APPLICATION DEVELOPMENT FOR MOBILE AND UBIQUITOUS COMPUTING

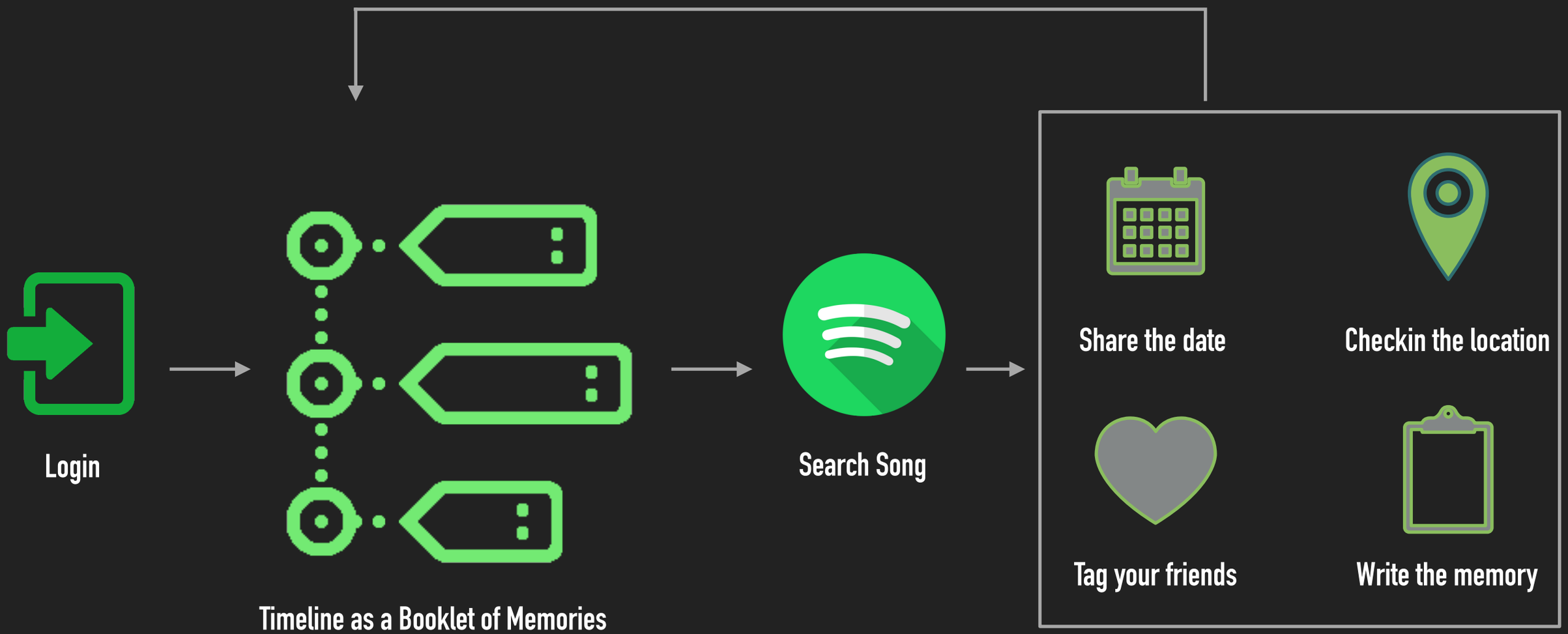
MEMOSONG

Final Presentation

Group No. 9
Team: Shermin Shojaei - Niloofar Gheibi

Dresden - 26.01.18

APP SCENARIO



CONTEXT

Location

Time (Date)

PHYSICAL

Network

TECHNICAL

Spotify Account (email)

PERSONAL

OFFLINE CHALLENGE

Splash Activity

- Responsibility:

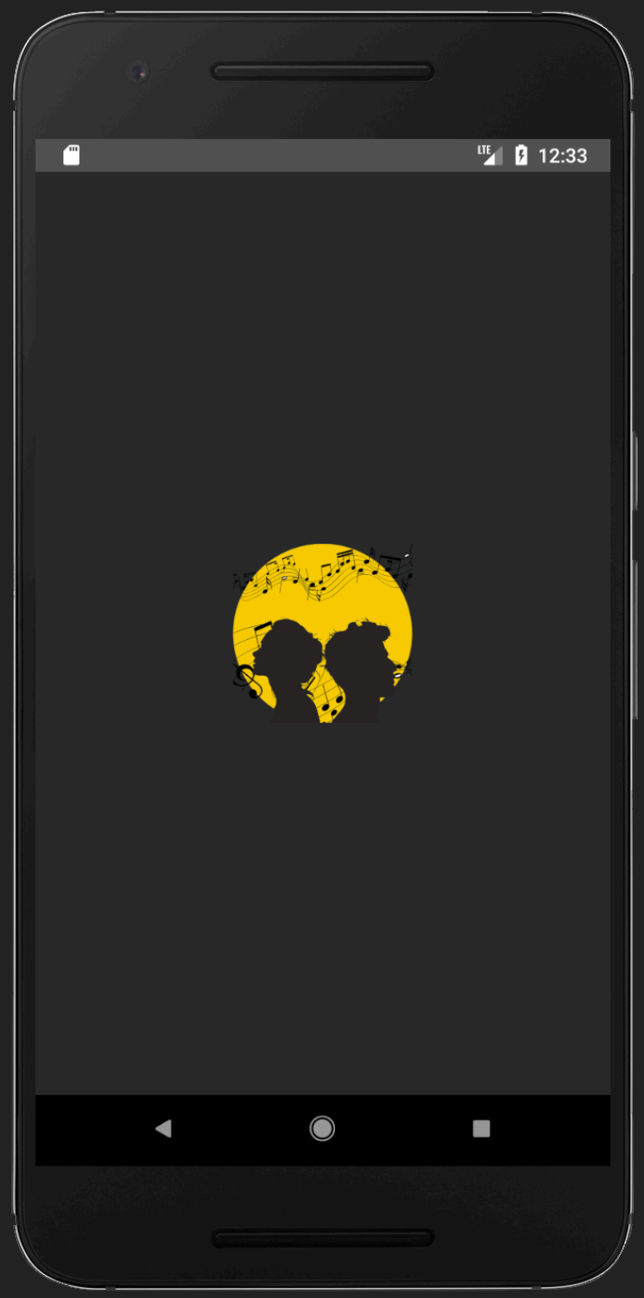
Check connectivity of device to the network

Timeline

Login

```
private Runnable endSplash = () → {

        if (!isFinishing()) {

            handler.removeCallbacks( r: this);

            boolean isNetworkAvailable = NetworkStatus.hasNetworkConnection( context: SplashActivity.this);
            Intent intent;
            if (isNetworkAvailable) {
                intent = new Intent( packageContext: SplashActivity.this, MainActivity.class);
            } else {
                intent = new Intent( packageContext: SplashActivity.this, TimelineActivity.class);
            }

            startActivity(intent);
            finish();
        }
};
```

SPLASH ACTIVITY



OFFLINE CHALLENGE

Timeline

Based on:

- **Network Status**
- **Availability of Web-Server**

Fetches Data from relevant DB

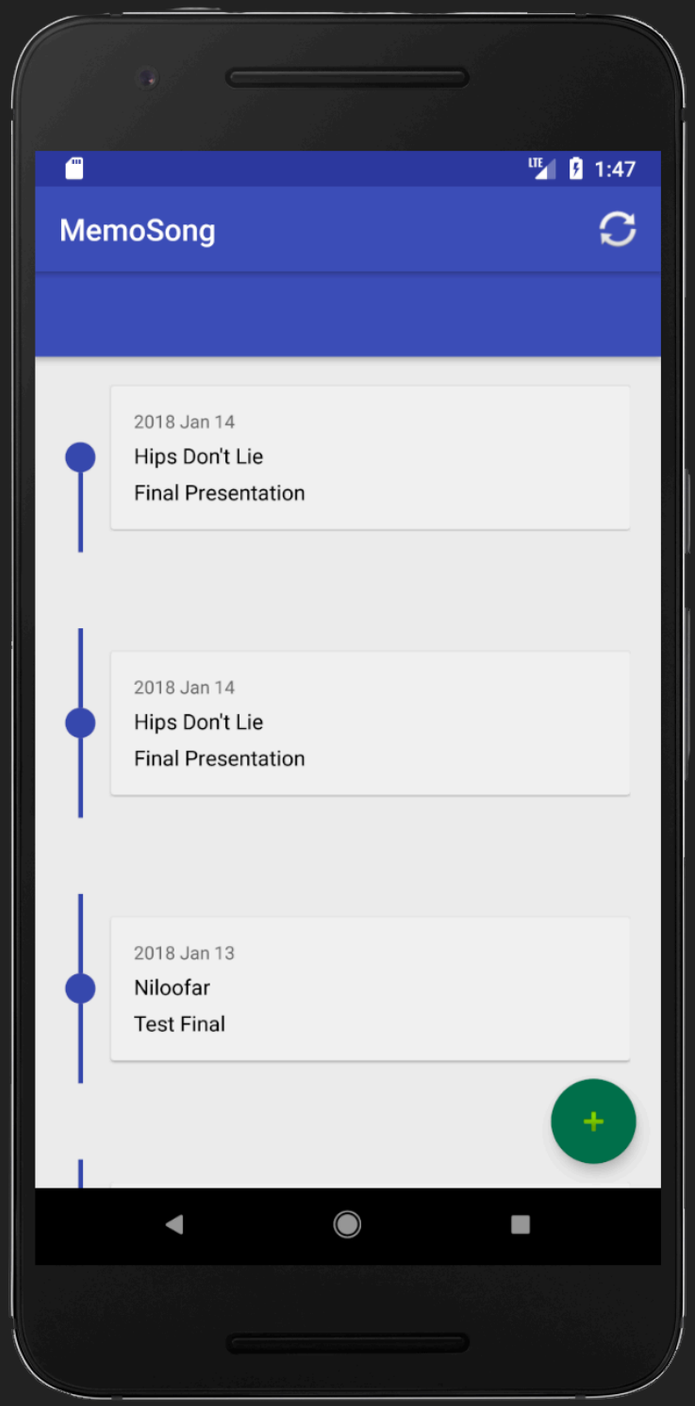
```
JsonObjectRequest requestJson = new JsonObjectRequest(input[1], new JSONObject(params),
        (response) → {
                try {

                    JSONArray array = response.getJSONArray( name: "Memories");
                    Log.i( tag: "onResponse",  msg: "");

                    for (int i = 0; i < array.length(); i++) {
                        JSONObject o = array.getJSONObject(i);
                        TimeLineModel mdata = new TimeLineModel(

                                o.getString( name: "Date"),
                                o.getString( name: "Song"),
                                o.getString( name: "Memory"),
                                o.getString( name: "Friend"),
                                o.getString( name: "Location"),
                                o.getString( name: "Uri"),
                                o.getString( name: "Url")

                        );
                        mDataList.add(mdata);
                    }

                    updateUI();
                } catch (JSONException e) {
                    fetchFromLocalDatabase();
                }
            },
            (error) → { fetchFromLocalDatabase(); });

    RequestQueue requestQueue = Volley.newRequestQueue(mContext);
    requestQueue.add(requestJson);
}
```

Connection Error

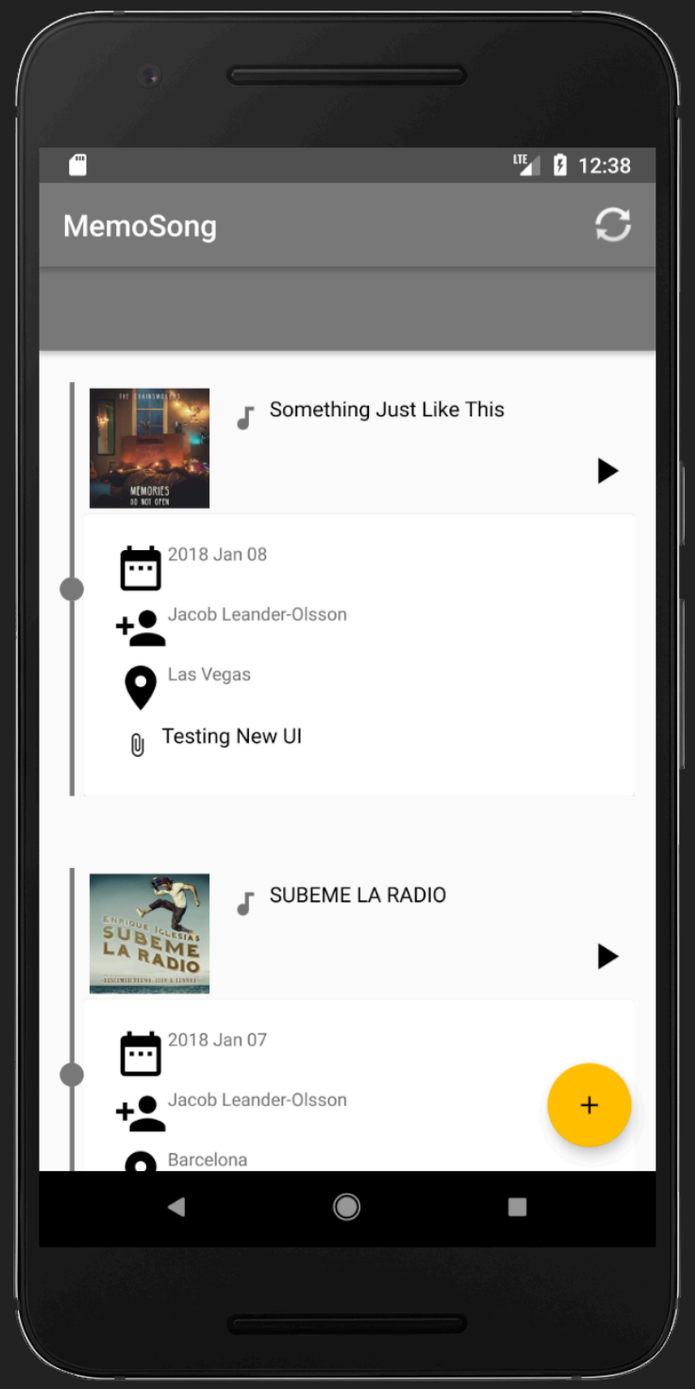
Server Unavailable

```
    private void setDataListItems() {
        if (NetworkStatus.hasNetworkConnection( context: this)) {
            FetchFromRemoteDatabase task = new FetchFromRemoteDatabase( context: this,email, URL_DATA,mTimeLineAdapter);
            task.fetch(email, URL_DATA);
        } else {
            FetchFromLocalDatabase task = new FetchFromLocalDatabase( context: this,mTimeLineAdapter);
            task.execute(email, URL_DATA);
        }
    }
}
```

TIMELINE ACTIVITY



BEFORE



AFTER

OFFLINE CHALLENGE

- What if while storing a new memory network is gone?

```
SendMemory( url: "http://10.0.2.2:3000",parseJSONStringToJSONObject(LoadCurrentData(memory)),memory);

List<MemoryEntity> tobeuploaded = FetchNotUploadedData.Fetch(memo[6]);
for(int i=0;i<tobeuploaded.size();i++) {
    Log.i( tag: "tobeuploaded : ", msg: "" +tobeuploaded.get(i).getMemory());
    SendMemory( url: "http://10.0.2.2:3000", parseJSONStringToJSONObject(LoadCurrentData(tobeuploaded.get(i))),
            tobeuploaded.get(i));
}
```

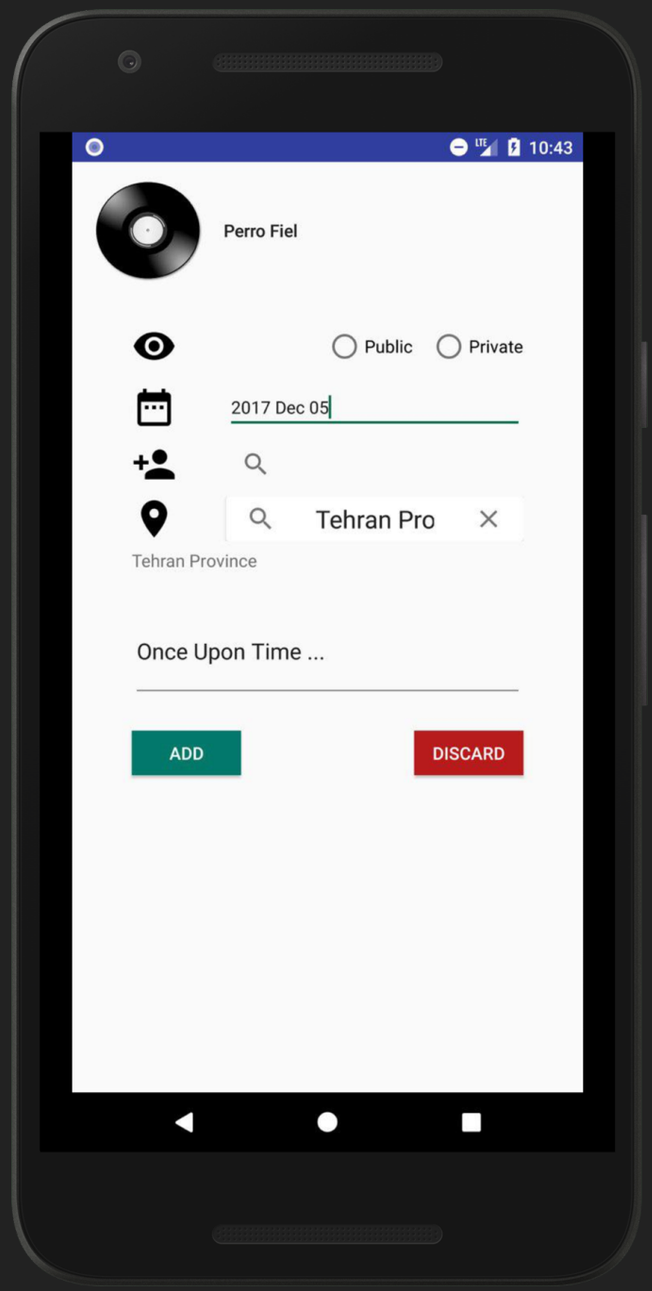
In the best case we only should upload the latest memory that user just added

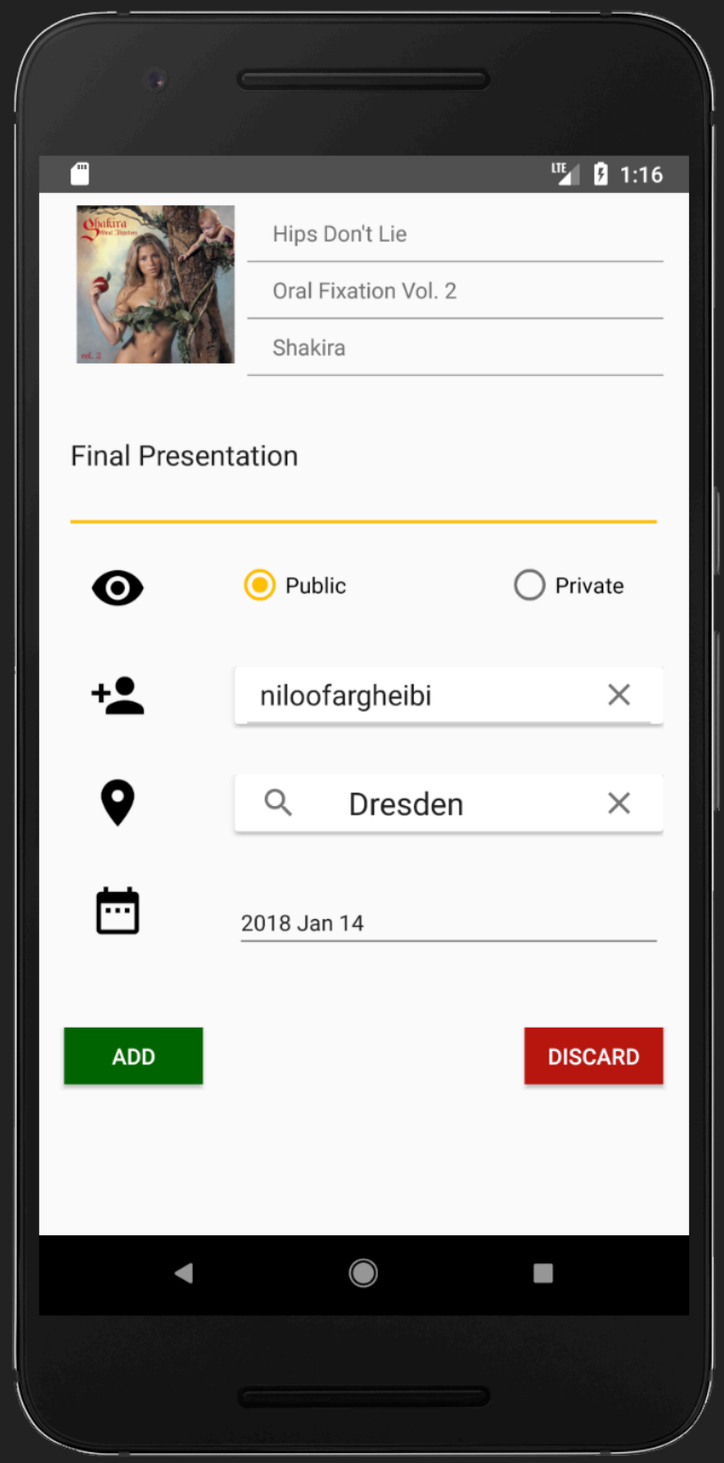
There are some memories that are not synced with local database and remote database

POSSIBLE NETWORK FAILURES

```
public static JSONObject SendMemory(String url, JSONObject json,MemoryEntity memory) {
    JSONObject jsonObjectResp = null;

    try {

        MediaType JSON = MediaType.parse("application/json; charset=utf-8");
        OkHttpClient client = new OkHttpClient();

        okhttp3.RequestBody body = RequestBody.create(JSON, json.toString());
        okhttp3.Request request = new okhttp3.Request.Builder()
                .url(url)
                .put(body)
                .build();

        okhttp3.Response response = client.newCall(request).execute();
        String networkResp = response.body().string();

        if (!networkResp.isEmpty()) {
            Log.i( tag: "SendingMemory", msg: "Done"+networkResp);
            // Save to local database -> no need to be uploaded
            memory.setUploaded("1");
            App.get().getDB().MemoryDao().insertAll(memory);
            App.get().setForceUpdate(false);
        }else{
            // Save to local database -> need to be uploaded
            Log.i( tag: "SendingMemory", msg: "Error Happened"+networkResp);
            memory.setUploaded("0");
            App.get().getDB().MemoryDao().insertAll(memory);
            App.get().setForceUpdate(false);
        }
    } catch (Exception ex) {
        // Connection Failure to the Server
        Log.i( tag: "SendingMemory", msg: "Server Failure"+ex.getMessage());
        // Save to local database -> need to be uploaded
            memory.setUploaded("0");
            App.get().getDB().MemoryDao().insertAll(memory);
            App.get().setForceUpdate(false);
    }
```

MEMORY ACTIVITY





BEFORE

AFTER

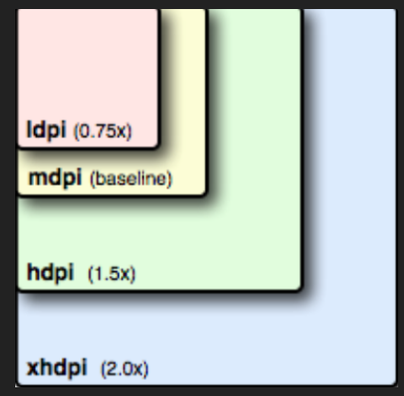
COMMUNICATION ADAPTATION

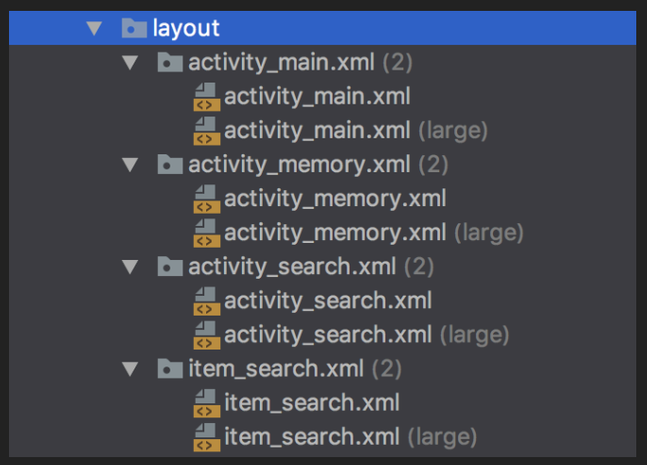
- Queuing Requests which suppose to be sent to the REST server
- Library: com.birbit:android-priority-jobqueue

```
public static void StoreInWebServerDatabaseTask(String...memo) {

    App.getInstance().getJobManager().addJobInBackground(new StoreWeb(memo));
}
```

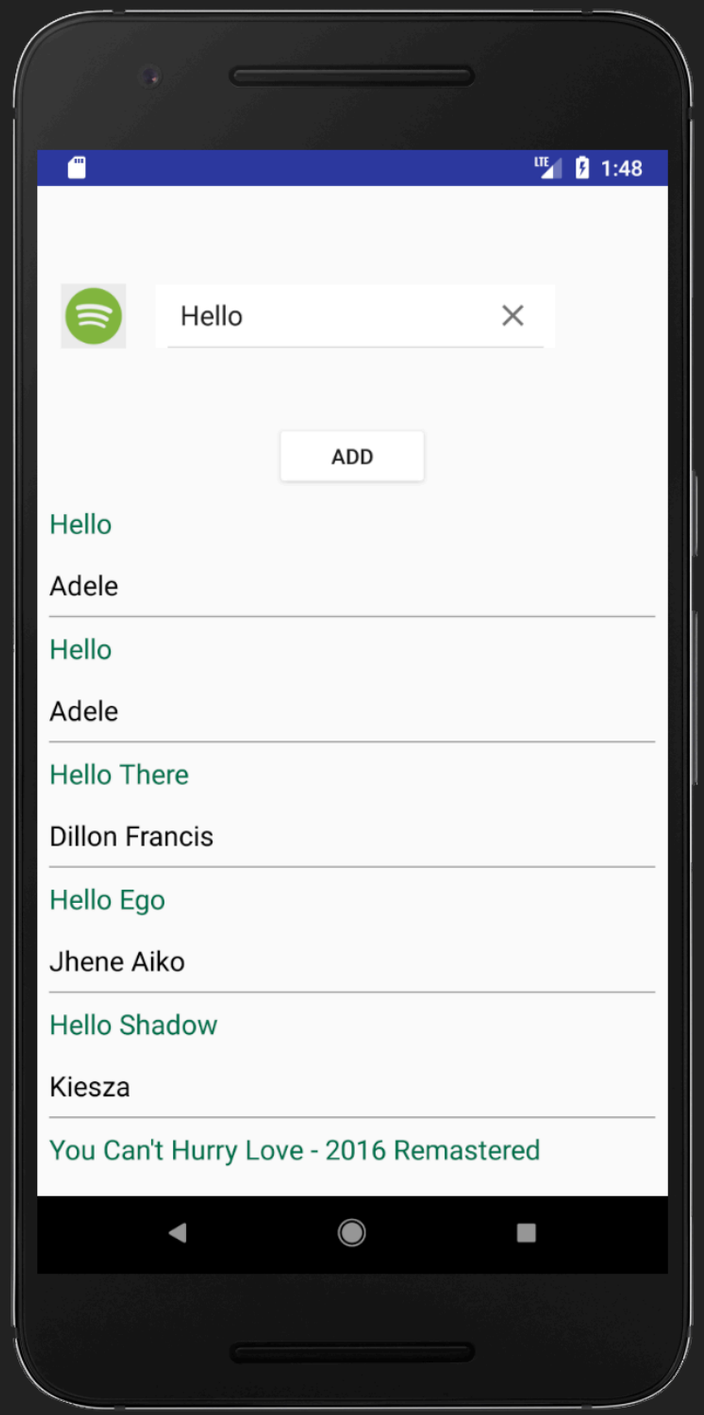
USABILITY CHALLENGE ▸ Supporting Different Screen Sizes

- Using **RelativeLayout** instead of AbsoluteLayout
- Generating density-specific **Resources** (mipmap-drawable)
- Avoiding **hard-coded sizes**
- Using **Size Qualifiers**
- ➡ Creating Different layouts for large screens

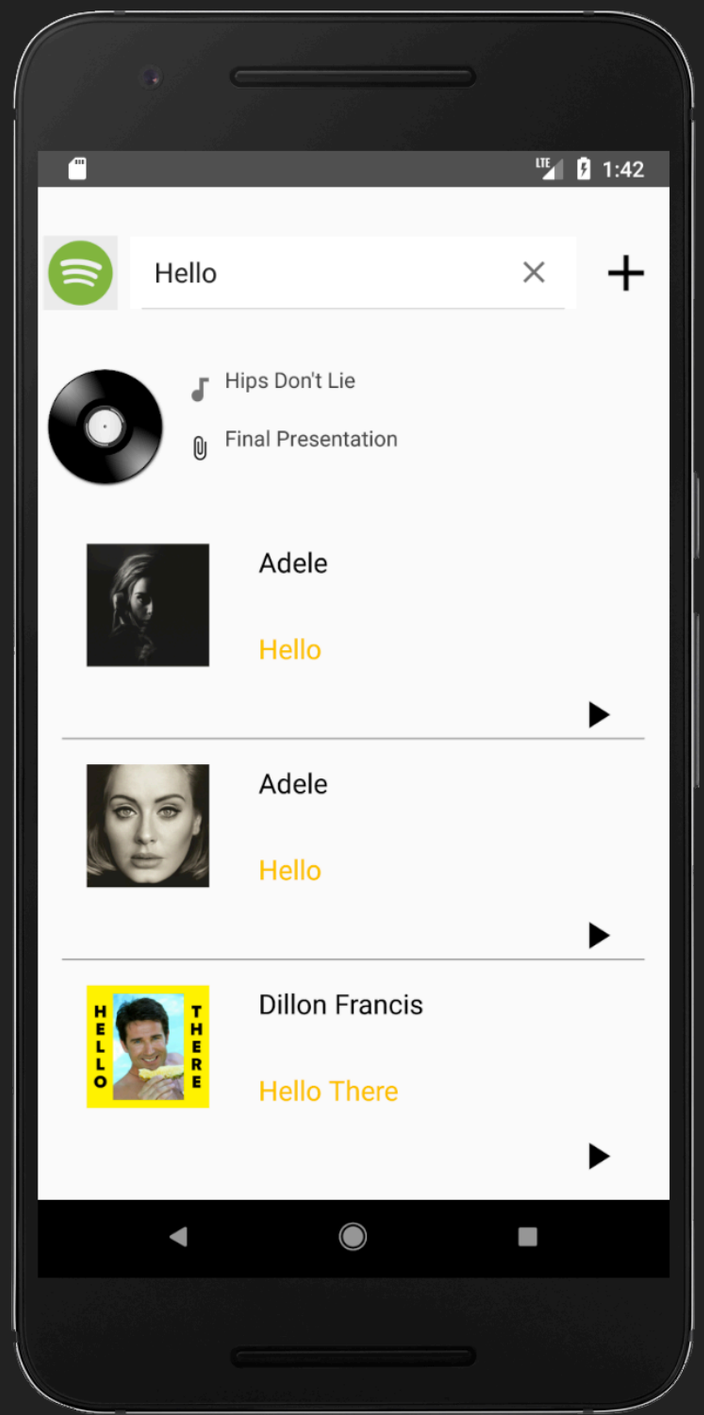




USER INTERFACE ENHANCEMENT

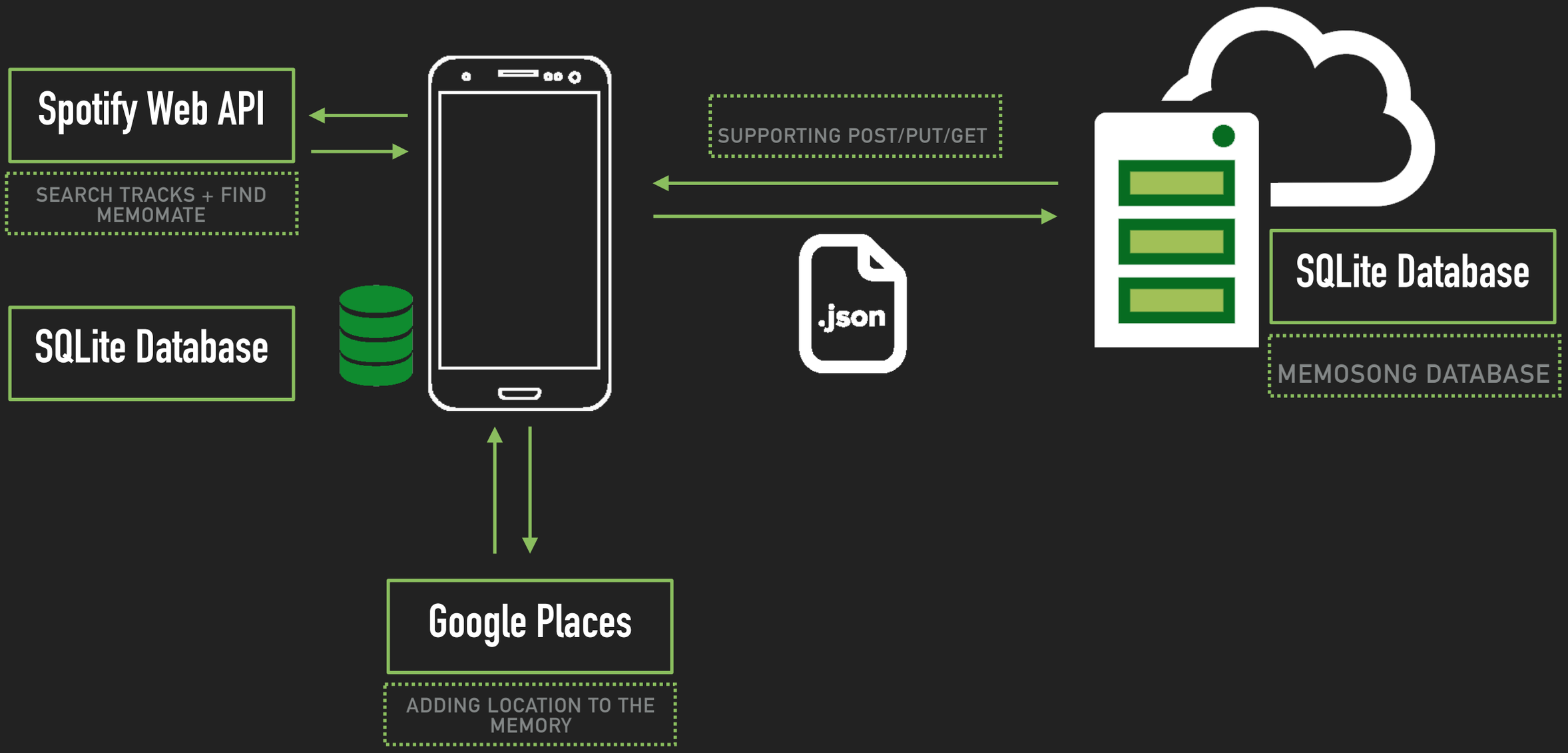


BEFORE



AFTER

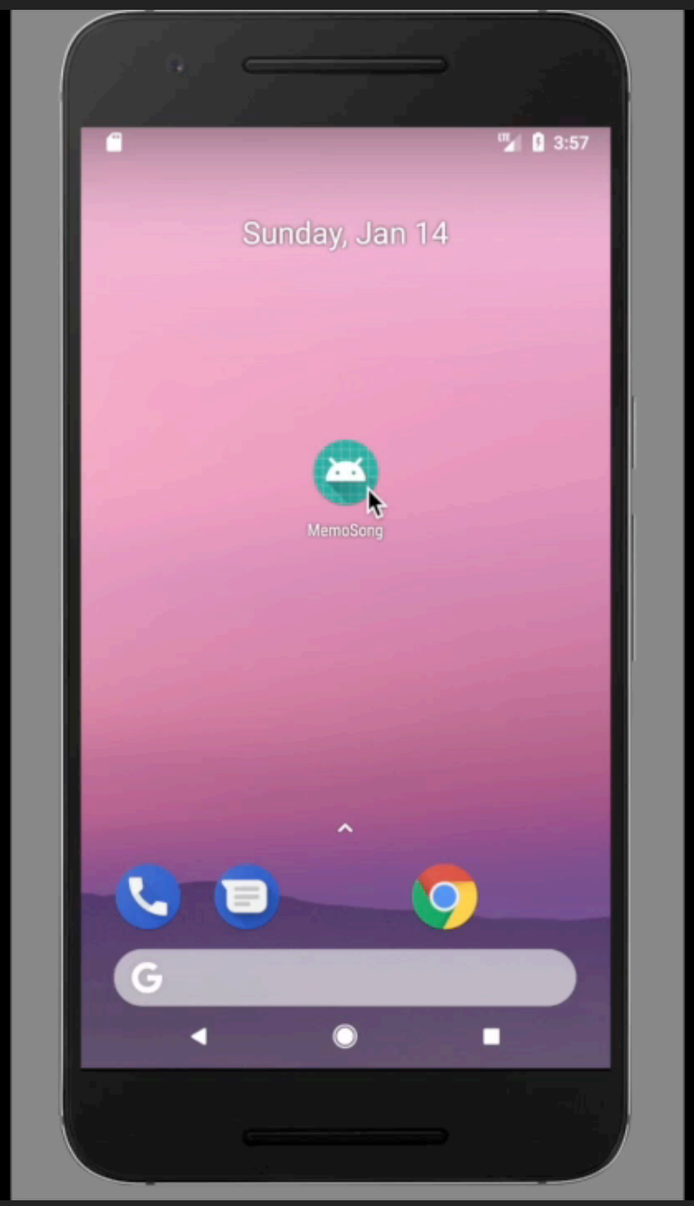
ARCHITECTURE



LESSONS LEARNED

- Database access should be done asynchronous to do not block UI
- Network communication is already done asynchronous and adding as a async Task does not make sense
- JobManager helps to handle networking failures automatically
- Using **Singleton Pattern** for heavy weight instances like Database
- Using OnActivityResult for updating memories in the timeline
- Wrapping Components in linear layout could make layouts cleaner

DEMO



QUESTIONS?