

Report of CS 522 Assignment 7

Name: Yunfeng Wei

CWID: 10394963

E-mail: ywei14@stevens.edu

Video: In the ZIP archive file

Part 1: Cloud Chat App:

1. Create a new class called Client, the Client stores the client's id, name and UUID in the app as Figure 1.1 shows. Create a new class called ClientContract to operate the database related operation as Figure 1.2

```
public class Client implements Parcelable {
    public long id;
    public String name;
    public UUID uuid;

    @Override
    public int describeContents() { return 0; }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeLong(id);
        dest.writeString(name);
        ParcelUuid parcelUuid = new ParcelUuid(uuid);
        dest.writeParcelable(parcelUuid, flags);
    }

    public Client(long id, String name, UUID uuid) {
        this.id = id;
        this.name = name;
        this.uuid = uuid;
    }

    public Client(String name, UUID uuid) {
        this.id = 0;
        this.name = name;
        this.uuid = uuid;
    }

    public Client(String name) {
        this.id = 0;
        this.name = name;
        this.uuid = null;
    }

    public Client(Parcel parcel) {
        this.id = parcel.readLong();
        this.name = parcel.readString();
        ParcelUuid parcelUuid = parcel.readParcelable(ParcelUuid.class.getClassLoader());
        this.uuid = parcelUuid.getUuid();
    }
}
```

Figure 1.1

```

public class ClientContract {
    public static final String CLIENT_ID = "id";
    public static final String NAME = "name";
    public static final String UUID = "uuid";

    public static final String TABLE_NAME = "Clients";
    public static final String CONTENT = "Client";

    public static final Uri CONTENT_URI = MessageDbProvider.CONTENT_URI(MessageDbProvider.AUTHORITY, TABLE_NAME);

    public static Uri CONTENT_URI(String id) {
        return MessageDbProvider.withExtendedPath(CONTENT_URI, id);
    }

    public static String CONTENT_PATH = MessageDbProvider.CONTENT_PATH(CONTENT_URI); // Messages
    public static String CONTENT_ITEM_PATH = MessageDbProvider.CONTENT_PATH(CONTENT_URI("#")); // Messages/#

    public static long getClientId(Uri uri) { return Long.parseLong(uri.getLastPathSegment()); }

    public static long getClientId(Cursor cursor) {
        return cursor.getLong(cursor.getColumnIndexOrThrow(CLIENT_ID));
    }

    public static void setClientId(ContentValues values, long id) { values.put(CLIENT_ID, id); }

    public static String getName(Cursor cursor) {
        return cursor.getString(cursor.getColumnIndexOrThrow(NAME));
    }

    public static void setName(ContentValues values, String name) { values.put(NAME, name); }

    public static java.util.UUID getUUID (Cursor cursor) {
        String uuid = cursor.getString(cursor.getColumnIndexOrThrow(UUID));
        if (uuid.equals("")) {
            return null;
        } else {
            return java.util.UUID.fromString(uuid);
        }
    }
}

```

Figure 1.2

2. Modify the DatabaseHelper to create a new table called “Clients” as Figure 2.1. And Modify the MessageDbProvider to support insert, query, update, delete in the “Clients” table. Figure 2.2 shows the how the Content Provider query all messages from the database.

```

public class DatabaseHelper extends SQLiteOpenHelper {
    public static final String TAG = DatabaseHelper.class.getSimpleName();

    private static final String MESSAGE_TABLE_CREATE =
        "CREATE TABLE " + MessageContract.TABLE_NAME + " ( " + MessageContract.MESSAGE_ID + " INTEGER PRIMARY KEY AUTOINCREME\n"
        + MessageContract.CHATROOM + " TEXT NOT NULL, " +\n"
        + MessageContract.MESSAGE_TEXT + " TEXT NOT NULL, " + MessageContract.TIMESTAMP + " INTEGER NOT NULL, " +\n"
        + MessageContract.SEQNUM + " INTEGER NOT NULL, " +\n"
        + MessageContract.SENDER_ID + " INTEGER NOT NULL, " + "FOREIGN KEY (" + MessageContract.SENDER_ID + ") REFERENC\n"
        + "(" + ClientContract.CLIENT_ID + ") ON DELETE CASCADE));";

    private static final String CLIENT_TABLE_CREATE =
        "CREATE TABLE " + ClientContract.TABLE_NAME + " ( " + ClientContract.CLIENT_ID + " INTEGER PRIMARY KEY, " +\n"
        + ClientContract.NAME + " TEXT NOT NULL, " + ClientContract.UUID + " TEXT );";

    private static final String CREATE_INDEX =
        "CREATE INDEX MessageIndex ON " + MessageContract.TABLE_NAME + "(" + MessageContract.SENDER_ID + ")";

    public DatabaseHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("PRAGMA foreign_keys = ON");
        db.execSQL(MESSAGE_TABLE_CREATE);
        db.execSQL(CLIENT_TABLE_CREATE);
        db.execSQL(CREATE_INDEX);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.v(TAG, "Upgrading from " + oldVersion + " to " + newVersion);
        db.execSQL("DROP TABLE IF EXISTS " + MessageContract.TABLE_NAME);
        db.execSQL("DROP TABLE IF EXISTS " + ClientContract.TABLE_NAME);
        onCreate(db);
    }
}

```

Figure 2.1

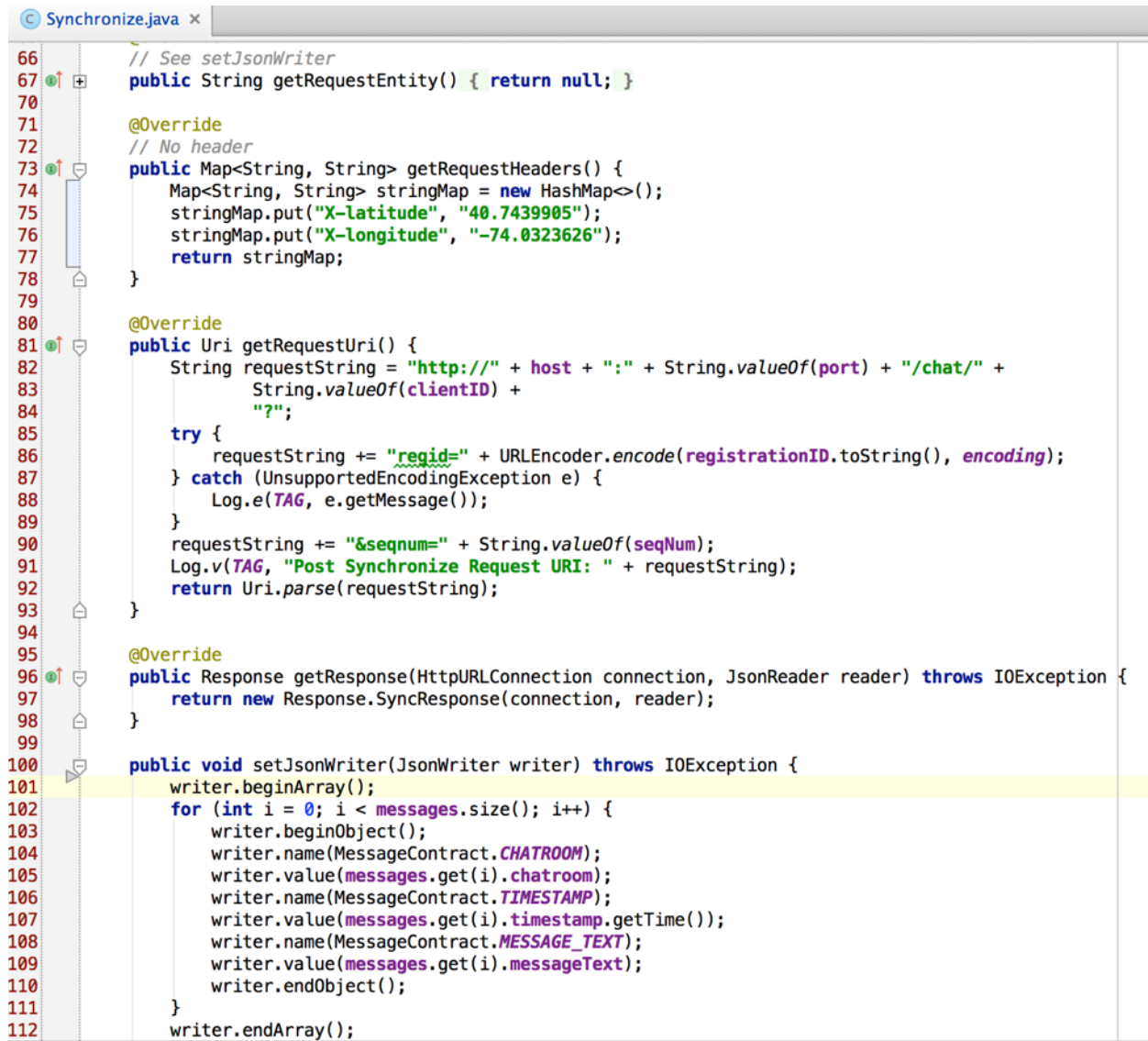
```

@Override
public Cursor query(Uri uri, String[] projection, String selection,
    String[] selectionArgs, String sortOrder) {
    SQLiteDatabase database;
    try {
        database = databaseHelper.getWritableDatabase();
    } catch (SQLException e) {
        database = databaseHelper.getReadableDatabase();
    }
    Cursor cursor;
    String query;
    switch (uriMatcher.match(uri)) {
        case MESSAGE_ALL_ROWS: // return all messages
            if (projection == null) {
                Log.i(TAG, "Query all messages");
                query = "SELECT " + MessageContract.TABLE_NAME + "." + MessageContract.MESSAGE_ID + " AS " + MessageContract.MESSAGE_ID + ", " +
                    MessageContract.TABLE_NAME + "." + MessageContract.CHATROOM + " AS " + MessageContract.CHATROOM + ", " +
                    MessageContract.TABLE_NAME + "." + MessageContract.MESSAGE_TEXT + " AS " + MessageContract.MESSAGE_TEXT + ", " +
                    MessageContract.TABLE_NAME + "." + MessageContract.TIMESTAMP + " AS " + MessageContract.TIMESTAMP + ", " +
                    MessageContract.TABLE_NAME + "." + MessageContract.SEQNUM + " AS " + MessageContract.SEQNUM + ", " +
                    ClientContract.TABLE_NAME + "." + ClientContract.NAME + " AS " + ClientContract.NAME + ", " +
                    MessageContract.TABLE_NAME + "." + MessageContract.SENDER_ID + " AS " + MessageContract.SENDER_ID +
                    " FROM " + ClientContract.TABLE_NAME + " LEFT OUTER JOIN " + MessageContract.TABLE_NAME + " ON " +
                    ClientContract.TABLE_NAME + "." + ClientContract.CLIENT_ID + " = " + MessageContract.TABLE_NAME + "." +
                    " ORDER BY " + MessageContract.TIMESTAMP + ";";
                cursor = database.rawQuery(query, null);
                cursor.setNotificationUri(getContext().getContentResolver(), uri);
                return cursor;
            } else {
                Log.i(TAG, "Query a particular message");
                return database.query(MessageContract.TABLE_NAME, projection, selection, selectionArgs, null, null, sortOrder);
            }
        case MESSAGE_SINGLE_ROW: // return a message
            Log.i(TAG, "Query a message by _id");
            query = "SELECT " + MessageContract.TABLE_NAME + "." + MessageContract.MESSAGE_ID + " AS " + MessageContract.MESSAGE_ID + ", " +
                MessageContract.TABLE_NAME + "." + MessageContract.CHATROOM + " AS " + MessageContract.CHATROOM + ", " +
                MessageContract.TABLE_NAME + "." + MessageContract.MESSAGE_TEXT + " AS " + MessageContract.MESSAGE_TEXT + ", " +
                MessageContract.TABLE_NAME + "." + MessageContract.TIMESTAMP + " AS " + MessageContract.TIMESTAMP + ", " +
                MessageContract.TABLE_NAME + "." + MessageContract.SEQNUM + " AS " + MessageContract.SEQNUM + ", " +
                ClientContract.TABLE_NAME + "." + ClientContract.NAME + " AS " + ClientContract.NAME + ", " +
                MessageContract.TABLE_NAME + "." + MessageContract.SENDER_ID + " AS " + MessageContract.SENDER_ID +
                " FROM " + ClientContract.TABLE_NAME + " LEFT OUTER JOIN " + MessageContract.TABLE_NAME + " ON " +
                ClientContract.TABLE_NAME + "." + ClientContract.CLIENT_ID + " = " + MessageContract.TABLE_NAME + "." +
                MessageContract.MESSAGE_ID + ";";
            cursor = database.rawQuery(query, null);
            cursor.setNotificationUri(getContext().getContentResolver(), uri);
            return cursor;
    }
}

```

Figure 2.2

3. Define a new Class called Synchronize which extends Request. This class is used to synchronize with the ChatServer. In the class, implement a method called setJsonWriter() which is used to write the data into the JsonWriter as Figure 3.1 shows.



```

66 // See setJsonWriter
67 public String getRequestEntity() { return null; }
70
71 @Override
72 // No header
73 public Map<String, String> getRequestHeaders() {
74     Map<String, String> stringMap = new HashMap<>();
75     stringMap.put("X-latitude", "40.7439905");
76     stringMap.put("X-longitude", "-74.0323626");
77     return stringMap;
78 }
79
80 @Override
81 public Uri getRequestUri() {
82     String requestString = "http://" + host + ":" + String.valueOf(port) + "/chat/" +
83         String.valueOf(clientID) +
84         "?";
85     try {
86         requestString += "&regid=" + URLEncoder.encode(registrationID.toString(), encoding);
87     } catch (UnsupportedEncodingException e) {
88         Log.e(TAG, e.getMessage());
89     }
90     requestString += "&seqnum=" + String.valueOf(seqNum);
91     Log.v(TAG, "Post Synchronize Request URI: " + requestString);
92     return Uri.parse(requestString);
93 }
94
95 @Override
96 public Response getResponse(HttpURLConnection connection, JsonReader reader) throws IOException {
97     return new Response.SyncResponse(connection, reader);
98 }
99
100 public void setJsonWriter(JsonWriter writer) throws IOException {
101     writer.beginArray();
102     for (int i = 0; i < messages.size(); i++) {
103         writer.beginObject();
104         writer.name(MessageContract.CHATROOM);
105         writer.value(messages.get(i).chatroom);
106         writer.name(MessageContract.TIMESTAMP);
107         writer.value(messages.get(i).timestamp.getTime());
108         writer.name(MessageContract.MESSAGE_TEXT);
109         writer.value(messages.get(i).messageText);
110         writer.endObject();
111     }
112     writer.endArray();

```

Figure 3.1

4. In the RestMethod class, define a new class called StreamingResponse which is used for the response of streaming as Figure 4.1. Meanwhile, define a new method to implement synchronize operation as Figure 4.2.

```

public static class StreamingResponse {
    public HttpURLConnection connection;
    public Response response;

    public StreamingResponse(HttpURLConnection connection, Response response) {
        this.connection = connection;
        this.response = response;
    }
    // TODO
    public Response getResponse() { return this.response; }
}

```

Figure 4.1

```

public StreamingResponse perform(Synchronize request, IStreamingOutput streamingOutput) {
    URL url = request.getRequestUrl();
    Log.i(TAG, "URL to synchronize: " + url.toString());
    if (isOnline()) {
        try {
            connection = (HttpURLConnection)url.openConnection();
            connection.setRequestProperty("USER_AGENT", TAG);
            connection.setRequestMethod("POST");
            connection.setUseCaches(false);
            connection.setRequestProperty("CONNECTION", "Keep-Alive");
            //connection.setConnectTimeout(15000);
            //connection.setReadTimeout(10000);
            Map<String, String> headers = request.getRequestHeaders();
            for(Map.Entry<String, String> header : headers.entrySet()) {
                connection.addRequestProperty(header.getKey(), header.getValue());
            }
            connection.setRequestProperty("Content-Type", "application/json");
            connection.setDoInput(true);
            connection.setChunkedStreamingMode(0);
            streamingOutput.write(connection, request);
            connection.connect();
            throwErrors(connection);
            JsonReader reader = new JsonReader(new BufferedReader(new InputStreamReader(connection.getInputStream())));
            return new StreamingResponse(connection, request.getResponse(connection, reader));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return null;
}

```

Figure 4.2

5. In the RequestProcessor, implement a new method to realize the synchronization operation as Figure 5.1 shows.

```

public void perform(Synchronize request, IContinue<Response> iContinue) {
    RestMethod.StreamingResponse streamingResponse = restMethod.perform(request, (connection, request) -> {
        connection.setDoOutput(true);
        OutputStream os = connection.getOutputStream();
        JsonWriter writer = new JsonWriter(new BufferedWriter(new OutputStreamWriter(os, "UTF-8")));
        writer.setIndent(" ");
        request.setJsonWriter(writer);
        writer.flush();
        writer.close();
    });
    if (streamingResponse != null) {
        Response response = streamingResponse.getResponse();
        if (response != null) {
            iContinue.kontinue(response);
        } else {
            iContinue.kontinue(response);
            Log.e(TAG, "No Response");
        }
        streamingResponse.connection.disconnect();
    } else {
        iContinue.kontinue(null);
        Log.e(TAG, "No streamingResponse");
    }
}

```

Figure 5.1

6. In the RequestService, implement a new method to handle the synchronization as Figure 6.1 and Figure 6.2 shows.


```

private void handleRefresh(Intent intent) {
    Synchronize request = intent.getParcelableExtra(ServiceHelper.REQUEST_KEY);
    requestProcessor.perform(request, (value) -> {
        if (value != null && value.isValid()) {
            Response.SyncResponse response = (Response.SyncResponse)value;
            ArrayList<String> clients = response.client;
            ArrayList<ContentValues> messages = response.messageValues;
            ArrayList<Client> clientEntities = new ArrayList<>();
            Log.i(TAG, "Start synchronize clients");
            Cursor clientCursor = getContentResolver().query(ClientContract.CONTENT_URI, null, null, null, null);
            if (clientCursor.moveToFirst()) { // Store all clients
                do {
                    for (int i = 0; i < clients.size(); i++) {
                        String client = clients.get(i);
                        if (client.equals(ClientContract.getName(clientCursor))) {
                            Client temp = new Client(clientCursor);
                            clientEntities.add(temp);
                        } else {
                            Client temp = new Client(client);
                            temp.id = i;
                            manager.persistSync(temp);
                            if (temp.id != 0) {
                                clientEntities.add(temp);
                            }
                        }
                    }
                } while (clientCursor.moveToNext());
            } else { // No client in the database
                for (String client : clients) {
                    Client temp = new Client(client);
                    manager.persistSync(temp);
                    if (temp.id != 0) {
                        clientEntities.add(temp);
                    }
                }
            }
        }
    });
}

```

Figure 6.1

```

clientCursor.close();
Log.i(TAG, "Client update finished, start synchrnize message");
for (ContentValues values : messages) {
    String name = values.getAsString(ClientContract.NAME);
    for (Client client : clientEntities) {
        if (client.name.equals(name)) { // find match
            Cursor cursor = getContentResolver().query(MessageContract.CONTENT_URI, new String[] {MessageContract.MESSAGE_TEXT, MessageContract.TIMESTAMP + "? AND " + MessageContract.MESSAGE_TEXT + "? AND " + MessageContract.SEQNUM}, new String[] {String.valueOf(values.getAsLong(MessageContract.TIMESTAMP)), values.getAsLong(MessageContract.SEQNUM)}, null);
            if (cursor.moveToFirst()) {
                ContentValues valueToUpdate = new ContentValues();
                valueToUpdate.put(MessageContract.SEQNUM, values.getAsLong(MessageContract.SEQNUM));
                long rowId = getContentResolver().update(MessageContract.CONTENT_URI, valueToUpdate, MessageContract.MESSAGE_TEXT + "? AND " + MessageContract.TIMESTAMP + "? AND " + MessageContract.SEQNUM, new String[] {String.valueOf(values.getAsLong(MessageContract.TIMESTAMP)), values.getAsLong(MessageContract.SEQNUM)});
                if (rowId > 0) {
                    Log.i(TAG, "Update a message");
                }
            } else {
                String chatroom = values.getAsString(MessageContract.CHATROOM);
                String text = values.getAsString(MessageContract.MESSAGE_TEXT);
                long timestamp = values.getAsLong(MessageContract.TIMESTAMP);
                long seqnum = values.getAsLong(MessageContract.SEQNUM);
                ContentValues valueToUpdate = new ContentValues();
                Message message = new Message(chatroom, text, new Timestamp(timestamp), seqnum);
                manager.persistSync(message, client);
            }
            cursor.close();
        }
    }
}
Log.i(TAG, "Messages update finished");
resultReceiver.send(RESULT_SYNC_OK, null);
// TODO
} else {
    Log.i(TAG, "SYNCHRONIZE FAILED, NO RESPONSE");
    resultReceiver.send(RESULT_FAILED, null);
}
});
}

```

Figure 6.2

7. In the ChatAppActivity, modify the send() method. When the send button is clicked, it stores the message in the database and synchronize with the server as Figure 7.1 shows.

```

sendButton.setOnClickListener((v) -> {
    String text = messageText.getText().toString();
    final Message message = new Message("_default", text, new Timestamp(new Date().getTime()));
    manager.persistAsync(message, client);
    messageText.setText("");
    manager.QueryDetail(MessageContract.CONTENT_URI, (results) -> {
        if (results != null) {
            // Synchronoise with server
            long seqnum = 0; // Get seqnum
            List<Message> messages = new ArrayList<Message>(); // Store messages to be saved
            for (int i = 0; i < results.size(); i++) {
                long tempNum = results.get(i).seqnum;
                if (tempNum == 0) {
                    messages.add(results.get(i));
                }
                if (tempNum > seqnum) {
                    seqnum = tempNum;
                }
            }
            Synchronize request = new Synchronize(host, port, registrationID, clientID, seqnum, messages);
            receiver = (IReceiver) (resultCode, resultData) -> {
                if (resultCode == RequestService.RESULT_SYNC_OK) {
                    getContentResolver().notifyChange(MessageContract.CONTENT_URI, null);
                    Toast.makeText(getApplicationContext(), "Synchronize successfully", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(getApplicationContext(), "Failed to synchronize", Toast.LENGTH_SHORT).show();
                }
            };
            wrapper = new AckReceiverWrapper(new Handler());
            wrapper.setReceiver(receiver);
            serviceHelper.RefreshMessage(request, wrapper);
        }
    });
});
});

```

Figure 7.1

8. Create an Activity called ClientsActivity which shows all the clients in the database, as Figure 8.1 shows. When the client in the list view is clicked, it will start a new Activity to show every message sent by the client.

```

public class ClientsActivity extends ActionBarActivity {
    public static final String TAG = ClientsActivity.class.getCanonicalName();
    public static final String CLIENT_ACTIVITY_KEY = TAG;
    public static final int CLIENT_ACTIVITY_LOADER_ID = 2;
    SimpleCursorAdapter cursorAdapter;
    ClientManager manager;
    ListView listView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_clients);
        manager = new ClientManager(this, (cursor) -> {
            return new Client(cursor);
        }, CLIENT_ACTIVITY_LOADER_ID);
        listView = (ListView) findViewById(android.R.id.list);
        String[] from = new String[] {ClientContract.NAME};
        int[] to = new int[] {R.id.client_list};
        cursorAdapter = new SimpleCursorAdapter(this, R.layout.client_list, null, from, to, 0);
        listView.setAdapter(cursorAdapter);
        manager.QueryAsync(ClientContract.CONTENT_URI, new IQueryListener<Client>() {
            @Override
            public void handleResults(TypedCursor<Client> cursor) {
                cursorAdapter.swapCursor(cursor.getCursor());
            }

            @Override
            public void closeResults() { cursorAdapter.swapCursor(null); }
        });
        listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                Cursor cursor = cursorAdapter.getCursor();
                cursor.moveToPosition(position);
                Client client = new Client(cursor);
                client.id = ClientContract.getClientId(cursor);
                Intent intent = new Intent(ClientsActivity.this, MessagesActivity.class);
                intent.putExtra(CLIENT_ACTIVITY_KEY, client);
                startActivity(intent);
            }
        });
    }
}

```

Figure 8.1

9. Create a new Activity called MessagesActivity. It accepts a Client from the intent and query every message sent by the client from the database, then list the messages on the screen as Figure 9.1 shows.


```

public class MessagesActivity extends ActionBarActivity {
    public static final int MESSAGE_ACTIVITY_LOADER_ID = 3;
    public static final String TAG = MessagesActivity.class.getCanonicalName();
    MessageManager messageManager;
    SimpleCursorAdapter adapter;
    ListView listView;
    TextView nameView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_messages);
        Intent intent = getIntent();
        final Client client = intent.getParcelableExtra(ClientsActivity.CLIENT_ACTIVITY_KEY);
        nameView = (TextView)findViewById(R.id.messages_client_name);
        listView = (ListView)findViewById(R.id.messages_list);
        messageManager = new MessageManager(this, (cursor) -> {
            return new Message(cursor);
        }, MESSAGE_ACTIVITY_LOADER_ID);

        String[] from = new String[] {MessageContract.MESSAGE_TEXT};
        int[] to = new int[] {R.id.messages_client_row};
        adapter = new SimpleCursorAdapter(this, R.layout.client_message_row, null, from, to, 0);
        listView.setAdapter(adapter);
        messageManager.QueryAsync(ClientContract.CONTENT_URI(String.valueOf(client.id)), new IQueryListener<Message>() {
            @Override
            public void handleResults(TypedCursor<Message> cursor) {
                adapter.swapCursor(cursor.getCursor());
                cursor.getCursor().setNotificationUri(getContentResolver(), ClientContract.CONTENT_URI(String.valueOf(client.id)))
            }

            @Override
            public void closeResults() { adapter.swapCursor(null); }
        });
    }
}

```

Figure 9.1

10. Finally, test the App and please see the result in the video file.