Android Workshop

Luke and Jake Klinker
Source Allies Inc.

Introduction



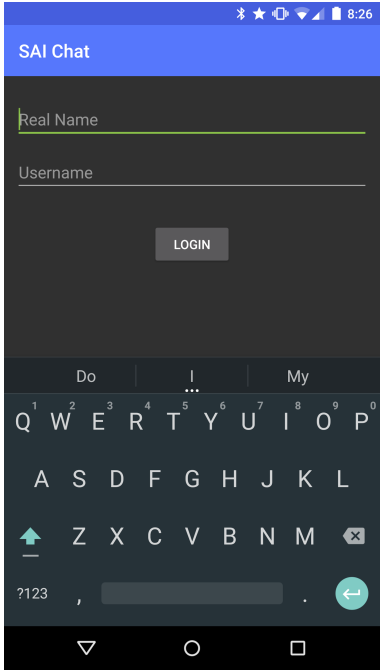


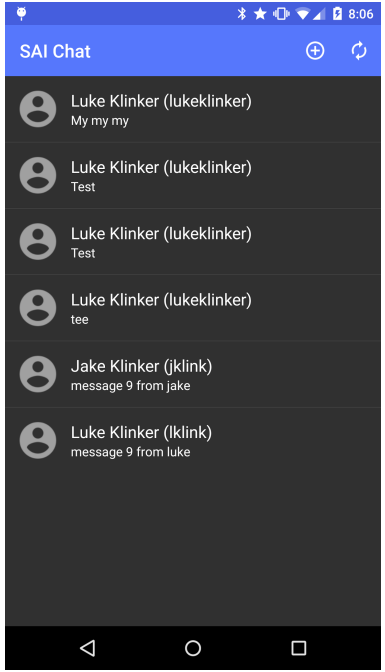


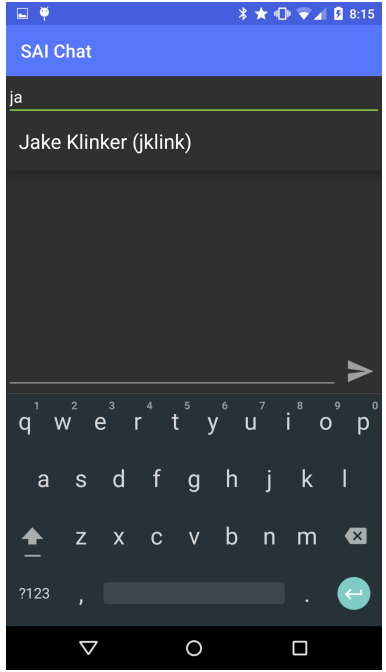
Today's Workshop

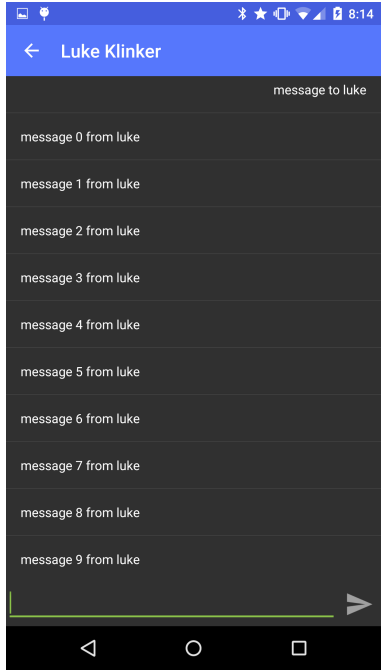
- Making an instant messaging app
  - iMessage, WhatsApp, Hangouts
- Backend powered by Google App Engine and Google Cloud Messaging
- Focus on the Android side
  - Mostly on the UI and user interactions
  - Backend, databases, and data calls are already coded

Today's Workshop









Getting Started

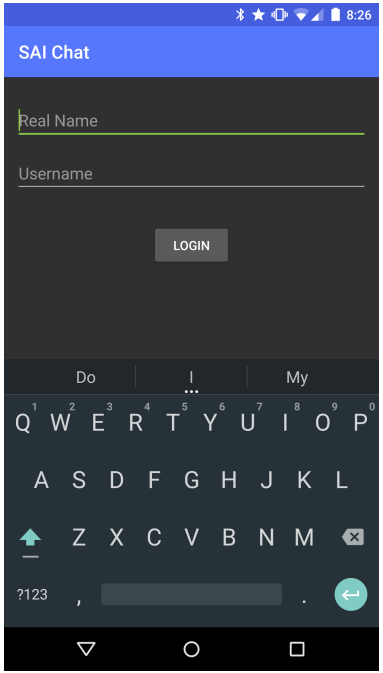
- GitHub Project: http://goo.gl/qQhTES
  - Checkout the “part_1” branch to get started
  - There are new branches for every step, but feel free to continue off of your old code instead as you finish the previous parts.
- Add Lombok plugin to Android Studio
  - File -> Settings -> Plugins
  - Browser repositories…
  - Search for “Lombok”

Getting Started

- In Android Studio hit the *Open Existing Android Studio Project* button
- Open the root directory for *source-allies-android-workshop*

    - About gradle build system: http://goo.gl/vJQGVq

Part 1: Register and Login



1.) Create the LoginActivity class using Android Studio's wizard in the */activities/* package

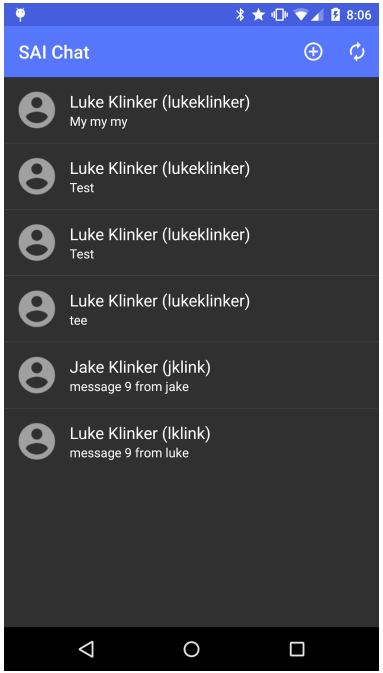
- Extend the AbstractToolBarActivity
- Add the class to the AndroidManifest

2.) Create a new activity_login.xml in */res/layout/*

- Needs a real name editText, username editText, and a login button

3.) Implement the views and the clicks into the UI on the LoginActivity class

Part 2: Read/Display Conversations



Step through todo's contained in:

1.) ConversationListActivity
2.) DatabaseHelper
3.) ConversationFragment
4.) ThreadArrayAdapter

Part 2: Read/Display Conversations

- ConversationListActivity

1.) Create the Conversation Fragment

- FragmentTransaction class to add it to the activity

2.) Create a button in the app bar to sync data (Example is the new message button)

- Add it to the menu xml file

- Call the refreshDataFunction() when it is clicked

3.) Implement the AysncTask background work for downloading the data

- Use the thread api as an example.

- Fill in the data for the Users and the Messages

Part 2: Read/Display Conversations

- DatabaseHelper

1.) Fill in the DatabaseHelper.findAllConversations() method.

```
/**
 * MESSAGES DATA SOURCE
 */
public List<Message> findThreadMessages(Long threadId) {
    List<Message> messages = new ArrayList<>();

    Cursor cursor = messageData.getThreadCursor(threadId);
    if (cursor != null && cursor.moveToFirst()) {
        do {
            Message message = new Message(context);
            message.fillFromCursor(cursor);
            messages.add(message);
        } while (cursor.moveToNext());
    }

    return messages;
}
```

Use the other methods in the class as examples.

Iterating over a cursor and adding the items to a List<Thread>

Part 2: Read/Display Conversations

- ConversationFragment

1.) Add an IntentFilter and BroadcastReceiver to the onResume() for Sender. SENT_BROADCAST

2.) Fill in the AsyncTask to find the conversations on a background thread.

3.) Follow the substeps to implement opening a conversation to view its messages

```
// the intent filter is used to 'listen' for a broadcast
IntentFilter filter = new IntentFilter();

// we add the registration complete action because the RegistrationUtils will send out this
// action after the server gets our info
filter.addAction(RegistrationUtils.REGISTRATION_COMPLETE);

// register a new receiver for the above action
registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
```

Part 2: ThreadArrayAdapter Examples

Example View holder:

```
public static class ViewHolder {
    public TextView message;
    public LinearLayout parent;
}
```

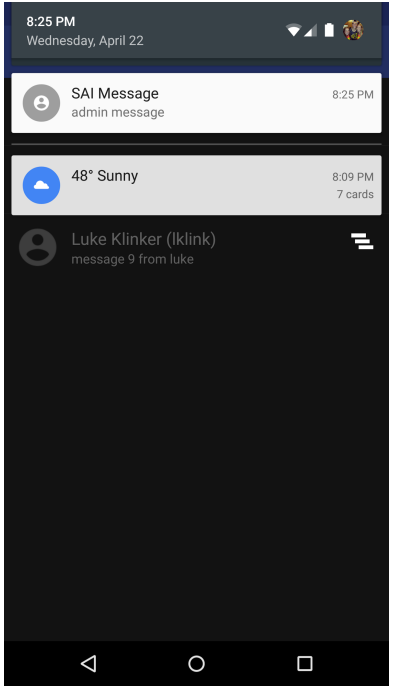
Example creating view holder:

```
// create a new view holder object
final ViewHolder holder = new ViewHolder();

// assign the children for the view holder
holder.message = (TextView) v.findViewById(R.id.message_text);
holder.parent = (LinearLayout) v.findViewById(R.id.parent);

// set the tags so that we can find all of these view without searching for them every time
// (when the view is recycled)
v.setTag(holder);
```

Finding the view holder object:

```
final ViewHolder holder = (ViewHolder) view.getTag();
```

Part 3: Receive Messages



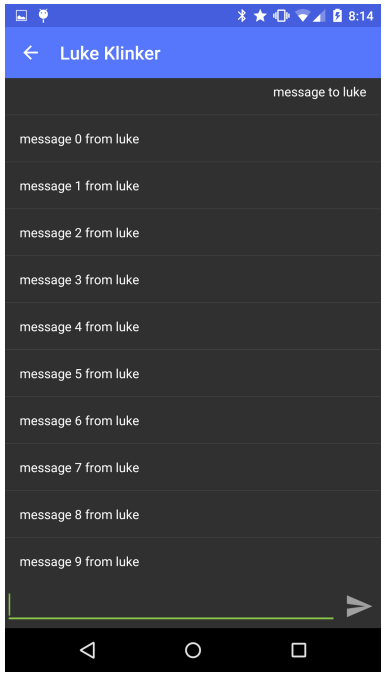
- Implement the notifications. Each needs contentText, contentTitle, and smallIcon.

```
.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);

PendingIntent resultPendingIntent =
        PendingIntent.getActivity(
                this,
                0,
                resultIntent,
                PendingIntent.FLAG_UPDATE_CURRENT
        );

builder.setContentIntent(resultPendingIntent);
```

- Add to the Android Manifest for BroadcastReceiver and Service

```
android:permission="com.google.android.c2dm.permission.SEND" >
<intent-filter>
    <action android:name="com.google.android.c2dm.intent.RECEIVE" />
    <category android:name="com.klinker.android.sai_chat" />
</intent-filter>
```

Part 4: View Messages



Step through todo's contained in:

1.) MessageListActivity
2.) MessageListFragment
3.) MessageArrayAdapter

Part 4: View Messages

- MessageListActivity

1.) Create and add the MessageFragment

2.) Set the title of the activity with AbstractToolbarActivity.setActivityTitle(String)

3.) What should the “<-” arrow in the top left of the window do?

Part 4: View Messages

- MessageListFragment

Use ideas from the ConversationListFragment to finish the 3 items here.

1.) put a <ListView … /> element in the res/layout/fragment_message_list.xml file

2.) Inflate and add the layout

3.) implement what happens when the send button is clicked

4.) Create the array adapter and add it to the list

Part 4: View Messages

- MessageArrayAdapter

Implement this class, taking note of performance enhancements. Similar to the ThreadArrayAdapter.

```
/**
 * This is the adapter to fill the message list for a thread.
 *
 * Adapter's are annoying in Android. Complex ones take a log of code and you just want to get to the
 * end of it to see the results... But, it isn't difficult work.
 *
 * What to learn from this:
 *      - This is just an ArrayAdapter. Android has a new RecyclerView class (New in 2014) that CAN be
 *          more effient, but it is also more complex with lot of different peices, they are a lot more powerful though.
 *          ArrayAdapter's and CursorAdapter's are simple, and can be made very efficient as well.
 *          Recycler views are based off of the ViewHolder pattern that originated in basic Array and Cursor Adapters.
 *          It is the pattern that we use in these classes.
 *
 *      - ViewHolder keeps an instance of the different attributes of your views so that you do not have
 *          to find them within your layouts for every single element in the list (using the findViewById() function).
 *          It recycles the views from the already inflated layout and reuses their tags.
 *          Hard to explain, so lets check out the code.
 */
```

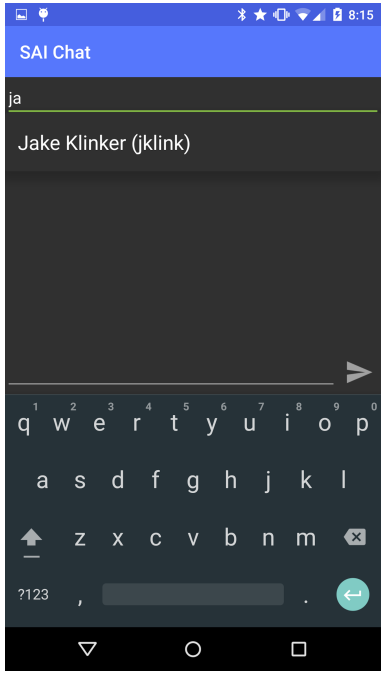
Part 5: Reply to Conversations

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:layout_gravity="bottom" >

    <EditText
        android:id="@+id/reply_text"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:capitalize="sentences"
        android:layout_height="wrap_content" />

    <ImageButton
        android:id="@+id/send_button"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:src="@drawable/ic_action_send"
        android:background="@android:color/transparent"
        android:layout_gravity="center_vertical"/>

</LinearLayout>
```

Reply to Messages

- Add to the xml
- Implement their functionality

Part 6: Compose New Conversations



- Fill and add the layout
- Find the views
- Implement the clicks
- Implement the AutoComplete

```
<AutoCompleteTextView
    android:id="@+id/user_auto_complete"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

```
userAutoComplete.setThreshold(1);
userAutoComplete.setAdapter(adapter);
```

What can I do by myself?

- Implement the *Deleter* methods:
  - How would you go about deleting those items?
  - Long click on the list view
  - *Deleter* util will remove them on backend, remove them on the device with the DataSources
- Settings page:
  - Text size, color, notification settings, etc

Extra Topics

- Databases
- Data Calls
- Custom Widget
  - LabelledEditText
- System BroadcastReceivers
  - BootReceiver
- Toolbar & Material Design
  - AbstractToolbarActivity