Report of CS 522 Assignment 6

Name: Yunfeng Wei
CWID: 10394963
E-mail: ywei14@stevens.edu
Video: In the ZIP Archive File

Part 1 : Simple Cloud Chat App

1. First create a sub package called Entities. In the package, define the Message class as Figure 1.1. The Message class has a messageID, a messageText, a timestamp and a senderID.

```
public class Message implements Parcelable{
    public long messageID;
    public String messageText;
    public Timestamp timestamp;
    public long senderID;

    public static final Creator<Message> CREATOR = new Creator<Message>() {
        @Override
        public Message createFromParcel(Parcel source) { return new Message(source); }

        @Override
        public Message[] newArray(int size) { return new Message[size]; }
    };

    public Message(String messageText, Timestamp timestamp, long senderID) {
        this.messageText = messageText;
        this.timestamp = timestamp;
        this.senderID = senderID;
    }

    public Message(String messageText, Timestamp timestamp, long senderID, long messageID) {
        this.messageText = messageText;
        this.timestamp = timestamp;
        this.senderID = senderID;
        this.messageID = messageID;
    }

    public Message(Parcel parcel) {
        this.messageID = parcel.readLong();
        this.messageText = parcel.readString();
        this.timestamp = new Timestamp(parcel.readLong());
        this.senderID = parcel.readLong();
    }

    public Message(Cursor cursor) {
        this.messageID = MessageContract.getMessageId(cursor);
        this.messageText = MessageContract.getMessageText(cursor);
        this.timestamp = MessageContract.getTimestamp(cursor);
        this.senderID = MessageContract.getSenderId(cursor);
    }
```

Figure 1.1

2. Then define the abstract class Request as Figure 2.1. Define the Register and PostMessage class which extends Request as Figure 2.2 and Figure 2.3. The Register class and PostMessage class store the information of the register operation and post message operation.

```java
/**
 * Created by wyf920621 on 3/12/15.
 */
public abstract class Request implements Parcelable {
    public static final String TAG = Request.class.getCanonicalName();
    protected static final String encoding = "UTF-8";
    public long clientID;
    public static final int UUIDFlag = 1;
    public UUID registrationID; // sanity check
    public URL getRequestUrl() {
        Uri uri = this.getRequestUri();
        URL url = null;
        try {
            url = new URL(uri.toString());
            Log.v(TAG, "Request URL: " + url.toString());
        } catch (MalformedURLException e) {
            Log.e(TAG, e.getMessage());
        }
        return url;
    }
    // App-specific HTTP request headers
    public abstract Map<String, String> getRequestHeaders();
    // Chat service URI with parameters e.g. query string parameters
    public abstract Uri getRequestUri();
    // JSON body (if not null) for request data not passed in headers
    public abstract String getRequestEntity() throws IOException;
    // Define your own Response class including HTTP response code.
    public abstract Response getResponse(HttpURLConnection connection, JsonReader reader) throws IOException; /* Null for streaming
}
```

Figure 2.1

```java
 */
public class Register extends Request {
    public static final String TAG = Register.class.getCanonicalName();
    public String host;
    public int port;
    public String clientName;

    public static final Creator<Register> CREATOR = new Creator<Register>() {...};


    public Register(String host, int port, String clientName, long clientId, UUID registrationID) {...}

    public Register(String host, int port, String clientName, UUID registrationID) {...}

    public Register(Parcel parcel) {...}
    @Override
    public Map<String, String> getRequestHeaders() {
        Map<String, String> stringMap = new HashMap<>();
        stringMap.put("X-latitude", "40.7439905");
        stringMap.put("X-longitude", "-74.0323626");
        return stringMap;
    }

    @Override
    public Uri getRequestUri() {
        String RequestString = "http://" + host + ":" + String.valueOf(port) + "/chat?";
        try {
            RequestString += "username=" + URLEncoder.encode(clientName, encoding);
            RequestString += "&regid=" + URLEncoder.encode(String.valueOf(registrationID.toString()), encoding);
            Log.v(TAG, "Register Request URI: " + RequestString);
        } catch (UnsupportedEncodingException e) {
            Log.e(TAG, e.getMessage());
        }
        return Uri.parse(RequestString);
    }

    @Override
    public String getRequestEntity() { return null; }

    @Override
    public Response getResponse(HttpURLConnection connection, JsonReader reader) throws IOException{
        return new Response.RegisterResponse(connection, reader);
    }
```

Figure 2.2

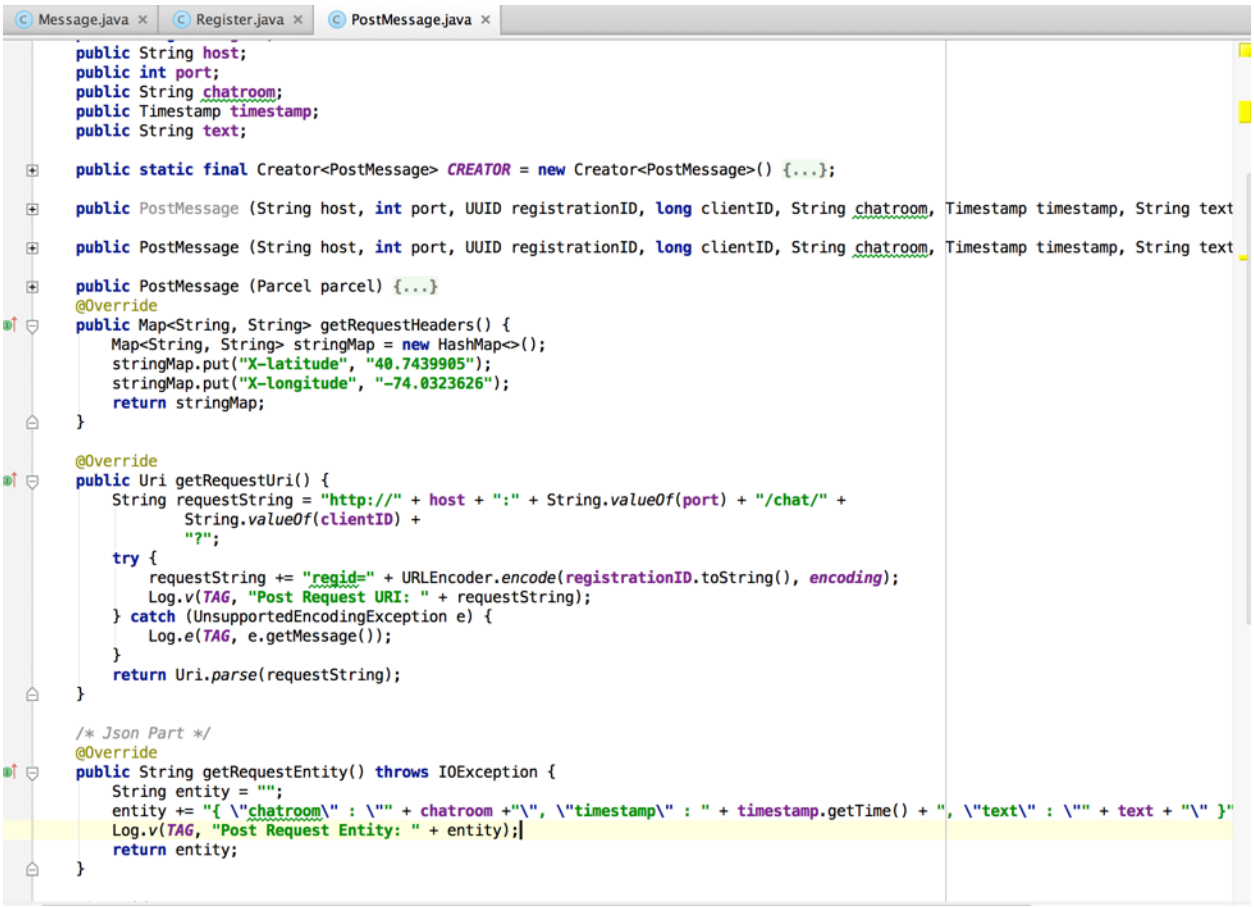
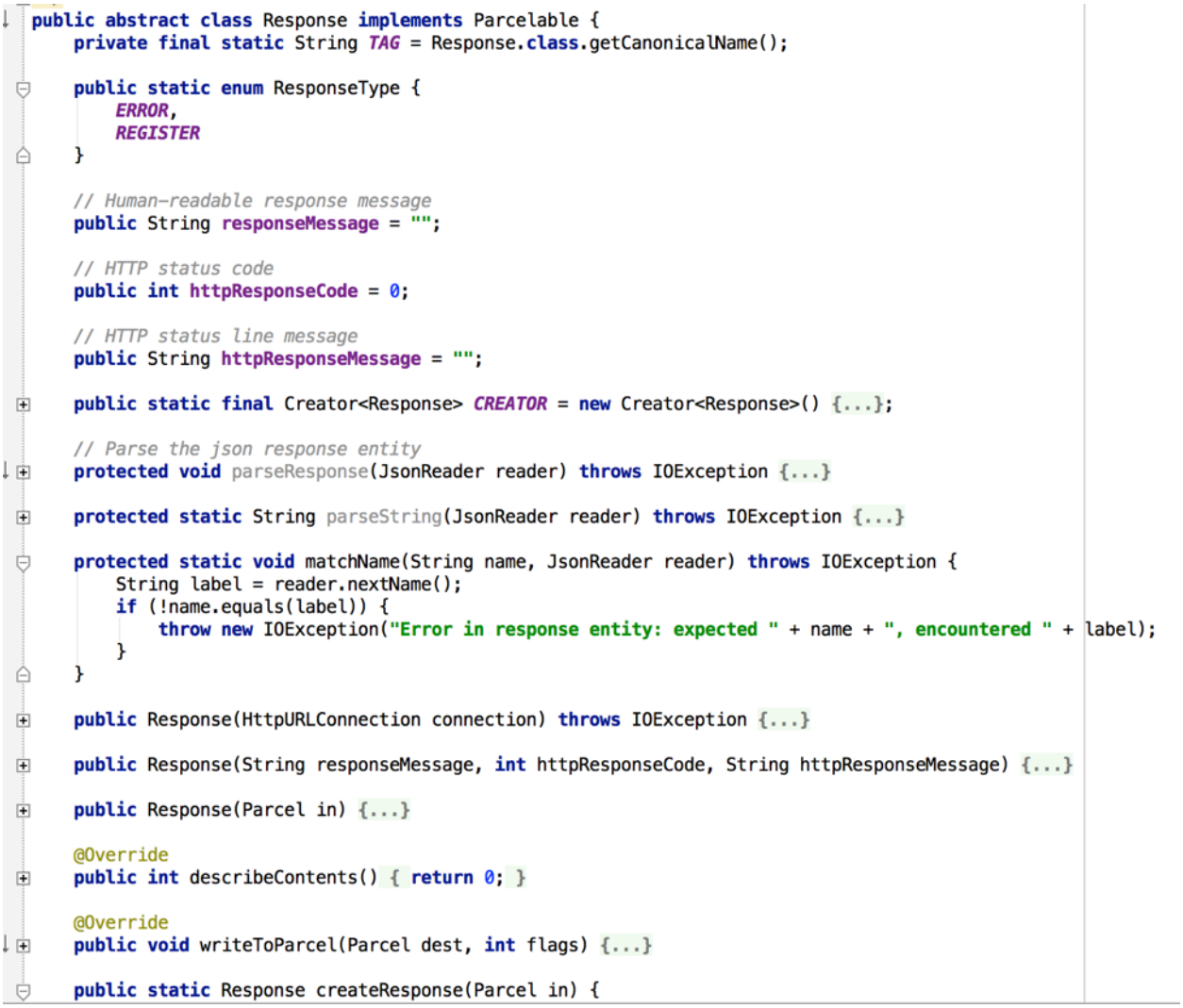


Figure 2.3

3. Define the Response abstract class which is used to store the HttpResponse Information as Figure 3.1. In the class, define the RegisterResponse class and the ErrorResponse class as Figure 3.2 and Figure 3.3. If the response is valid, then stores the response message in the RegisterResponse class. If the response is invalid, then stores the response message in the ErrorResponse class.



```
public abstract class Response implements Parcelable {
    private final static String TAG = Response.class.getCanonicalName();

    public static enum ResponseType {
        ERROR,
        REGISTER
    }

    // Human-readable response message
    public String responseMessage = "";

    // HTTP status code
    public int httpResponseCode = 0;

    // HTTP status line message
    public String httpResponseMessage = "";

    public static final Creator<Response> CREATOR = new Creator<Response>() {...};

    // Parse the json response entity
    protected void parseResponse(JsonReader reader) throws IOException {...}

    protected static String parseString(JsonReader reader) throws IOException {...}

    protected static void matchName(String name, JsonReader reader) throws IOException {
        String label = reader.nextName();
        if (!name.equals(label)) {
            throw new IOException("Error in response entity: expected " + name + ", encountered " + label);
        }
    }

    public Response(HttpURLConnection connection) throws IOException {...}

    public Response(String responseMessage, int httpResponseCode, String httpResponseMessage) {...}

    public Response(Parcel in) {...}

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

    @Override
    public void writeToParcel(Parcel dest, int flags) {...}

    public static Response createResponse(Parcel in) {
```

Figure 3.1

```
/* ErrorResponse */
public static class ErrorResponse extends Response implements Parcelable {
    public enum Status {
        NETWORK_UNAVAILABLE,
        SERVER_ERROR,
        SYSTEM_ERROR,
        APPLICATION_ERROR
    }
    public Status status;

    public ErrorResponse(int responseCode, Status status, String message) {...}

    public ErrorResponse(int responseCode, Status status, String message, String httpMessage) {...}

    @Override
    public boolean isValid() { return false; }

    @Override
    public void writeToParcel(Parcel dest, int flags) {...}

    public ErrorResponse(Parcel in) {
        super(in);
        this.status = Status.valueOf(in.readString());
    }

    public static final Creator<ErrorResponse> CREATOR = new Creator<ErrorResponse>() {...};
}
```

Figure 3.2

```
public static class RegisterResponse extends Response implements Parcelable {
    public long id;
    public RegisterResponse(HttpURLConnection connection, JsonReader reader) throws IOException {...}

    @Override
    public boolean isValid() { return this.id > 0; }

    @Override
    protected void parseResponse(JsonReader reader) throws IOException {
        reader.beginObject();
        matchName("id", reader);
        this.id = reader.nextInt();

        reader.endObject();
    }

    public void writeToParcel(Parcel out, int flags) {...}

    public RegisterResponse(Parcel in) {...}

    public static final Creator<RegisterResponse> CREATOR = new Creator<RegisterResponse>() {...};
}
```

Figure 3.3

4. Define the MessageContract, DatabaseHelper, MessageDbProvider, AsyncContentResolver, SimpleQueryBuilder, QueryBuilder, TypedCursor, Manager and MessageManager which are used store the message information into the database and query from the database asynchronously.

5. Define a RestMethod class which is used to connect to the service and get RESTful information. In the class, define isOnline, outputRequestEntity and throwErrors method as Figure 5.1. Define two kinds of perform method to send Register Request or Post Message Request to the server as Figure 5.2 and Figure 5.3.

```
private boolean isOnline() {
    ConnectivityManager connectivityManager = (ConnectivityManager)context.getSystemService(Context.CONNECTIVITY_SERVICE);
    return connectivityManager.getActiveNetworkInfo() != null && connectivityManager.getActiveNetworkInfo().isConnectedOrConnect
}

private void outputRequestEntity(Request request) throws IOException {
    String requestEntity_= request.getRequestEntity();
    if (requestEntity != null) {
        connection.setDoOutput(true);
        connection.setRequestProperty("Content-Type", "application/json");
        byte[] outputEntity = requestEntity.getBytes("UTF-8");
        connection.setFixedLengthStreamingMode(outputEntity.length);
        OutputStream out = new BufferedOutputStream(connection.getOutputStream());
        out.write(outputEntity);
        out.flush();
        out.close();
    }
}

private static void throwErrors (HttpURLConnection connection) throws IOException {
    final int status = connection.getResponseCode();
    if (status < 200 || status >= 300) {
        String exceptionMessage = "Error response " + status + " " + connection.getResponseMessage() + " for " + connection.getU
        throw new IOException(exceptionMessage);
    }
}
```

Figure 5.1

```
public Response perform(Register request) {
    URL url = request.getRequestUrl();
    Log.i(TAG, "URL to request: " + url.toString());
    if(isOnline()) {
        try {
            // prepare request
            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());
            }

            // execute request
            outputRequestEntity(request);
            connection.setDoInput(true);
            connection.connect();
            throwErrors(connection);

            JsonReader jsonReader = new JsonReader(new BufferedReader(new InputStreamReader(connection.getInputStream())));
            Response response = request.getResponse(connection, jsonReader);
            jsonReader.close();
            connection.disconnect();
            return response;
        } catch (IOException e) {
            Log.e(TAG, e.getMessage());
        }
    }
    return null;
}
```

Figure 5.2

```
public Response perform(PostMessage request) {
    URL url = request.getRequestUrl();
    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.setDoInput(true);
            // execute
            outputRequestEntity(request);
            connection.connect();
            throwErrors(connection);
            JsonReader reader = new JsonReader(new BufferedReader(new InputStreamReader(connection.getInputStream())));
            Response response = request.getResponse(connection, reader);
            reader.close();
            connection.disconnect();
            return response;
        } catch (IOException e) {
            Log.e(TAG, e.getMessage());
        }
    }
    return null;
}
```

Figure 5.3

6. Define a RequestProcessor class, it perform the RestMethod operation and execute the callback method as Figure 6.1.

```
/**
 * Created by wyf920621 on 3/13/15.
 */
public class RequestProcessor {
    public static final String TAG = RequestProcessor.class.getCanonicalName();
    private RestMethod restMethod;
    public RequestProcessor(Context context) { restMethod = new RestMethod(context); }
    public void perform(Register register, IContinue<Response> iContinue) {
        Response response = restMethod.perform(register);
        if (response != null) {
            iContinue.kontinue(response);
        } else {
            Log.e(TAG, "No Response");
        }
    }

    public void perform(PostMessage postMessage, IContinue<Response> iContinue) {
        Response response = restMethod.perform(postMessage);
        if (response != null) {
            iContinue.kontinue(response);
        } else {
            Log.e(TAG, "No Response");
        }
    }
}
```

Figure 6.1

7. Define a ResultReceiver called AckReceiverWrapper which is used to notice the App whether the POST method results in success as Figure 7.1.

```java
/**
 * Created by wyf920621 on 3/14/15.
 */
public class AckReceiverWrapper extends ResultReceiver {
    private IReceiver receiver;
    public AckReceiverWrapper(Handler handler) { super(handler); }

    @Override
    protected void onReceiveResult(int resultCode, Bundle resultData) {
        if (receiver != null) {
            receiver.onReceiveResult(resultCode, resultData);
        }
    }

    public void setReceiver(IReceiver receiver) { this.receiver = receiver; }

    public interface IReceiver {
        public void onReceiveResult(int resultCode, Bundle resultData);
    }
}
```

Figure 7.1

8. Define an Intent Service called RequestService, it receives a PostMessage or a Register message, then use the RequestProcessor to send the message to the server and process the result as Figure 8.1 and Figure 8.2.

```java
public class RequestService extends IntentService {
    public static final String TAG = RequestService.class.getCanonicalName();
    // IntentService can perform, e.g. ACTION_FETCH_NEW_ITEMS
    public static final String ACTION_REGISTER = "edu.stevens.cs522.simplecloudchatapp.Services.action.REGISTER";
    public static final String ACTION_POST_MESSAGE = "edu.stevens.cs522.simplecloudchatapp.Services.action.POST_MESSAGE";

    private static int messageCount = 1;
    public static final int RESULT_REGISTER_OK = 0;
    public static final int RESULT_MESSAGE_OK = 1;
    public static final int RESULT_FAILED = 2;

    public static final int REQUEST_SERVICE_LOADERID = 2;
    private RequestProcessor requestProcessor = null;
    private MessageManager manager;
    private ResultReceiver resultReceiver;

    @Override
    public void onCreate() {
        manager = new MessageManager(this, (cursor) -> {
                return new Message(cursor);
        }, REQUEST_SERVICE_LOADERID);
        super.onCreate();
    }

    public RequestService() { super("RequestService"); }

    @Override
    protected void onHandleIntent(Intent intent) {
        if (intent != null) {
            resultReceiver = intent.getParcelableExtra(ServiceHelper.ACK);
            requestProcessor = new RequestProcessor(this);
            if (ACTION_REGISTER.equals(intent.getAction())) {
                handleRegister(intent);
            } else if (ACTION_POST_MESSAGE.equals(intent.getAction())) {
                handlePostMessage(intent);
            } else {
                throw new UnsupportedOperationException("Unsupported Action");
            }
        }
        stopSelf();
    }
```

Figure 8.1

```
private void handleRegister(Intent intent) {
    final Register request = intent.getParcelableExtra(ServiceHelper.REQUEST_KEY);
    requestProcessor.perform(request, (value) -> {
            if (value != null && value.isValid()) {
                Response.RegisterResponse response = (Response.RegisterResponse)value;
                request.clientID = response.id;
                Log.i("Client to be saved: ", request.clientName + " " + request.host + ":" + String.valueOf(request.port) + " '
                SharedPreferences preferences = getSharedPreferences(SettingActivity.MY_SHARED_PREF, Context.MODE_PRIVATE);
                SharedPreferences.Editor editor = preferences.edit();
                editor.putString(SettingActivity.PREF_USERNAME, request.clientName);
                editor.putLong(SettingActivity.PREF_IDENTIFIER, request.clientID);
                editor.putString(SettingActivity.PREF_HOST, request.host);
                editor.putInt(SettingActivity.PREF_PORT, request.port);
                editor.apply();
                Log.i(TAG, "Register save successfully");
                resultReceiver.send(RESULT_REGISTER_OK, null);
            } else {
                Log.i(TAG, "Register save failed");
                resultReceiver.send(RESULT_FAILED, null);
            }
    });
}

private void handlePostMessage(Intent intent) {
    final PostMessage postMessage = intent.getParcelableExtra(ServiceHelper.REQUEST_KEY);
    requestProcessor.perform(postMessage, (value) -> {
            if (value != null && value.isValid()) {
                Response.RegisterResponse response = (Response.RegisterResponse)value;
                postMessage.messageID = response.id;
                Log.i("Message to be saved: ", postMessage.text + " " + postMessage.messageID);
                manager.persistSync(new Message(postMessage.text, postMessage.timestamp, postMessage.clientID, postMessage.messa
                resultReceiver.send(RESULT_MESSAGE_OK, null);
            } else {
                Log.i(TAG, "message save failed");
                resultReceiver.send(RESULT_FAILED, null);
            }
    });
}
```

Figure 8.2

9. Define a ServiceHelper which has the methods called RegisterUser and PostMessage as Figure 9.1. The ServiceHelper starts the service to send the message to the web server.

```java
/**
 * Created by wyf920621 on 3/12/15.
 */
public class ServiceHelper {
    public static final String REQUEST_KEY = "edu.stevens.cs522.simplecloudchatapp.request_key";
    public static final String ACK = "edu.stevens.cs522.simplecloudchatapp.ACK";
    public Context context;

    public ServiceHelper(Context context) { this.context = context; }


    public void RegisterUser(Register register, AckReceiverWrapper wrapper) {
        Intent intent = new Intent(context, RequestService.class);
        intent.setAction(RequestService.ACTION_REGISTER);
        intent.putExtra(REQUEST_KEY, register);
        intent.putExtra(ACK, wrapper);
        context.startService(intent);
    }

    public void PostMessage(PostMessage postMessage, AckReceiverWrapper wrapper) {
        Intent intent = new Intent(context, RequestService.class);
        intent.setAction(RequestService.ACTION_POST_MESSAGE);
        intent.putExtra(REQUEST_KEY, postMessage);
        intent.putExtra(ACK, wrapper);
        context.startService(intent);
    }
}
```

Figure 9.1

10. Define a ChatAppActivity as the Launcher Activity, it first check if the App has registered in the web server. If yes, it sets the button to enabled, otherwise it sets the button to not enabled and shows the warning message on the screen. If the button is clicked, it gets the message from the screen ,sends the message to the server to get the message identifier and stores the message into the database. The structure of the ChatAppActivity is shown as Figure 10.1, Figure 10.2, Figure 10.3 and Figure 10.4.

```java
public class ChatAppActivity extends ActionBarActivity {
    public static final String TAG = ChatAppActivity.class.getCanonicalName();
    public static final String DEFAULT_HOST = SettingActivity.DESTINATION_HOST_DEFAULT;
    public static final int DEFAULT_PORT = SettingActivity.DESTINATION_PORT_DEFAULT;

    public static final int CHAT_APP_LOADER_ID = 1;
    public static final int REQUEST_CODE = 1;
    private ServiceHelper serviceHelper = null;
    private MessageManager manager = null;
    private SimpleCursorAdapter cursorAdapter = null;

    private ListView messageList;
    private EditText messageText;
    private Button sendButton;
    private TextView warningView;

    private String clientName;
    private long clientID;
    private String host;
    private int port;
    private UUID registrationID;
    private AckReceiverWrapper.IReceiver receiver = null;
    private AckReceiverWrapper wrapper = null;
```

Figure 10.1

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_chat_app);

    messageList = (ListView)findViewById(R.id.message_list);
    messageText = (EditText)findViewById(R.id.message_text);
    warningView = (TextView)findViewById(R.id.warning_view);
    messageText.setText("");
    sendButton = (Button)findViewById(R.id.send_button);
    manager = new MessageManager(this, (cursor) -> {
            return new Message(cursor);
    }, CHAT_APP_LOADER_ID);
    String[] from = new String[] {MessageContract.MESSAGE_TEXT};
    int[] to = new int[] {R.id.message_row};
    cursorAdapter = new SimpleCursorAdapter(this, R.layout.message_row, null, from, to, 0);
    messageList.setAdapter(cursorAdapter);

    manager.QueryAsync(MessageContract.CONTENT_URI, new IQueryListener<Message>() {
        @Override
        public void handleResults(TypedCursor<Message> cursor) {
            cursorAdapter.swapCursor(cursor.getCursor());
        }

        @Override
        public void closeResults() { cursorAdapter.swapCursor(null); }
    });
    serviceHelper = new ServiceHelper(this);
    resetInfo();
    sendButton.setOnClickListener((v) -> {
            String message = messageText.getText().toString();
            Date date = new Date();
            PostMessage postMessage = new PostMessage(host, port, registrationID, clientID, "_default", new Timestamp(date.getT
            receiver = (IReceiver) (resultCode, resultData) -> {
                    if (resultCode == RequestService.RESULT_MESSAGE_OK) {
                        getContentResolver().notifyChange(MessageContract.CONTENT_URI, null);
                        Log.v(TAG, "Message post success");
                    } else {
                        Log.e(TAG, "Message post failed");
                    }
            };
            wrapper = new AckReceiverWrapper(new Handler());
            wrapper.setReceiver(receiver);
            serviceHelper.PostMessage(postMessage, wrapper);
            messageText.setText("");
```

Figure 10.2

```
private void resetInfo() {
    SharedPreferences sharedPreferences = getSharedPreferences(SettingActivity.MY_SHARED_PREF, MODE_PRIVATE);
    clientName = sharedPreferences.getString(SettingActivity.PREF_USERNAME, "");
    clientID = sharedPreferences.getLong(SettingActivity.PREF_IDENTIFIER, -1);
    host = sharedPreferences.getString(SettingActivity.PREF_HOST, DEFAULT_HOST);
    port = sharedPreferences.getInt(SettingActivity.PREF_PORT, DEFAULT_PORT);
    registrationID = new UUID(sharedPreferences.getLong(SettingActivity.PREF_REGID_MOST, 0), sharedPreferences.getLong(SettingA
    if (clientID == -1) {
        Log.i(TAG, "NEED TO REGISTER APP");
        sendButton.setEnabled(false);
        warningView.setText("Please first register the app");

    } else {
        Log.i(TAG, "APP INSTALLATION VALID");
        sendButton.setEnabled(true);
        warningView.setText("");
    }
}
```

Figure 10.3

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_chat_app, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        Intent intent = new Intent(this, SettingActivity.class);
        startActivityForResult(intent, REQUEST_CODE);
        return true;
    }

    return super.onOptionsItemSelected(item);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            Log.v(TAG, "Register successfully");
            resetInfo();
        } else {
            Log.v(TAG, "Register failed");
        }
    }
}
```

Figure 10.4

11. Define an Activity called SettingActivity. When the “save” button is clicked, it gets the destination host, port and client name from the EditText parts, generates a UUID number, creates a Register object and sends the message to the web server. If the register operation is successful, it saves the client identifier into the SharedPreference and returns RESULT_OK to the ChatAppActivity, or RESULT_CANCELED otherwise. The structure is shown as Figure 11.1, Figure 11.2.

```java
public class SettingActivity extends ActionBarActivity {
    public static final String TAG = SettingActivity.class.getCanonicalName();
    public static final String MY_SHARED_PREF = "edu.stevens.cs522.simplecloudchatapp.SharedPreference";
    public static final String PREF_USERNAME = "edu.stevens.cs522.simplecloudchatapp.SharedPreference.username";
    public static final String PREF_IDENTIFIER = "edu.stevens.cs522.simplecloudchatapp.SharedPreference.identifier";
    public static final String PREF_REGID_MOST = "edu.stevens.cs522.simplecloudchatapp.SharedPreference.regid.most";
    public static final String PREF_REGID_LEAST = "edu.stevens.cs522.simplecloudchatapp.SharedPreference.regid.least";
    public static final String PREF_HOST = "edu.stevens.cs522.simplecloudchatapp.SharedPreference.host";
    public static final String PREF_PORT = "edu.stevens.cs522.simplecloudchatapp.SharedPreference.port";

    public static final String DESTINATION_HOST_DEFAULT = "127.0.0.1";
    public static final int DESTINATION_PORT_DEFAULT = 8080;
    public static final String CLIENT_NAME_DEFAULT = "";

    private static UUID uuid;
    private EditText textHost;
    private EditText textPort;
    private EditText textClientName;
    private Button saveButton;

    private ServiceHelper serviceHelper = null;
    private AckReceiverWrapper.IReceiver receiver = null;
    private AckReceiverWrapper wrapper = null;
```

Figure 11.1

```java
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_setting);
        SharedPreferences sharedPreferences = getSharedPreferences(MY_SHARED_PREF, MODE_PRIVATE);
        long mst = sharedPreferences.getLong(PREF_REGID_MOST, 0);
        if (mst == 0) {
            uuid = UUID.randomUUID();
            SharedPreferences.Editor editor = sharedPreferences.edit();
            editor.putLong(PREF_REGID_MOST, uuid.getMostSignificantBits());
            editor.putLong(PREF_REGID_LEAST, uuid.getLeastSignificantBits());
            editor.apply();
        } else {
            uuid = new UUID(sharedPreferences.getLong(PREF_REGID_MOST, 0), sharedPreferences.getLong(PREF_REGID_LEAST, 0));
        }
        Log.i(TAG, "UUID: " + uuid.toString());
        textHost = (EditText)findViewById(R.id.destination_host);
        textPort = (EditText)findViewById(R.id.destination_port);
        textClientName = (EditText)findViewById(R.id.client_name);
        saveButton = (Button)findViewById(R.id.save_setting_button);
        textHost.setText(sharedPreferences.getString(PREF_HOST, DESTINATION_HOST_DEFAULT));
        textPort.setText(String.valueOf(sharedPreferences.getInt(PREF_PORT, DESTINATION_PORT_DEFAULT)));
        textClientName.setText(sharedPreferences.getString(PREF_USERNAME, CLIENT_NAME_DEFAULT));
        serviceHelper = new ServiceHelper(this);
        saveButton.setOnClickListener((v) -> {
                String host = textHost.getText().toString();
                int port = Integer.parseInt(textPort.getText().toString());
                String clientName = textClientName.getText().toString();
                Register register = new Register(host, port, clientName, uuid);
                receiver = (IReceiver) (resultCode, resultData) -> {
                        if (resultCode == RequestService.RESULT_REGISTER_OK) {
                            setResult(RESULT_OK);
                            finish();
                        } else {
                            setResult(RESULT_CANCELED);
                            finish();
                        }
                };
                wrapper = new AckReceiverWrapper(new Handler());
                wrapper.setReceiver(receiver);
                serviceHelper.RegisterUser(register, wrapper);
        });
    }
```

Figure 11.2

12. Finally, run the App as the video shows.

Plus: It seems that the web server has some bugs. The register operation is successful but the post message operation is not. After a message is post, the web server always returns { ‘id’ : 0 } to the phone and the HTTP response code is HTTP/1.1 #, the number # seems to be the actual identifier of the message. See the details in the video.