Android development

Tiberiu Vilcu
Prepared for EECS 411
Sugih Jamin
15 September 2017

Outline

1. Set up Android Studio
2. Create sample app
3. Add UI to see how the design interface works
4. Add some code to give the app functionality
5. Run and test to verify and debug

Android Studio

- Modern IDE for Java, XML, C++
- Built-in Java environment
- Built-in debugger and Android Debug Bridge
- Android product emulators
- Capable of version control (git)
- Cross-platform (Windows and macOS)

Setting up Android Studio

https://developer.android.com/studio/index.html

1. Install and select “Recommended” installation
2. Download and install all components it asks

Warnings about Android Studio

- Android compatibility is fragile
- Different API versions have widely different use percentages
- Code may sometimes be underlined while correct as it takes time to refactor
- Build often to view errors

Outline

1. Set up Android Studio
2. Create sample app
3. Add UI to see how the design interface works
4. Add some code to give the app functionality
5. Run and test to verify and debug

Sample project: Chatter

- Has view to write a post and submit to server
- Has timeline view to see posts
- Is cheaply inspired by Twitter
- Has a live web API already:
  http://159.203.172.151/getchatts/
  http://159.203.172.151/addchatt/
  (we will create this in a later lecture as well)

Setting up an Android project

1. “Start a new Android Studio Project”
2. Set the application name, domain, and location (domain like example.com)
3. Select phone and tablet (only) and API version 17
4. Add an empty activity
5. Pick a good activity name and title
6. Make sure it generates a layout and has backwards compatibility

Android Studio window

- Left: project directory, structure, and capture panes
- Right: design and text editor
- Bottom: debugging output, terminal, version control panes

Adding GitHub remote

1. VCS —> Enable Version Control Integration (select Git)
2. File —> Other Settings —> Default Settings —> Version Control —> GitHub (add your login and password)
3. VCS —> Import into Version Control —> Share Project on GitHub
4. Select a new repo name and share (select all subdirectories)

Android project structure

- AndroidManifest.xml: general app settings and activity list
- /java: activity class code and more
- /res/layout: activity interface XML documents
- /res/values: constants for strings and designs
- Gradle: build scripts

Outline

1. Set up Android Studio
2. Create sample app
3. Add UI to see how the design interface works
4. Add some code to give the app functionality
5. Run and test to verify and debug

Adding to the timeline layout

1. Delete the default TextView
2. Drag and drop a FloatingActionButton
3. Select a drawable icon, name postChattActionButton
4. Add constraints to bottom and right of the button
5. Drag and drop a ListView, name chattListView
6. Set the width/height to match_parent, add 0 constraints around

Creating a new activity

1. Select java/com.example.chatter/ directory
2. Right click, New —> Activity —> Empty Activity
3. Name PostActivity, select Generate Layout File and Backwards Compatibility but not launcher Activity

Adding to the post activity

1. Add a TextView for the username and constrain to top center
2. Add a Multiline Text for the message, constrain left
3. Add a Button, constrain to right of message TextView
4. Make width and height of MultilineText wrap_content, give ID
5. Change button text and give ID

Creating a list cell layout

1. Select res/layout/ directory
2. Right click New —> Layout resource file
3. Name file chatt_item
4. Make Root element ConstraintLayout

Adding to the Chatt list cell layout

1. Add a TextView for the username, constrain to top left
2. Add a TextView for the timestamp, constrain to top right
3. Add a TextView for the message, make full-width with match_parent and height wrap_content, constrain to bottom
4. Name all widgets

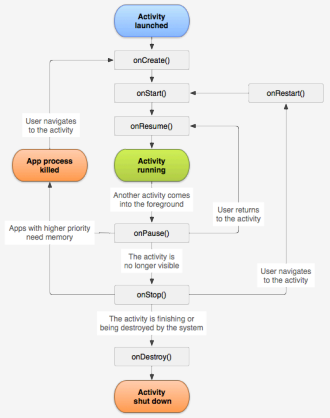
Outline

1. Set up Android Studio
2. Create sample app
3. Add UI to see how the design interface works
4. Add some code to give the app functionality
5. Run and test to verify and debug

Android app activity lifecycle

- onCreate(): activity first loads up
- onStart(): activity becomes visible to the user
- onResume(): activity comes to foreground and user interacts
- onPause(): user is leaving activity (but not destroying)
- onStop(): activity is no longer visible to user
- onDestroy(): activity being destroyed by user or system

Android app activity lifecycle



Adding navigation between activities

In TimelineActivity.java:

```
public void postChatt(View view) {
    Intent intent = new Intent(this, PostActivity.class);
    startActivity(intent);
}
```

For the action button, make onClick postChatt()

[alt+enter (macOS) to auto-import classes]

Adding navigation between activities

In AndroidManifest.xml:

```
<activity android:name=".PostActivity"
    android:parentActivityName=".TimelineActivity" >
    <!-- The meta-data tag is required if you support API level 15 and
lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".TimelineActivity" />
</activity>
```

Adds the back arrow in the PostActivity

Setting up HTTP connections

In AndroidManifest.xml manifest:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

In build.gradle dependencies:

```
compile 'com.android.volley:volley:1.0.0'
```

Volley is a library used to make HTTP requests

Making an HTTP POST request

```
public void submitChatt(View view) {
    RequestQueue queue = Volley.newRequestQueue(this);

    String url = "http://159.203.172.151/addchatt/";
    TextView usernameTextView = (TextView) findViewById(R.id.usernameTextView);
    TextView messageTextView = (TextView) findViewById(R.id.messageEditText);
    String username = usernameTextView.getText().toString();
    String message = messageTextView.getText().toString();
    HashMap<String, String> params = new HashMap<String, String>();
    params.put("username", username);
    params.put("message", message);

    JsonObjectRequest postRequest = new JsonObjectRequest(url, new JSONObject(params),
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    finish();
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            finish();
        }
    });

    queue.add(postRequest);
}
```

Add the onClick listener to the submitButton

Adding a custom Java class

1. Select java/com.example.chatter directory
2. Right click —> New —> Java Class
3. Name Chatt and leave default options

Adding a custom Java class for Chatt

```
public class Chatt {
    public String username;
    public String message;
    public String timestamp;

    public Chatt(String username, String message, String timestamp) {
        this.username = username;
        this.message = message;
        this.timestamp = timestamp;
    }
}
```

Adding a list adapter

Create a new Java class ChattAdapter the same as before

List adapters used to link elements in lists to ListViews

Adding a list adapter

```
public class ChattAdapter extends ArrayAdapter<Chatt> {
    public ChattAdapter(Context context, ArrayList<Chatt> users) {
        super(context, 0, users);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Chatt chatt = getItem(position);
        if (convertView == null) {
            convertView = LayoutInflater.from(getContext()).inflate(R.layout.chatt_item, parent,
false);
        }
        TextView usernameTextView = (TextView) convertView.findViewById(R.id.usernameTextView);
        TextView messageTextView = (TextView) convertView.findViewById(R.id.messageTextView);
        TextView timestampTextView = (TextView) convertView.findViewById(R.id.timestampTextView);
        usernameTextView.setText(chatt.username);
        messageTextView.setText(chatt.message);
        timestampTextView.setText(chatt.timestamp);
        return convertView;
    }
}
```

Setting up the list adapter

```
private ArrayList<Chatt> chattArrayList;
private ChattAdapter chattAdapter;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_timeline);

    chattArrayList = new ArrayList<Chatt>();
    chattAdapter = new ChattAdapter(this, chattArrayList);
    ListView lView = (ListView) findViewById(R.id.chattListView);
    lView.setAdapter(chattAdapter);
}
```

Adding a function to refresh

Call function in onCreate() as well

```
private void refreshTimeline() {
    chattAdapter.clear();
    RequestQueue queue = Volley.newRequestQueue(this);
    final String url = "http://159.203.172.151/getchatts/";
    queue.add(getRequest);
}
```

HTTP GET request and adding to list

```
JsonObjectRequest getRequest = new JsonObjectRequest(Request.Method.GET, url, null,
          new Response.Listener<JSONObject>()
          {
              @Override
              public void onResponse(JSONObject response)  {
                  try {
                      JSONArray array = response.getJSONArray("chatts");
                      for (int i = 0; i < array.length(); i++) {
                          String username = array.getJSONArray(i).getString(0);
                          String message = array.getJSONArray(i).getString(1);
                          String timestamp = array.getJSONArray(i).getString(2);
                          chattAdapter.add(new Chatt(username, message, timestamp));
                      }
                  }
                  catch (JSONException e) {
                  }
              }
          },
          new Response.ErrorListener()
          {
              @Override
              public void onErrorResponse(VolleyError error) {
              }
          }
   );
```

Pull-to-refresh the timeline

Around ListView in activity_timeline.xml:

```
<android.support.v4.widget.SwipeRefreshLayout xmlns:android="http://
schemas.android.com/apk/res/android"
    android:id="@+id/refreshContainer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

</android.support.v4.widget.SwipeRefreshLayout>
```

Pull-to-refresh the timeline

```
private SwipeRefreshLayout refreshContainer;
```

Set up in onCreate():

```
refreshContainer = (SwipeRefreshLayout) findViewById(R.id.refreshContainer);
refreshContainer.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
    @Override
    public void onRefresh() {
        refreshTimeline();
    }
});
```

To stop refreshing animation whenever needed

```
refreshContainer.setRefreshing(false);
```

Outline

1. Set up Android Studio
2. Create sample app
3. Add UI to see how the design interface works
4. Add some code to give the app functionality
5. Run and test to verify and debug

Running on an Android emulator

1. Hit the green play button
2. Create New Virtual Device
3. Pick a device and system image (Nexus 5X and O x86 are good)
4. Give it a name and leave the default settings
5. Use same selection for future launches and hit OK
6. Use yellow lightning icon in Android Studio to quickly rebuild and refresh app
7. Close emulator entirely with Cmd+Q

Possible improvements

- Error handling
- Using constant string values
- Automatically add new post to timeline without refreshing
- Don't delete and refresh the entire timeline

Tips before you begin

- Learn how to use Java and XML as you go
- Plan your activities and how they interact ahead of time
- Name your widgets and variables carefully; refactoring can cause problems
- Don't waste too much time on icons

Resources

- Sample project repo: https://github.com/UM-EECS-441/android-project-sample-f17
- Android Studio: https://developer.android.com/studio/index.html
- Android programming tutorial: https://developer.android.com/training/index.html
- App activity lifecycle: https://developer.android.com/guide/components/activities/activity-lifecycle.html
- Google Play developer account: https://support.google.com/googleplay/android-developer/answer/6112435?hl=en