RW748: Final Report

Kyle Titus

17 August 2014

1 Introduction

The following document describes the final project for RW748: Software Development for Mobile Devices. The app "LeaderBoard" was implemented.

The bulk of the functionality decided upon at the start of this project [3] that was neglected due to time constraints. The following sections outline the work that was done.

2 High-level design

2.1 UI

The app consists of 2 main UI components:

- Login screen
- Main tab Layout

2.1.1 Login screen

The Login screen consists of a basic login form for inserting a username and password. The "Login" button submits the form details to the server and responds correctly to unregistered usernames and incorrect passwords.

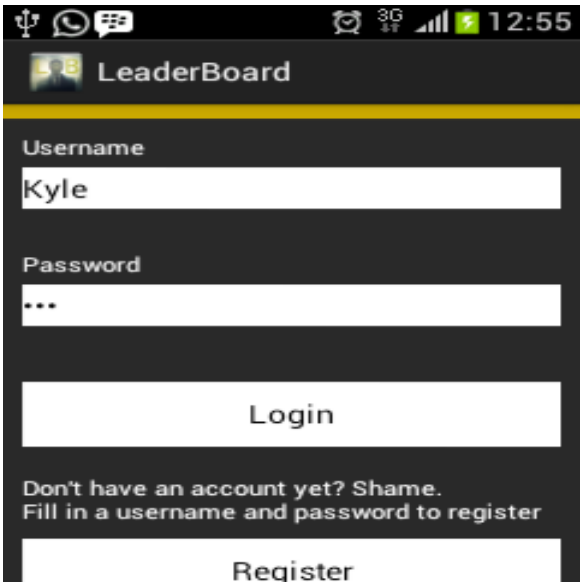

Figure 1: Login screen

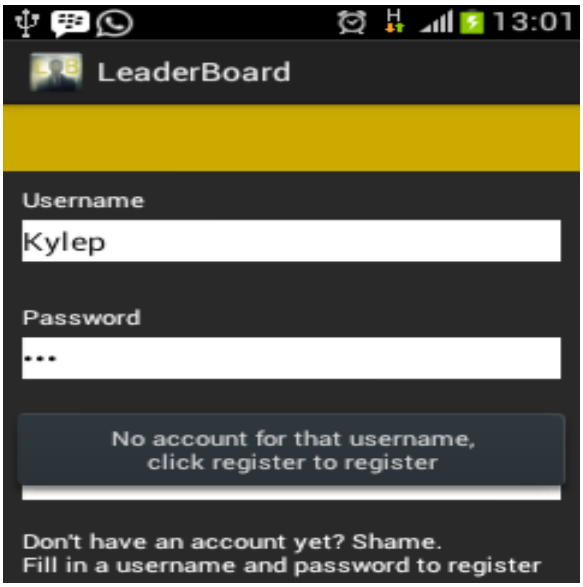

Figure 2: Login screen: Unregistered username

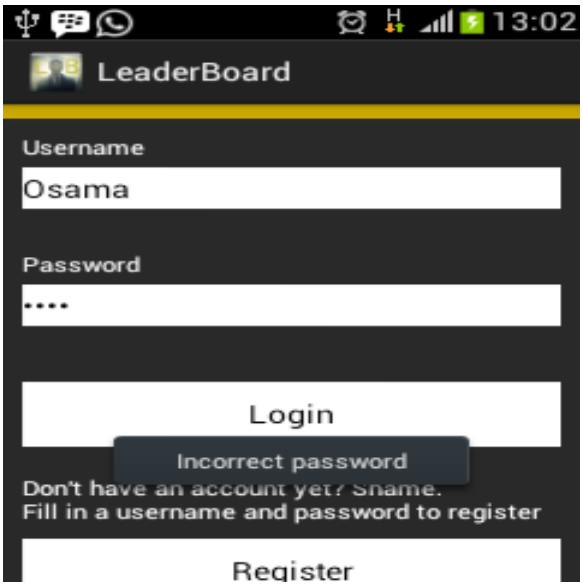

Figure 3: Login screen: Incorrect username

Once the correct username and password is inserted, the user is taken to the main tab layout.

2.1.2 Main tab layout

The Main tab layout consists of three tabs.

- Feeds
- Rivals
- Leaderboard

The rivals is the tab selected by default as the Main Activity is started. This is a list view which contains all the rivals for the currently logged in user. Each item in the list view is a linear layout that contains an image view (an avatar), text view (the rival's name), and another image view (a button to used to challenge the rival. The avatar upload function was not implemented so a instead a default avatar is shown for each rival.

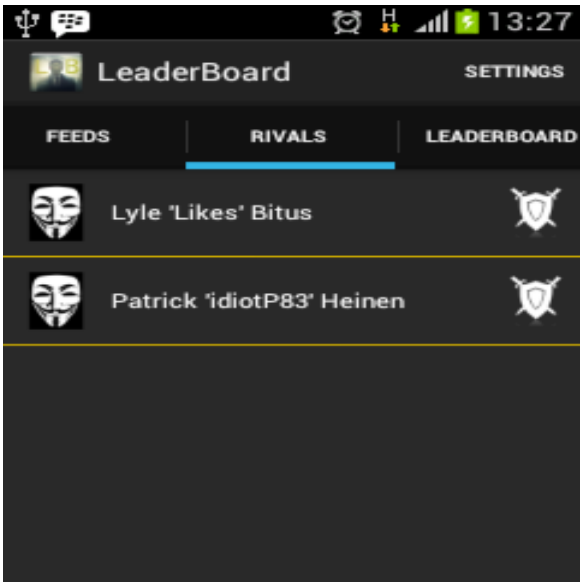

Figure 4: Main tab layout: Rivals tab

Clicking on a rival allows the user to challenge the rival at a particular game. A pop up window will allow the user to choose the game with a spinner UI element. Once the submit button is clicked the challenge is sent to the selected rival. The cancel button cancels the challenge request.

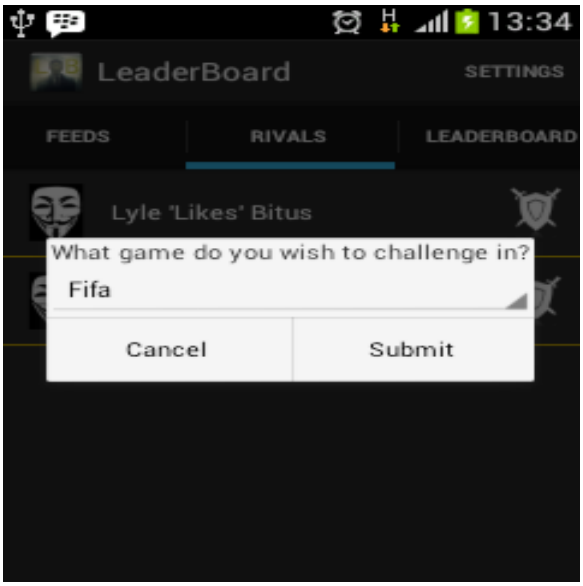

Figure 5: Submitting a challenge

Once a challenge is submitted the rival that has been challenged will be notified with a Toast. The Toast specifies the rival that issued the challenge and the game that he/she wants to challenge in.

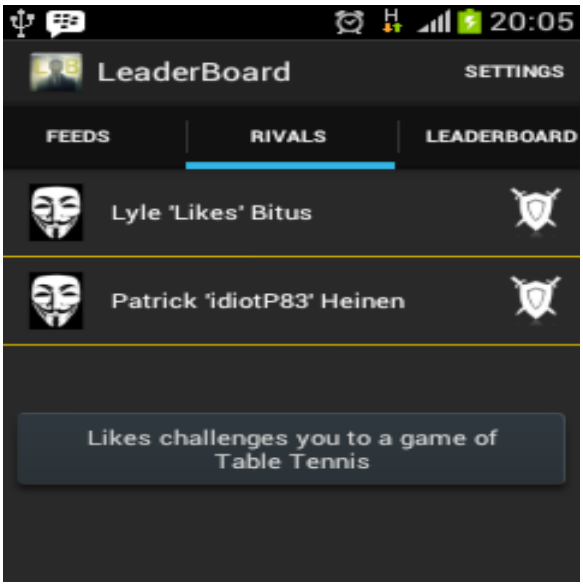

Figure 6: Toast showing a challenge request

The functionality for accepting and scoring a challenge was not implemented.

The Leaderboard tab shows where the user ranks among his/her rivals. The app contains dummy data for this tab. This does not function fully because the scoring function was not implemented (mentioned above).

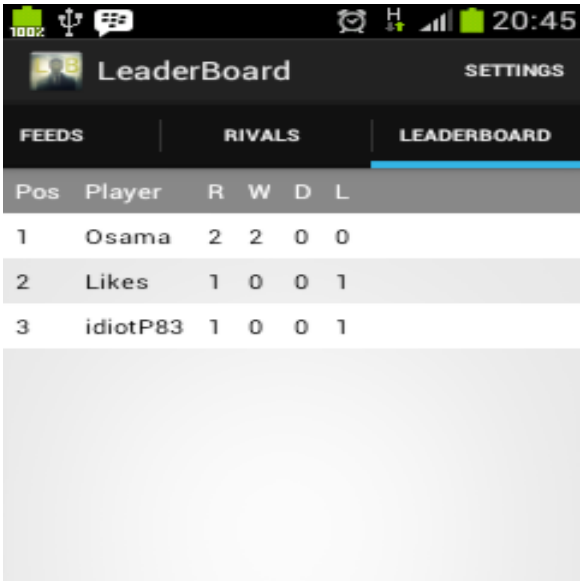

Figure 7: Main tab layout: Leaderboard tab

The feeds tab shows the user events that have happened. Currently, it only shows the user challenge requests due to lack of functionality.

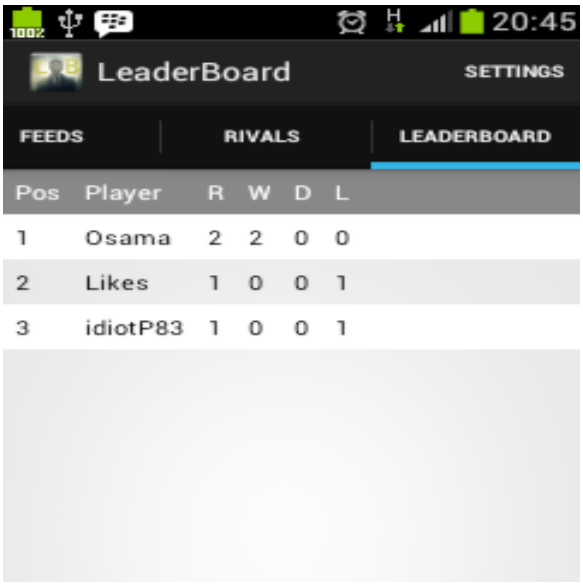

Figure 8: Main tab layout: Leaderboard tab

3 Backend

The server for LeaderBoard is hosted by a free web hosting company called Free Web Hosting Area [1]. The database consists of 7 tables:

- Challenges
- Feeds
- Games
- Leaderboard
- Results
- Rivals
- Users

The app currently only uses the Challenges, Games, Rivals and Users tables. Challenges contains information for each challenge issued, who they are issued by (referencing Users), who they are issued to (referencing Users), and what game it is (referencing Games). Games contain the details of each game (name,type) and are referenced by other tables by their id. The rivals table indicates which users are rivals by rows that simply contain the id's of two users that are rivals which refer to entries in the Users table. This is a symmetrical relationship. Users contain all user's information.

The server uses simple `echo` messages to respond to request from the client app.

4 Android Functionality

Content providers were used for the following content:

- Users
- Games
- Challenges
- Feeds
- Leaderboard

The provider and db helper functions and data contract is implemented for each of these in the `UserProvider.java`, `UserDbHelper.java`, and `UserContract.java` classes respectively.

```
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    Cursor retCursor;
    switch (sUriMatcher.match(uri)) {

        case USER: {
            retCursor = mOpenHelper.getReadableDatabase().query(
                    UserContract.UserEntry.TABLE_NAME,
                    projection,
                    selection,
                    selectionArgs,
                    null,
                    null,
                    sortOrder
            );
            break;
        }
        case USER_WITH_ID: {
            retCursor = mOpenHelper.getReadableDatabase().query(
                    UserContract.UserEntry.TABLE_NAME,
                    projection,
                    UserContract.UserEntry.COLUMN_USER_ID + " = '" + ContentUris.parseId(uri) + "'",
                    null,
                    null,
                    null,
                    sortOrder
            );
            break;

        }

        case GAME: {
            retCursor = mOpenHelper.getReadableDatabase().query(
                    UserContract.GamesEntry.TABLE_NAME,
                    projection,
                    selection,
                    selectionArgs,
                    null,
                    null,
                    sortOrder
            );
            break;
        }
```

Figure 9: Code snippet from UserProvider.java

```java
public UserDbHelper(Context context) { super(context, DATABASE_NAME, null, DATABASE_VERSION); }

@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {

    final String SQL_CREATE_GAMES_TABLE = "CREATE TABLE " + GamesEntry.TABLE_NAME + " (" +
            GamesEntry.COLUMN_GAME_ID + " INTEGER PRIMARY KEY," +
            GamesEntry.COLUMN_NAME + " TEXT UNIQUE NOT NULL, " +
            GamesEntry.COLUMN_TYPE + " TEXT NOT NULL" +
            " );";

    final String SQL_CREATE_USER_TABLE = "CREATE TABLE " + UserEntry.TABLE_NAME + " (" +
            // Why AutoIncrement here, and not above?
            // Unique keys will be auto-generated in either case.  But for weather
            // forecasting, it's reasonable to assume the user will want information
            // for a certain date and all dates *following*, so the forecast data
            // should be sorted accordingly.
            UserEntry.COLUMN_USER_ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +

            // the ID of the location entry associated with this weather data
            UserEntry.COLUMN_NAME + " TEXT NOT NULL, " +
            UserEntry.COLUMN_SURNAME + " TEXT NOT NULL, " +
            UserEntry.COLUMN_USERNAME + " TEXT NOT NULL, " +
            UserEntry.COLUMN_DATE_JOINED + " TEXT NOT NULL);";

    final String SQL_CREATE_CHALLENGES_TABLE = "CREATE TABLE " + ChallengesEntry.TABLE_NAME + " (" +
            ChallengesEntry._ID + " INTEGER PRIMARY KEY," +

            ChallengesEntry.COLUMN_USER1_KEY + " INTEGER NOT NULL, " +
            ChallengesEntry.COLUMN_USER2_KEY + " INTEGER NOT NULL, " +
            ChallengesEntry.COLUMN_GAME_KEY  + " INTEGER NOT NULL, " +
            ChallengesEntry.COLUMN_ACCEPTED + " INTEGER NOT NULL, " +

            " FOREIGN KEY (" + ChallengesEntry.COLUMN_USER1_KEY + ") REFERENCES " +
            UserEntry.TABLE_NAME  + "(" + UserEntry.COLUMN_USER_ID + "), " +

            " FOREIGN KEY (" + ChallengesEntry.COLUMN_USER2_KEY + ") REFERENCES " +
            UserEntry.TABLE_NAME  + "(" + UserEntry.COLUMN_USER_ID + "), " +

            " FOREIGN KEY (" + ChallengesEntry.COLUMN_GAME_KEY + ") REFERENCES " +
            GamesEntry.TABLE_NAME + "(" + GamesEntry.COLUMN_GAME_ID + "));";
```

Figure 10: Code snippet from UserDbHelper.java

```java
public static final class FeedsEntry implements BaseColumns {

    public static final Uri CONTENT_URI =
            BASE_CONTENT_URI.buildUpon().appendPath(PATH_FEED).build();

    public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/" + CONTENT_AUTHORITY + "/" + PATH_FEED;
    public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/" + CONTENT_AUTHORITY + "/" + PATH_FEED;

    public static final String TABLE_NAME = "Feeds";

    public static final String COLUMN_USER1_KEY = "user1_id";

    public static final String COLUMN_USER2_KEY = "user2_id";

    public static final String COLUMN_FEED = "event";

    public static final String COLUMN_FEED_TIME = "event_time";

    public static Uri buildLocationUri(long id) {
        return ContentUris.withAppendedId(CONTENT_URI, id);
    }

}

public static final class LeaderboardEntry implements BaseColumns {

    public static final Uri CONTENT_URI =
            BASE_CONTENT_URI.buildUpon().appendPath(PATH_LEADERBOARD).build();

    public static final String CONTENT_TYPE =
            "vnd.android.cursor.dir/" + CONTENT_AUTHORITY + "/" + PATH_LEADERBOARD;
    public static final String CONTENT_ITEM_TYPE =
            "vnd.android.cursor.item/" + CONTENT_AUTHORITY + "/" + PATH_LEADERBOARD;

    public static final String TABLE_NAME = "Leaderboard";

    public static final String COLUMN_USER_KEY = "user_id";

    public static final String COLUMN_RIVALS = "rivals";

    public static final String COLUMN_WINS = "wins";

    public static final String COLUMN_LOSSES = "losses";

    public static final String COLUMN_POSITION = "position";

    public static Uri buildLocationUri(long id) {
        return ContentUris.withAppendedId(CONTENT_URI, id);
```

Figure 11: Code snippet from UserContract.java

An AsyncTask was implemented to do the login request and another to fetch the rivals for the current user. The login response was processed on the `onPostExecute` method of the AsyncTask. The AsyncTask for fetching rivals is tied to the array adapter used to show the rivals as list view on the rivals tab.

A service is used to send challenges to rivals and to listen for challenges from other rivals. The listening for challenges is set as an alarm that goes off every 10 seconds to check the Challenges table for updates.

```
@Override
public void onDestroy() {

        AlarmManager alarm = (AlarmManager) getSystemService(ALARM_SERVICE);
        alarm.set(
                alarm.RTC_WAKEUP,
                System.currentTimeMillis() + (7000),
                PendingIntent.getService(this, 0, new Intent(this, ChallengeService.class), 0)
        );
}
```

Figure 12: Alarm for service that listens for challenges

A broadcast receiver is implemented to listen for booting of the android device to start the above mentioned service.

A share action provider is implemented. A share option was added to the action bar. The content shared is simply a an advertising message about the Leaderboard app.

5 Testing

JUnit testing was done for the SQLite database creations, inserts, updates, and deletes.

Robotium [2] was used to do simple user UI testing. Robotium allows users to record tasks done with the app and allows automatic rerunning of these tasks to ensure that the app does not crash unexpectedly.

6 References

References

[1] *Free Web Hosting Area* `http://www.freewebhostingarea.com/`

[2] *Robotium* `http://robotium.com/`

[3] K. Titus, *RW748: Project Outline*