Meet-Meet Design Document

The design for Meet-Meet is made up of several individual components. The first component is the Graphical User Interface (GUI) that the user interacts with. The second component is the C# code that is compiled to native code which runs on a user’s device. Finally, we have created a backend with a webapi and a database to store user data for the long term. To help us meet these goals we used a program called Xamarin to compile C# code and provide a GUI builder, the free service tier of Amazon Web Services (AWS) to host our server and database. Finally, we used Google Cloud Messaging (GCM) to push notifications to our user’s devices. Our application must be able to access the user’s location and use that information to match them with other users in the same area. Thus, it is essential for us find a user’s location, as well as efficiently finding other users within the same area.

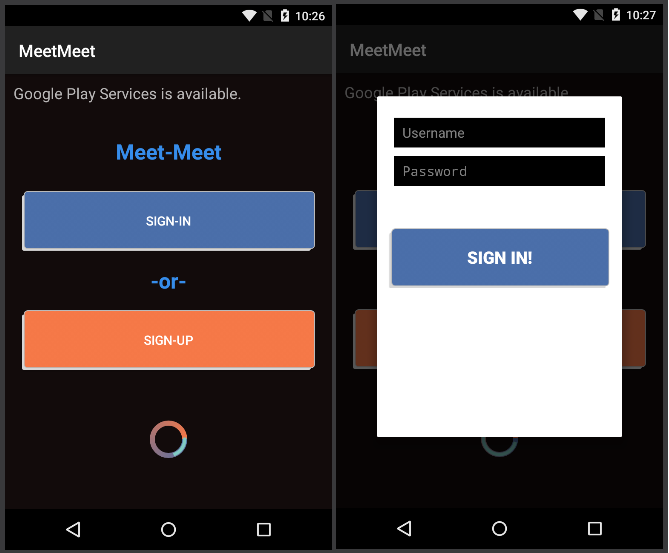
The GUI for Meet-Meet is fairly simplistic, we allow a user to sign up for the service in the app. The first menu has buttons for signing up and signing in, when the user clicks a button, they are given an appropriate dialog where they can input any required information. The signup dialog includes a “sign up” button and the following fields: username, password and password confirmation. The login dialog simply has the username and a single password field, in addition to the login button. Once users are signed up, they are taken to an edit profile page where they can set their gender and bio. After that, users are taken to the home menu where they can set their location, find nearby users, send messages, logout, edit their profile or delete their account. The “find nearby users” menu displays a list of usernames within a 5-mile radius. When a username is selected, that user’s gender, bio, upvotes and downvotes are displayed. A user may be selected by double clicking, which will take the current user to viewed user’s profile. The profile screen displays the following information: username, gender, bio, the rating of the user (positive and negative votes), buttons for upvoting and downvoting, inviting (to a group), and finally, buttons for blocking and unblocking. The user can also send messages to nearby users using the “send message” button from the main menu. Once in the send message screen, the user can select another nearby username, write a message, and send it. The “group message” menu is almost identical to the single message screen, the list of users has been removed and it merely has a text field and a button, which the user can use to send a message to their entire group (if they are currently in one). We made the conscious decision to not include profile pictures, mainly because the users are in close proximity to each other and we don’t want to allow one user to find another with the consent of both.

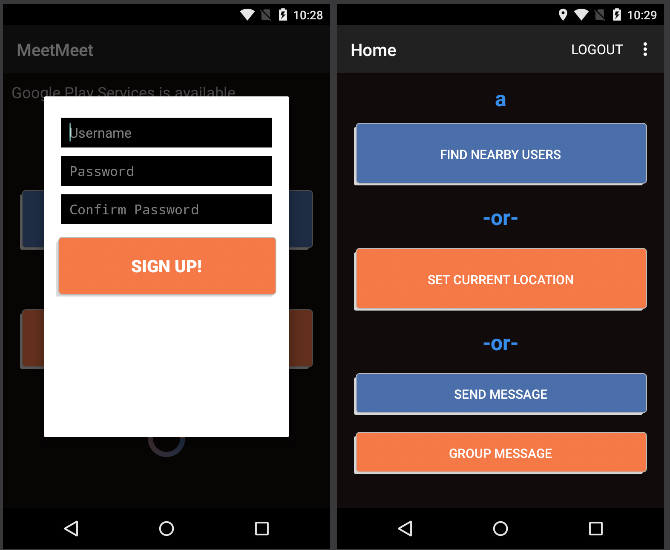
The most important aspect of Meet-Meet is the location based grouping. This is the backbone of our application and it is imperative that it works properly. To accomplish this goal we set up a database containing information about our users as well as their current location. When a user asks our application to get the list of other users nearby, we send a request to our server which uses a great-circle distance formula to find all the other users within a set radius (in our case, 5 miles). Our application presents users with the profiles of others in the area who have set their locations within the last half hour. The time constraint helps us to remove users who are inactive, but may have forgotten to log out.

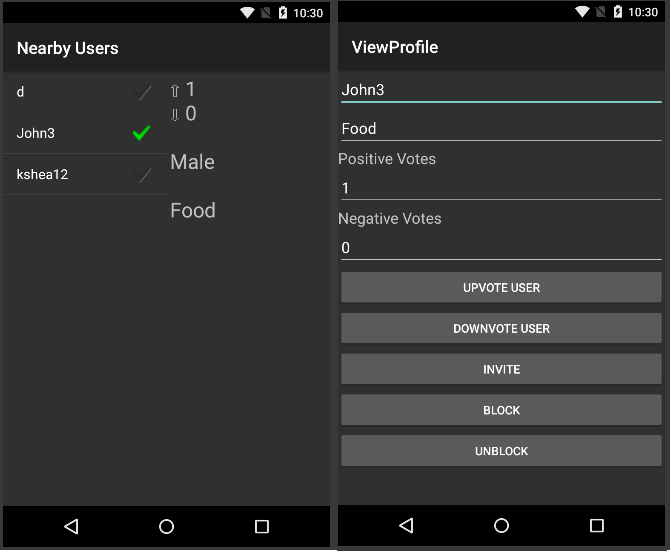
Presenting our users with nearby profiles allows the user to view the information about the other users and potentially send them a message. Contact is done as a messaging service through the app using Google Cloud Messaging (GCM). When a user sends a message, the message is first sent to our AWS server, from there we process it. The first thing we check for is to make sure that the users have not blocked each other. If they have not, we then match our custom username to a GCM regid, which is used by the Google servers. Once the regid(s) has been found (if they exist), the message and the regid(s) of the receiving user(s) are forwarded on to the GCM servers which handle the actual message delivery. Once a message has been forwarded by Google and received by our mobile application, we display a notification to the user, or perform some message specific action (such as opening an invite menu). We use messages, not only to communicate between users, but also to do utility tasks, such as notify a user of a group invite or if their group has been disbanded. If a user accepts an invite, a group is formed (or the user is added to a pre-existing group). Once a group has been formed, the users in the group may make use of the group messaging functionality to message all of the users in the group at once. There is no way for a user to be forcibly removed from a group, however, users are free to leave a group whenever they desire.

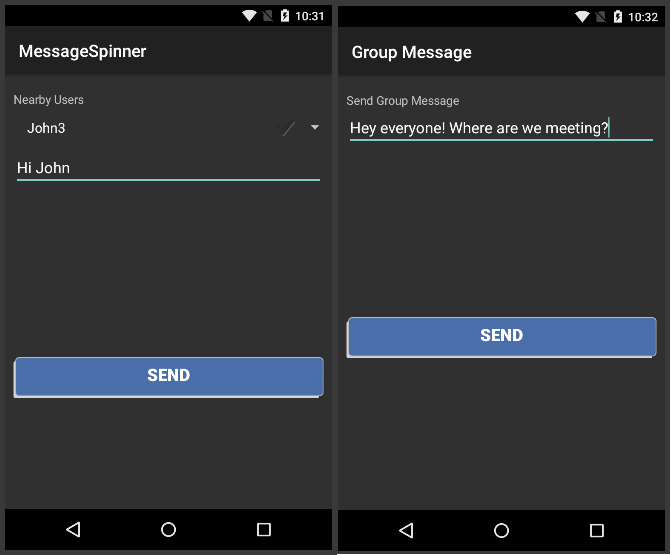
We utilized Amazon Web Service’s (AWS) free service tier to host our server side operations. Our server is running Linux, we are using Node.js for our webapi, BCrypt for securely hashing passwords, and MySQL for our data storage needs. Our database has evolved over time as we updated and refined what we wanted our application to be. We removed much of the personal information about users and added a lot of utility information. The only personal information that our database currently stores is the usernames, hashed passwords, bios and the gender of our users. However, we have had to add information which the user generates through their use of the app. This information includes: session tokens, the id of the group the user is in (if applicable), the users that the user has blocked, the rating of a user, the GCM regid, used to identify the user’s device and their latitude and longitude. We initially planned to store the interests and language of a user, to be used, if possible, for matching users based on more than location. However, we were unable to attain these stretch goals, and as such, we decided it would make more sense to remove these restrictions and simply allow the user to create a general bio where they could write any information they wanted.

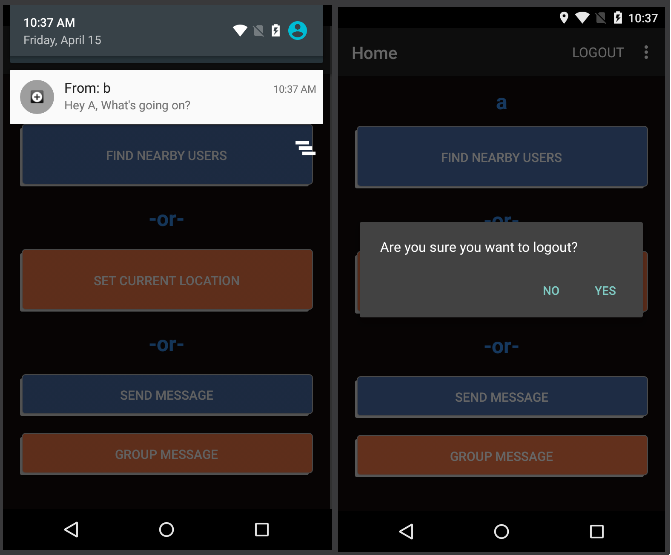
**Example user sign up and match GUIs:**

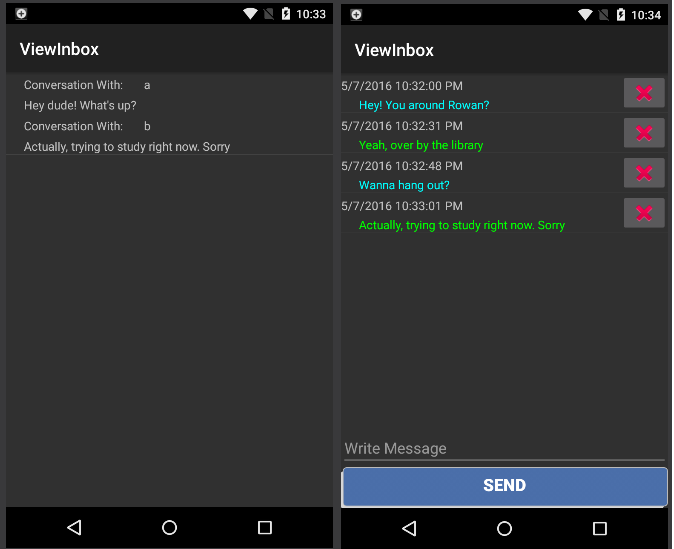










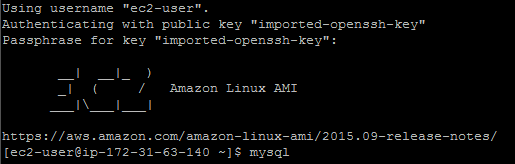


**Choices of IDE**

We originally were going to exclusively develop on the Android device, therefore we used the official IDE called Android Studio. We had then planned on attempting to create a cross platform app, which led us to an IDE called Xamarin, who provided us with a free, one year, student license.

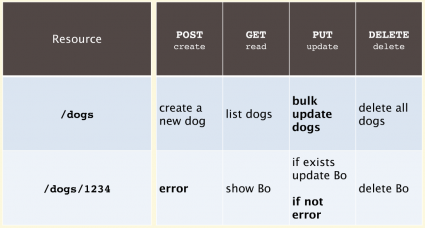
**Choices of Servers**

We thought about using Heroku as our web server provider as it did provide a free level though a little more limited than Amazon Web Service. Heroku highlights the ability to link GitHub files straight onto the server. This seemed promising but Heroku lacked a good tutorial and API to use it, so we looked into Amazon Web Service instead. AWS’s free tier service provides 24/7 server uptime and a relational database with respectable processing speed. They also had a very detailed walkthrough on how to setup the server, database, apache web service, and MySQL.

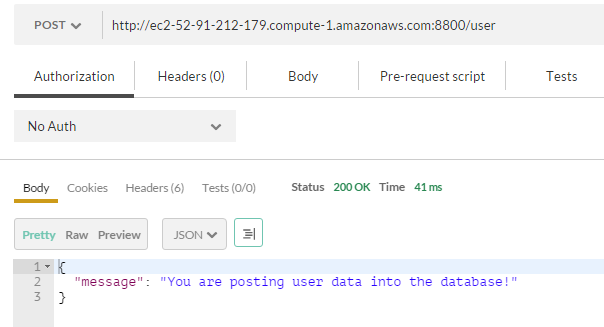


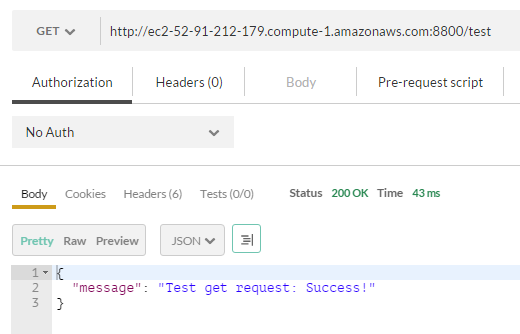
**Web API**

We are developing our web API using Node.js with the Express package framework, REST software architectural style and utilizing the JSON format of information passing. It will be designed RESTfully with nouns in the URI to avoid confusing verbs. The principle behind this is because it is cumbersome to have URI for actions such as “deletePerson” and “writePerson”. It is better to create noun keywords in the URI such as “Person” with a delete request will delete all persons. “Person/John” with a put request will add a person named John and “Person/John with delete request will delete John. This utilizes the four different requests used in RESTful calls.



**Sample Web API requests**

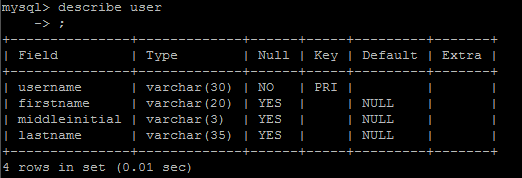


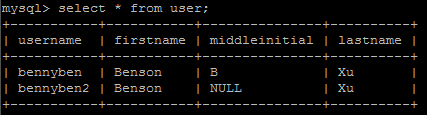


**Choices of Databases**

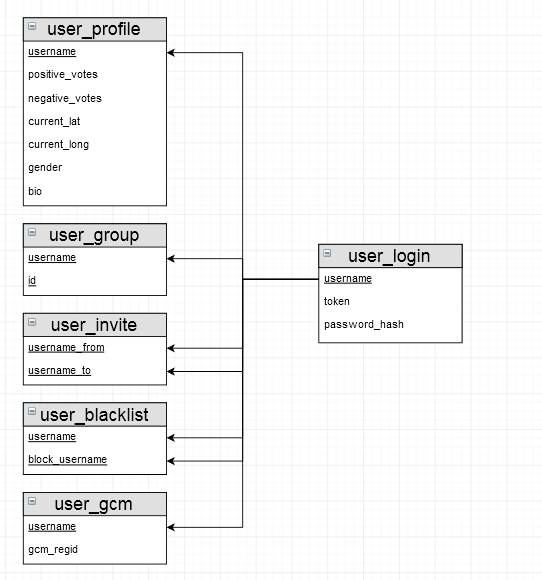
mySQL is offered by Amazon Web services and we choose it because we are most familiar with it. The database is relational so we can abstract our tables if needed.

**Test database tables**

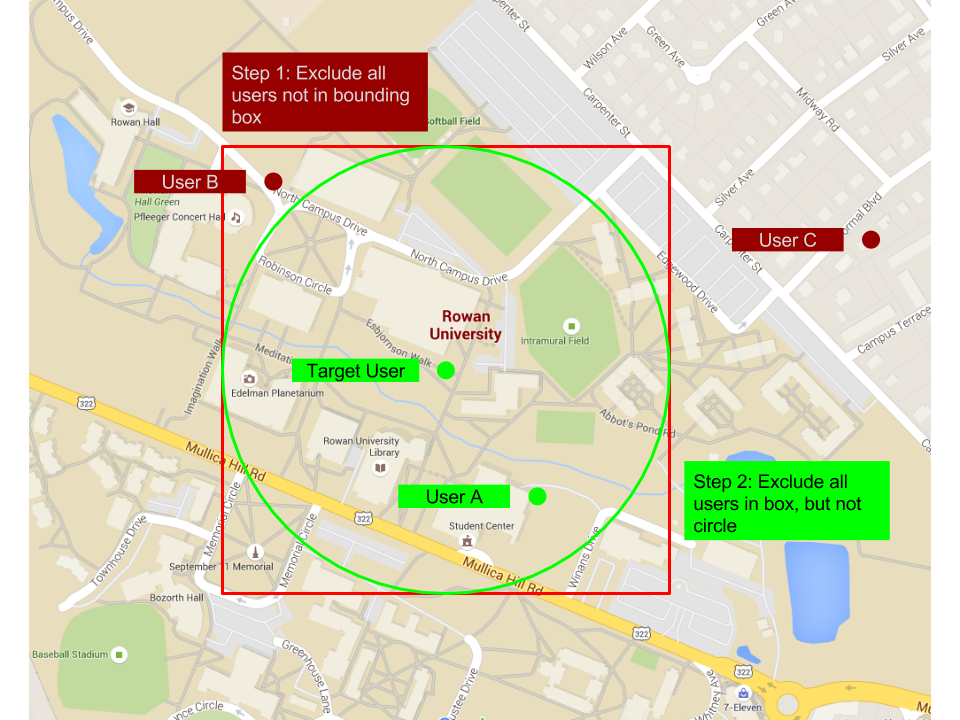




**Sample database table design**



**Geolocation matching overview**



This picture shows how our software will determine geolocation matches. In this first step, we exclude all points outside of the bounding box. This is a fairly efficient task to do and it will greatly reduce the number of possible matches. After we have excluded points using the box, we individually calculate the distance between the target point and the remaining points, discarding any that aren’t within the target radius.

In this example, user C would be excluded by step 1, users A and B would not. Step 2 would eliminate user B. And finally, the program would present the target user with user A as a potential match.

**References**

1. <http://janmatuschek.de/LatitudeLongitudeBoundingCoordinates>

**Document revisions**

This document was originally a Word document hosted on GitHub. It was then moved to Google Docs where the majority of the work was done. In order to see the revision history, the Google Doc can been found at this link: <https://docs.google.com/a/students.rowan.edu/document/d/1i59oqPVazMxky7acjp7WOxpwQclsC4_p24nLkEL3YfU/edit?usp=sharing>