Meet-Meet Design Document

The design for Meet-Meet is made up of several individual components. The first component will be a Graphical User Interface (GUI) that the user will interact with. The second component will be the c# code that will run on a user’s device. Finally, we will create the backend database to store user data for the long term. To help us meet these goals we will be using a program called Xamarin to compile c# code and provide a GUI builder. We will also be using the free service tier of Amazon Web Services (AWS) to host our server and database. 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 to the idea that we are able to efficiently find a user’s location, as well as easily finding users within the same area.

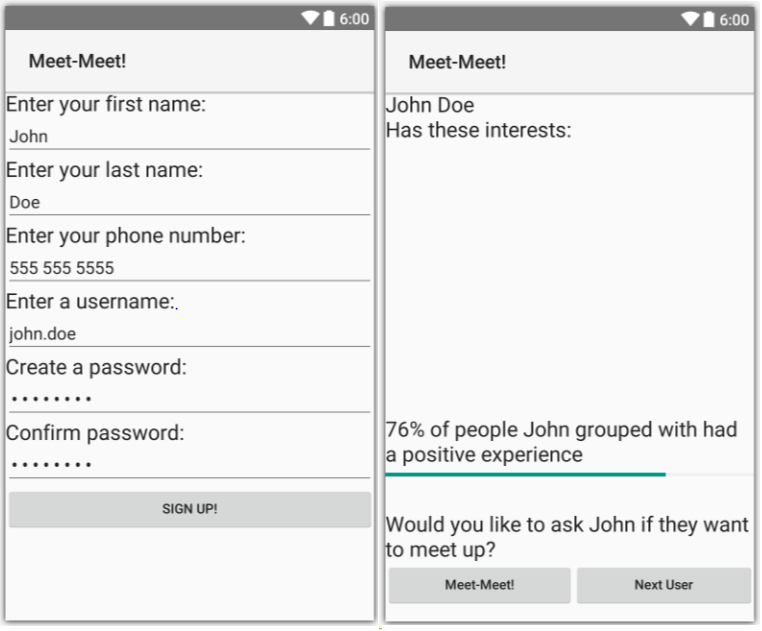
The GUI for Meet-Meet will be fairly simplistic, we will allow a user to sign up for the service in the app. The signup page will include a “sign up” button and the following fields: username, phone number, first name, last name, password and password confirmation. Users will also be able to view the profile of other users in the area, the profile screen will display the following information: Full name, the language(s) the user speaks, the user’s interest(s), the report rating of the user, and two buttons. The buttons will be used to either skip the current profile, or to contact the other user and see if they would be willing to message. If the other user agrees to communicate, their profile will change to have positive and negative feedback buttons, rather than the skip and message buttons. At our current time, we will not be including profile pictures, mainly because the users are in such 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 will be the backbone of our application and it is imperative that it works properly. Our application will present users with the profiles of others in the area. This will allow the user to view the interests listed on the other profiles and will let them contact the other user through the app. Contact will either be done as a messaging service through the app itself, or it will merely push a notification to the other user’s device and if that user agrees, the application will distribute the phone numbers of the two users to each other (In this scenario, no information will change hands until both users have consented). Once a group has been formed, one user must be designated as the leader, this will either happen in an automatic way, or the users may designate a leader themselves. Once a group has been formed and a leader has been designated, that new group will act in the same manner as an individual, and the application will allow the group to enter the grouping “pool” again and make more matches.

The information of a user profile will be stored in a database hosted on the AWS free tier of service. The database will also store the current location of a user which will be retrieved from the user’s device. There are two possibilities for the matching of locations, our first choice would be to find an API that will give us all the locations within a certain radius around a latitude and longitude. If we are unable to find an API to do the matching for us, we will implement the functionality ourselves. We have found a paper describing, in detail, the steps involved in finding points within a certain radius that we should be able to implement ourselves with minimal effort (Ref 1). The paper in question even goes so far as to show how to implement SQL queries to pull the required information from our database.

At our current time, we are attempting to use Amazon Web Service’s (AWS) free service tier to host our server side operations. We will use this to store user information in a database which we will pull from to suit our needs. The design of the database will be basic, as none of us have much experience with formal database coding. We will store custom information about a user including their name, phone number, username/email, a random number specific ID bestowed to the user in order to be used for salting their password hash, and finally, a hashed version of their password. In addition to custom information, we will require the user to choose one or more options from predetermined lists in the following categories: Interests and Languages. We must also store the current user rating, which will be composed of two parts: the positive and negative votes that the user has received in the past. Finally, we will temporarily store user-to-user messages.

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

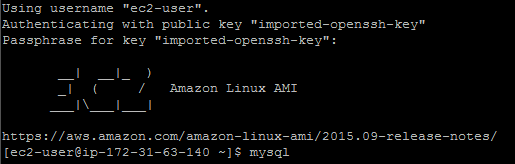


**Choices of IDE**

We originally were going to exclusively develop on the Android device, therefore we used the official IDE called Android Studio. Some group members had trouble with the IDE and used what Android Studio was based off of and that was IntelliJ. Then we decided to go to the cross platforming IDE for our app and decided to use Xamarin which provided us with a free one year license.

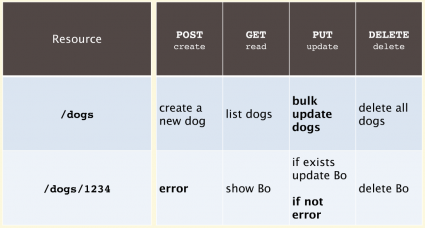
**Choices of Servers**

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.

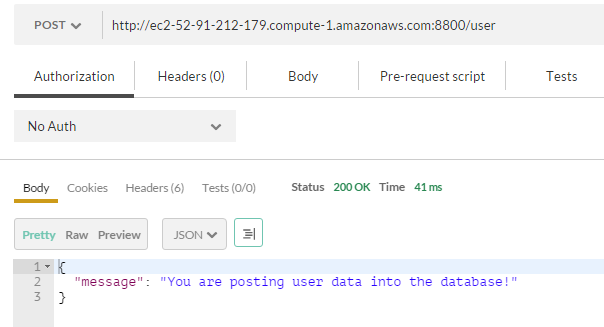


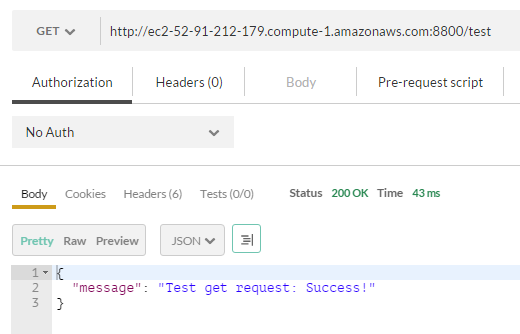
**Web API**

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**

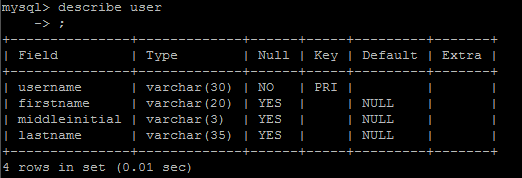


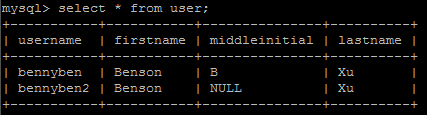


**Choices of Databases**

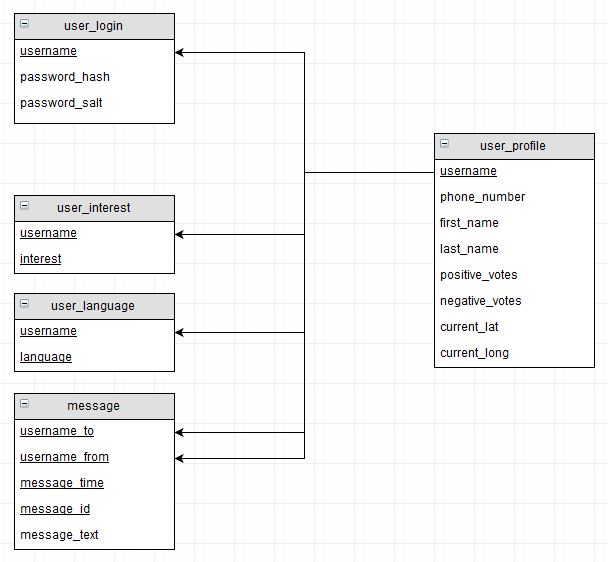
mySQL is offered by Amazon Web services and we choose it because we are most familiar with it. The database is relational so we can abstract our tables if needed.

**Test database tables**

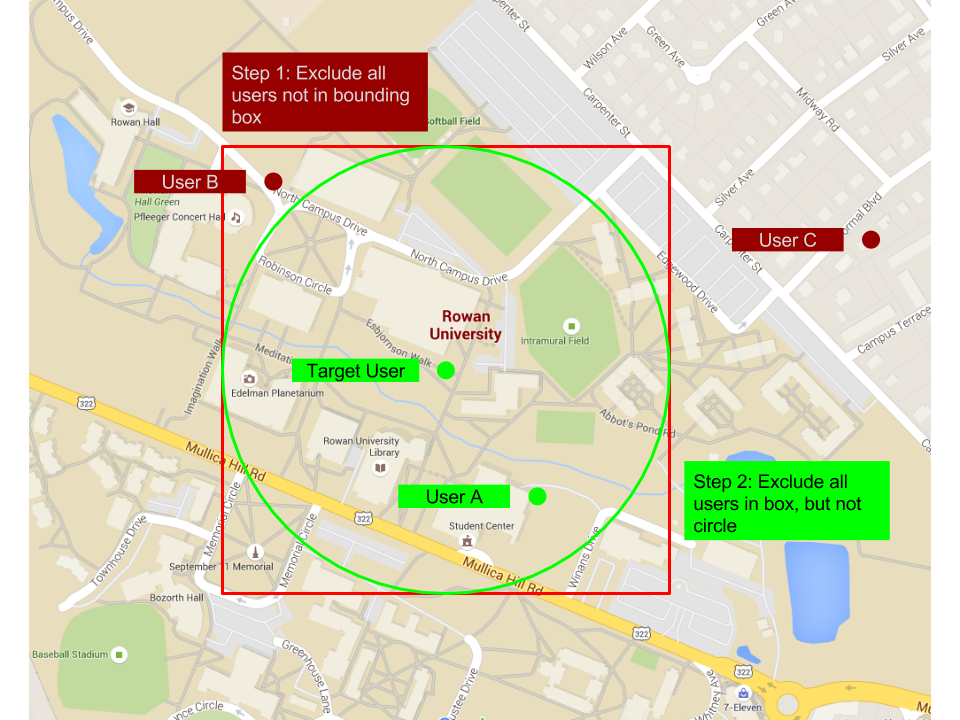




**Sample database table design**



**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>