CS 495: Capstone Progress Report 5
Karen Dana
18 November 2014

On the server, I installed CentOS 7 with the minimal installation to make it lightweight. On top of CentOS, I installed Secure Shell (SSH), Apache 2.4.6, OpenSSL, and MariaDB [1]. First, I set the server's Internet Protocol address (IP address) to be static so it could easily be referenced. Then I installed SSH so I could remotely log on to the server. I had to set up port-forwarding on my router to allow the port to be accessed from outside the local network. Then I installed Apache 2.4.6 to handle Hypertext Transfer Protocol Secure (HTTPS). I installed OpenSSL, used it to generate Secure Socket Layer (SSL) certificates, and then configured the SSL configuration settings for Apache to use SSL and give it the path to the certificate and certificate key that were generated. For the database, I installed MariaDB, the CentOS replacement for MySQL, and created a new database for the *FriendFinder* application. I then made tables for the database according to the database schema that I created. Data was then added to the tables to test them and ensure they were functioning correctly.

For the Android side, I implemented the authentication token system using the classes `ServerAuthentication`, the Android class `AccountManager`, `AccountAuthenticator`, and using `Login` as the `AccountAuthenticatorActivity`. Whenever a user accesses the application, `getAuthToken()` is called from `AccountManager`, which in turn calls `AccountAuthenticator`. If a token exists, the `AccountAuthenticator` returns it, but if it doesn't, it will send a request to `ServerAuthentication` which will try to sign the user in and return an authentication token, but if that fails it will send the user to the login screen where they can re-enter their credentials. This was accomplished using the Android Developer pages for these classes and the article "Write your own Android Authenticator" [2].

To get the data from the application to the server, I set up a `KeyStore` that holds the certificate of the server so the application can have access to it. Then I have a class interact with the `KeyStore` so it can access the certificate in order to connect to the server. Another class will get the `SSLContext` from that `KeyStore` which will get passed to the main class that then creates a request using the package `Volley`.

[1] https://mariadb.org/
[2] http://udinic.wordpress.com/2013/04/24/write-your-own-android-authenticator/