CS 495: Capstone Report 5
Tyler Allen
11/17/2014

The goals for this report were to switch over to JavaScript Object Notation (JSON) for networking support, a content provider would be constructed and functional, and users will be able to view groups and events[1]. Groups are not functional yet, but the other requirements are in place. In addition, old code is being progressively refactored and missing documentation is being added in place.

JSON has been implemented on both server and client side. The server now uses a JSON model to store information to be sent to the client as well as retrieve information from the client.

A `ContentProvider` has been created that allow the client to use and modify a SQLite3[2] database. This client side database is a projection of the server side database containing only the data that is relevant to the user. `ContentProviders` are a new topic that required a good amount of research before starting. The decision to use SQLite was made up front, but actually implementing a `ContentProvider` backed by an SQLite database proved to be challenging. The documentation for `ContentProviders` is written abstractly, and provides only method stubs[3][4]. After implementing a `ContentProvider`, it proved difficult to test without an actual scenario. Implementing user events was a logical decision for testing the content provider. The difficulties implementing this will be covered from the server to client side in the following paragraphs.

Creating more complex queries, such as relational JOINS, using Java Database Connector (JDBC) also required research, and proved to be more difficult than typical Structured Query Language statements[5][6]. Substituting variable values into queries requires the usage of the JDBC Application Programming Interface (API) above and beyond basic SQL. The more complex statements are required in order to filter events down to only the events that are relevant to the user.

Creating a parseable format for sending database table data to the client required exploration of the `ResultSetMetaData` API in addition to the typical `ResultSet` API[7][8]; this

---

[1] "JSON." 2003. 17 Nov. 2014 <http://www.json.org/>
[2] "android.database.sqlite | Android Developers." 2009. 17 Nov. 2014 <http://developer.android.com/reference/android/database/sqlite/package-summary.html>
[3] "Content Provider Basics | Android Developers." 2012. 17 Nov. 2014 <http://developer.android.com/guide/topics/providers/content-provider-basics.html>
[4] "Creating a Content Provider | Android Developers." 2012. 17 Nov. 2014 <http://developer.android.com/guide/topics/providers/content-provider-creating.html>
[5] "JDBC Basics - Oracle Documentation." 2011. 17 Nov. 2014 <http://docs.oracle.com/javase/tutorial/jdbc/basics>
[6] "SQL Introduction - W3Schools." 17 Nov. 2014 <http://www.w3schools.com/sql/sql_intro.asp>
[7] "ResultSet (Java Platform SE 7) - Oracle Documentation." 2012. 17 Nov. 2014 <https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html>

allowed the inclusion of column names in the format, which are required for the client to be able to abstractly parse data from the format. A number of variations of the format were tested, but a new one may need to be devised in the future; due to nature of text fields like “event descriptions”, most character delimiters have the possibility of appearing in the text fields. It currently works under the condition that users do not use colons, commas, or tildas in their text fields.

An additional difficulty became apparent in the slight differences between Dalvik and standard Java. A method of writing data as a stream of bytes is interpreted correctly by the Java-based server, but the same implementation causes the client to hang. This is related to the data format, as well, as Newline characters seem to be interpreted slightly differently depending on the system and how data is written to the socket.

Writing data to the SQLite database client side created a number of difficulties. The SQLite database error messages are uninformative, and any given issue could be attributed to an error in the database creation, the content Uniform Resource Identifier (URI) required to access the database, or by an error in the data actually being sent to the SQLite handler. When added to unfamiliarity with the nature of `ContentProvider` errors, other bugs became obfuscated by the SQLite error messages.

Finally, a `ListView` was implemented to the events page. This requires a custom implementation of a `ListAdapter`[9][10]. As of this point, events are successfully retrieved from the database and are created in a `ListView` when the user selects the home page. Users may select an event and view details. Currently, the location functionality and description are not functioning on the event page. Groups were not completed this week due to the difficulties mentioned above, but the framework is in place for groups to be added in a short amount of time. Groups have related data, are accessed in the same manner, and are transmitted in the same fashion as events. The data synchronization process is also written in such a way that the `SyncAdapter` updates the `ContentProvider` in an abstract manner, such that the table that is being synced does not require additional code; the server-side data synchronization code is written in the same manner. Therefore, groups should be straightforward because the framework is already in place.

---

[8] "ResultSetMetaData - Oracle Documentation." 2012. 17 Nov. 2014 <https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSetMetaData.html>

[9] "List View | Android Developers." 2012. 17 Nov. 2014 <http://developer.android.com/guide/topics/ui/layout/listview.html>

[10] "ListAdapter | Android Developers." 2009. 17 Nov. 2014 <http://developer.android.com/reference/android/widget/ListAdapter.html>