Computer Science and Engineering, University of Nevada Reno

Party Up!

Assignment 2:

Design and Specification
3/17/17

Group Members:
Martin Revilla, Jaime Moreno, Henry Huffman, Brian Parawan

Instructors:
Devrin Lee, Eelke Folmer

External Advisor:
Dr. Michael Ekedahl ekedahl@unr.edu

Table of contents

Abstract
There are currently services that are used to plan and share events ahead of time which allow for formal planning of events but there is are no services/applications that allow people to broadcasts current events to others who might be looking for something to do at that moment. When creating a social event there needs to be invitations sent out to each person and often times friends will be left out by accident. For people with larger networks of friends, which is especially true for college students, it become a task to invite every single person they know to an event they are hosting. There are also cases where a person may not have anything planned for the day and would like to see what other friends are doing. In these cases a person would need to contact each friend to find a planned or ongoing event for the day. Although not difficult, it becomes a task for a person to both find an event to which they would be welcome to attend and also to share a hosted event to all friends.

Introduction
The application PartyUP will connect different users through posted events allowing for information sharing via application messaging and interfaces. Events are posted onto Google Maps using the Google Maps API. Posting will show up as soon as the user posts and will display to all friends through Google Maps. Users associated with the host can all see the posted location. Each user will have the ability to post events but live events cannot be planned ahead. Users are limited to one live event at a time but can have other planned events as the application should have a live feel to it. In order to keep a live feel, planned events will not show up on the map until the time of the event is reached but will show up in a user's invitations.

Users will build a friend network by adding other registered users. Friends can be further divided into custom groups, according to the user, so that only certain people can see posted events. Users will be given options to categorize posted events. A public event can be viewed by everyone as long as they are your friend. Private events can be seen by only the user's friends and group posting can only be seen by the group to which the event is applied to. With the given ability to edit who is able to see each event Events will only be posted for up to 6 hours and will be removed from the map. After an event is removed, the user can add another event. Each event can be modified by the host at any time given and can be canceled as well.

The main interface will be the map displaying the different posted event pins. Navigating through the map will allow users to see the ongoing events and choose to contact the host for any information. The map will provide a simple user interface for the purpose of displaying locations. When clicking on an event pin the application will display the host's name and provide information regarding the event details such as the occasion. The main interface will have a toggle to the settings activity where user can add friends and make simple edits to account attributes. The application will consist of a simple UI and essential functionality as it is aimed to be a quick resource to nearby events. This application should fit well in a college environment and offers enough functionality to be used by anyone.

Interview Questions/Summary

Questions

1) How many events should a user be allowed to make?
2) Should users be able to make public events viewable outside of friends list?
3) Can there be multiple events at the same location?
4) Should the location be identified through GPS coordinates or address details?
5) Will there be a log of past events for each users?
6) How long can an event be scheduled ahead of its official date?
7) Is there a limit for invitations sent for an event?
8) Can a group of users create a shared event?
9) Will users have the ability to cancel events?
10) Can other users modify the event details?
11) Should the application include a messaging feature or hook up to the already existing messaging system?
12) Who can see contact information for each user?

User Interview: Amber Gaab - agaab@nevada.unr.edu

1) The user shouldn't have a limit on the number of events that can be created. Limiting the number might limit how much a user actually uses the application. Perhaps limiting the number of events a person can create within a certain time limit would be a nice compromise.
2) No, that seems like something only businesses would use to spam events or for people to have outrageous parties.
3) Yes, weekly meetings with groups usually meet in the same location. Also, multiple people may be have events at the same location concurrently.
4) Probably address location, I'm not sure how accurate GPS location would be especially if I'm inside.
5) Keeping a log of events would be a neat idea because it could act almost like facebook memories.
6) At a maximum, five years. This would give people enough time to schedule things like long engagements or family trips ahead of time. More than five years is unrealistic and unpredictable.
7) There shouldn't be a limit. As long as you are mutual friends there shouldn't be an issue. That being said, maybe limit the number of times you can invite and individual. This would prevent people from spamming invites after its turned down.
8) Yes, this would help groups set up and manage appointments and meetings.
9) Yes, events fall through. It would be confusing to leave the event up.
10) No, if someone changed my event to a time or location that I couldn't attend, it would defeat the purpose of setting the event.
11) The messaging application feature would be neat, but there so many other applications out there it wouldn't make the application stand out necessarily.
12) Only my friends, I wouldn't use the application if my information was public.

Developer Interview: Henry Huffman - hhuffman@nevada.unr.edu

1) There should not be a limit for the number of events that a user can create. Although this may create a lot of data in the database, if organized correctly, it should easily handle a large number of events.
2) For legal purposes, the application will not be making any public events that can be broadcasted to users. This will prevent unsolicited invitations and dangerously large events.
3) Yes, there can be multiple events at the same location. This will allow users to have meetings at the same location. Perhaps in later updates beyond the basic prototype, there will be a feature that prevents users from double booking times.
4) Both GPS location and address location should be used. This will ensure that the location is correct and it does not limit the location to an address because some locations may not have an address.
5) For the prototype, there will not be a logging feature of events. This is a feature that may be implemented beyond the prototyping phase.
6) For the prototype, there will not be a limit for how far an individual can schedule an event ahead of its scheduled date. Limiting the time span of scheduling seems unnecessary and not as fun. It would be fun for users to schedule an event in the distant future and see whether or not it comes to fruition.
7) No, there will not be a limit for the number of invitations sent by the user. That being said, users will only be able to invite users that are in their friends list.
8) For the prototype, there will not be a group managed event. It should be sufficient that an individual group member make an event and send invitations to the other group members. Furthermore, manipulation of appointment times and location may lead to frustrating the users.
9) Yes, users will be able to cancel events.
10) No, other users will not be able to modify the events for this prototype. Multiple users accessing the same event may lead to conflicting agreements between users and would require more event security protocols.
11) The application will not have a built in messaging application. Instead, it will call the preferred messaging application that the user already has installed.
12) Only friends will be able to see contact information. People outside of the friends application can only see that you are a user.

Advisor Interview: Michael Ekedahl - ekedahl@unr.edu

1) This can be a tuneable. By that I mean something that can be set now, but altered later after core functionality has been developed.
2) Yes, most social sites already allow features like that already. Perhaps setting a boolean flag in user settings to allow for public and private options would be the solution. Users generally want to have control over what they can and can not display.
3) Yes, think of concurrent conferences. Sometimes events are combined or on tracks so they run very close or overlap.
4) Storing both would give the best of both world. I recommend staying away from UTM, no one really uses it. Longitude and Latitude are generalized and help if there isn't an address or the user doesn't know the address, plus most systems use longitude and latitude.
5) For the prototype, simply don't delete them. The amount of data we will be dealing with is small enough to just leave them after the event has passed. For future development I recommend archiving events. This will provide better reporting and create a trail for auditing if needed.

6) This is another tuneable feature than can be set to something arbitrary now and altered later. For future development, keep in mind that planning ahead can vary depending on what the event is. For the moment, limiting to a year in advance is fine.
7) This is also something that can be adjusted later. Perhaps set a count feature so limitations can be applied later.
8) If time allows for it, yes. Have an event flag to determine if it is a group or user modifiable event.
9) This is a function that should be configured later. Flags within the user's profile can be set to public or private.
10) For the prototype focus on one user modification only. This is a tuneable, so I would recommend having a flag that can be set later on to allow others to modify or see it.
11) Both ways are valid. The group can perform CRUD operations on a server to push this data, or just hook up to sms that is built into the device.
12) Put the users in charge of data. Design the application so users can have settings for public and private profiles.

High Level Business Requirements

1) Any person with an updated android device should be able to download the app from the Google Play Store.
2) Application needs to open up to the registration page if the person does not already have an account.
3) Account login should be saved so that the person does not have to login every time.
4) If person already has an account then application should open up to the map with refreshed data.
5) Events can be posted as soon as the person has registered.
6) People should be able to see events within a minute of the posted time.
7) Friends should be approved before they show up as a friend of that person and also be removed at any time the person wants to.
8) Anyone should be able to create a group of specific friends .
9) Scrolling through the map should not be limited, even if beyond the radius of event list.
10) Live events need to posted directly to map.

Technical Requirements Specification

Functional

Priority Level 1 (High)

The application must allow users to create a profile and log into the application.
This function is the highest priority because if users are unable to login and access their data, the entire application is inaccessible.

Priority Level 2 (Medium)

The application must allow users to create events with accurate details such as who, what , when, where, and why. This is the core functionality of the application.

Priority Level 3 (Low)

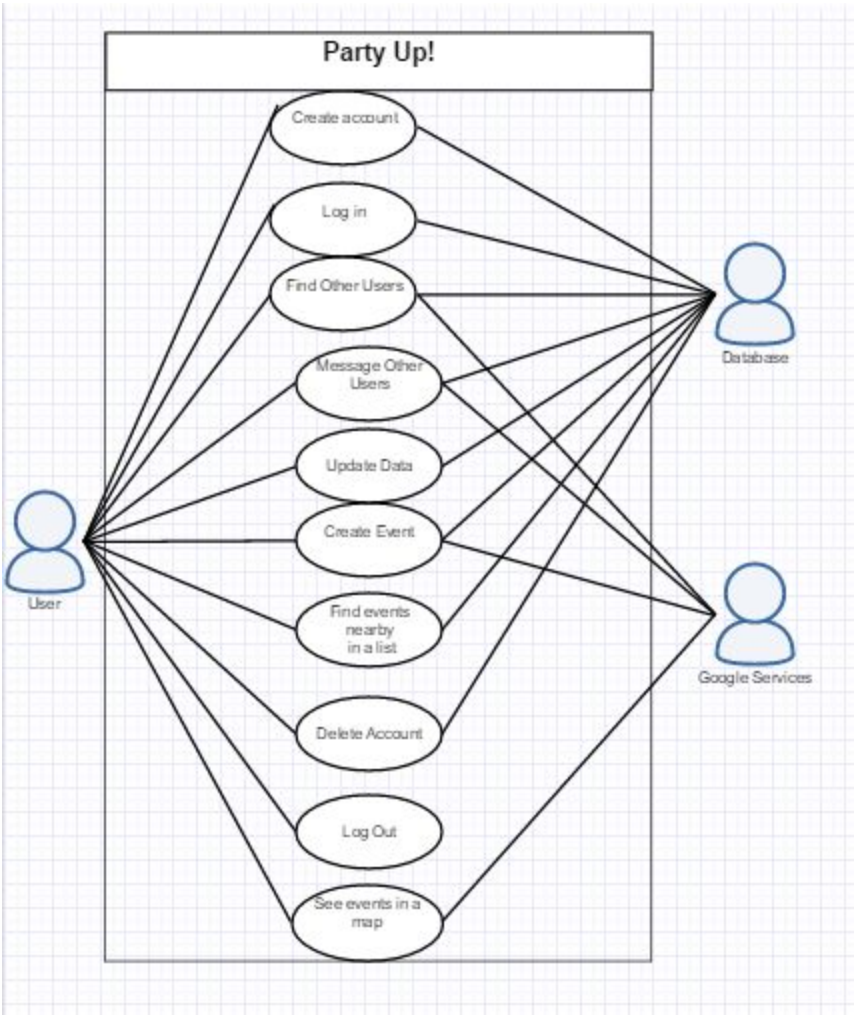
The event must show as a marker on google maps for the event owner as well as all of the invited guests or friends. This functionality is in addition to the core functionality of creating and organizing events. It is an aesthetic that will help users make their way to an event and provide an interface most users are familiar with.

Non-Functional
Priority Level 1 (High)
Passwords must be hashed to prevent security breaches of user profiles. Due to the high value and high risk of user data, this non-functional requirement is ranked the highest.

Priority Level 2 (Medium)
The data must be accessible to the user at all times. This means running the database and web service on a reliable server that is accessible to user. Moreover, redundancy clusters should be in place to ensure data accessibility.

Priority Level 3 (Low)
Invitations can not be sent to old or expired events. This is not as critical because nothing is lost or jeopardized by an old invitation; consequently, it will annoy and confuse users.

Use Case Modeling



1) Create account - On the Title/Starting page of the app, a new user interested in Party Up! can create an account that asks for details such as name, age, email, and a proposed password for the creation of their new account.

2) Log in - On the Title/Starting page of the app, a returning user will use an email/username and password to log into their account. The database will handle the authentication.

3) Find Other Users - A button allowing a user to look up their friends or other users using their emails or usernames. Once found, they can see their profile.

4) Message Other Users - If a user is looking at another user's profile, the current user can communicate with the other user via a message, if the other user allows others to communicate with them. This use case also includes adding that other user as a friend.

5) Update Data - A user can update their profile settings such as privacy or basic information like birthday. Any update is sent to the database to publish the new user's data.

6) Create Event - A user can host an event at the location they are currently located. The user can invite friends, write details about the event, and adjust privacy settings such as the audience of who can see this event in the map or events list.

7) Find Events Nearby in a List - As opposed to seeing nearby events in a map, a user can see these events in a list. Event details such as location is then listed if tapped. A user can specify the nearest and farthest distances Party Up! filters events.

8) See Events in a Map - As opposed to a list, users can see nearby events in a map. They can zoom in and out to see nearby events, and they can tap on events to see who is hosting and the users that will be going.

9) Log Out - A current user can choose to log out of their Party Up! App. The Title/Starting page is then shown if the user wishes to log in again.

10) Delete Account - A user can delete their account and have their profile removed from Party Up! And its database. Authentication and security details are asked first before deletion is completed.

The following are the use case templates for 4 use cases. They explain more in detail what preconditions, the flow of events, the postconditions that will occur when the use case is conducted.

Use Case: Log Out
ID: UC9
Actors: User
Preconditions: 1: The user is logged in.
Flow of Events: 1. The user is logged in and taps the log out button. The user is shown a verification message. 2. If the user taps “Yes, I want to log out.” 2.1. The user is logged out of their account. 3. If the user taps “No, I want to stay logged in.” 3.1. The user is returned to the current screen they are on.
Postconditions:
Alternative flow 1: The user logs out and the user sees the Title/Start page.
Alternative flow 2:
The user stays on the current screen they were on.

Use Case: Update Data
ID: UC5
Actors: User Database
Preconditions: 1: The user is logged in.
Flow of Events: 1. The user is logged in and wishes to update their profile. 2. The user taps on a basic information and chooses to edit it. The application asks if they are sure of the change. 3. If the user taps “Yes, I am sure of the change” 3.1. The user is returned to the current screen and their information is updated. 4. If the user taps “Cancel” 4.1 The user is returned to the current screen and their information is not updated.
Postconditions:
Alternative flow 1: The user’s profile has been updated.
Alternative flow 2:
The user’s profile has not been updated.

Use Case: Find Other Users
ID: UC3
Actors: User Database
Preconditions: 1: The user is logged in.
Flow of Events: 1. The user searches for a user, either through their email or the username of the other users. Full names can also be searched. 2. The application will display the results of showing potential matches, if potential matches are found in the database.
Postconditions: The user has either found the user(s) they searched for or the database has returned nothing.

Use Case: Create Event
ID: UC6
Actors: User Database
Preconditions: 1: The user is logged in.
Flow of Events: 1. The user wishes to host an event. By tapping on a button, they are shown an event creation screen. 2. On the creation screen, they are ask to fill out text forms such as the name of the event and a description. They are also allowed to invite other users they added as friends.
Postconditions: The user has created an event is shown to be hosting it.

Requirement Traceability Matrix

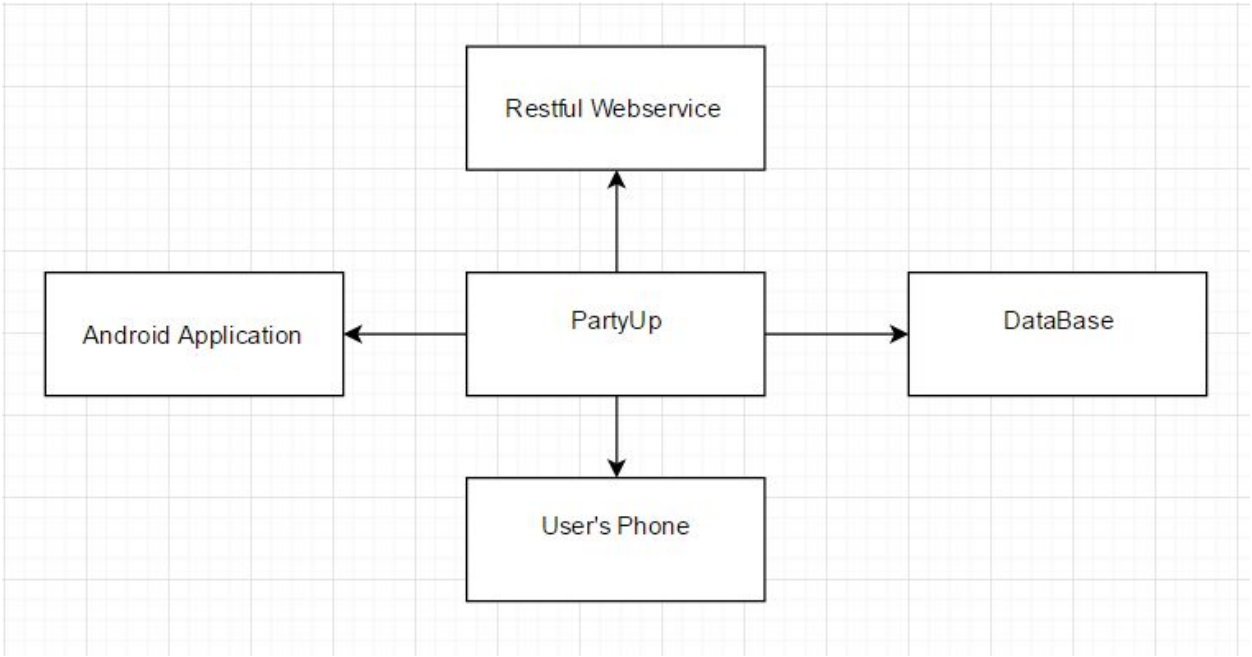
Use Cases/Functional Requirements	Create Profile	Log in	Create Events	Event seen on map	Event seen on list
Create Account					
Log in					
Find Other Users					
Message other Users					
Update Data					
Create Event					
Find Events nearby in a List					
See Events in a Map					
Log Out					
Delete Account					

Potential Legal Issues

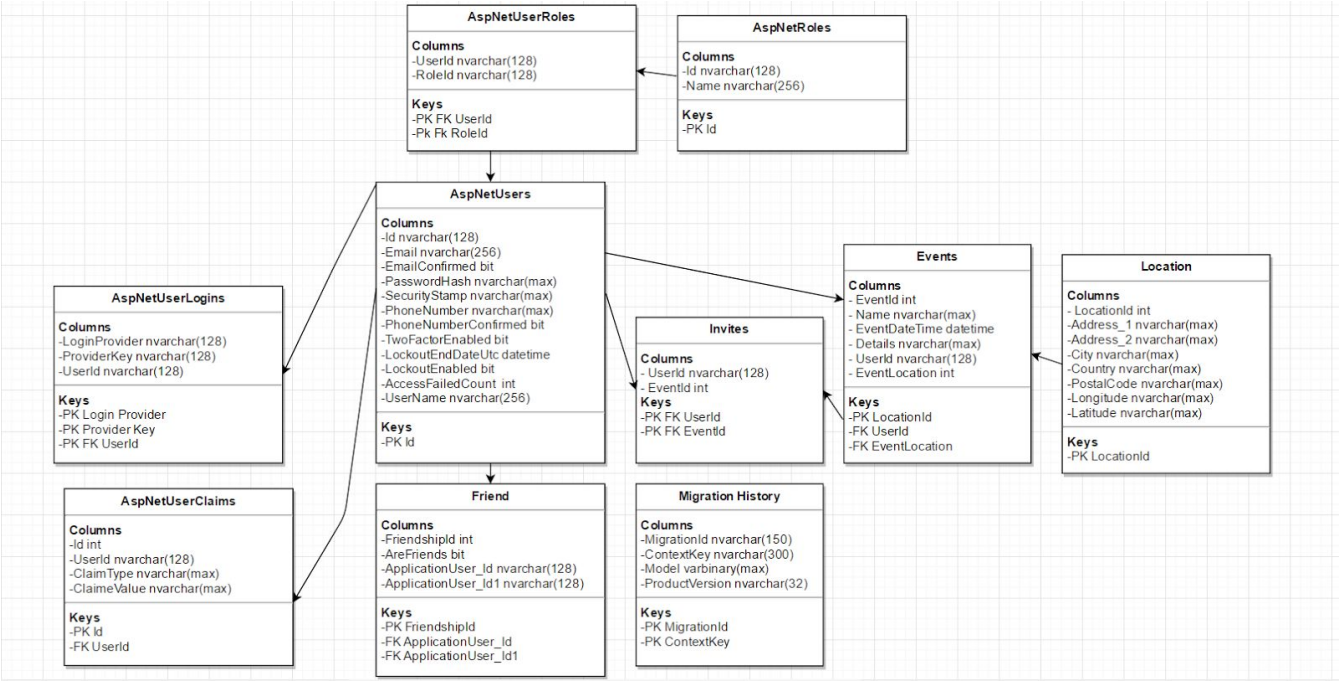
1. Revealing location of users to the public. Releasing this type of information could jeopardize the safety and well being of users. This can be solved through only allowing invited friends or authorized contacts to view event information.
2. Events may be set up to mug or scam users. This issue was seen with the rise of the Pokemon Go application. By only allowing invited friends, PartyUp aims to avoid this legal issue entirely.
3. Users focusing on GPS rather than being aware of surroundings can lead to bodily harm. This legal issue has been seen in popular applications such as Pokemon Go and Google Maps. The application merely provides a service, when and how the user chooses to use the service is entirely at their own risk. Perhaps addendum included in the licensing and legal agreement.
4. Tracking and accumulating location based data of user. Including in the license agreement that users understand datalocation is tracked in their events would help resolve this issue. Furthermore, encrypting data and following a service agreement where this data is not released to third parties would also help prevent this legal issue.
5. User profiles stolen and impersonating individual. Acknowledgement of this risk must be agreed to by the user, and encryption strategies such as hashed passwords and encapsulating user profile details will help resolve this potential legal issue.

High-Level and Medium-Level Design

Context Model



Database Model



Database Documentation

AspNetUsers

-Id: a unique string that identifies the user in the database; Primary Key
-Email: string representing user specified email
-EmailConfirmed: bit represents if the email has been confirmed
-PasswordHash: string that holds the hashed password
-SecurityStamp: string that verifies session
-PhoneNumber: string that represents the user's phone number
-PhoneNumberConfirmed: bit represents if the phone number has been confirmed
-TwoFactorEnabled: bit represents if two factor security is enabled
-LockoutEndDate: timedate representing when account will be unlocked
-LockoutEnabled: bit represents if user has lockout feature enabled
-AccessFailedCount: integer that keeps track of the number of failed attempts
-UserName: string that will be used to store user's unique username

AspNetUserLogins

-LoginProvider: string representing who is providing the login; Primary Key
-ProviderKey: string representing unique provider key; Primary Key
-UserId: string representing the specified user; Foreign Key

AspNetUserClaims

-Id: integer representing unique claim; Primary Key
-UserId: string representing the user making the claim; Foreign Key
-ClaimType: string used to describe the claim made

-ClaimValue: string containing the claim itself

AspNetUserRoles
-UserId: string representing a unique user; Primary and Foreign Key
-RoleId: string representing a specified role; Primary and Foreign Key

AspNetRoles
-Id: unique identifier of a role; Primary Key
-Name: string representing the name of the role

Friend
-FriendshipId: integer representing the unique relationship between two users; Primary Key
-AreFriends: bit representing if friendship has been established by both parties
-ApplicationUser_Id: The id of the user who sent friend request; Foreign Key
-ApplicationUser_Id1: The id of the user who has received friend request; Foreign Key

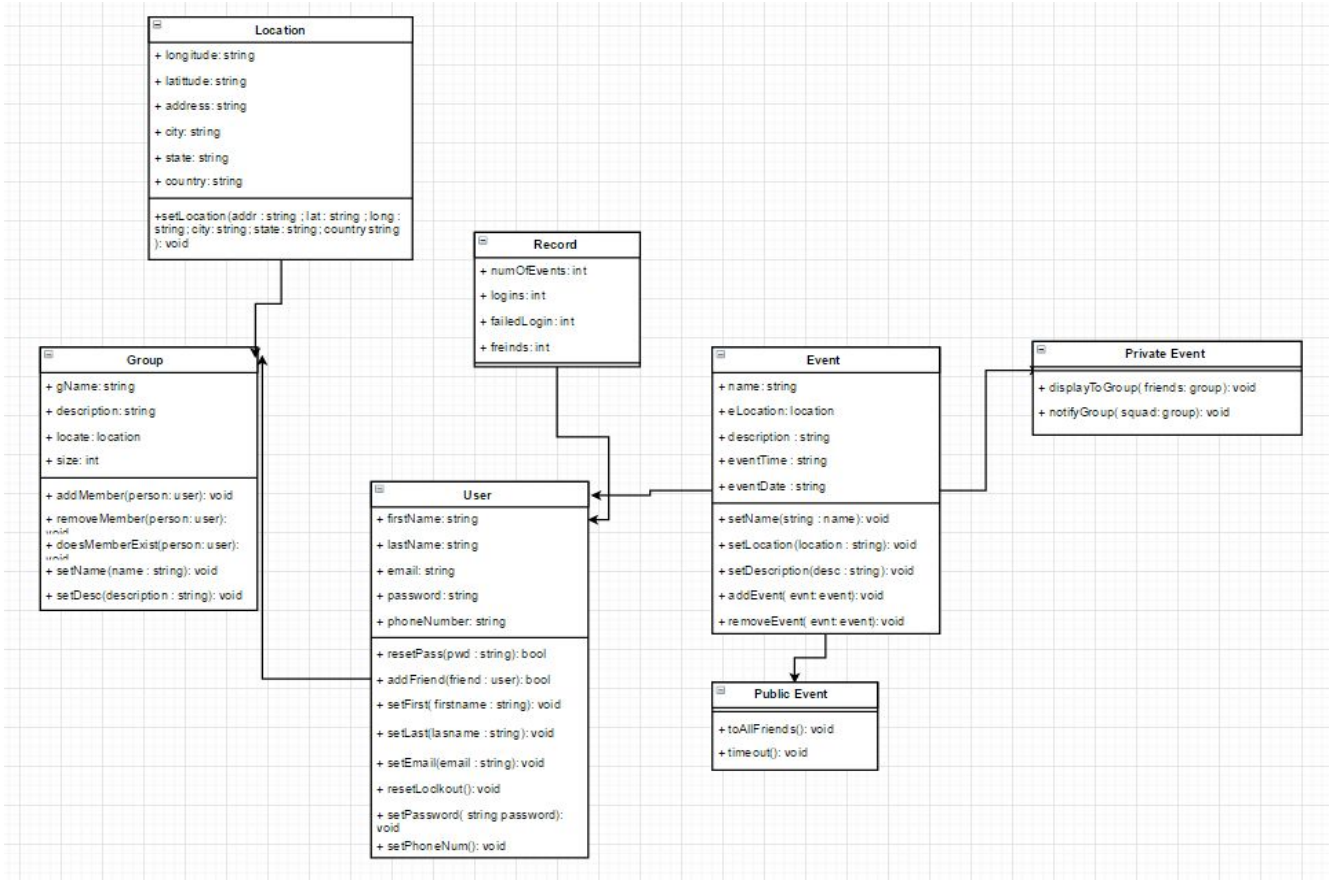
Invites
-UserId: string representing id of user that has been invited; Primary and Foreign Key
-EventId: integer representing the id of the event that a user has been invited to; Primary and Foreign Key

Events
-EventId: integer uniquely representing the event; Primary Key
-Name: string used to store the event name
-EventDateTime: datetime value for when the event will be held
-Details: string that contains all the details related to the event
-UserId: the user who created the event; Foreign Key
-EventLocation: integer for unique location; Foreign Key

Location
-LocationId: integer uniquely representing a location; Primary Key
-Address_1: string that hold basic address info
-Address_2: string that holds basic address info
-City: string that holds the city name
-Country: string that hold the country name
-PostalCode: string that hold postal code of event location
-Longitude: string that holds coordinate position
-Latitude: string that holds coordinate position

Migration History
-MigrationId: string holding the unique name of the migration; Primary Key
-ContextKey: string that represents when and where migration was made; Primary Key
-Model: variable bits that represent the database after this migration
-ProductVersion: string representing what version of migration was used

Class Model



User

-firstName: string containing first name of user.

-lastName: string containing last name of user.

-email : string containing email used to register and sign in to account.

-password: string containing password used to sign into account.

-phoneNumber: string containing phone number of user.

-resetPassword(password: string): resets the user password if forgotten.

-resetLockout(): resets sign in lockout count of user.

-addFriend(friend: user): adds friend to user profile.

-setFirst(firstName : string) : sets first name of user in database using API.

-setLast(lastName : string): sets last name of user in database using API.

-setEmail(email: string): sets the email for the user in database using API .
-setPassword(password: string): sets the password for user in database using API.
-setPhoneNum(num: string): sets the phone number of user in database using API.

Event
-name: string used as name of event.
-eLocation: location used to pinpoint event on map .
-description: string containing details of event.
-eventTime: string indicating the time of event.
-eventDate: string indicating date of event.
-setName(): sets the name of event in database.
-sendInvite(): sends invite to friends.
-setLocation(): sets the location of the event.
-setDescription(): sets details of event.
-addEvent(evnt: event): adds passed event to database.
-removeEvent(evnt:event): removes passed event from database.

Public Event
-toAllFriends(): sends event information to all friends.
-timeout(): sets the timeout of public event.

Private Event
-displayToGroup(friends: group): displays post to designated group for access.
-notifyGroup(squad : group): notifies group of created event.

Record
-numOfEvents: integer containing total number of events for user.
-logins: integer containing total number of login history.
-failedLogin: integer tracking number of failed login attempts.
-friends: integer containing total number of friends.

Group
-gName: string containing the name of group.
-description: string containing group details.
-locate: location type referencing the location of event.

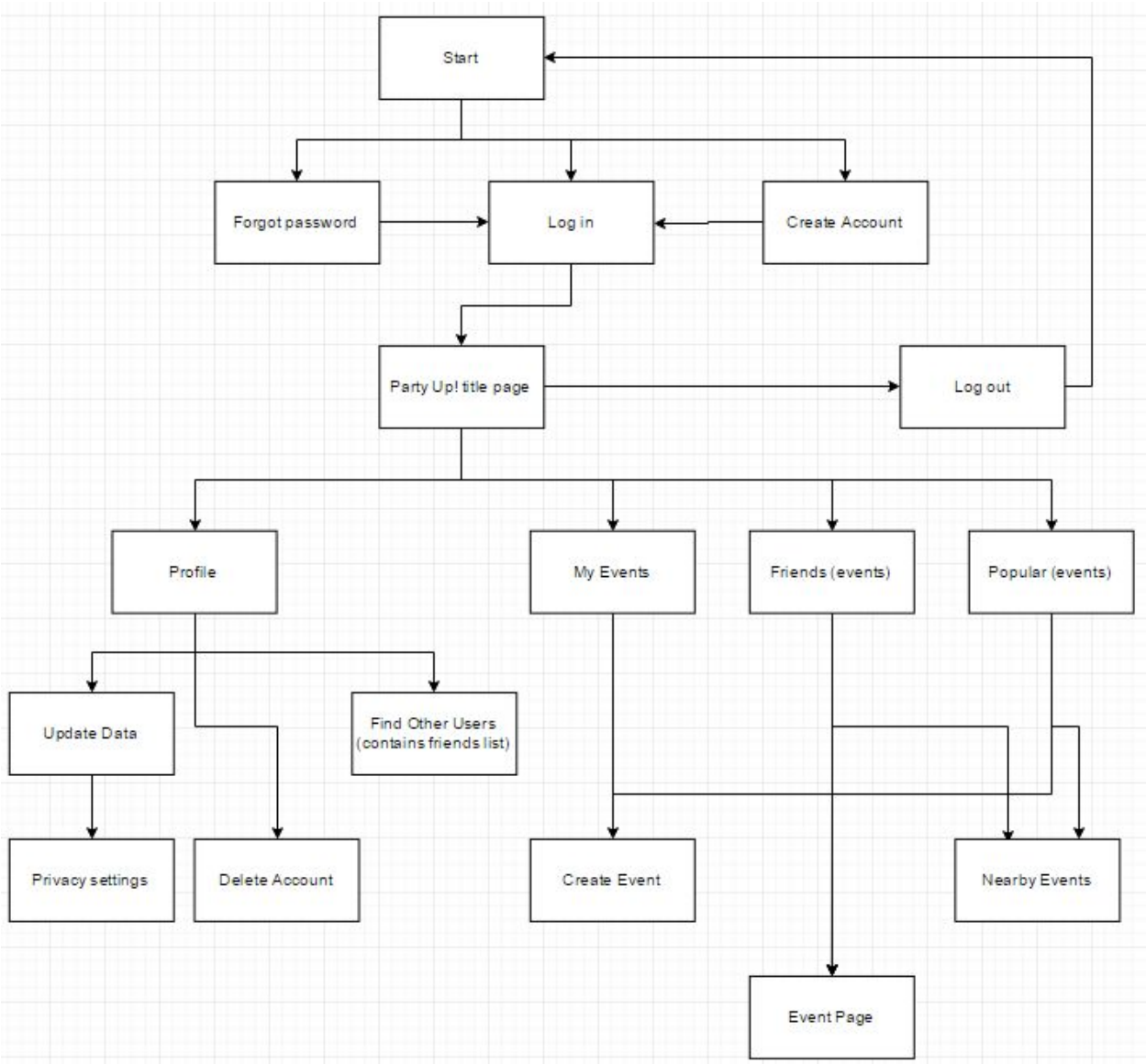
-size: integer indicating member size of group.
-addMembers(person: user): adds passed user to group.
-removeMember(person: user): removes passed user from group.
-doesMemberExist(person: user): check to see if user is already member of group.
-setName(name: string): sets name of group to string passed.
-setDesc(description: string): sets details of group.

Location

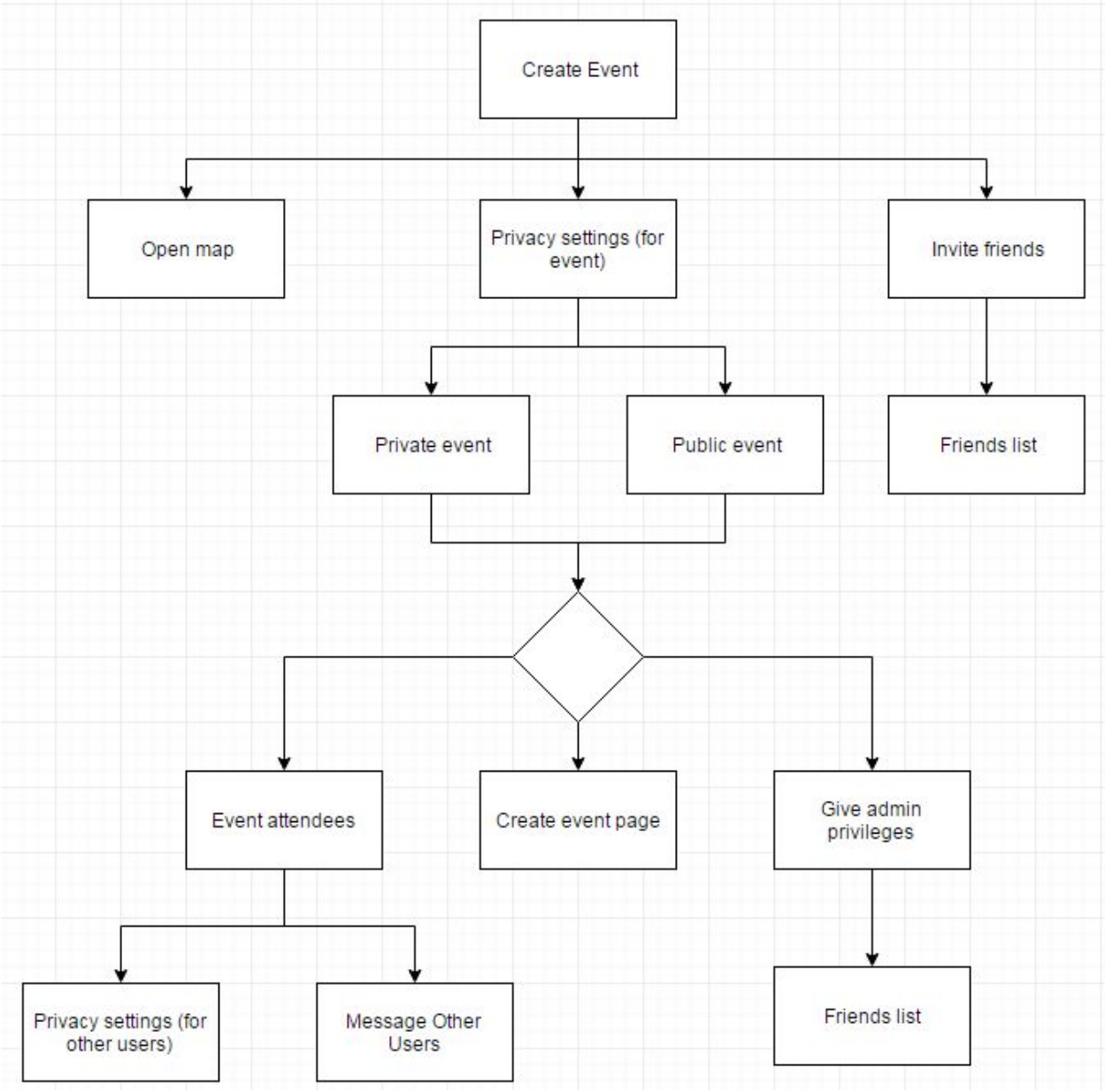
-longitude: string containing the longitude attribute.
-latitude: string containing the latitude of location.
-address: contains the address of location in string.
-city: string containing city of location.
-state: contains state of location in string.
-country: string containing the country of location.
-setLocation(addr: string ; lat : string; long: string; city : string; state : string; country : string;):
Initializing the location with specified attributes.

Detailed design

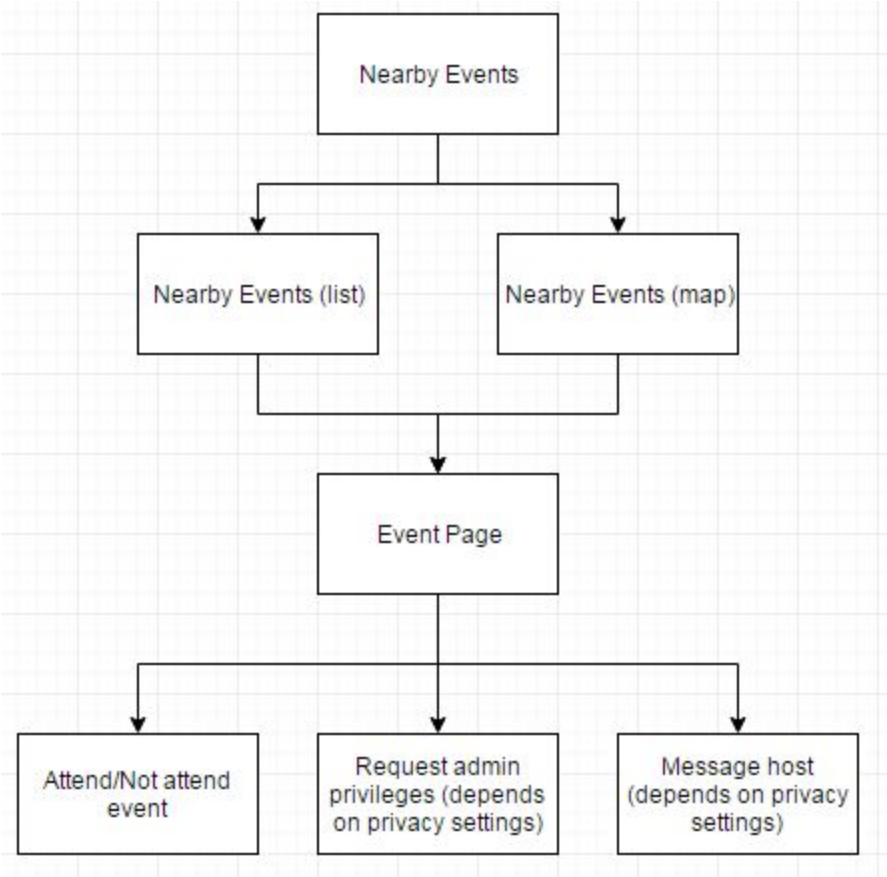
1. Main screen navigation



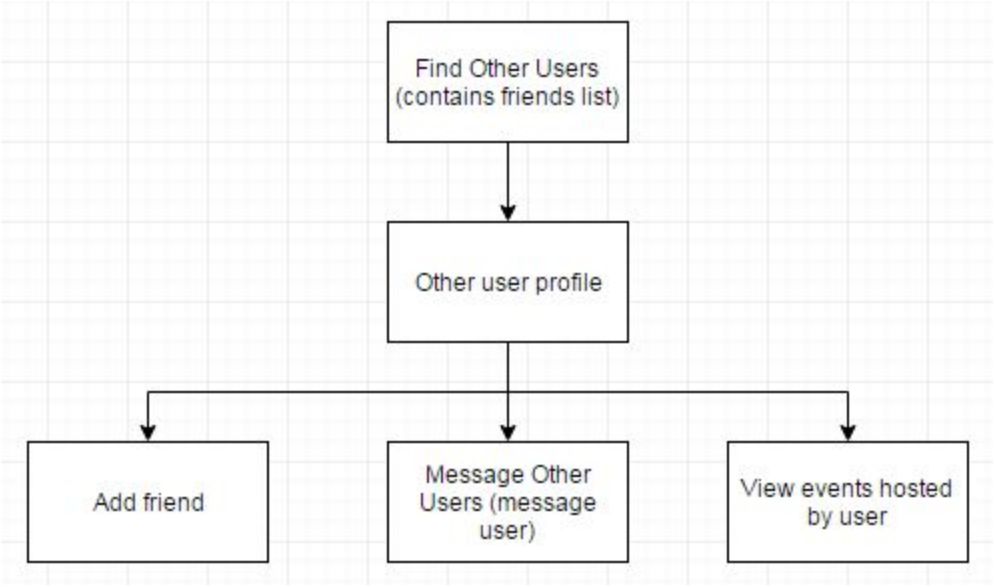
2. Events by user



3. Events by other users

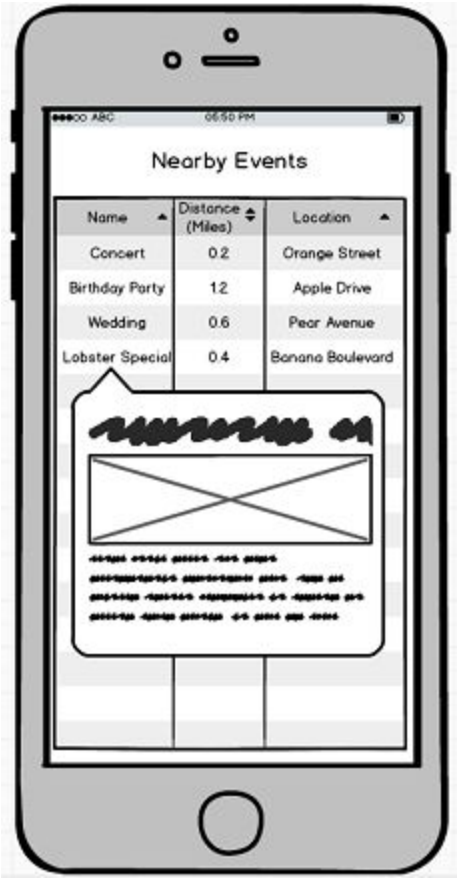


4. Interacting with other users



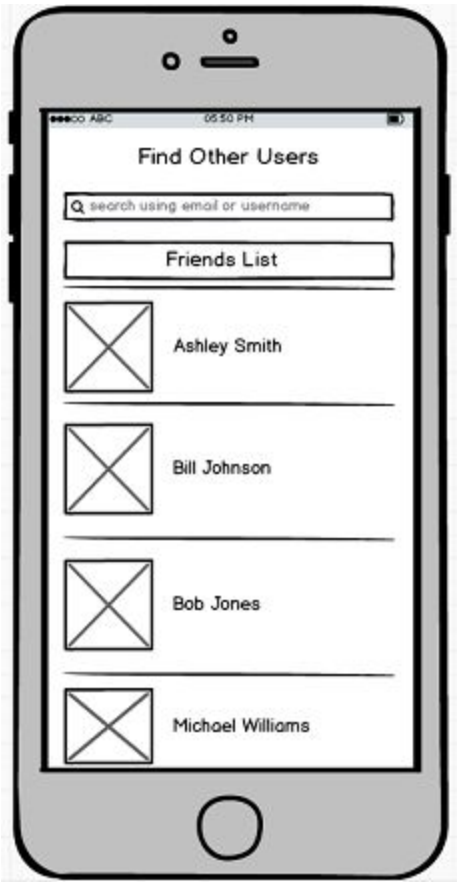
Initial Snapshots/User Interface Design

1. See Events in a Map allows users to see nearby events on a map as opposed to in a list format. The user can zoom in and out and tap on events to know more information about them.



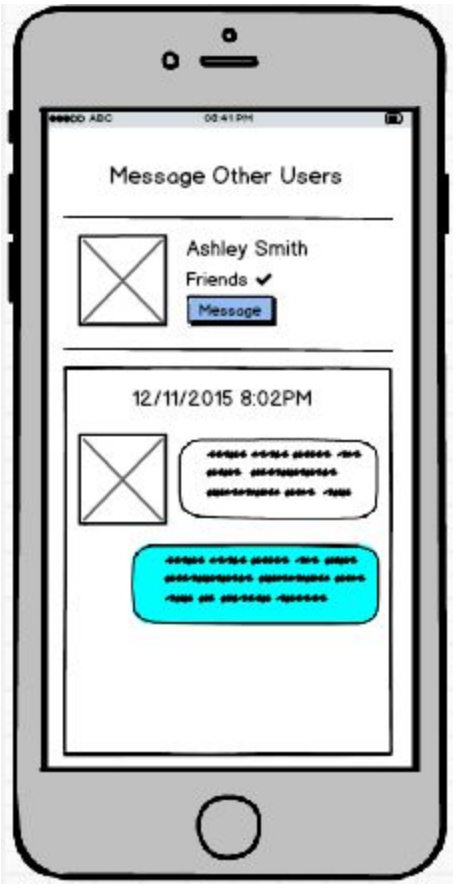
2. Find Nearby Events in a List allows users to see nearby events in a list format as opposed to on a map. The user can list by alphabetical order, distance (miles), or location.

3. Log in features the screen that existing users will see when first opening the app. The user will input their username and password to log into their account and use the service. The database handles the authentication.

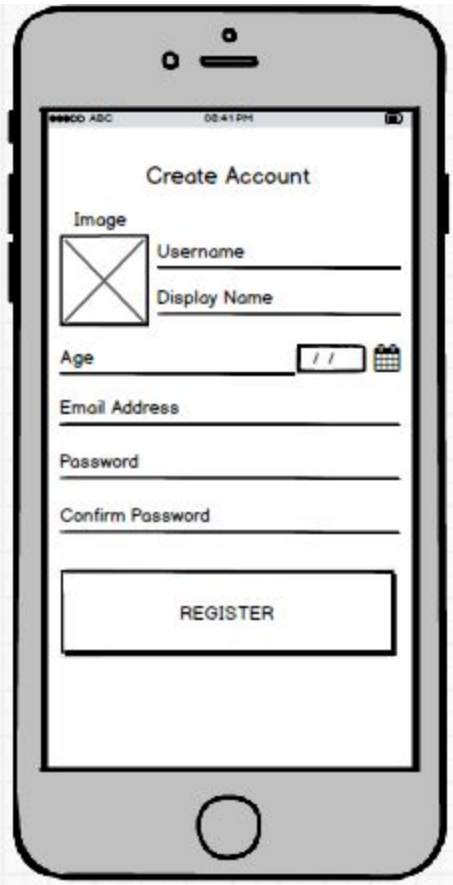


4. Find Other Users allows users to search other existing users by using email or username. As the user types, a pop-up will drop down according to their input.

5. Create Event allows the user to either create an event where they are currently located or somewhere else. The user can immediately change the default privacy settings and invite their friends to their event.

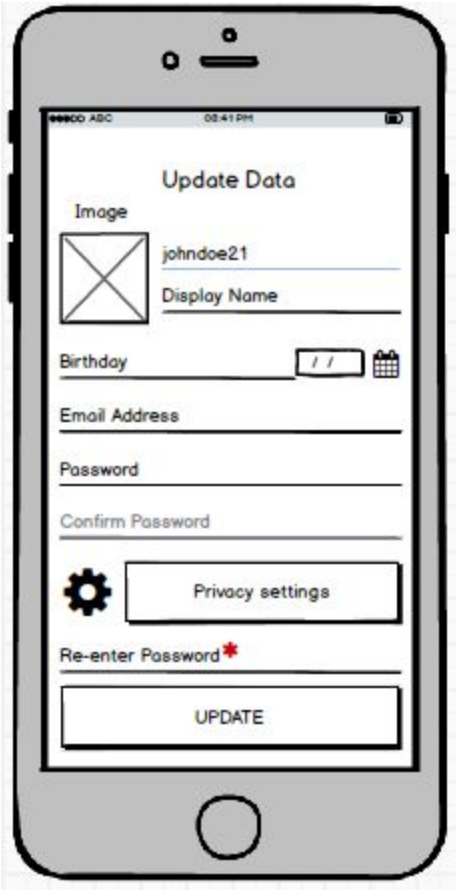


6. Message Other Users allows users to message other users upon being on their profile page. By tapping the “Message” button, a message window will pop-up below their name in which both users can send/receive messages to/from one another.

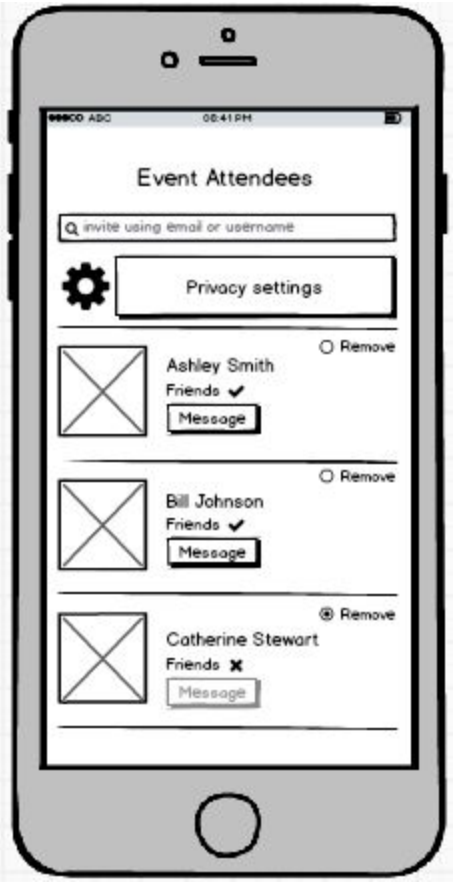


7. Create Account allows users to become a new user to the service. The user inputs their login details (username and password), which they should always remember, and other details that other users can see.

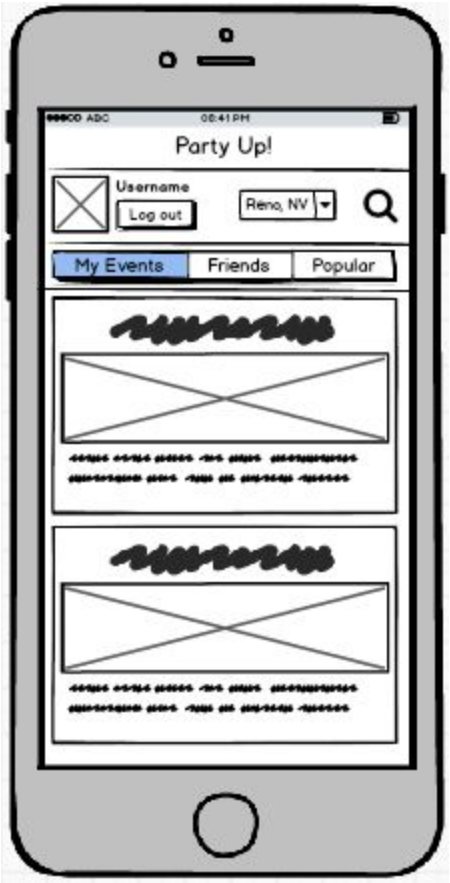
8. Delete Account allows an existing user to delete their account, which means every trace of their account in the service and security database is deleted. The user must re-enter their password to ensure that they are the owner of the account that is being deleted.



9. Update Data allows users to update their private and public information as shown on their profile. The user must re-enter their password if at least one change is made to their data. Updates are sent to the database to update the user's data.



10. Event Attendees allows the user to see the current attendees for an event they are hosting. The user can invite more users using email or username, change the event's privacy settings, see whether they are friends with attendees, message attendees, and remove attendees. It also appears that this user cannot message users who they are not friends with. This privacy setting can be changed.



11. Party Up! title/starting page of the app allows users to see three different lists of events categorized by their own hosted events, events hosted by their friends, and popular events hosted by other people in the current location (Reno, NV in this case). The search function allows users to do a search that permeates the whole service, not just the details on this screen.

12. Privacy Settings allows the user to label an event as a private event or public event. The user can tap on the “Event Attendees” button to see who is currently attending the event and remove attendees if desired. The user can also give admin privileges for the event to other users.

Annotated references

1)https://www.codeproject.com/Articles/112470/Developing-a-REST-Web-Service-using-C-A-walkthroug
In this article/tutorial, the author talks about the basic CRUD operations that go into building a web service. Moreover, it also discusses how web services can utilize REST rather than SOAP to perform http get and post methods. This example is developed in C# and utilizes the .NET framework that our team has decided to use. Although this does include a brief tutorial on how to actually set up a simple web service, it does not include how to connect the service to a database and how to consume the service itself.

2) https://www.w3schools.com/sql/default.asp
This is a link to a SQL tutorial on w3schools. This provides a starting point for learning just the basics of SQL and will be referenced throughout the development of the database portion of this project. The SQL tutorial includes everything from developing a normalized schema to complex queries against the database. Lastly, the w3schools tutorial provides understanding of core concepts of databases such as primary keys, foreign keys, and the many relationships tables can share.

3)
https://www.codeproject.com/Articles/304302/Calling-Asp-Net-Webservice-ASMX-From-an-Android-Ap
In this link, it gives a brief example on how to consume a web service from an android application. This is helpful in defining and understanding the client to server and server to client relationships that the RESTful web service needs to provide. Although the tutorial shows only how to consume the webservice using SOAP, most of the key concepts translate over. This tutorial serves more to provide an understanding of consumption rather than the implantation process itself.

4)https://www.lynda.com/Development-Tools-tutorials/Xamarin-Essential-Training
For the development of the android application, our team will be using Xamarin. This Lynda tutorial provides the team with simple application examples and tutorials to learn android development. Furthermore, all programming will be done in C# and XML. These tutorials combine both languages and show how android applications combine the functionality of both these languages to create an application. This tutorial also provides a nice explanation on how to set up the developing environment for Xamarin.

5) http://business.unr.edu/faculty/ekedahl/
This is a link to our external advisor's website. Professor Ekedahl has many helpful links that guide users on how to set up their developing environment. His website also has many labs that will help the team complete and understand various aspects of the project. Two of the most helpful links are his IS 389 course which guides android application development, and his IS 460 course which details connecting to a database and setting up a simple web service.

6) https://www.microsoft.com/net/tutorials/csharp/getting-started
In this tutorial, Microsoft details on important aspects of the .Net framework that will be used in developing parts of this application. The tutorial also includes helpful insights while learning C# and understanding what is being accomplished. The tutorials can be adjusted to meet each of the team members needs by selecting what experience the team member has and what knowledge they already know. Some of these tutorials will allow developers to learn to automate certain developing processes.

Contributions of team members

Henry Huffman - Interview Questions/Summary, Technical Requirements, Legal Issues, High/Medium Level Design, Annotated References, Glossary

Jaime Moreno - Abstract, Introduction, High Level Business Requirements, Glossary, High/Medium Level Design

Martin Revilla - Use Cases, Use Case Diagrams, Requirement Traceability Matrix

Brian Parawan - Initial snapshots, User interface design, Detailed design

Glossary updates

1. REST: Representational State Transfer; is a style of building web services.
2. Layout: Defines the visual structure for a user interface.
3. API: Application program interface is a set of routines, protocols for building software applications.
4. ASP.NET: Open source server side web application framework designed for web development.
5. HttpClient: An interface for HTTP client.
6. Framework: a basic structure underlying a system concept.
7. Activity: A single, focused interaction point for the user.
8. View: A basic block on the user interface which occupies a rectangular area on screen.
9. Widgets: Miniature application views that can be embedded in other applications.
10. Adapters: Object that acts as a bridge between a view and underlying data for that view.
11. Fragments: represents a behavior or a portion of a user interface in an Activity.
12. Web Service: A method of communication between Xamarin and other required resources.
13. Dialog: A small window that prompts user to make a decision.
14. Toasts: A view containing a quick little message for the user.
15. Generics: Classes that take in a data type as a parameter when created.
16. Primary Key: a special relational database table column designated to to uniquely identify all table records.
17. Foreign Key: a field in one table that uniquely identifies a row of another table or the same table.
18. Connection String: a string that specifies information about a datasource and the means of connecting to it.
19. Schema: organization of data as a blueprint of how the database will be constructed.
20. Schema Migration: the process of updating or reverting a database to a different

version.

21. Consuming a Web Service: calling or use a web service from a client.
22. WCF: Windows Communication Foundation; framework for building service-oriented applications.
23. Bundle: provides a means of saving and restoring state data of an activity.
24. Extra: an object that allows you to pass data from one activity to another.
25. Intent: used to start and pass data between activities.
26. Attributes: used throughout code to tell the compiler what to do.
27. Gravity: a way to define which side of a specified object another object is anchored to.
28. Manifest: tells the system how to compile the project, defines SDKs, requests permissions, tells the system about the default activity, and keeps track of all activities.
29. Resources: XML documents that contain all the resources (strings constants, layouts, etc.) that are referenced by an android application.
30. SHA - Secure Hash Algorithm; typically used to encrypt data such as passwords.