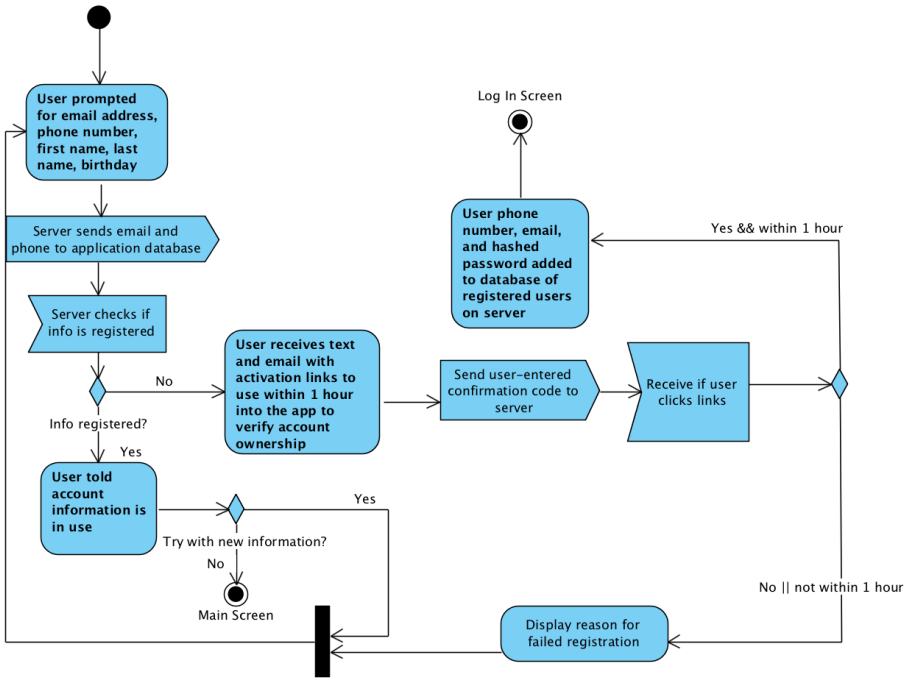


The user can choose a user to remove from their block list. A confirmation dialog then confirms if they would like to remove the user from their block list. If the user wants to remove the blocked user from the block list, the server is notified on this and the blocked user is removed from their block list. A user will also be able to report a user on his or her block list, without having to first remove the user from their block list.

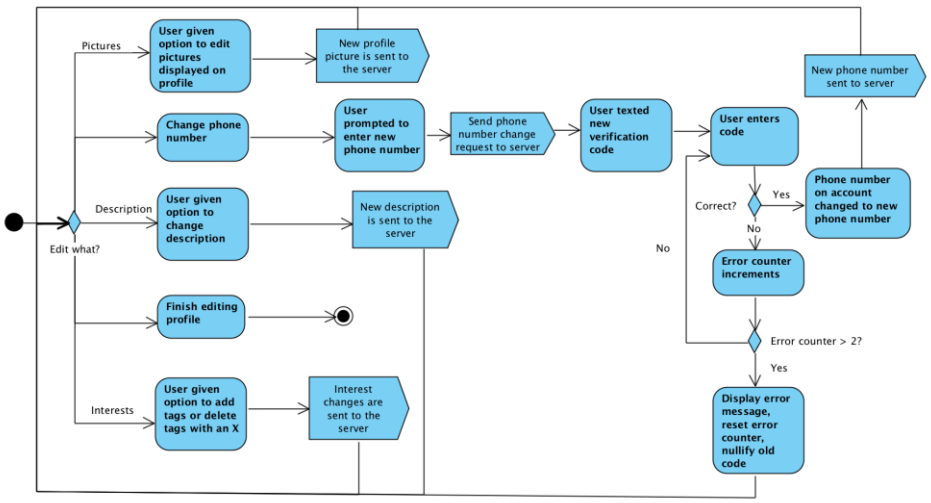
Contact Developers Activity Diagram



The user can choose to provide feedback on the app and is prompted to enter their feedback into a text form. The feedback is sent to the server when the user finishes filling out the form.

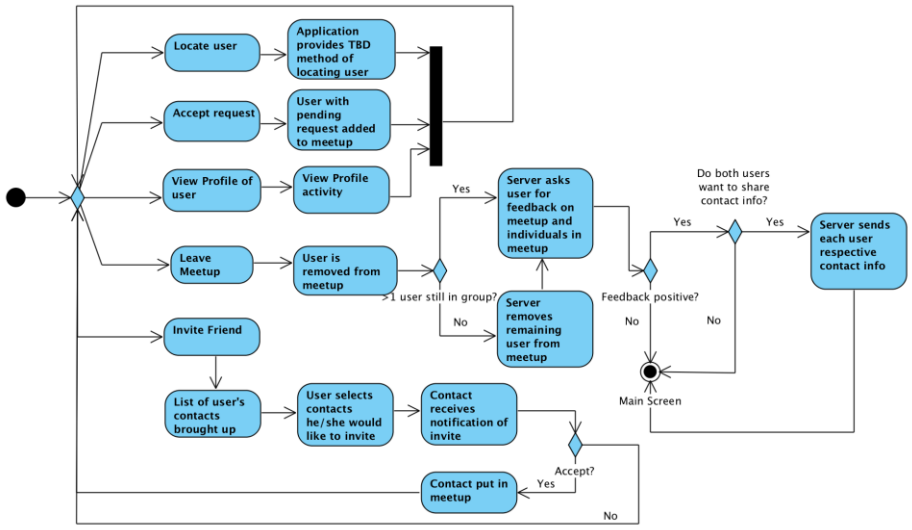


The user is prompted for their email address, password, and phone number. The server will check if the email or phone number is current associated with a user account. If either is associated with a current user account, the user is informed of this and is allowed to restart account creation or leave account creation. Otherwise, the user will receive email and phone confirmation codes that they must input into the app to verify that the email and phone number provided is their email. The confirmation codes expires in an hour. When the user types in their confirmation code, the server checks if the codes have been entered within an hour from when it was created and if the codes are correct. If so, the account is created and the user is sent to the login screen.



The user can choose which information they would like to change. For all information besides phone number, the changes can be made in the app and then the server updates the user profile. For the phone number, the user enters the desired new phone number and a phone number change request is sent to the server. The server then texts a verification code to the new number. The user has three attempts to type in the correct verification code. If the verification code is correct, the new phone number is sent to the server and the phone number is changed. Otherwise, the user fails and is taken back to edit profile.

Meetup Management Activity Diagram

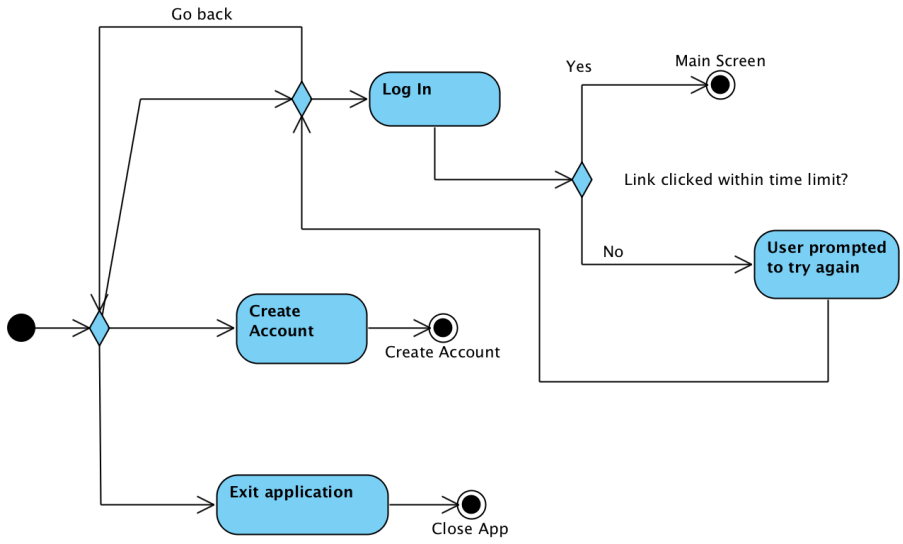


While in a meetup, the invite users in their friends lists to join the meetup, view profiles of users in the meetup, and leave the meetup. Users can also locate other users who are in the meet up. The method used to locate people has not been decided yet. Potential methods are using gps to create a compass, making the screen glow a certain color that users hold up to find each other, and allowing people to text each other through the app. The user can also search for other users to bring into the meetup in the background.

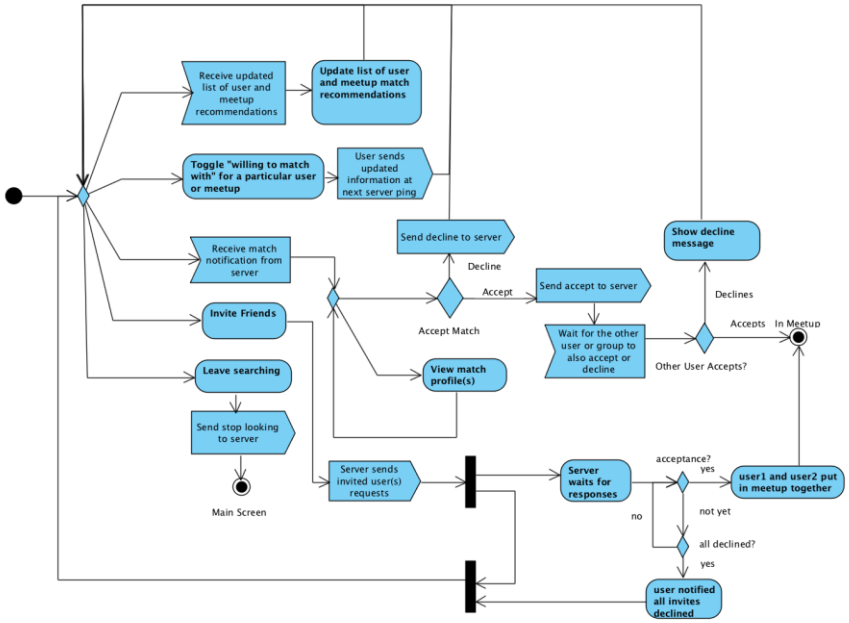
A potential feature is the ability to invite friends to the meetup. A list of the user's friends comes up, and the user selects all the people they want invited to the meetup. The server receives this list and sends invite notifications to the friends. When a friend accepts the request, they are put into the meetup.

When a user leaves a meetup, if the meetup is now empty, it is closed and finished. The user is then prompted to send feedback to the server about how the meetup went. This feedback can be used later on to help sort potential future meetups.

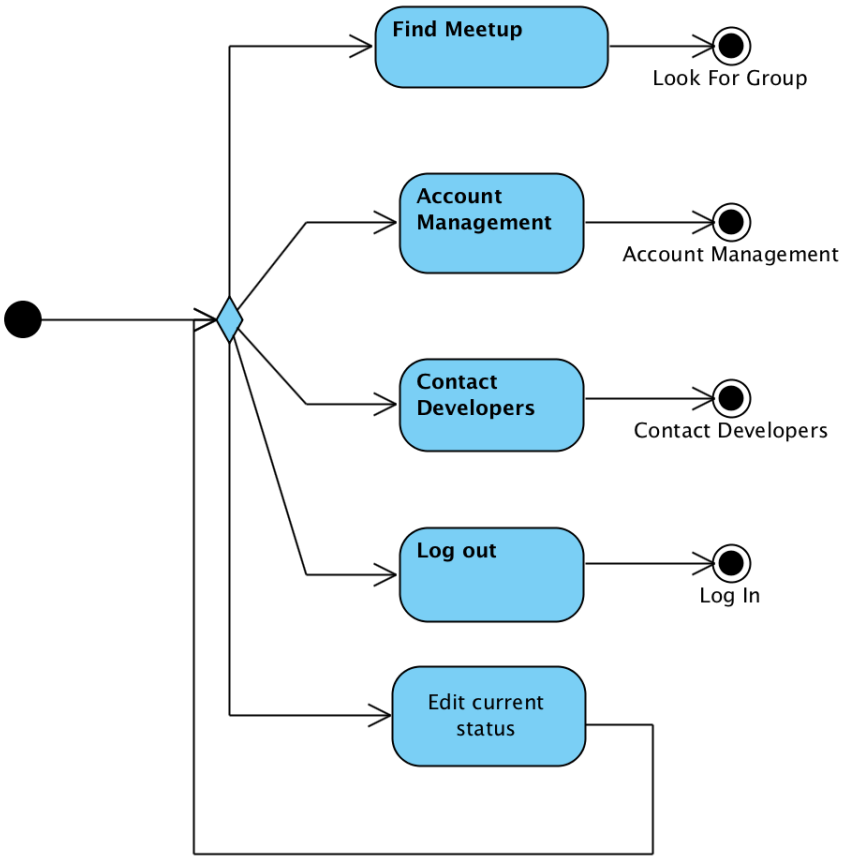
Login Activity Diagram



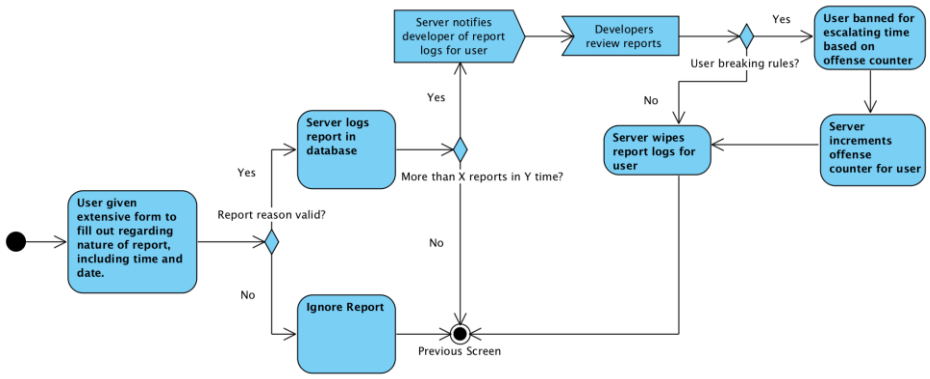
The user can choose to create their account or log into their account. If the user chooses to log into an account, the user types in their login information and the information is sent to the server. If the information is valid, the server allows the user to log in and is taken inside the app. If the information is not valid, the user is prompted to retry entering their information. Creating an account is covered in the “Create Profile” activity diagram. The user can also choose to exit the application.



When searching for a meetup, the user will continuously receive a list of user and meetup recommendations from the server. The user can toggle which users and/or meetups they are willing to join. The user will receive match notifications from the server when a different user/meetup that this user is willing to meetup with also is willing to meetup with this user. If the user wants to go finalize the match, the user confirms to meetup. If not, the searching continues. If a user/meetup does not confirm to meetup, then a decline messages is sent to the other user. The user can also choose to stop searching so they will no longer match with other users. A potential feature is the ability for users to invite friends during this stage, to start their own meetup. Users will send out invites to contacts, and then go back into the normal meetup searching. When a contact accepts, the user will be pulled from wherever they are in the activity and placed into the “Meetup Management” activity, along with their invited contact.

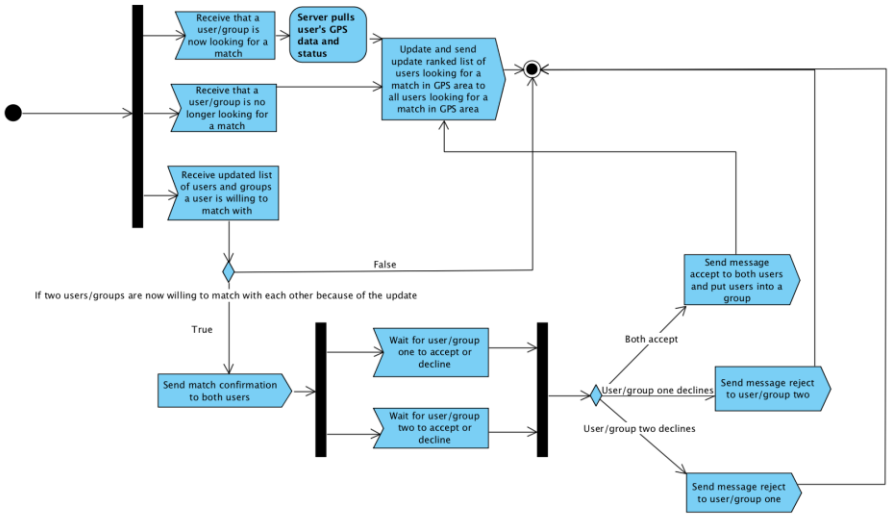


From the main screen, the user can choose to start searching for a meetup, manage their account, contact the developers, or log out. A potential feature is the ability to edit a user's status, should statuses be implemented.



A potential feature is the ability to report a user. To report a user for abuse of the app, the user selects the “report user” option from the other user’s profile. The possible options for reporting include for misconduct such as hateful speech, unwanted lewdness, or stalking. The menu will also include dummy options such as “I did not like the user” to prevent useless reports.

Server Meetup Search Activity Diagram

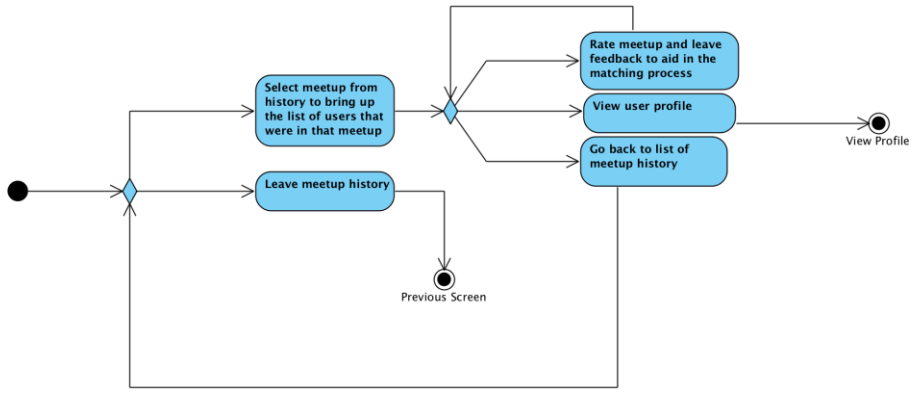


For creating matches, the server will send a list of ranked meetups that a user could match with to a user who sends “a start searching request” to the server. This will pull the user’s GPS data and status, to be used in the searching process. The server also adds that user to the list of

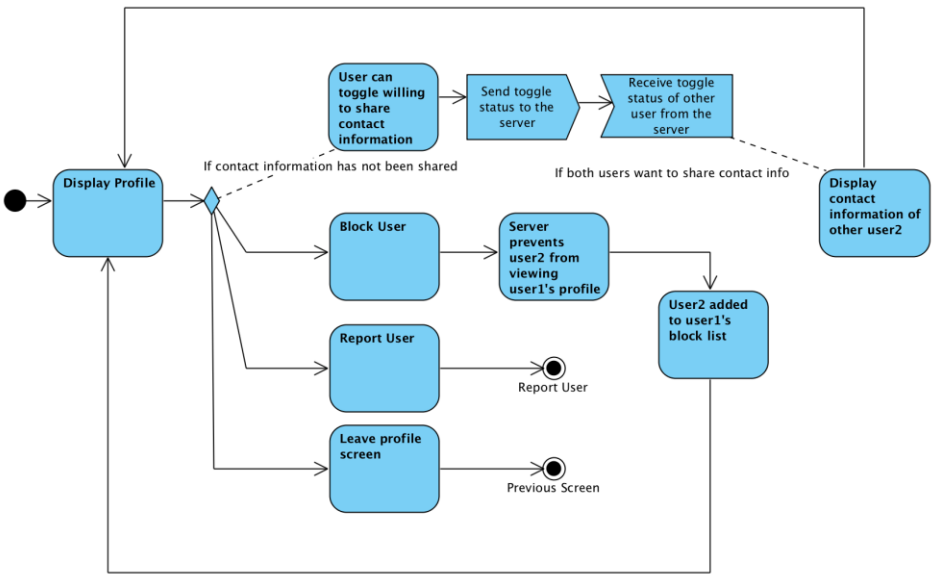
users looking for a meetup. If a user cancels searching for a meetup, the user will be removed from the list of users looking for a meetup.

When users change the list of users they are willing to match with, the server checks if two users are now willing to match with each other. If so, confirmation notifications are sent to each user. The server waits for both users to accept or decline. It should also timeout if the a user does not respond. If both users/meetups accept, then they are added to one meetup and are removed from searching for a meetup. If a user declines, the other user is sent a decline notification.

View Meetup History Activity Diagram



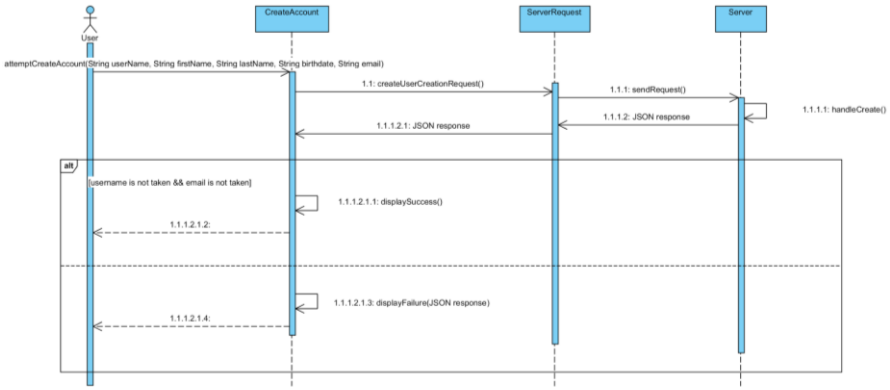
The app displays a list of previous meetups and the user can choose a meetup to further inspect. From there, the user can view the profile of anyone user who participated in the group. Ideally, we will provide a feature to provide feedback on the group that will assist in our matching algorithm. If we cannot implement the feedback in a way that will tangibly affect our algorithm, then this feedback feature will not be implemented.



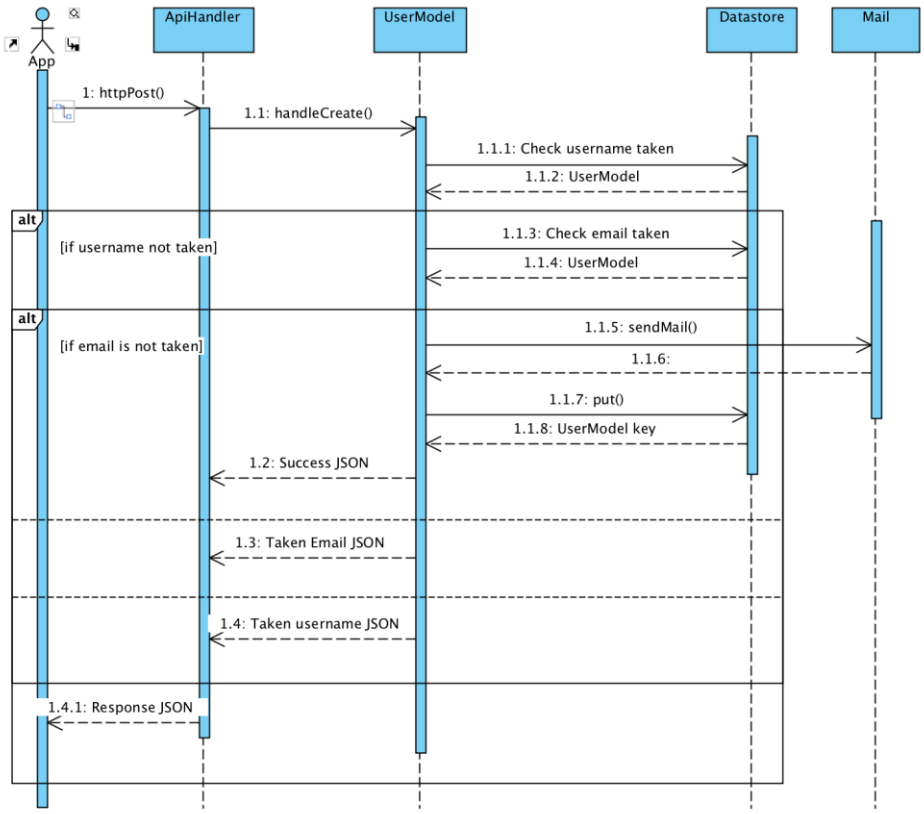
The app displays the profile of the viewed user either from a cache or from pulling the data from the server. The user from there can toggle whether the they want to share contact information with the viewed user, as well as magnify parts of the profile. If both users toggle that they desire to share contact information with each other, then contact information is displayed instead of the share toggle. A potential feature is the implementation of reporting and blocking of users directly into this screen. Blocking would be implemented before reporting.

VII. Sequence Diagrams

Create User Diagram (App)

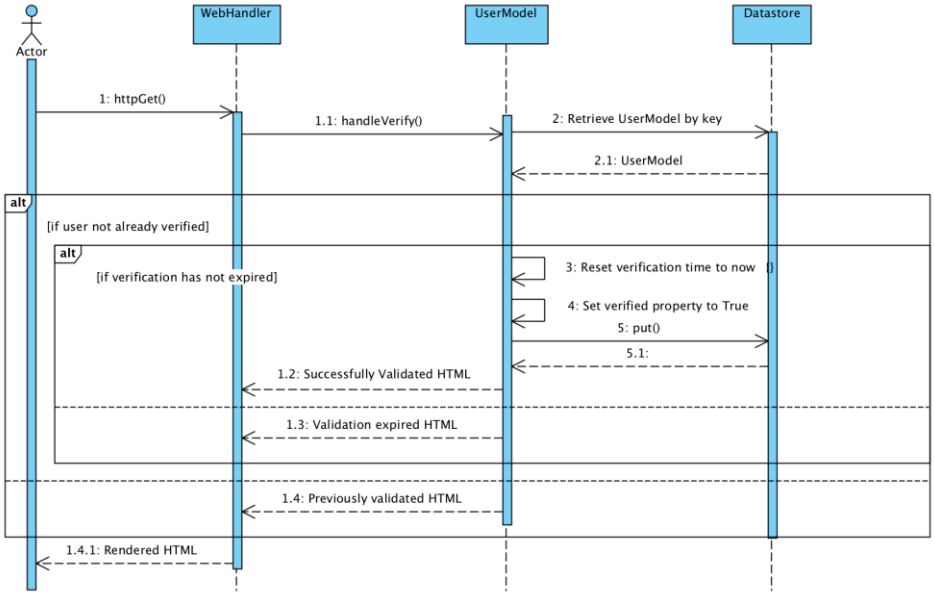


In this sequence, a user will attempt to create his or her profile. They will pass a username, first name, last name, birthdate, and email into the server. The server responds with a JSON containing data about the success or failure of the creation of the account. Should it succeed, the user will be notified, and they will be able to log in with this information provided. Should it fail, either due to the username or email being taken, or potentially both, the user will be notified of this as well. They will need to reattempt the sequence if the creation fails.



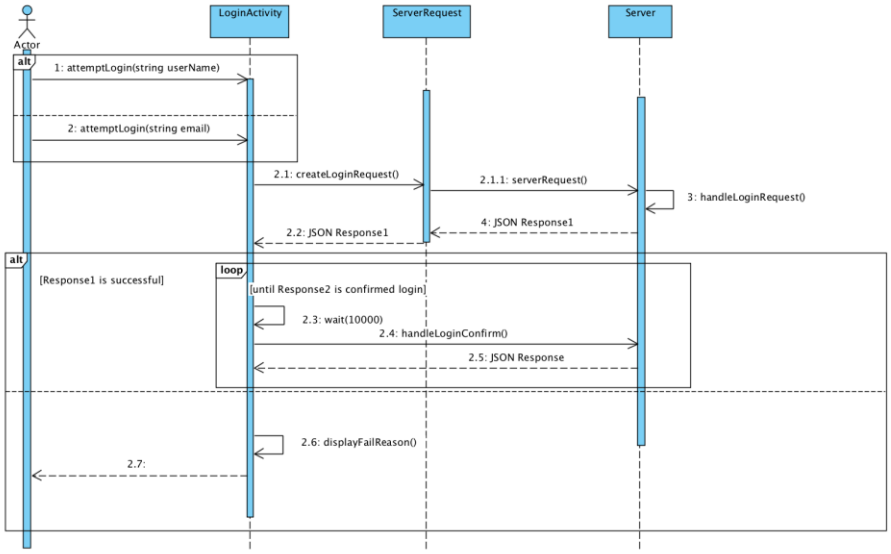
When the user sends an account creation request to server, the server checks if the supplied username or password is already in use by another account that is existing and validated. If the username and email are not taken, the server sends an email to the supplied email address. This email contains a link that the user must click to validate their email address. The server then sends back to the app the appropriate response message depending on whether the username was taken, the email was taken, or if account creation succeeded.

Handle Verify User Diagram (Server)



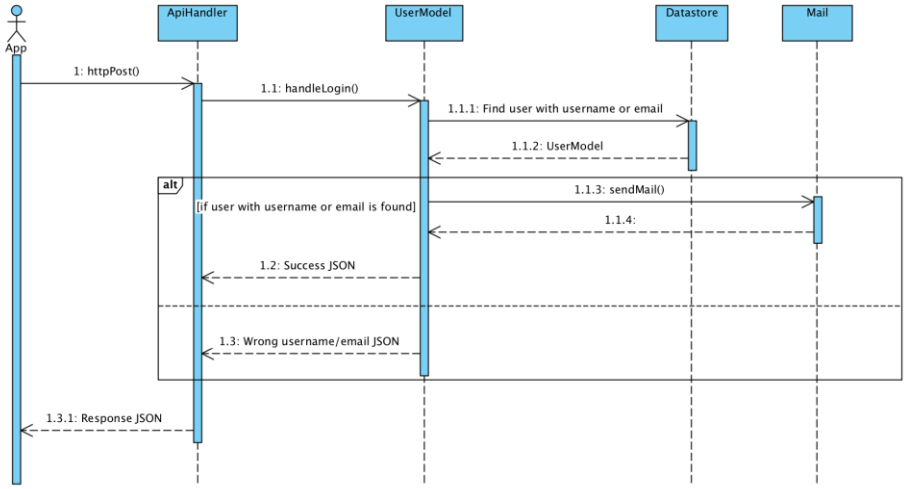
When the user clicks the link to verify their email address, the server will receive a request to verify the account the link belongs to. Using the user key provided in the link, the server will retrieve the user model in the database corresponding to the key. If the user is already verified, the HTML the server responds with indicates this. Otherwise, the server checks if the verification link has expired. If it has expired, the HTML the server responds with indicates this. If the link has not expired, the verified property of the user is set to true. The time when the verification was sent is reset to the current time as well. This will allow the user to login immediately (see Handle Login Check Diagram (Server)). The HTML the server responds with will indicate a successful email validation.

Login User Diagram (App)



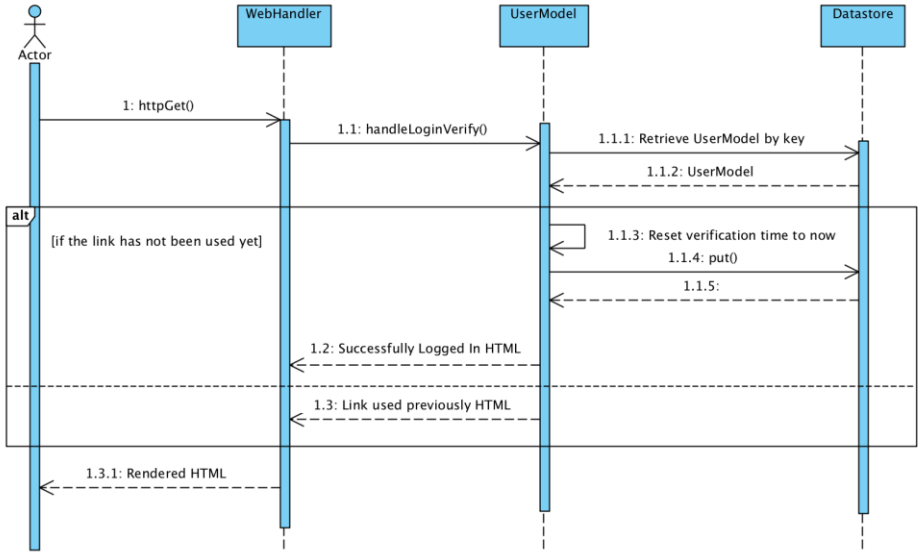
This sequence diagram involves a user attempting to log-in to the application. The user will first invoke the attemptLogin() method, and will pass a username or an email address. The server will then check to see if that username or email address exists in the server. The server will respond with a JSON containing data about the success or failure of the login attempt. If this attempt succeeds, the app will request to see if the link has been clicked every, at the moment, 10 seconds, although this time window could be subject to change. Once the link has been clicked, the user will be allowed to log in. Should the JSON contain failure data, the reason for the failure will be displayed, whether it be the username or email being non-existent. Additionally, should the user's login link expire, the user will be notified of that upon clicking the link, and will need to re-attempt the entire login process to acquire a new link.

This method of logging in may seem inefficient, but it was chosen due to the infrequency of having to log into the application. Users may stay logged in for months, or even years, without having to re-enter their passwords. This creates a very real possibility of the user having to recover their account, should the infrequently used password be forgotten. This method of logging in does present some challenges. It is possible that a user who does not own an account to attempt a login at the same time as a real user. In this case, the user would receive two emails, with different verification links. Since these links are device bound, clicking the wrong link could result in an unauthorized party gaining access to your account. To minimize this, we plan to include certain device data in the email, to ensure that users have access to enough information to verify their logins. Additionally, due to the infrequency of login attempts and the short lived length of the verification links, this scenario would rarely, if ever, occur.

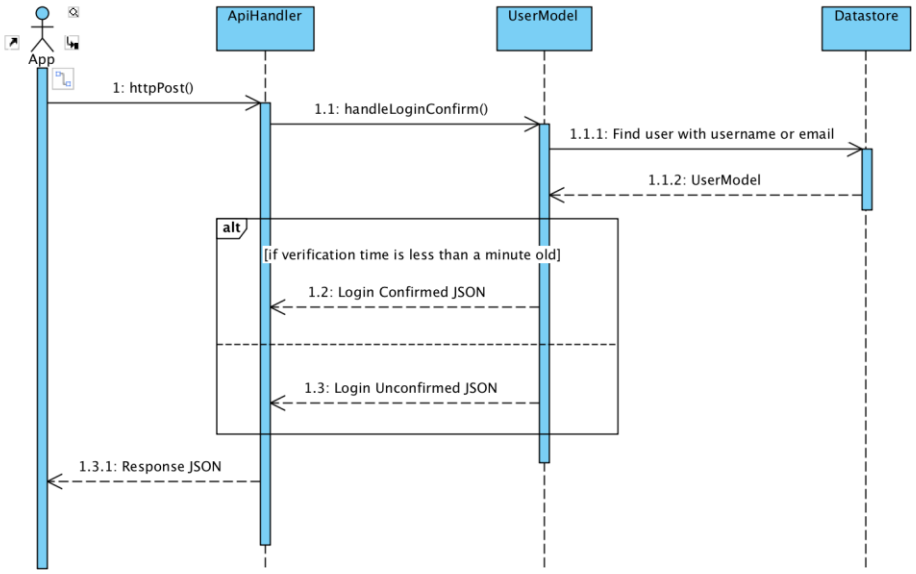


When the user wants to log in, an email is sent to their email address to confirm they want to log in using their device. The server first validates that a user exists with either the username or email address that the user has provided to log in. If an account exists, then a login email is sent to the email address of the user. Otherwise the server responds with a message indicating no user exists with the specified username or email.

Handle Verify User Login Diagram (Server)



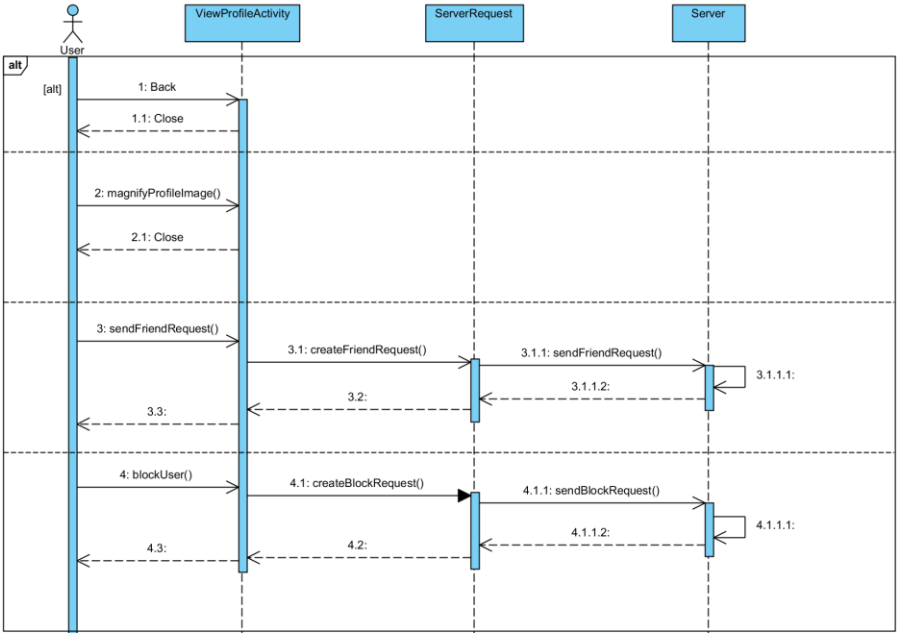
When the user clicks the link to verify their email when logging in, the link will send them to a web page that displays whether the user has been logged in or not. If the login link has not been used yet, the verification time of the user is reset to the current time. The HTML displays whether login was successful or if the link was used previously.



Commented [1]: Respond with the key of the user in the database

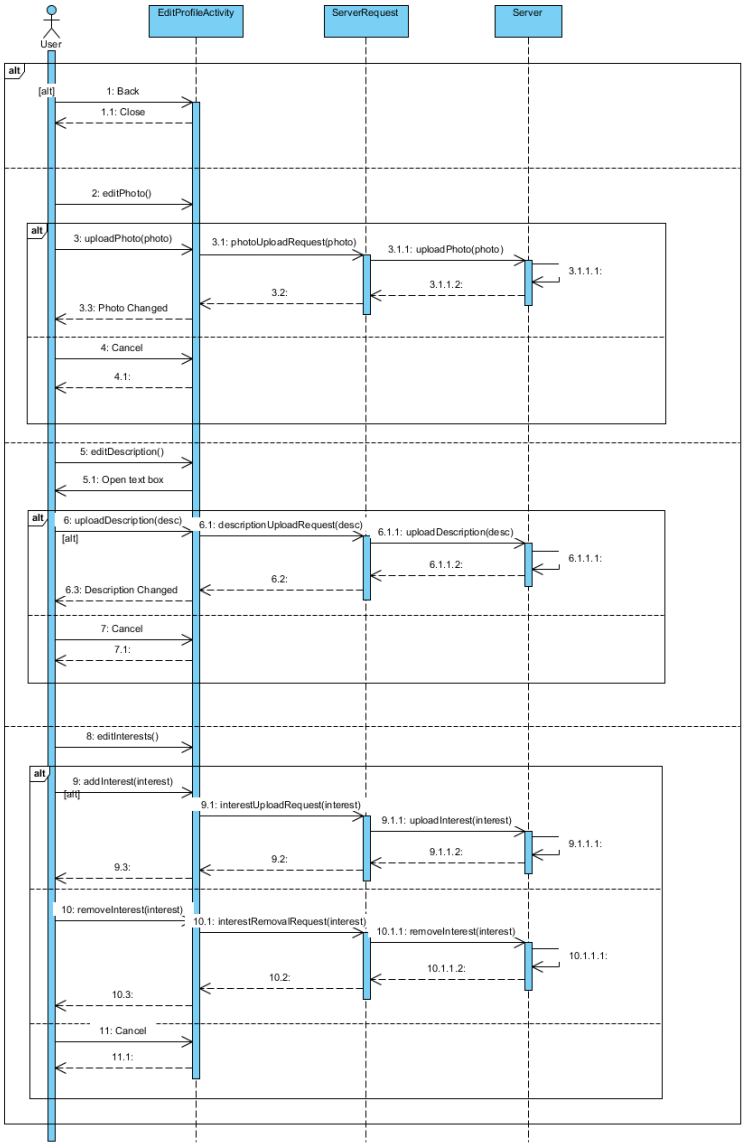
After the user has attempted to log in, the app sends periodic messages to the server checking if the email link has been clicked and the user has been logged in. The server receives this message and retrieves the user model associated with it from the database. If the clicked the link, the verification time will be recent. If the verification time is within a threshold from now (threshold set at one minute for now), the the login is confirmed and the user receives their user key; otherwise, the login cannot be confirmed.

View Profile Diagram (App)



This sequence diagram is for viewing the profile of another user. This is a fairly nonlinear sequence diagram because there is no guarantee that any action should occur before any other action. Once the user is viewing another user's profile, the viewProfileActivity class displays buttons to request the other user as a friend, block the other user, and magnify the profile image, as well as displaying the other user's description and interests with no associated interactions. The sequence diagram consists of a large alt block detailing the sequences associated with interacting with each of the buttons described above. If the user tries to request or block the other user, a ServerRequest is used to notify the server to appropriately handle the communication between users. Magnifying the profile image occurs locally and does not involve any classes besides the viewProfileActivity.

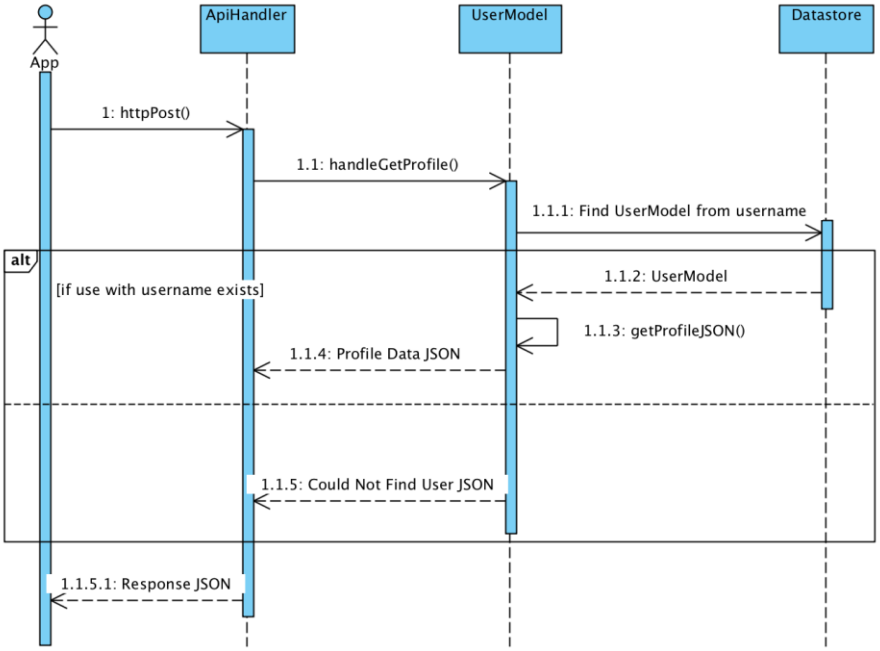
Edit Profile Diagram (App)



This sequence diagram is for editing a user's profile. The editProfileActivity class inherits from viewProfileActivity because of the similarities of the actions, although the sequence diagram is

fairly different. The editProfileActivity will display a user's profile with the option to edit the photo, description, or interests visible to other users. The sequence diagram consists of a large alt block each with nested alt blocks to represent that path from choosing one of these options to completing a change or cancelling it. Additionally, editInterest has functionality to add and remove interests, rather than editing a single object like editDescription or editPhoto.

Handle View Profile Diagram (Server)

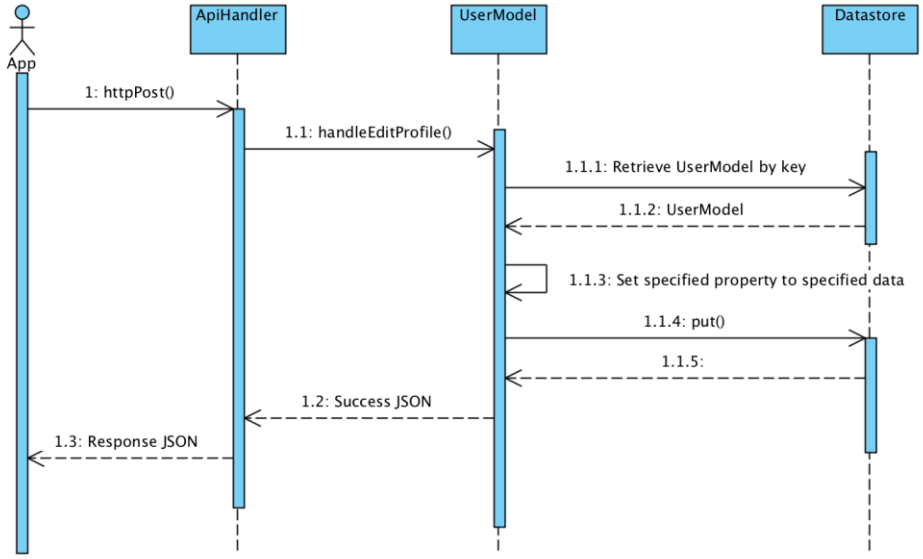


When the user issues a request to retrieve the contents of a user account, the server simply looks for the user with the specified username, and formats the data in a JSON format that can be returned to the app. If the user cannot be found, the response to the app will indicate this.

Commented [2]: change 'use' to 'user'

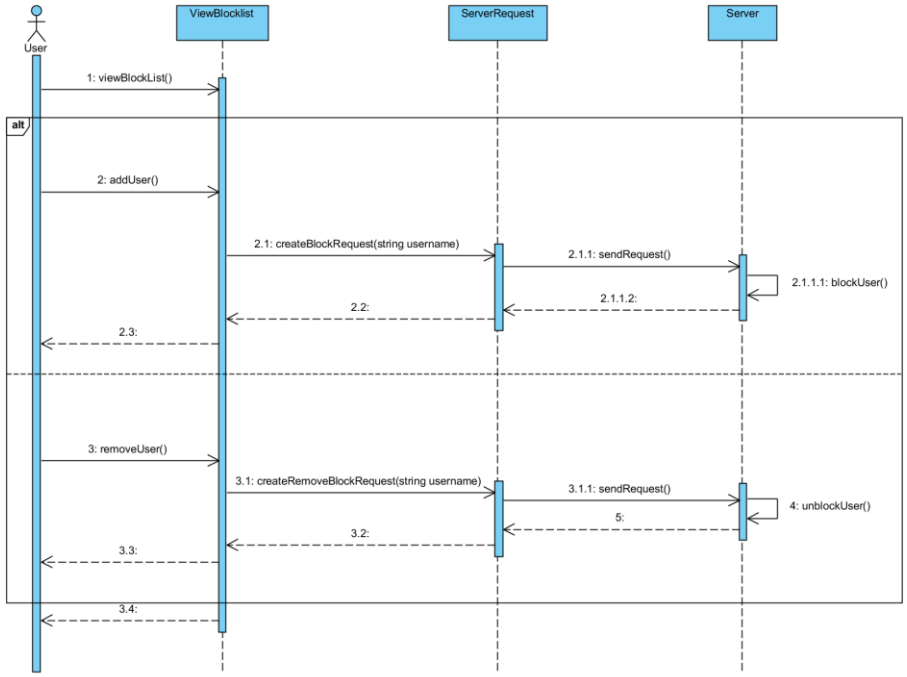
Commented [3]: Change to accept a list of usernames

Handle Edit Profile Diagram (Server)



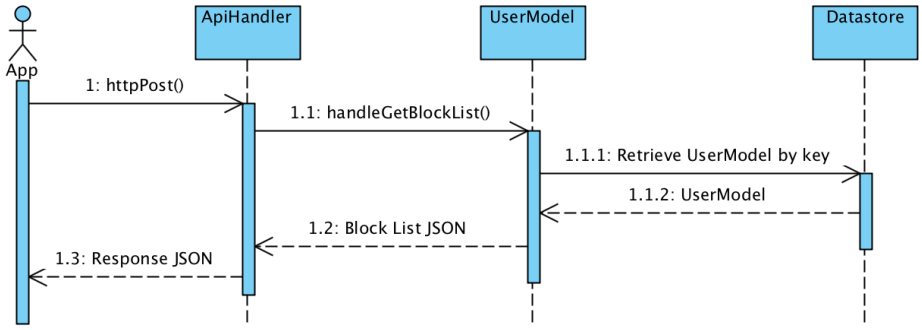
When the user edits their profile, the user issues a request to the server with the key of the user, the content to be edited, and the contents of the edit. The server retrieves the data of the user from the datastore using the key of the user. The property specified in the request is then set to the newly edited contents and the datastore is updated with this information. The server then responds to the app indicating the edit was a success.

View Blocklist Diagram (App)



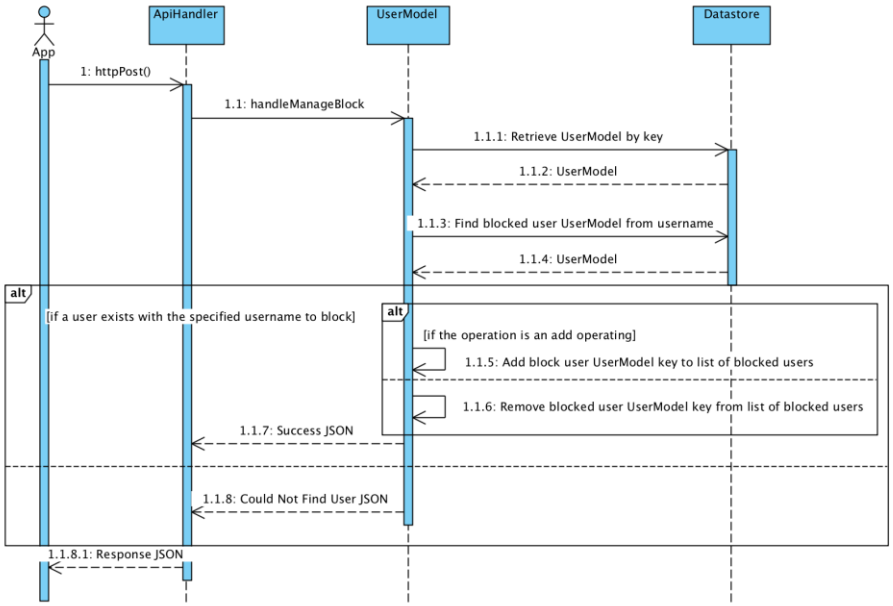
A user will invoke the viewBlockList() method in AccountManagement to start this sequence. Once viewing the list, the user will have two options, the first is to add a new user to the list, which will be placed on the server to prevent matching blocked users with their blockers. The second is to remove a user from the block list, which will result in a server request to remove that blocked user from the requester's block list. The server, as usual, handles most of the actual difficult tasks.

Handle View Blocklist Diagram (Server)



When the user wants to view their block list, the server retrieves the user's data from the database through the use of the user key provided when the user logged in. The block list containing usernames of blocked users is returned back to the app.

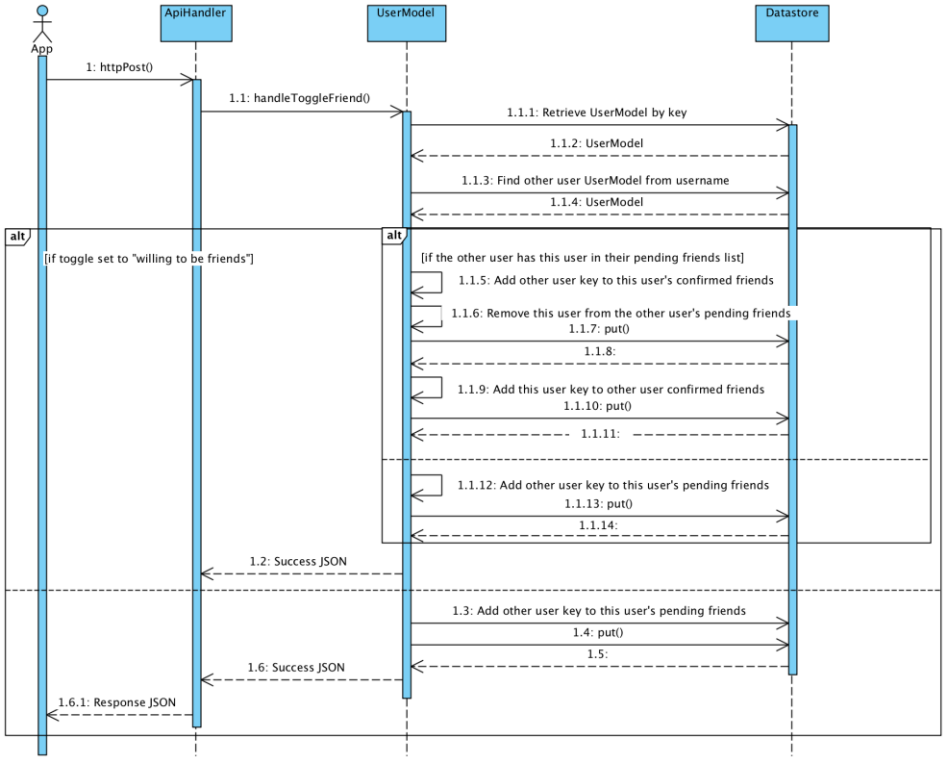
Handle Edit Blocklist Diagram (Server)



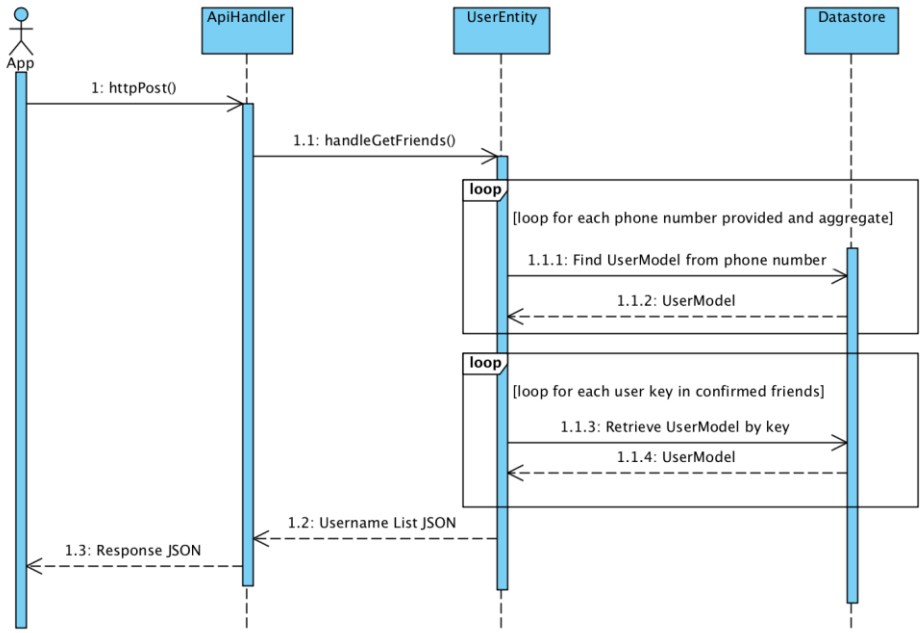
When the user edits their block list by adding or removing a user from the list, the app makes a request to the server to edit the list. The server retrieves the block list of the user by retrieving

the data of the user from the datastore using the user key provided in the request. The server also attempts to find the user of the specified username to block or unblock. If the server cannot find this user, the response to the app indicates this. Otherwise, the user is either added or removed from the list depending on which operation the request indicates.

Handle Toggle Friend Status (Server)

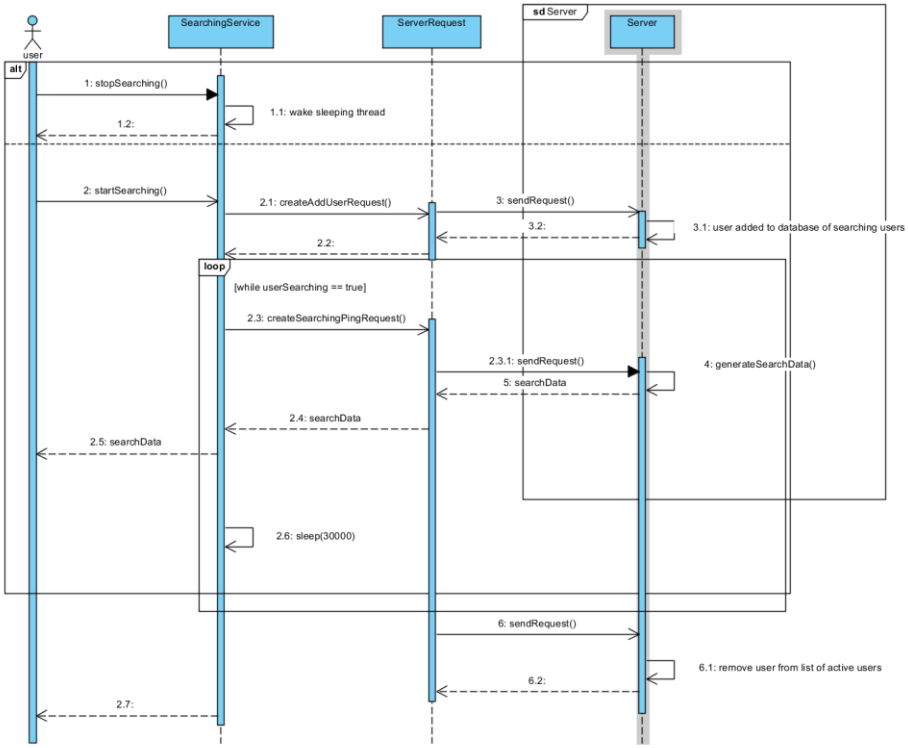


When the user toggles whether they would like to share their phone number with the another user, the app issues a request to the server detailing this. The server receives the request, the server retrieves the data associated with the current user and the other user. If the user toggled the status to “willing to be friends with” then the other user is added to the current user’s pending friends list. If the status is toggled otherwise, the other user will be removed from the pending user list. If both users have each other in their pending user list, they will remove each other from their pending users lists and add each other to their confirmed users list. Confirmed users are friends through the app and their phone number can be shared.

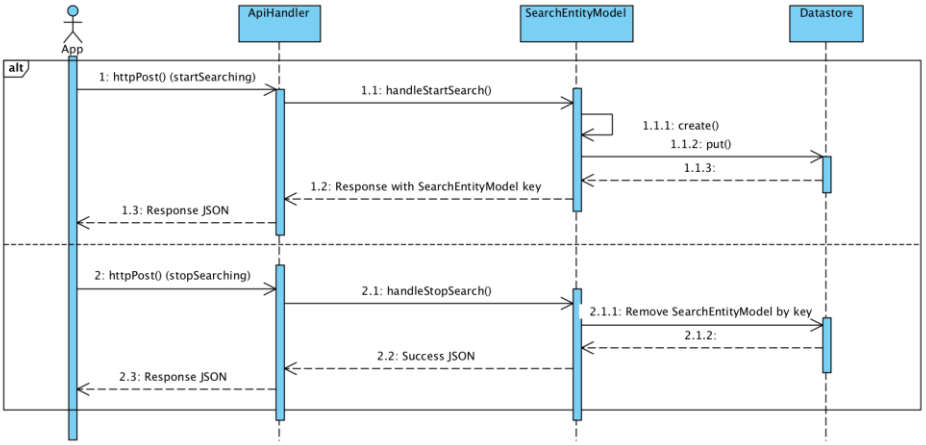


When the user requests to view their friends list, the app issues a request to the server with all the phone numbers in the user's phone. The server attempts to find a user for each phone number. For all the phone numbers associated a user account, the usernames of those profiles are aggregated into a list. The server also retrieves the usernames of all the users that the user has become friends but not yet received the friend confirmation. These usernames will later be added to the user's phone as contacts

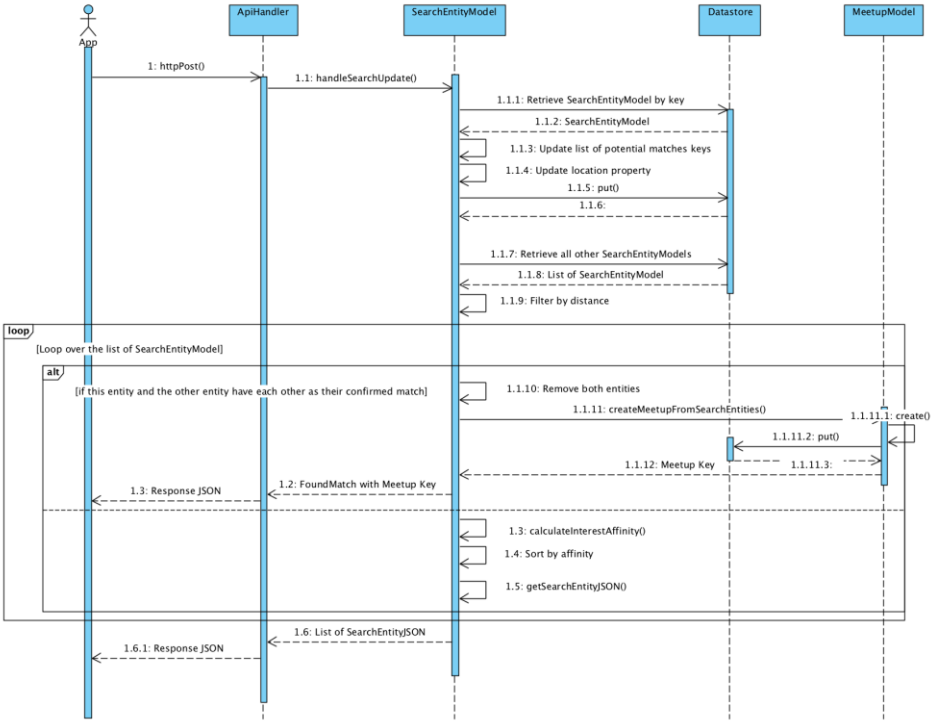
Meetup Search Start/Stop/Update (App)



This sequence diagram is relatively straightforward, but contains many paths. It has two alternate run patterns, depending on whether or not the user has requested to stop searching. For simplicities sake, the second part of the diagram will be covered first. The user initiates the search process on the main screen, and invokes the startSearching() method. A request is then made by the SearchingService to create a ping request, which is passed to the ServerRequest class. The ServerRequest class handles the creation of all requests, and it sends a request to add the user to the table of all active users. It then enters a loop that will repeat every 30 seconds unless, in which the server sends a request to the server for all active searching users within a specified radius. The server generates this data, and then replies to the request, sending back an object with all the searching users as well as their match compatibility. This thread then sleeps for 30 seconds, although it can be woken by the user invoking stopSearching(). This will set the userSearching flag to false, and send a wakeup call to the thread. When the thread attempts to loop once more, it will check the flag and see that it is false, ending it's searching attempts and quitting the search sequence. The user will additionally be removed from the list of searching users.

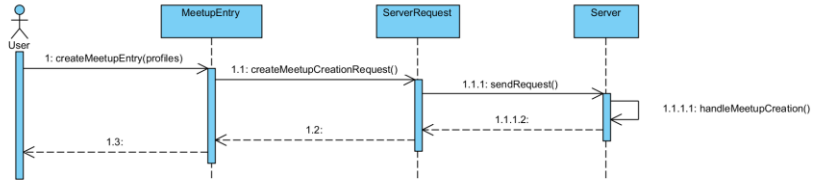


When a user desires to start searching for a person or meetup to join, the server creates a new search entity for the user and places it into the datastore. When the user wants to stop searching, the search entity that was created when they started searching is removed from the datastore.

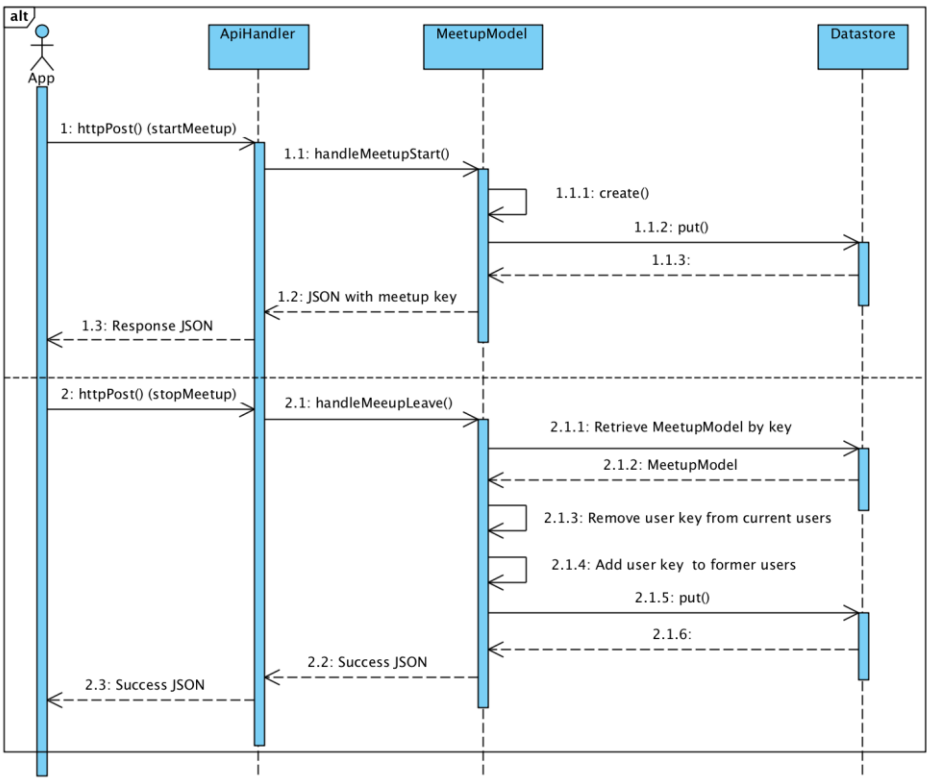


When the app is searching for a meetup or user to match with, the app constantly pings the server for matching updates. The server receives updates from the server of who the user is willing to match with. The server sets the location of the entity to the location of the user that issued the update request. It then retrieves all the other entities searching nearby, sorts them by affinity to the interests of the entity and returns the list of other searching entities. This sequence also handles if two entities indicate that they are willing to match with each other. When two users indicate this, they will both be notified that they can match with each other. If both confirm that they want to join the other, then their SearchEntityModels will be removed from the datastore and a meetup will be formed of the combination of those two entities.

Start Meetup (App)

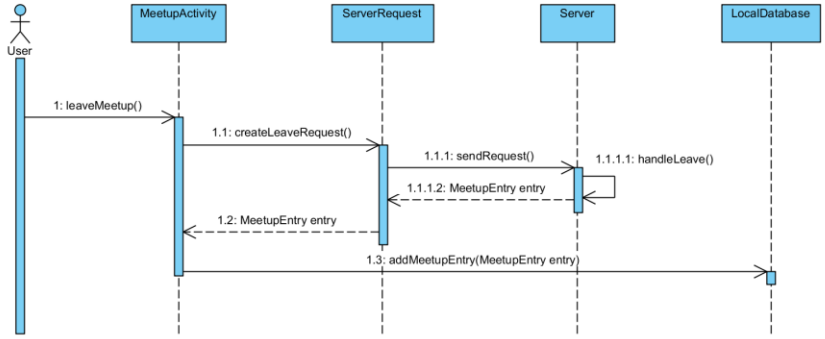


In this sequence, a meetup is first being made. A meetup entry will be made with the profiles of the matched users, and then placed onto the database, along with a list of the users. This second list will only ever be added to, and will be used for profile history.



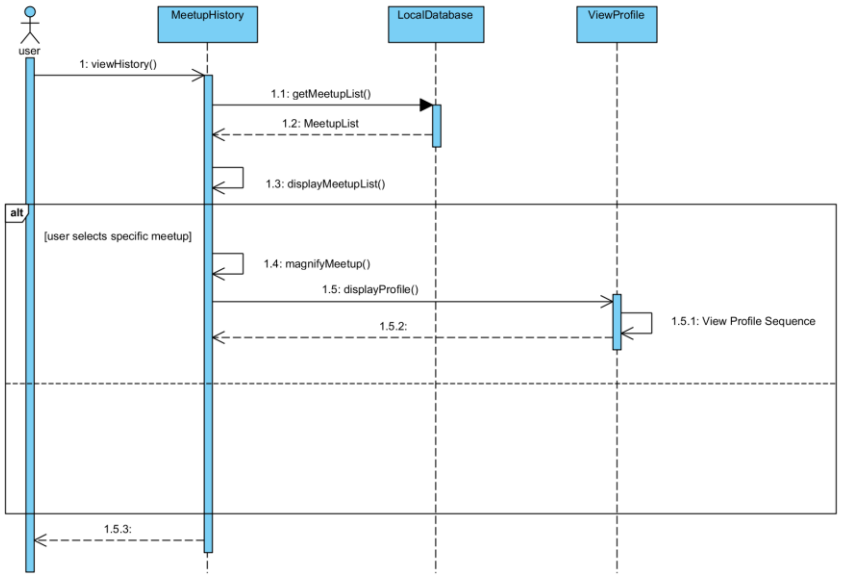
When a user forms a meetup without searching for matches, the app will send a request to the server indicating the creation of a meetup. When a user leaves the meetup, the key of the user that was stored in the meetup will be removed from the current users list and added to the former users list. This will allow the app to keep track of meetups and members.

Leave Meetup (App)

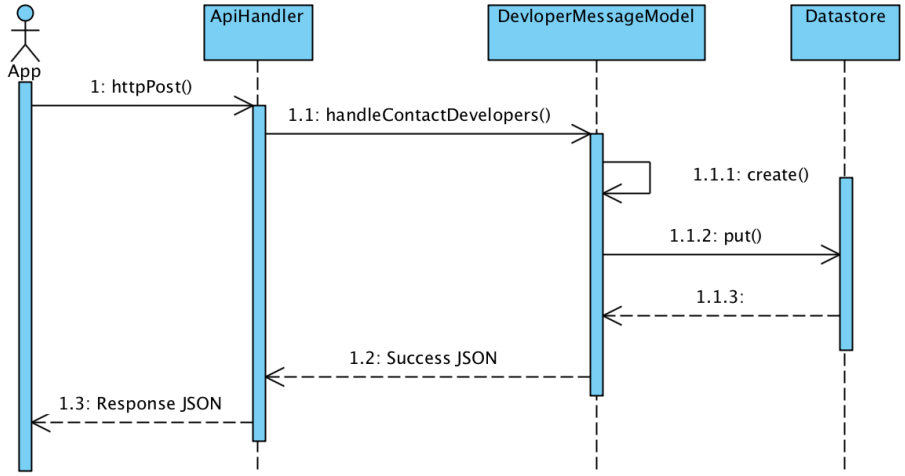


In this sequence, a user leaves a meetup they are currently in. A request will be made to the server to remove the user from the meetup. The server will handle the removal of the user, and return a Meetup Entry. This Meetup Entry consists of all previous users in the meetup, which is stored separately than the list of active users. The Meetup Entry will then be placed on the phone or device's local database, as part of the user's meetup history.

View Meetup History (App)

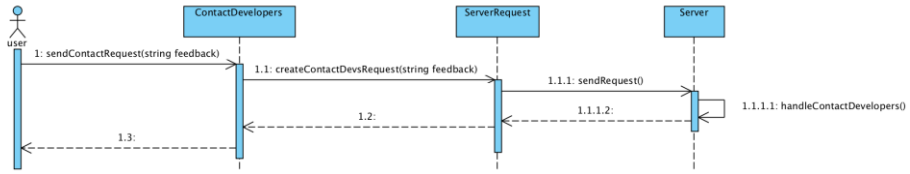


The View Meetup History sequence has minimized contact with the server, due to the method we have chosen to store meetups, outlined in Leave Meetup (app). The LocalDatabase will return a list of past meetups, and display them. A user will be able to magnify a meetup to see more detail about it, and view user's profiles that were part of the meetup. The View Profile sequence takes place at this time, and this is the only point in which this sequence has server contact.



When the user desires to send a message to the developers of the app, the app sends the user key and the message to the server. The server receives this request and creates a new developer message that is stored in the datastore. The server then responds indicating the request has been successfully sent.

Contact Developers Diagram (App)



An extremely simple sequence; the user will invoke the sendContactRequest() method with a parameter string, feedback, which contains developer feedback. This will then be placed on the server using server requests, for the developers to look at whenever they wish.

Afterword

Some of the recommended changes to the Use Case diagrams and the Activity diagrams cannot actually be made, due to the limitations of Visual Paradigm. For the most part, these limitations are limited to not being able to send an activity to a line of another activity, but being forced to send an activity to another activity. Thus, while we appreciate and acknowledge the feedback provided, we are genuinely unable to act on a few of the recommendations without converting our entire suite of diagrams into a different program. Additionally, our Use Case and Activity diagrams both include many potential features. Should these not make it to the final release, the diagrams will be edited. As such, our **sequence** diagrams reflect the sequences of required features, while our **activity and use case** reflect more possibilities that, should they be implemented, will receive sequence diagrams if necessary.