CS 495: Progress Report 1
Tyler Allen, Karen Dana
09/15/2014

This week we have completed the Unified Modelling Language (UML) design and storyboard for the FriendFinder application. We have made the decision to split the design into three pieces: a storyboard layout for screens, a UML diagram for the server, and a UML diagram for the intermediate layer in the application. It is important to note that we have chosen Java as the language for our server implementation. Android requires Java (Dalvik), so we will be using Java for the full project. It is possible that we will explore Java 8 for the server implementation. The C/C++, and D languages are targeted at high performance applications. We chose Java over C/C++ because, while C/C++ is capable of running more efficiently, it would take us significantly longer to develop the server because C++ is a lower level language than Java, and we both have significantly more experience with Java. D was also considered but neither of us have ever used D, and we were not able to find a reliable database connector available in D during our research. All mentioned classes can be seen in the UML diagrams provided with this document.

The storyboard is a visual representation of each screen available in the application. It shows the appearance of screens as they will appear in the application. The storyboard also shows what order the screens will appear on based on which options on an individual screen are shown. Taking into account the project plans previously that we previously created together, Karen drew the storyboard.

The server UML class diagram contains our current design for the server. The RequestServer class will contain a ServerSocket that awaits incoming requests from instances of the application. When a request is received, the data received will be used to create a Request object. The Request class will contain logic for authenticating users, performing necessary calculations and database lookups required by the application's request, and then generating a Response object. The Request object will likely use the Builder Method design pattern to return the appropriate subclass of Request. For example, a Sync object will be generated when the application makes a request to refresh its current store of information. The Response class will be capable of providing data in a format that can be sent back over the socket. We have not chosen what manner the Request will provide the data back to the socket because we need to do more research on idioms and best practices when networking with an Android device. We will need to do additional research on an appropriate API for encrypting user credentials. We are currently planning to use Java Database Connectivity (JDBC) to connect with our database, although more research will be done to verify that it is the best choice. Tyler used the UML tool available at yuml.me to generate the UML for the server.

We made a decision to split the application into 3 different design documents. We found that it the majority of code associated with creating the visual portion of an Android application is boilerplate code. The actual visualization is more important, so we created a storyboard in its place. There is logic associated with all screens however, which we chose to abstract. This includes code for communicating with the server, as well as wrapper classes for wrapping data received from the Server in order to make the data accessible in a manner that is easy to code and allows for uniform data management across the different screens. This is the “intermediate layer” between the application screens and the server. We also detail the use of Android APIs, such as the Google Maps API, that will be used on the screens but are not necessarily required for communication with the server. We have chosen the Google Maps API for now but we will research Open Street Maps in the future. We both created the UML for this portion using the Google Drive App, *draw.io*.