**Android Help File**

by Talha Saqib

**Table of Contents**

[1. EXIT APP 4](#_Toc34608121)

[2. REMOVE ACTION BAR 4](#_Toc34608122)

[3. MOVE TO ANOTHER ACTIVITY 4](#_Toc34608123)

[4. TABS 4](#_Toc34608124)

[5. BACK BUTTON 6](#_Toc34608125)

[6. BOTTOM NAVIGATION BAR 6](#_Toc34608126)

[7. CIRCULAR IMAGE 8](#_Toc34608127)

[8. NESTED TABS 8](#_Toc34608128)

[9. ROUNDED BUTTON 11](#_Toc34608129)

[10. RECYLER VIEW 12](#_Toc34608130)

[11. CHECKBOXES IN RECYCLER VIEW 15](#_Toc34608131)

[12. INCLUDE ANOTHER LAYOUT FILE 16](#_Toc34608132)

[13. MOVE TO OTHER ACTIVITY 16](#_Toc34608133)

[14. ASYNC TASK 16](#_Toc34608134)

[15. ON SAVED INSTANCE STATE 18](#_Toc34608135)

[16. HIDE KEYBOARD 18](#_Toc34608136)

[17. CHECKING INTERNET CONNECTION 19](#_Toc34608137)

[18. HITTING API VIA INTERNET 19](#_Toc34608138)

[19. READING THE (above) API'S JSON RESPONSE 21](#_Toc34608139)

[20. CONVERTING STRING (having JSON response) TO JSON OBJECT 22](#_Toc34608140)

[21. REGISTER SYSTEM BROADCAST 22](#_Toc34608141)

[22. CUSTOM LOCAL BROADCAST 23](#_Toc34608142)

[23. IMPLICIT INTENT 24](#_Toc34608143)

[24. SHARING (text) 26](#_Toc34608144)

[25. RECEIVING IMPLICIT INTENT 26](#_Toc34608145)

[26. NOTIFICATION CHANNEL 27](#_Toc34608146)

[27. CREATE AND SEND NOTIFICATION 28](#_Toc34608147)

[28. UPDATE AND CANCEL NOTIFICATION 29](#_Toc34608148)

[29. ADDING ACTION TO NOTIFICATION 30](#_Toc34608149)

[30. JOB SCHEDULER 32](#_Toc34608150)

[31. WORK MANAGER 35](#_Toc34608151)

[32. TAG AND DISPLAY WORK STATUS 38](#_Toc34608152)

[33. CANCELLING WORK 40](#_Toc34608153)

[34. WORK CONSTRAINTS 40](#_Toc34608154)

[35. INTENT SERVICE 41](#_Toc34608155)

[36. SERVICE 42](#_Toc34608156)

[37. FOREGROUND SERVICE 43](#_Toc34608157)

[38. SHARED PREFERENCES 44](#_Toc34608158)

[39. SETTING 45](#_Toc34608159)

[40. PERMISSIONS 50](#_Toc34608160)

[41. RETROFIT 52](#_Toc34608161)

[42. ROOM 56](#_Toc34608162)

[43. RECYLER VIEW SWIPE 67](#_Toc34608163)

[44. Firebase 68](#_Toc34608164)

[45. Firebase Storage 72](#_Toc34608165)

[46. Firebase Cloud 74](#_Toc34608166)

(Do include try catches in functions)

# ****EXIT APP****

finish();

System.exit(0);

# REMOVE ACTION BAR

In Manifest file:

<activity android:name=".SignUp"

android:theme="@style/AppTheme.NoActionBar">

# MOVE TO ANOTHER ACTIVITY

Intent intent = new Intent(getApplicationContext(), activity.class);

startActivity(intent);

# TABS

- Start a "tabbed" activity

- Delete Placeholder Fragment

- Create a new blank fragment along with resource files

- In SectionsPagerAdapter:

**public Fragment getItem(int position)**

{

Fragment frag = null;

switch(position)

{

case 0:

frag = new CustomFragment();

break;

}

return frag;

}

private static final int[] TAB\_TITLES = new int[]{R.string.tab\_text\_1, R.string.tab\_text\_2, R.string.tab\_text\_3, R.string.tab\_text\_4};

**public CharSequence getPageTitle(int position)**

{

return mContext.getResources().getString(TAB\_TITLES[position]);

}

**public int getCount()**

{

// For 4 fragments

return 4;

}

**// TAB ICONS**

**private void setupTabIcons()**

{

tabs.getTabAt(0).setIcon(R.drawable.ic\_assessment\_black\_24dp);

tabs.getTabAt(1).setIcon(R.drawable.ic\_av\_timer\_black\_24dp);

tabs.getTabAt(2).setIcon(R.drawable.ic\_brightness\_5\_black\_24dp);

tabs.getTabAt(3).setIcon(R.drawable.ic\_assignment\_black\_24dp);

}

# BACK BUTTON

- In manifest, inside the activity: android:parentActivityName=".ActivityName"

- In resource file, add Toolbar view

- In activity:

**void setBackButton()**

{

Toolbar toolbar = findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

ActionBar actionBar = getSupportActionBar();

if (actionBar != null)

getSupportActionBar().setDisplayHomeAsUpEnabled(true);

}

# BOTTOM NAVIGATION BAR

**- Create a menu file having items:**

<item android:id="home"

android:icon="@drawable/ic\_home\_black\_24dp"

android:title="@string/home"/>

**- Then in activity xml:**

<com.google.android.material.bottomnavigation.BottomNavigationView

android:id="@+id/bottom\_nav"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="bottom"

app:menu="@menu/menu\_bottom\_nav"

android:background="?android:attr/windowBackground"/>

**- To make the selection listener:**

BottomNavigationView bottomNav = findViewById(R.id.bottom\_nav);

bottomNav.setOnNavigationItemSelectedListener(navListener);

private BottomNavigationView.OnNavigationItemSelectedListener navListener =

new BottomNavigationView.OnNavigationItemSelectedListener()

{

@Override

public boolean onNavigationItemSelected(@NonNull MenuItem menuItem)

{

Fragment frag = null;

switch(menuItem.getItemId())

{

case R.id.main:

frag = new Constraint();

break;

}

getSupportFragmentManager().beginTransaction().replace(R.id.frame, frag).commit();

return true;

};

<FrameLayout

android:layout\_width="match\_parent"

android:layout\_height="250dp"

android:id="@+id/frame"

android:layout\_marginTop="250dp" />

# CIRCULAR IMAGE

<androidx.cardview.widget.CardView

android:layout\_width="60dp"

android:layout\_height="60dp"

android:id="@+id/profilepic"

app:cardCornerRadius="40dp"

android:layout\_below="@+id/settings"

android:layout\_margin="18dp">

<ImageView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:src="@mipmap/ic\_launcher"

android:scaleType="centerCrop"

android:layout\_gravity="center"

/>

</androidx.cardview.widget.CardView>

# NESTED TABS

public class Relative extends Fragment

{

private static final String TAG = "MyActivity";

TabLayout tabs;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState)

{

**// Inflate the layout for this fragment**

View view = inflater.inflate(R.layout.fragment\_relative, container, false);

try

{

NestedSectionsPagerAdapter mSectionsPagerAdapter = new NestedSectionsPagerAdapter(getChildFragmentManager());

ViewPager mViewPager = view.findViewById(R.id.view\_pager1);

mViewPager.setAdapter(mSectionsPagerAdapter);

tabs = view.findViewById(R.id.tabs1);

tabs.setupWithViewPager(mViewPager);

setupTabIcons();

}

catch (Exception e)

{

Log.e(TAG, "onCreateView: Error = " + e);

}

return view;

}

private void setupTabIcons()

{

try

{

tabs.getTabAt(0).setIcon(R.drawable.ic\_center\_focus\_strong\_black\_24dp);

tabs.getTabAt(1).setIcon(R.drawable.ic\_equalizer\_black\_24dp);

}

catch (Exception e)

{

Log.e(TAG, "setupTabIcons: Error = " + e);

}

}

}

class NestedSectionsPagerAdapter extends FragmentPagerAdapter

{

public NestedSectionsPagerAdapter(FragmentManager fm) {

super(fm);

}

@Override

public Fragment getItem(int position) {

Fragment frag = null;

switch (position) {

case 0:

frag = new NestedFrag1();

break;

case 1:

frag = new NestedFrag2();

break;

}

return frag;

}

@Override

public int getCount() {

return 2;

}

}

**XML:**

<androidx.viewpager.widget.ViewPager

android:id="@+id/view\_pager1"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"

android:layout\_below="@id/web"

android:layout\_marginTop="12dp">

<com.google.android.material.tabs.TabLayout

android:id="@+id/tabs1"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

/>

</androidx.viewpager.widget.ViewPager>

# ROUNDED BUTTON

**- In drawable, create a file round\_button.xml**

<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item android:state\_pressed="false">

<shape android:shape="rectangle">

<corners android:radius="40dp" />

<solid android:color="#fa09ad"/>

</shape>

</item>

<item android:state\_pressed="true">

<shape android:shape="rectangle">

<corners android:radius="40dp" />

<solid android:color="#c20586"/>

</shape>

</item>

</selector>

**- Set this as background in button tag**

android:background="@drawable/round\_button"

# RECYLER VIEW

**- Tutorial: https://www.androidhive.info/2016/01/android-working-with-recycler-view/**

**- XML**

<androidx.recyclerview.widget.RecyclerView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/recycler\_view"

android:scrollbars="vertical"

android:layout\_marginBottom="30dp"

/>

**- Also make row.xml**

{

recyclerView = view.findViewById(R.id.recycler\_view);

**// use this setting to improve performance if you know that changes**

**// in content do not change the layout size of the RecyclerView**

recyclerView.setHasFixedSize(true);

**// use a linear layout manager**

layoutManager = new LinearLayoutManager(getContext());

recyclerView.setLayoutManager(layoutManager);

mAdapter = new MyAdapter(data);

recyclerView.setAdapter(mAdapter);

data.clear();

filldata();

mAdapter.notifyDataSetChanged();

}

class MyAdapter extends RecyclerView.Adapter<MyAdapter.MyViewHolder>

{

private List<String> strings;

private final SparseBooleanArray array1 =new SparseBooleanArray();

public class MyViewHolder extends RecyclerView.ViewHolder

{

**// Provide a reference to the views for each data item**

public TextView string;

public MyViewHolder(View view)

{

super(view);

string = view.findViewById(R.id.textView99);

}

}

public MyAdapter(List<String> strings)

{

this.strings = strings;

}

**// Create new views (invoked by the layout manager)**

@Override

public MyAdapter.MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType)

{

View itemView = LayoutInflater.from(parent.getContext())

.inflate(R.layout.recycler\_row, parent, false);

return new MyViewHolder(itemView);

}

**// Replace the contents of a view (invoked by the layout manager)**

@Override

public void onBindViewHolder(MyViewHolder holder, final int position)

{

try

{

String string = strings.get(position);

holder.string.setText(string);

}

catch (Exception e)

{

Log.e("TAG", "onBindViewHolder: Error " + e);

}

}

**// Return the size of your dataset (invoked by the layout manager)**

@Override

public int getItemCount()

{

return strings.size();

}

}

# CHECKBOXES IN RECYCLER VIEW

public MyViewHolder(View view)

{

checkBox = view.findViewById(R.id.checkBox);

checkBox.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v)

{

if(array1.get(getAdapterPosition()))

array1.put(getAdapterPosition(), false);

else

array1.put(getAdapterPosition(), true);

notifyDataSetChanged();

}

});

}

public void onBindViewHolder(MyViewHolder holder, final int position)

{

**// To avoid duplications of checked checkboxes**

if(array1.get(position))

holder.checkBox.setChecked(true);

else

holder.checkBox.setChecked(false);

}

# INCLUDE ANOTHER LAYOUT FILE

-In xml:

<include layout="@layout/another\_layout" />

# MOVE TO OTHER ACTIVITY

Intent i = new Intent(getApplicationContext(),NextActivity.class);

startActivity(i);

# ASYNC TASK

public class MyAsyncTask extends AsyncTask<Void, Void, String>

{

WeakReference<TextView> mtextView;

**MyAsyncTask(TextView textView)**

{ mtextView = new WeakReference<>(textView); }

@Override

**protected String doInBackground(Void... voids)**

{

Random r = new Random();

int n = r.nextInt(11);

int s = n\*200;

try

{

Thread.sleep(s);

}

catch(InterruptedException e)

{

Log.e("TAG", "doInBackground: " + e );

}

return "Awake at last after sleeping for " + s + " milliseconds!";

}

@Override

**protected void onPostExecute(String result)**

{

//Weak References are accessed by .get()

mtextView.get().setText(result);

}

{

**//Initiating Activity**

new MyAsyncTask(mTextView).execute();

} }

# ON SAVED INSTANCE STATE

private static final String KEY\_TEXT = "currentText";

**- In oncreate():**

// Restore TextView if there is a savedInstanceState

if(savedInstanceState!=null)

{

mTextView.setText(savedInstanceState.getString(KEY\_TEXT));

}

**- In onSaveInstanceState(@NonNull Bundle outState):**

{

// these are stored in key-value pairs

outState.putString(KEY\_TEXT, mTextView.getText().toString());

}

# HIDE KEYBOARD

**- Put the following code in a function that is called when a button is pressed e.g search button**

//Hide keyboard

InputMethodManager inputManager = (InputMethodManager)

getSystemService(Context.INPUT\_METHOD\_SERVICE);

if (inputManager != null)

{

inputManager.hideSoftInputFromWindow(view.getWindowToken(),

InputMethodManager.HIDE\_NOT\_ALWAYS);

}

# CHECKING INTERNET CONNECTION

**//Getting Network info**

ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY\_SERVICE);

NetworkInfo networkInfo = null;

if (connMgr != null)

{

**// ConnectivityManager.TYPE\_WIFI, ConnectivityManager.TYPE\_MOBILE are params of the below method to get WIFI and mobile network info respectively.**

networkInfo = connMgr.getActiveNetworkInfo();

}

**// Checking internet connection**

if (networkInfo != null && networkInfo.isConnected())

{

//do something

}

# HITTING API VIA INTERNET

-In manifest, include these before application tag:

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

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

class NetworkUtils

{

private static final String LOG\_TAG = NetworkUtils.class.getSimpleName();

// Base URL for Books API.

private static final String BOOK\_BASE\_URL = "https://www.googleapis.com/books/v1/volumes?";

// Parameter for the search string.

private static final String QUERY\_PARAM = "q";

// Parameter that limits search results.

private static final String MAX\_RESULTS = "maxResults";

// Parameter to filter by print type.

private static final String PRINT\_TYPE = "printType";

**static String getBookInfo(String queryString)**

{

// use "HttpsURLConnection" instead of http..

HttpURLConnection urlConnection = null;

BufferedReader reader = null;

try

{

Uri builtURI = Uri.parse(BOOK\_BASE\_URL).buildUpon()

.appendQueryParameter(QUERY\_PARAM, queryString)

.appendQueryParameter(MAX\_RESULTS, "10")

.appendQueryParameter(PRINT\_TYPE, "books")

.build();

// Converting URI to URL

URL requestURL = new URL(builtURI.toString());

urlConnection = (HttpURLConnection) requestURL.openConnection();

urlConnection.setRequestMethod("GET");

urlConnection.connect();

// Get the InputStream.

InputStream inputStream = urlConnection.getInputStream();

}

catch (IOException e)

{

e.printStackTrace();

}

finally

{

if (urlConnection != null) { urlConnection.disconnect(); }

if (reader != null)

{

try { reader.close(); }

catch (IOException e)

{ e.printStackTrace(); }

}

}

}

}

# READING THE (above) API'S JSON RESPONSE

BufferedReader reader = null;

reader = new BufferedReader(new InputStreamReader(inputStream));

// Use a StringBuilder to hold the incoming response.

StringBuilder builder = new StringBuilder();

// Reading Input

String line;

while ((line = reader.readLine()) != null)

{

builder.append(line);

builder.append("\n");

}

// Checking if result is empty

if (builder.length() == 0) { return null; }

bookJSONString = builder.toString();

return bookJSONString;

# CONVERTING STRING (having JSON response) TO JSON OBJECT

// Convert the response into a JSON object.

JSONObject jsonObject = new JSONObject(string);

// Get the JSONArray of book items.

JSONArray itemsArray = jsonObject.getJSONArray("items");

# REGISTER SYSTEM BROADCAST

public class CustomReceiver extends BroadcastReceiver

{

@Override

**public void onReceive(Context context, Intent intent)**

{

String intentAction = intent.getAction();

if (intentAction != null)

{

switch (intentAction)

{

case Intent.ACTION\_POWER\_CONNECTED:

toastMessage = "Power connected!";

break;

}

}

}

**- onCreate():**

mReceiver = new CustomReceiver();

filter = new IntentFilter();

filter.addAction(Intent.ACTION\_POWER\_DISCONNECTED);

filter.addAction(Intent.ACTION\_POWER\_CONNECTED);

// Registering the receiver using the activity context.

this.registerReceiver(mReceiver, filter);

**- onDestroy():**

//Unregistering the receiver to save system resources and memory leaks

this.unregisterReceiver(mReceiver);

super.onDestroy();

# CUSTOM LOCAL BROADCAST

// Our custom class that extends Broadcast Receiver and implements onReceive

private CustomReceiver mReceiver new CustomReceiver();

// declaring and initializing custom intent action string

private static final String ACTION\_CUSTOM\_BROADCAST =

BuildConfig.APPLICATION\_ID + ".ACTION\_CUSTOM\_BROADCAST";

**// Registering the local receiver.**

LocalBroadcastManager.getInstance(this)

.registerReceiver(mReceiver,

new IntentFilter(ACTION\_CUSTOM\_BROADCAST));

**// Unregistring Local Broadcast**

LocalBroadcastManager.getInstance(this).unregisterReceiver(mReceiver);

**// Sending custom broadcast**

public void sendCustomBroadcast(View view)

{

Intent customBroadcastIntent = new Intent(ACTION\_CUSTOM\_BROADCAST);

LocalBroadcastManager.getInstance(this).sendBroadcast(customBroadcastIntent);

}

**//Now handle the receiving of the broadcast in onRecive() of Custom Broadcast class**

case ACTION\_CUSTOM\_BROADCAST:

toastMessage = "Custom Broadcast Received";

break;

# IMPLICIT INTENT

public void openWebsite(View view)

{

String url = mWebsiteEditText.getText().toString();

Uri webpage = Uri.parse(url);

// Implicit Intent

Intent intent = new Intent(Intent.ACTION\_VIEW, webpage);

// Find an activity to handle the intent and start that activity.

if (intent.resolveActivity(getPackageManager()) != null)

{

startActivity(intent);

}

else

{

Toast.makeText(this, "Open Website intent didn't work", Toast.LENGTH\_SHORT).show();

}

}

public void openLocation(View view)

{

String loc = mLocationEditText.getText().toString();

Uri addressUri = Uri.parse("geo:0,0?q=" + loc);

Intent intent = new Intent(Intent.ACTION\_VIEW, addressUri);

if (intent.resolveActivity(getPackageManager()) != null)

{

startActivity(intent);

}

}

# SHARING (text)

String txt = mShareTextEditText.getText().toString();

String mimeType = "text/plain";

//Sharing data with other apps

ShareCompat.IntentBuilder

.from(this)

.setType(mimeType)

.setChooserTitle("Share this text with: ")

.setText(txt)

.startChooser();

# RECEIVING IMPLICIT INTENT

- In manifest.xm/activity (which you want to receive intent):

// Intent to recieve URLS

<intent-filter>

// Any Intent with view actions.

<action android:name="android.intent.action.VIEW" />

// This category must be included for your Activity to receive any implicit Intent.

<category android:name="android.intent.category.DEFAULT" />

// Requests for browsable links from web pages, email, or other sources.

<category android:name="android.intent.category.BROWSABLE" />

// 'data' tag is restricting the urls

<data android:scheme="https" android:host="\*.com" />

</intent-filter>

**- onCreate():**

Intent intent = getIntent();

//Intent data is always URI

Uri uri = intent.getData();

if (uri != null)

{

// In this case, it was a url

String uri\_string = "URI: " + uri.toString();

Toast.makeText(this, ""+uri\_string, Toast.LENGTH\_SHORT).show();

}

# NOTIFICATION CHANNEL

**- call this method in onCreate()**

//Creates a Notification channel, for OREO and higher.

public void createNotificationChannel()

{

mNotifyManager = (NotificationManager) getSystemService(NOTIFICATION\_SERVICE);

//Because notification channels are only available in API 26 and higher,

//add a condition to check for the device's API version.

if (android.os.Build.VERSION.SDK\_INT >= android.os.Build.VERSION\_CODES.O)

{

// Create a NotificationChannel

NotificationChannel notificationChannel = new NotificationChannel(PRIMARY\_CHANNEL\_ID,

"First Notification", NotificationManager.IMPORTANCE\_HIGH);

// Changing channel default settings

notificationChannel.enableLights(true);

notificationChannel.setLightColor(Color.RED);

notificationChannel.enableVibration(true);

notificationChannel.setDescription("Notification from Mascot");

mNotifyManager.createNotificationChannel(notificationChannel);

}

}

# CREATE AND SEND NOTIFICATION

private static final String PRIMARY\_CHANNEL\_ID = "primary\_notification\_channel";

private NotificationManager mNotifyManager;

private static final int NOTIFICATION\_ID = 0;

**// Create Notification**

private NotificationCompat.Builder getNotificationBuilder()

{

Intent notificationIntent = new Intent(this, MainActivity.class);

**// To open this activity when user taps on notification**

PendingIntent notificationPendingIntent = PendingIntent.getActivity(this,

NOTIFICATION\_ID, notificationIntent, PendingIntent.FLAG\_UPDATE\_CURRENT);

**//To clear notification**

Intent clearIntent = new Intent(ACTION\_CLEAR\_NOTIFICATION);

PendingIntent clearPendingIntent = PendingIntent.getBroadcast

(this, NOTIFICATION\_ID, clearIntent, PendingIntent.FLAG\_ONE\_SHOT);

**// Last 2 are for backward compatibility as we already specified these in channel**

NotificationCompat.Builder notifyBuilder =

new NotificationCompat.Builder(this, PRIMARY\_CHANNEL\_ID)

.setContentTitle("You've been notified!")

.setContentText("This is your notification text.")

.setSmallIcon(R.drawable.ic\_android)

.setDeleteIntent(clearPendingIntent)

.setContentIntent(notificationPendingIntent)

.setAutoCancel(true)

.setPriority(NotificationCompat.PRIORITY\_HIGH)

.setDefaults(NotificationCompat.DEFAULT\_ALL);

return notifyBuilder;

}

**public void sendNotification(View view)**

{

NotificationCompat.Builder notifyBuilder = getNotificationBuilder();

mNotifyManager.notify(NOTIFICATION\_ID, notifyBuilder.build());

}

# UPDATE AND CANCEL NOTIFICATION

public void updateNotification(View view)

{

// Adding this image in notification

Bitmap androidImage = BitmapFactory

.decodeResource(getResources(),R.drawable.mascot\_1);

NotificationCompat.Builder notifyBuilder = getNotificationBuilder();

notifyBuilder.setStyle(new NotificationCompat.BigPictureStyle()

.bigPicture(androidImage)

.setSummaryText("This is updated content")

.setBigContentTitle("Notification Updated!"));

mNotifyManager.notify(NOTIFICATION\_ID, notifyBuilder.build());

}

**public void cancelNotification(View view)**

{

mNotifyManager.cancel(NOTIFICATION\_ID);

}

# ADDING ACTION TO NOTIFICATION

private static final String ACTION\_UPDATE\_NOTIFICATION =

"com.example.android.notifyme.ACTION\_UPDATE\_NOTIFICATION";

//Nested class (not necessary but makes sense)

public class NotificationReceiver extends BroadcastReceiver

{

public NotificationReceiver() { }

View view = null;

String intentAction = intent.getAction();

if (intentAction != null)

{

switch (intentAction)

{

case ACTION\_UPDATE\_NOTIFICATION:

// Update the notification

updateNotification(view);

break;

case ACTION\_CLEAR\_NOTIFICATION:

cancelNotification(view);

break;

}

}

}

**- On create():**

private NotificationReceiver mReceiver = new NotificationReceiver();

// It may seem as if the broadcast sent by the notification only concerns your app and

// should be delivered with a LocalBroadcastManager. However, using a PendingIntent

// delegates the responsibility of delivering the notification to the Android framework.

// Because the Android runtime handles the broadcast, you cannot use LocalBroadcastManager.

IntentFilter filter = new IntentFilter();

filter.addAction(ACTION\_UPDATE\_NOTIFICATION);

filter.addAction(ACTION\_CLEAR\_NOTIFICATION);

registerReceiver(mReceiver, filter);

@Override

**protected void onDestroy()**

{

unregisterReceiver(mReceiver);

super.onDestroy();

}

**- At the beginning of sendNotification():**

// our custom action

Intent updateIntent = new Intent(ACTION\_UPDATE\_NOTIFICATION);

**//To make sure that this pending intent is sent and used only once, set FLAG\_ONE\_SHOT.**

PendingIntent updatePendingIntent = PendingIntent.getBroadcast

(this, NOTIFICATION\_ID, updateIntent, PendingIntent.FLAG\_ONE\_SHOT);

**//After notifyBuilder definition**

notifyBuilder.addAction(R.drawable.ic\_update, "Update Notification",

updatePendingIntent);

# JOB SCHEDULER

// works only with API >= 21

// The onStartJob() method is executed on the main thread, and therefore any long-running tasks

// must be offloaded to a different thread. In this app, you are simply posting a notification,

// which can be done safely on the main thread.

// onStartJob() returns a boolean indicating whether the job needs to continue on a separate thread. If true,

// the work is offloaded to a different thread, and your app must call jobFinished() explicitly in

// that thread to indicate that the job is complete. If false, the system knows that the job is

// completed by the end of onStartJob(), and the system calls jobFinished() on your behalf.

// The onStopJob() callback returns a boolean that determines what to do if the job is not finished.

// If the return value is true, the job is rescheduled; otherwise, the job is dropped.

**- In Manifest:**

<service

android:name=".NotificationJobService"

android:permission="android.permission.BIND\_JOB\_SERVICE"/>

public class JobServiceExample extends JobService

{

@Override

**public boolean onStartJob(JobParameters params)**

{

// false means the job will be done on main thread.

return false;

}

@Override

**public boolean onStopJob(JobParameters params)**

{

// true means that the job will be rescheduled upon failure.

return true;

}

}

**- In MainActivity:**

private JobScheduler mScheduler;

private static final int JOB\_ID = 0;

public void scheduleJob(View view)

{

// some arguments for setRequiredNetworkType

JobInfo.NETWORK\_TYPE\_NONE, JobInfo.NETWORK\_TYPE\_ANY, JobInfo.NETWORK\_TYPE\_UNMETERED

mScheduler = (JobScheduler) getSystemService(JOB\_SCHEDULER\_SERVICE);

**//getting the JobService class**

ComponentName serviceName = new ComponentName(getPackageName(),

JobServiceExample.class.getName());

**//creating jobbuilder**

JobInfo.Builder builder = new JobInfo.Builder(JOB\_ID, serviceName)

.setRequiredNetworkType(selectedNetworkOption)

.setRequiresDeviceIdle(mDeviceIdleSwitch.isChecked()) //didn't work last time

.setRequiresCharging(mDeviceChargingSwitch.isChecked()); // didn't work last time

//additional condition with an if check

if (seekBarSet)

{

builder.setOverrideDeadline(seekBarInteger \* 1000);

}

boolean constraintSet = selectedNetworkOption != JobInfo.NETWORK\_TYPE\_NONE

|| mDeviceChargingSwitch.isChecked() || mDeviceIdleSwitch.isChecked()

|| seekBarSet;

if (constraintSet)

{

**//schedule job**

mScheduler.schedule(builder.build());

Toast.makeText(this, "Job Scheduled, job will run when " +

"the constraints are met.", Toast.LENGTH\_SHORT).show();

}

else

{ Toast.makeText(this, "Please set at least one constraint", Toast.LENGTH\_SHORT).show(); } }

**public void cancelJobs(View view)**

{

if (mScheduler!=null)

{

mScheduler.cancelAll();

mScheduler = null;

Toast.makeText(this, "Jobs cancelled", Toast.LENGTH\_SHORT).show();

}

}

# WORK MANAGER

**- Theory:**

**Worker:** This is where you put the code for the actual work you want to perform in the background. You'll extend this class and override the doWork() method.

**WorkRequest**: This represents a request to do some work. You'll pass in your Worker as part of creating your WorkRequest. When making the WorkRequest you can also specify things like Constraints on when the Worker should run.

**WorkManager:** This class actually schedules your WorkRequest and makes it run. It schedules WorkRequests in a way that spreads out the load on system resources, while honoring the constraints you specify.

**- app/build.gradle:**

dependencies

{

implementation "androidx.work:work-runtime:$versions.work"

}

**- build.gradle:**

versions.work = "2.2.0"

//Worker Class

public class WorkerA extends Worker

{

private static final String TAG = WorkerA.class.getSimpleName();

public WorkerA(@NonNull Context context, @NonNull WorkerParameters workerParams)

{

super(context, workerParams);

}

@NonNull

@Override

**public Result doWork()**

{

if()

{

//success

return Result.success();

}

else

{

//failure

return Result.failure();

}

}

}

**// Add WorkRequest to save the image to the filesystem**

OneTimeWorkRequest save =

new OneTimeWorkRequest.Builder(SaveImageToFileWorker.class)

.build();

continuation = continuation.then(save);

**// Actually start the work**

continuation.enqueue();

}

**//Chaining Work**

private WorkManager mWorkManager;

mWorkManager = WorkManager.getInstance(application);

WorkContinuation continuation = mWorkManager.beginWith(workA);

continuation.then(workB) // FYI, then() returns a new WorkContinuation instance

.then(workC)

.enqueue(); // Enqueues the WorkContinuation which is a chain of work

**//Actual example**

**void applyBlur(int blurLevel)**

{

// Add WorkRequest to Cleanup temporary images

// WorkContinuation continuation =

// mWorkManager.beginWith(OneTimeWorkRequest.from(CleanupWorker.class));

// To make our work unique so that only one image can be blurred at a time

// REPLACE, KEEP, APPEND - 3 options for unqiue work

WorkContinuation continuation = mWorkManager

.beginUniqueWork(IMAGE\_MANIPULATION\_WORK\_NAME,

ExistingWorkPolicy.REPLACE,

OneTimeWorkRequest.from(CleanupWorker.class));

**// Add WorkRequest to blur the image**

OneTimeWorkRequest blurRequest = new OneTimeWorkRequest.Builder(BlurWorker.class)

.setInputData(createInputDataForUri())

.build();

continuation = continuation.then(blurRequest);

**// Add WorkRequest to save the image to the filesystem**

OneTimeWorkRequest save =

new OneTimeWorkRequest.Builder(SaveImageToFileWorker.class)

.addTag(TAG\_OUTPUT) // This adds the tag

.build();

continuation = continuation.then(save);

**// Actually start the work**

continuation.enqueue();

}

# TAG AND DISPLAY WORK STATUS

**- Add a tag when creating a OneTimeWorkRequest:**

// Add WorkRequest to save the image to the filesystem

OneTimeWorkRequest save =

new OneTimeWorkRequest.Builder(SaveImageToFileWorker.class)

.addTag(TAG\_OUTPUT) // This adds the tag

.build();

**- We can get WorkInfo by following:**

- getInfoByIdLiveData

- getWorkInfosForUniqueWorkLiveData

- getWorkInfosByTagLiveData

**- Getting WorkInfo:**

// New instance variable for the WorkInfo class

private LiveData<List<WorkInfo>> mSavedWorkInfo;

// Placed this code in the BlurViewModel constructor

mSavedWorkInfo = mWorkManager.getWorkInfosByTagLiveData(TAG\_OUTPUT);

// Add a getter method for mSavedWorkInfo

LiveData<List<WorkInfo>> getOutputWorkInfo() { return mSavedWorkInfo; }

**- Display WorkInfo:**

// Show work status, added in onCreate()

mViewModel.getOutputWorkInfo().observe(this, listOfWorkInfos -> {

// If there are no matching work info, do nothing

if (listOfWorkInfos == null || listOfWorkInfos.isEmpty()) {

return;

}

// We only care about the first output status.

// Every continuation has only one worker tagged TAG\_OUTPUT

WorkInfo workInfo = listOfWorkInfos.get(0);

boolean finished = workInfo.getState().isFinished();

if (!finished) {

showWorkInProgress();

} else {

showWorkFinished();

**// To show output from WorkInfo object**

Data outputData = workInfo.getOutputData();

String outputImageUri = outputData.getString(Constants.KEY\_IMAGE\_URI);

**// If there is an output file show "See File" button**

if (!TextUtils.isEmpty(outputImageUri)) {

mViewModel.setOutputUri(outputImageUri);

mOutputButton.setVisibility(View.VISIBLE);

}

}

});

# CANCELLING WORK

// Cancel work using the work's unique name

void cancelWork() {

mWorkManager.cancelUniqueWork(IMAGE\_MANIPULATION\_WORK\_NAME);

}

# WORK CONSTRAINTS

// Create charging constraint

Constraints constraints = new Constraints.Builder()

.setRequiresCharging(true)

.build();

// Add WorkRequest to save the image to the filesystem

OneTimeWorkRequest save = new OneTimeWorkRequest.Builder(SaveImageToFileWorker.class)

.setConstraints(constraints) // This adds the Constraints

.addTag(TAG\_OUTPUT)

.build();

# INTENT SERVICE

// This is a subclass of Service that uses a worker thread to handle all of the start requests, one at a time.

// This is the best option if you don't require that your service handle multiple requests simultaneously.

// Implement onHandleIntent(), which receives the intent for each start request so that you can complete the background work.

**- In manifest:**

<service

android:name="MyService"

android:icon="@drawable/ic\_android"

android:label="My Service">

</service>

public class MyService extends IntentService

{

public MyService("MyService")

{ super(); }

@Override

protected void onHandleIntent(Intent intent)

{ }

// For other callback methods, call the 'super'

}

# SERVICE

This is the base class for all services. When you extend this class, it's important to create a

new thread in which the service can complete all of its work; the service uses your application's

main thread by default, which can slow the performance of any activity that your application is running.

public class MyService extends Service

{

@Override

**public int onStartCommand(Intent intent, int flags, int startId)**

{

//TODO do something useful

return Service.START\_NOT\_STICKY;

}

@Override

**public IBinder onBind(Intent intent)**

{

//TODO for communication return IBinder implementation

return null;

}

}

**START\_NOT\_STICKY**

If the system kills the service after onStartCommand() returns, do not recreate the service unless there are pending intents to deliver.

This is the safest option to avoid running your service when not necessary and when your application can simply restart any unfinished jobs.

**START\_STICKY**

If the system kills the service after onStartCommand() returns, recreate the service and call onStartCommand(), but do not redeliver the last

intent. Instead, the system calls onStartCommand() with a null intent unless there are pending intents to start the service. In that case, those intents are delivered.

This is suitable for media players (or similar services) that are not executing commands but are running indefinitely and waiting for a job.

**START\_REDELIVER\_INTENT**

If the system kills the service after onStartCommand() returns, recreate the service and call onStartCommand() with the last intent that was delivered to the service.

Any pending intents are delivered in turn. This is suitable for services that are actively performing a job that should be immediately resumed, such as downloading a file.

**// STARTING THE SERVICE**

Intent intent = new Intent(this, MyService.class);

startService(intent);

**// STOPPING THE SERVICE**

- stopSelf(), called internally

- or stopService(Intent service), called externally

# FOREGROUND SERVICE

- For foreground service, we need an active notification

Intent notificationIntent = new Intent(this, ExampleActivity.class);

PendingIntent pendingIntent =

PendingIntent.getActivity(this, 0, notificationIntent, 0);

Notification notification =

new Notification.Builder(this, CHANNEL\_DEFAULT\_IMPORTANCE)

.setContentTitle(getText(R.string.notification\_title))

.setContentText(getText(R.string.notification\_message))

.setSmallIcon(R.drawable.icon)

.setContentIntent(pendingIntent)

.setTicker(getText(R.string.ticker\_text))

.build();

startForeground(ONGOING\_NOTIFICATION\_ID, notification);

# SHARED PREFERENCES

private SharedPreferences mPreferences;

// shared pref file can have any name, but conventionally, it is package's name.

private String sharedPrefFile = "com.example.android.packagename";

**- Putting data in shared pref:**

@Override

protected void onPause()

{

super.onPause();

SharedPreferences.Editor preferencesEditor = mPreferences.edit();

preferencesEditor.putInt(COUNT\_KEY, mCount);

// 'apply()' is for asynchronous operation and 'commit()' is for synchronous.

preferencesEditor.apply();

}

**- Getting data from shared pref:**

- In onCreate():

// MODE\_WORLD\_READABLE, MODE\_WORLD\_WRITEABLE are deprecated and are highly discouraged.

mPreferences = getSharedPreferences(sharedPrefFile, MODE\_PRIVATE);

// 'getInt()' takes a key value and a default value

mCount = mPreferences.getInt(COUNT\_KEY, 0);

mShowCountTextView.setText(String.format("%s", mCount));

**- Resetting shared pref:**

public void reset(View view)

{

// Clear preferences

SharedPreferences.Editor preferencesEditor = mPreferences.edit();

preferencesEditor.clear();

preferencesEditor.apply();

}

# SETTING

- Add "implementation 'androidx.preference:preference:1.1.0'" in 'app gradle' file

- We can create a settings activity via File->New->Activity->SettingsActivity

- By default, 'root\_preferences.xml' will be created that will contain the settings, their values. Its heirarchy should be: res/xml/root\_prefernces.xml

Sample 'root\_preferences.xml':

<PreferenceScreen xmlns:app="http://schemas.android.com/apk/res-auto">

<PreferenceCategory app:title="@string/messages\_header">

<EditTextPreference

app:key="signature"

app:title="@string/signature\_title"

app:useSimpleSummaryProvider="true" />

<ListPreference

app:defaultValue="reply"

app:entries="@array/reply\_entries"

app:entryValues="@array/reply\_values"

app:key="reply"

app:title="@string/reply\_title"

app:useSimpleSummaryProvider="true" />

</PreferenceCategory>

<PreferenceCategory app:title="@string/sync\_header">

<SwitchPreferenceCompat

app:key="sync"

app:title="@string/sync\_title" />

<SwitchPreferenceCompat

app:dependency="sync"

app:key="attachment"

app:summaryOff="@string/attachment\_summary\_off"

app:summaryOn="@string/attachment\_summary\_on"

app:title="@string/attachment\_title" />

</PreferenceCategory>

</PreferenceScreen>

**// arrays.xml for ListPreference above**

<resources>

<!-- Reply Preference -->

<string-array name="reply\_entries">

<item>Reply</item>

<item>Reply to all</item>

</string-array>

<string-array name="reply\_values">

<item>reply</item>

<item>reply\_all</item>

</string-array>

</resorces>

- You don't need to use Shared Preferences to store the values of these settings as they already use Shared Preferences by default.

- Here is the sample Settigns Activity:

public class SettingsActivity extends AppCompatActivity

{

@Override

**protected void onCreate(Bundle savedInstanceState)**

{

super.onCreate(savedInstanceState);

setContentView(R.layout.settings\_activity);

// To add, attach, detach, remove, replace a fragment we use a transaction via FragmentManager.

// Here, we need to attach the SettingsFragment to our SettingsActivity.

getSupportFragmentManager()

.beginTransaction()

.replace(R.id.settings, new SettingsFragment())

.commit();

ActionBar actionBar = getSupportActionBar();

if (actionBar != null)

{

actionBar.setDisplayHomeAsUpEnabled(true);

}

}

**// This is the actual settings fragment that contains the settings from the preferences xml**

public static class SettingsFragment extends PreferenceFragmentCompat

{

@Override

public void onCreatePreferences(Bundle savedInstanceState, String rootKey)

{

setPreferencesFromResource(R.xml.root\_preferences, rootKey);

}

}

}

**- To read settings data, we use 'DefaultSharedPreferences':**

SharedPreferences sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);

String signature = sharedPreferences.getString("signature", "");

**- To listen to Settings change event:**

public static SharedPreferences.OnSharedPreferenceChangeListener getSharedPreferencesListener()

{

SharedPreferences.OnSharedPreferenceChangeListener listener =

new SharedPreferences.OnSharedPreferenceChangeListener()

{

@Override

**public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key)**

{

if (key.equals("reply"))

{

Toast.makeText(context, sharedPreferences.getString(key, ""), Toast.LENGTH\_SHORT).show();

// do something here

}

}

};

return listener;

}

**- We have to register this listener, and for proper lifecycle management, we do it like this:**

@Override

public void onResume()

{

super.onResume();

sharedPreferences.registerOnSharedPreferenceChangeListener(getSharedPreferencesListener());

}

@Override

public void onPause()

{

super.onPause();

sharedPreferences.unregisterOnSharedPreferenceChangeListener(getSharedPreferencesListener());

}

# PERMISSIONS

- On all versions of Android, to declare that your app needs a permission, put a <uses-permission>

element in your app manifest, as a child of the top-level <manifest> element.

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

- Beginning with Android 6.0 (API level 23), users can revoke permissions from any app at any time, even if the app targets a lower API level. So even if the app used the camera yesterday, it can't assume it still has that permission today. To check premission:

public boolean checkPermission(String permission, int requestCode)

{

// Checking if permission is not granted

if (ContextCompat.checkSelfPermission(this, permission) == PackageManager.PERMISSION\_DENIED)

{

// Note: At this point, control transfers to the onRequestPermissionResult, so return false is probably 'useless'

ActivityCompat.requestPermissions(this, new String[] { permission }, requestCode);

return false;

}

else

{

Toast.makeText(MainActivity.this, "Permission already granted", Toast.LENGTH\_SHORT).show();

return true;

}

}

private static final int CONTACTS\_PERMISSION\_CODE = 1;

private static final int STORAGE\_PERMISSION\_CODE = 2;

// This function is called when user accept or decline the permission.

// Request Code is used to check which permission called this function.

// This request code is provided when user is prompt for permission.

@Override

**public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults)**

{

super.onRequestPermissionsResult(requestCode, permissions, grantResults);

if (requestCode == CAMERA\_PERMISSION\_CODE)

{

// Checking whether user granted the permission or not.

if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED)

{

// Do the required task

Toast.makeText(MainActivity.this,"Camera Permission Granted", Toast.LENGTH\_SHORT).show();

}

else

{

// Disable Functionality

Toast.makeText(MainActivity.this,"Camera Permission Denied", Toast.LENGTH\_SHORT).show();

}

}

else if (requestCode == STORAGE\_PERMISSION\_CODE)

{

if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED)

{

// Do the required task

Toast.makeText(MainActivity.this, "Storage Permission Granted", Toast.LENGTH\_SHORT).show();

}

else

{

// Disable Functionality

Toast.makeText(MainActivity.this,"Storage Permission Denied", Toast.LENGTH\_SHORT).show();

}

}

// 'else ifs' of other permissions

}

**//If any of the permission has been denied with “Never ask again” check, then that permission can only be allowed from the Settings of the app.**

**- To check multiple permission at one time**

Example code: <https://medium.com/mindorks/multiple-runtime-permissions-in-android-without-any-third-party-libraries-53ccf7550d0>

# RETROFIT

- In app's build.gradle:

implementation 'com.squareup.retrofit2:retrofit:2.1.0'

implementation 'com.squareup.retrofit2:converter-gson:2.1.0'

implementation 'com.squareup.okhttp3:logging-interceptor:4.3.1'

**- Also add the internet permission**

**- Retrofit annotated methods**

public interface JsonPlaceHolderApi

{

// 'posts' is the url path

// "Call<List<Post>>" is the return type

// '@Query' is for query strings, e.g url?id=x

@GET("posts")

Call<List<Post>> getPosts(@Query("userId") Integer[] userIds,

@Query("\_sort") String sort,

@Query("\_order") String order);

@GET("posts")

Call<List<Post>> getPosts(@QueryMap Map<String, String> params);

@GET("posts/{id}/comments")

Call<List<Comment>> getComments(@Path("id") int postId);

@GET

Call<List<Comment>> getComments(@Url String url);

@POST("posts")

Call<Post> createPost(@Body Post post);

@FormUrlEncoded

@POST("posts")

Call<Post> createPost(

@Field("userID") int userId,

@Field("title") String title,

@Field("body") String text

);

@FormUrlEncoded

@POST("posts")

Call<Post> createPost(@FieldMap Map<String, String> fields);

@Headers({"Static-Header1:123", "Static-Header2: 456"})

@PUT("posts/{id}")

Call<Post> putPost(@Header("Dynamic-Header") String header,

@Path("id") int id,

@Body Post post);

@PATCH("posts/{id}")

Call<Post> patchPost(@Path("id") int id, @Body Post post);

@DELETE("posts/{id}")

Call<Void> deletePost(@Path("id") int id);

}

**- Retrofit Setup:**

// Without this the null values get ignored

Gson gson = new GsonBuilder().serializeNulls().create();

**//For logging**

HttpLoggingInterceptor loggingInterceptor = new HttpLoggingInterceptor();

loggingInterceptor.setLevel(HttpLoggingInterceptor.Level.BODY);

OkHttpClient okHttpClient = new OkHttpClient.Builder()

.addInterceptor(loggingInterceptor)

.build();

Retrofit retrofit = new Retrofit.Builder()

.baseUrl("https://jsonplaceholder.typicode.com/")

.addConverterFactory(GsonConverterFactory.create(gson))

.client(okHttpClient)

.build();

jsonPlaceHolderApi = retrofit.create(JsonPlaceHolderApi.class);

**- Calling the method:**

public void updatePost()

{

// Post will be class containing variables and getters only

// We can use "@SerializedName("")" above variables to change their name in the request/response

Post post = new Post(12, null, null);

Call<Post> call = jsonPlaceHolderApi.putPost("Dynamic Header",5, post);

call.enqueue(new Callback<Post>()

{

@Override

**public void onResponse(Call<Post> call, Response<Post> response)**

{

if (!response.isSuccessful())

{

textViewResult.setText("Code: " + response.code());

return;

}

Post postResponse = response.body();

String content = "";

content += "Code: " + response.code() + "\n";

content += "ID: " + postResponse.getId() + "\n";

content += "User ID: " + postResponse.getUserId() + "\n";

content += "Title: " + postResponse.getTitle() + "\n";

content += "Text: " + postResponse.getText() + "\n\n";

textViewResult.setText(content);

}

@Override

**public void onFailure(Call<Post> call, Throwable t)**

{

textViewResult.setText(t.getMessage());

}

});

}

# ROOM

- In gradle file, "app":

// Room components

implementation "android.arch.persistence.room:runtime:$rootProject.roomVersion"

annotationProcessor "android.arch.persistence.room:compiler:$rootProject.roomVersion"

androidTestImplementation "android.arch.persistence.room:testing:$rootProject.roomVersion"

// Lifecycle components

implementation "android.arch.lifecycle:extensions:$rootProject.archLifecycleVersion"

annotationProcessor "android.arch.lifecycle:compiler:$rootProject.archLifecycleVersion"

**- In gradle file, "project":**

ext

{

roomVersion = '2.2.3'

archLifecycleVersion = '2.2.0'

}

**- ENTITY**

@Entity(tableName = "word\_table")

public class Word

{

@PrimaryKey

@NonNull

@ColumnInfo(name = "word")

private String mWord;

public Word(@NonNull String word) {this.mWord = word;}

// Every field that's stored in the database must either be public or have a "getter" method

public String getWord(){return this.mWord;}

}

**- DAO**

@Dao

public interface WordDao

{

@Insert(onConflict = OnConflictStrategy.IGNORE)

void insert(Word word);

@Query("DELETE FROM word\_table")

void deleteAll();

//LiveData help your app respond to data changes.

@Query("SELECT \* FROM word\_table ORDER BY word ASC")

LiveData<List<Word>> getAllWords();

@Query("SELECT \* from word\_table LIMIT 1")

Word[] getAnyWord();

@Delete

void deleteWord(Word word);

}

**- Room Database**

// Entities are tables in database.

@Database(entities = {Word.class}, version = 1, exportSchema = false)

public abstract class WordRoomDatabase extends RoomDatabase

{

public abstract WordDao wordDao();

private static WordRoomDatabase INSTANCE;

public static WordRoomDatabase getDatabase(final Context context)

{

if (INSTANCE == null)

{

synchronized (WordRoomDatabase.class)

{

if (INSTANCE == null)

{

// Create database here

INSTANCE = Room.databaseBuilder(context.getApplicationContext(),

WordRoomDatabase.class, "word\_database")

// Wipes and rebuilds instead of migrating

// if no Migration object.

// Migration is not part of this practical.

.fallbackToDestructiveMigration()

.addCallback(sRoomDatabaseCallback)

.build();

}

}

}

return INSTANCE;

}

private static RoomDatabase.Callback sRoomDatabaseCallback = new RoomDatabase.Callback()

{

//onOpen() creates and executes an AsyncTask to add content to the database.

@Override

public void onOpen (@NonNull SupportSQLiteDatabase db){

super.onOpen(db);

new PopulateDbAsync(INSTANCE).execute();

}

};

/\*\*

\* Populate the database in the background.

\*/

private static class PopulateDbAsync extends AsyncTask<Void, Void, Void> {

private final WordDao mDao;

String[] words = {"dolphin", "crocodile", "cobra"};

PopulateDbAsync(WordRoomDatabase db) {

mDao = db.wordDao();

}

@Override

protected Void doInBackground(final Void... params) {

for (int i = 0; i <= words.length - 1; i++)

{

Word word = new Word(words[i]);

mDao.insert(word);

}

return null;

}

}

}

**// Migration Scenerios:**

1: keep the database version unchanged — app crashes

2: version increased, but no migration provided — app crashes

3: version increased, fallback to destructive migration enabled — database is cleared

4: version increased, migration provided — data is kept

// Scenerio 4:

// Case 1, No Change in Schema:

@Database(entities = {User.class}, version = 2)

public abstract class UsersDatabase extends RoomDatabase {

…

static final Migration MIGRATION\_1\_2 = new Migration(1, 2) {

@Override

public void migrate(SupportSQLiteDatabase database) {

// Since we didn't alter the table, there's nothing else to do here.

}

};

…

database = Room.databaseBuilder(context.getApplicationContext(),

UsersDatabase.class, "Sample.db")

.addMigrations(MIGRATION\_1\_2)

.build();

// Case 2, Simple Change:

@Database(entities = {User.class}, version = 3)

public abstract class UsersDatabase extends RoomDatabase

static final Migration MIGRATION\_2\_3 = new Migration(2, 3) {

@Override

public void migrate(SupportSQLiteDatabase database) {

database.execSQL("ALTER TABLE users "

+ " ADD COLUMN last\_update INTEGER");

}

};

database = Room.databaseBuilder(context.getApplicationContext(),

UsersDatabase.class, "Sample.db")

.addMigrations(MIGRATION\_1\_2, MIGRATION\_2\_3)

.build();

// Case 3, Complex Change:

static final Migration MIGRATION\_3\_4 = new Migration(3, 4) {

@Override

public void migrate(SupportSQLiteDatabase database) {

// Create the new table

database.execSQL(

"CREATE TABLE users\_new (userid TEXT, username TEXT, last\_update INTEGER, PRIMARY KEY(userid))");

// Copy the data

database.execSQL(

"INSERT INTO users\_new (userid, username, last\_update) SELECT userid, username, last\_update FROM users");

// Remove the old table

database.execSQL("DROP TABLE users");

// Change the table name to the correct one

database.execSQL("ALTER TABLE users\_new RENAME TO users");

}

};

// For migrationg from 1 - 4 in single step:

static final Migration MIGRATION\_1\_4 = new Migration(1, 4)

and then: ".addMigrations(MIGRATION\_1\_2, MIGRATION\_2\_3, MIGRATION\_3\_4, MIGRATION\_1\_4)"

**- REPOSITORY**

public class WordRepository

{

private WordDao mWordDao;

private LiveData<List<Word>> mAllWords;

WordRepository(Application application)

{

WordRoomDatabase db = WordRoomDatabase.getDatabase(application);

mWordDao = db.wordDao();

mAllWords = mWordDao.getAllWords();

}

LiveData<List<Word>> getAllWords()

{ return mAllWords; }

public void insert (Word word)

{ new insertAsyncTask(mWordDao).execute(word); }

public void deleteAll() { new deleteAllWordsAsyncTask(mWordDao).execute(); }

public void deleteWord(Word word) { new deleteWordAsyncTask(mWordDao).execute(word); }

private static class insertAsyncTask extends AsyncTask<Word, Void, Void>

{

private WordDao mAsyncTaskDao;

insertAsyncTask(WordDao dao)

{ mAsyncTaskDao = dao; }

@Override

protected Void doInBackground(final Word... params)

{

mAsyncTaskDao.insert(params[0]);

return null;

}

}

private static class deleteAllWordsAsyncTask extends AsyncTask<Void, Void, Void>

{

private WordDao mAsyncTaskDao;

deleteAllWordsAsyncTask(WordDao dao)

{ mAsyncTaskDao = dao; }

@Override

protected Void doInBackground(Void... voids)

{

mAsyncTaskDao.deleteAll();

return null;

}

}

private static class deleteWordAsyncTask extends AsyncTask<Word, Void, Void>

{

private WordDao mAsyncTaskDao;

deleteWordAsyncTask(WordDao dao)

{ mAsyncTaskDao = dao; }

@Override

protected Void doInBackground(final Word... params) {

mAsyncTaskDao.deleteWord(params[0]);

return null;

}

}

}

- **VIEWMODEL**

public class WordViewModel extends AndroidViewModel

{

private WordRepository mRepository;

private LiveData<List<Word>> mAllWords;

public WordViewModel (Application application)

{

super(application);

mRepository = new WordRepository(application);

mAllWords = mRepository.getAllWords();

}

LiveData<List<Word>> getAllWords() { return mAllWords; }

public void insert(Word word) { mRepository.insert(word); }

public void deleteAll() {mRepository.deleteAll();}

public void deleteWord(Word word) {mRepository.deleteWord(word);}

}

**- CONNECT VIEWMODEL WITH UI**

private WordViewModel mWordViewModel;

mWordViewModel = ViewModelProviders.of(this).get(WordViewModel.class);

mWordViewModel.getAllWords().observe(this, new Observer<List<Word>>()

{

@Override

public void onChanged(@Nullable final List<Word> words)

{

// Update the cached copy of the words in the adapter.

adapter.setWords(words);

}

});

- in some function

mWordViewModel.insert(word);

# RECYLER VIEW SWIPE

- In adapter, a method to get return the position of item:

public Word getWordAtPosition (int position)

{

return mWords.get(position);

}

**- In activity, on create():**

// Add the functionality to swipe items in the

// recycler view to delete that item

ItemTouchHelper helper = new ItemTouchHelper(

new ItemTouchHelper.SimpleCallback(0,

ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT)

{

@Override

public boolean onMove(RecyclerView recyclerView,

RecyclerView.ViewHolder viewHolder,

RecyclerView.ViewHolder target) {

return false;

}

@Override

public void onSwiped(RecyclerView.ViewHolder viewHolder,

int direction)

{

// get the position of swiped viewholder

int position = viewHolder.getAdapterPosition();

Word myWord = adapter.getWordAtPosition(position);

Toast.makeText(MainActivity.this, "Deleting " +

myWord.getWord(), Toast.LENGTH\_LONG).show();

// Delete the word

mWordViewModel.deleteWord(myWord);

}

});

helper.attachToRecyclerView(recyclerView);

# Firebase

Don’t forget to lock the database when deploying the app by changing the **database rules**.

**- CONNECTIVITY:**

* Go to firebase website
* Add new project, Connect to project:
  + Write ‘package name’
  + Copy SHA-1 signature from Gradle->app->Tasks->android->signingReport
  + Download ‘services.json’ file
  + Copy the file in ‘app’ folder of the project
  + Copy the gradle lines in respective gradle files
  + Reinstall your app

Available Libraries: <https://firebase.google.com/docs/android/setup?authuser=0#available-libraries>

**-TO READ AND WRITE, WE NEED DATABASE REFERENCE:**

private DatabaseReference mDatabase;  
mDatabase = FirebaseDatabase.getInstance().getReference();

**-WRITING:**

//This will overwrite data

User user = new User(name, email);  
mDatabase.child("users").child(userId).setValue(user);

//To update

mDatabase.child("users").child(userId).child("username").setValue(name);

//if we don’t have an ‘ID’, use ‘push’ to generate a unique id

mDatabaseReference.push().setValue(friendlyMessageObj);

**-READING:**

* We don’t often need to request data manually, we can simply attach a listener to database reference which can automatically get values on change event:
* mChildEventListener = new ChildEventListener()  
  {  
   //'onChildAdded' gets called for newly added data and also once for already added data.  
   @Override  
   public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String s)  
   {  
   FriendlyMessage friendlyMessage = dataSnapshot.getValue(FriendlyMessage.class);  
   mMessageAdapter.add(friendlyMessage);  
   }  
   @Override  
   public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable String s) { }  
   @Override  
   public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) { }  
   @Override  
   public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable String s) { }  
   @Override  
   public void onCancelled(@NonNull DatabaseError databaseError) { }  
  };  
  mDatabaseReference.addChildEventListener(mChildEventListener);

**- DATABASE RULES CASCADING:**

When .read and .write rule permissions are evaluate to true, this cascades to all of the rule’s children. Only truth is cascading; falseness is not cascading. This means any child of the node that has true .read or .write rules is also true. If a parent has .read or .write true, this access cannot be revoked by a child node

**//An example of db rules with validation included**

{ "rules": {

"messages": {

// only authenticated users can read and write the messages node

".read": "auth != null",

".write": "auth != null",

"$id": {

// the read and write rules cascade to the individual messages

// messages should have a 'name' and 'text' key or a 'name' and 'photoUrl' key

".validate": "newData.hasChildren(['name', 'text']) && !newData.hasChildren(['photoUrl']) || newData.hasChildren(['name', 'photoUrl']) && !newData.hasChildren(['text'])" }}}

}

**- AUTHENTICATION:**

- Add dependencies from Firebase doc and Auth Github in gradle file.

public static final int *RC\_SIGN\_IN* = 1;

// In onCreate():

FirebaseAuth mFirebaseAuth = FirebaseAuth.*getInstance*();

FirebaseAuth.AuthStateListener mAuthStateListener = new FirebaseAuth.AuthStateListener() {  
 @Override  
 public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {  
 FirebaseUser user = firebaseAuth.getCurrentUser();  
 if (user != null)  
 {  
 // user is signed in  
 onSignedInInitialize(user.getDisplayName());  
 }  
 else  
 {  
 // user is signed out  
 onSignedOutCleanUp();

// This code block handles all the login/signup   
 startActivityForResult(  
 AuthUI.*getInstance*()  
 .createSignInIntentBuilder()  
 .setIsSmartLockEnabled(false)  
 .setAvailableProviders(Arrays.*asList*(  
 new AuthUI.IdpConfig.GoogleBuilder().build(),  
 new AuthUI.IdpConfig.EmailBuilder().build()))  
 .build(),  
 *RC\_SIGN\_IN*);  
 }  
 }  
};

// HELPER FUNCTIONS

private ChildEventListener mChildEventListener;

private void onSignedInInitialize(String username)  
{  
 mUsername = username;  
 attachDatabaseReadListener();  
}

private void onSignedOutCleanUp()  
{

// For this example  
 mUsername = *ANONYMOUS*;  
 mMessageAdapter.clear();

// General  
 detachDatabaseListener();  
}

private void attachDatabaseReadListener()  
{  
 if(mChildEventListener == null)  
 {  
 // Reading code written above.  
 }  
}

private void detachDatabaseListener()  
{  
 if(mChildEventListener != null)  
 {  
 mDatabaseReference.removeEventListener(mChildEventListener);  
 }  
 mChildEventListener = null;  
}

// TO AVOID ENDLESS LOOP WHILE CANCELLING LOGIN

@Override  
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data)  
{  
 super.onActivityResult(requestCode, resultCode, data);  
  
 if(requestCode == *RC\_SIGN\_IN*)  
 {  
 if(resultCode == *RESULT\_OK*)  
 {  
 Toast.*makeText*(this, "Signed in!", Toast.*LENGTH\_SHORT*).show();  
 }  
 else if(resultCode == *RESULT\_CANCELED*)  
 {  
 Toast.*makeText*(this, "Sign in Cancelled", Toast.*LENGTH\_SHORT*).show();  
 finish();  
 }  
 }  
}

// IT IS ALSO IMPORTANT TO DETACH A LISTNER AS UPON RECREATION OF THE ACTIVITY, LISTNEER WILL GET ATTACHED AGAIN.

// For correct lifecycle flow, attach and detach in ‘pause’ and ‘resume’

@Override  
protected void onPause() {  
 super.onPause();  
 if(mAuthStateListener != null) {  
 mFirebaseAuth.removeAuthStateListener(mAuthStateListener);  
 }  
 detachDatabaseListener();  
 mMessageAdapter.clear();  
}  
  
@Override  
protected void onResume() {  
 super.onResume();  
 mFirebaseAuth.addAuthStateListener(mAuthStateListener);  
}

// SIGN OUT

AuthUI.*getInstance*().signOut(this);

* **Change Firebase UI Theme:** <https://stackoverflow.com/questions/44492655/customize-firebase-ui-auth-android>

# Firebase Storage

private FirebaseStorage mFirebaseStorage;  
private StorageReference mChatPhotosStorageReference;

// In onCreate():

mFirebaseStorage = FirebaseStorage.*getInstance*();  
mChatPhotosStorageReference = mFirebaseStorage.getReference().child("chat\_photos");

// In onActivityResult():

else if(requestCode == *RC\_PHOTO\_PICKER* && resultCode == *RESULT\_OK*)  
{  
 Uri selectedImageUri = data.getData();  
  
 // Get a reference to store file at chat\_photos/<FILENAME>  
 final StorageReference photoRef = mChatPhotosStorageReference.child(selectedImageUri.getLastPathSegment());  
  
 // Upload file to Firebase Storage  
 photoRef.putFile(selectedImageUri).continueWithTask(new Continuation<UploadTask.TaskSnapshot, Task<Uri>>()  
 {  
 @Override  
 public Task<Uri> then(@NonNull Task<UploadTask.TaskSnapshot> task) throws Exception  
 {  
 if (!task.isSuccessful()) {  
 throw task.getException();  
 }  
 return photoRef.getDownloadUrl();  
 }  
 }).addOnCompleteListener(new OnCompleteListener<Uri>()  
 {  
 @Override  
 public void onComplete(@NonNull Task<Uri> task)  
 {  
 if (task.isSuccessful())  
 {  
 Uri downloadUri = task.getResult();  
 FriendlyMessage friendlyMessage = new FriendlyMessage(null, mUsername, downloadUri.toString());  
 mDatabaseReference.push().setValue(friendlyMessage);  
 }  
 else  
 {  
 Toast.*makeText*(MainActivity.this, "Upload Failed: " + task.getException().getMessage(), Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
 });  
}

**// Bonus: An image picker button**

// ImagePickerButton shows an image picker to upload a image for a message  
mPhotoPickerButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent intent = new Intent(Intent.*ACTION\_GET\_CONTENT*);  
 intent.setType("image/jpeg");  
 intent.putExtra(Intent.*EXTRA\_LOCAL\_ONLY*, true);  
 startActivityForResult(Intent.*createChooser*(intent, "Complete action using"), *RC\_PHOTO\_PICKER*);  
 }  
});

**// Firebase Storage Rules**

* Firebase Storage rules do not cascade to followed paths by default. In order to make them cascade, write the following syntax:
* match /{allPaths=\*\*}
* Firebase Storage rules start with the keyword ‘allow’ and are then followed by a condition.
* Conditions are based on contexts that either **‘request’** (for incoming data) or **‘resource’** (for existing data).
* For example rule that allows you to write to storage if the data is less than 3 MB:
* allow write: if request.resource.size < 3 \* 1024 \* 1024

**BONUS: FIREBASE CLOUD MESSAGING & REMOTE CONFIG**

* Add the ‘Cloud Messaging’ Library in the app
* Send Notifications via Firebase Console
* Remote Configs are very handy in changing some attributes in app as compared to sending full-scale updates.

# Firebase Cloud

// > COMMANDS AND STEPS:

// Firebase Cloud  
// npm install -g firebase-tools  
// firebase login  
// firebase init functions  
// ...Write the code  
// firebase deploy  
  
  
const functions = require('firebase-functions');  
  
// replaces keywords with emoji in the "text" key of messages  
// pushed to /messages  
exports.emojify = functions.database.ref('/messages/{pushId}/text')  
 .onWrite(event =>  
 {  
 // Database write events include new, modified, or deleted  
 // database nodes. All three types of events at the specific  
 // database path trigger this cloud function.  
 // For this function we only want to emojify new database nodes,  
 // so we'll first check to exit out of the function early if  
 // this isn't a new message.  
  
 // !event.data.val() is a deleted event  
 // event.data.previous.val() is a modified event  
 if (!event.data.val() || event.data.previous.val()) {  
 console.log("not a new write event");  
 return;  
 }  
  
 // Now we begin the emoji transformation  
 console.log("emojifying!");  
  
 // Get the value from the 'text' key of the message  
 const originalText = event.data.val();  
 const emojifiedText = emojifyText(originalText);  
  
 // Return a JavaScript Promise to update the database node  
 return event.data.ref.set(emojifiedText);  
 });  
  
// Returns text with keywords replaced by emoji  
// Replacing with the regular expression /.../ig does a case-insensitive  
// search (i flag) for all occurrences (g flag) in the string  
function emojifyText(text)  
{  
 var emojifiedText = text;  
 emojifiedText = emojifiedText.replace(/\blol\b/ig, "😂");  
 emojifiedText = emojifiedText.replace(/\bcat\b/ig, "😸");  
 return emojifiedText;  
}