

# Work Samples

## Summary: Glo – Android Studio Mobile App

Android Studio application that tracks a user's emotional state history by allowing them to create moods with 5 different states, specify the reason and social situation, upload a jpeg image that is attached to the mood, specify the mood instance location on a map, and shared their most recent mood instance with their friends.

Github Link:

<https://github.com/ApluUalberta/GroupProject1/tree/master/Code/app/src/main/java/com/example/myapplication>

Code that pulls user's data from a Firestore database:

```

57     protected void onCreate(Bundle savedInstanceState) {
58         super.onCreate(savedInstanceState);
59         setContentView(R.layout.activity_mood_history);
60         historyActivity = this;
61         filterPressed = getResources().getColor(android.R.color.darker_gray);
62         db = FirebaseFirestore.getInstance();
63         users = db.collection("Users");
64         users.addSnapshotListener(new EventListener<QuerySnapshot>() {
65             @Override
66             public void onEvent(@Nullable QuerySnapshot queryDocumentSnapshots, @Nullable FirebaseFirestoreException e) {
67                 Log.d(TAG, "Something changed");
68                 if (queryDocumentSnapshots != null) {
69                     for (DocumentChange doc : queryDocumentSnapshots.getDocumentChanges()) {
70                         if (doc.getDocument().get("Username").equals(user.getUserName())) {
71                             user = doc.getDocument().get("Participant", Participant.class);
72                             user.setUID(doc.getDocument().getId());
73                             moodArrayList = user.getMoodHistory();
74                             moodArrayAdapter.clear();
75                             moodArrayAdapter.addAll(user.getMoodHistory());
76                         }
77                     }
78                 }
79             }
80         });
81         Intent intent = getIntent();
82         user = (Participant) intent.getSerializableExtra("User");
83         moodArrayList = user.getMoodHistory();
84         moodArrayAdapter = new CustomList(this, moodArrayList, user, null);
85
86         moodHistory = findViewById(R.id.mood_history);
87         moodHistory.setAdapter(moodArrayAdapter);
88         filterScroll = findViewById(R.id.FilterScroll);
89
90         greatFilter = findViewById(R.id.GreatFilterButton);
91         goodFilter = findViewById(R.id.GoodFilterButton);
92         neutralFilter = findViewById(R.id.NeutralFilterButton);
93         badFilter = findViewById(R.id.BadFilterButton);
94         worstFilter = findViewById(R.id.WorstFilterButton);
95         buttonBackground = worstFilter.getBackground();
96
97     }

```

The code that updates the user data in the Firestore database after mood history edit, delete, adding, etc:

```

200         moodHistory.setAdapter(moodArrayAdapter);
201     }
202
203     @Override
204     protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
205         super.onActivityResult(requestCode, resultCode, data);
206         updateFilterColors();
207         resetFilter();
208         filterScroll.setVisibility(View.GONE);
209         if (requestCode == 1) {
210             if (resultCode == RESULT_OK) {
211                 Log.d(TAG, "Return from add");
212                 moodArrayList = (ArrayList<Participant>) data.getSerializableExtra("addmood");
213                 Log.d("MyTag", "mood image: " + moodArrayList.get(0).getPicture());
214                 user.addMoodHistory(moodArrayList);
215                 final HashMap<String, Object> userData = new HashMap<>();
216                 userData.put("Participant", user);
217                 users.update(userData, user.getUserName(), user.getUserName());
218                 get();
219                 .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
220                     @Override
221                     public void onComplete(@NonNull Task<QuerySnapshot> task) {
222                         if (task.isSuccessful()) {
223                             QuerySnapshot queryDocumentSnapshots = task.getResult();
224                             Participant updateUser = queryDocumentSnapshots.getDocuments().get(0).get("Participant", Participant.class);
225                             Log.d(TAG, "Updating user: " + updateUser.getUserName());
226                             users.document(queryDocumentSnapshots.getDocuments().get(0).getId())
227                                 .update(updateUser);
228                             .addOnFailureListener(new OnFailureListener() {
229                                 @Override
230                                 public void onFailure(@NonNull Exception e) {
231                                     Log.d(TAG, "Update failed: " + e);
232                                 }
233                             });
234                             .addOnSuccessListener(new OnSuccessListener<Void>() {
235                                 @Override
236                                 public void onSuccess(Void void) {
237                                     Log.d(TAG, "Updated user SuccessFully");
238                                 }
239                             });
240                         }
241                     }
242                 });
243             }
244         }
245     }
246 }

```

Allen Lu

Code that allows the user to edit, add, and delete the google map location that a mood instance was

```
36 @Override
37 protected void onCreate(Bundle savedInstanceState) {
38     super.onCreate(savedInstanceState);
39     setContentView(R.layout.activity_view_map);
40     Intent intent = getIntent();
41     LatLng = new LatLng(intent.getDoubleExtra("Lat",0),intent.getDoubleExtra("Long",0));
42
43     SupportMapFragment mapFrag = (SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.Viewmap);
44     mapFrag.getMapAsync(this);
45 }
46
47
48 @Override
49 public void onMapReady(GoogleMap googleMap) {
50     getLocationPermission();
51     if (mLocationPermissionGranted) {
52         googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(LatLng, 16.0f));
53
54         googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
55         googleMap.getUiSettings().setMapToolbarEnabled(false);
56         googleMap.getUiSettings().setZoomControlsEnabled(true);
57         googleMap.addMarker(new MarkerOptions().position(LatLng)
58             .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE)));
59     }
60 }
61
62
63 private void getLocationPermission() {
64     if (ContextCompat.checkSelfPermission(this.getApplicationContext(), Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
65         mLocationPermissionGranted = true;
66     } else {
67         ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
68     }
69 }
70
71 @Override
72 public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
73     mLocationPermissionGranted = false;
74     if (requestCode == PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION) {
75         if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
76             mLocationPermissionGranted = true;
77         } else {
78             Toast.makeText(this, "App must have permissions for your location if you want to use the map", Toast.LENGTH_LONG).show();
79             setResult(RESULT_CANCELED);
80             finish();
81         }
82     }
83 }
```

in:

## Summary: Crime Statistics – Embedded SQLite3 Database Program

A command-line interface that allows a user to generate bar/pie charts and maps of Edmonton's crime statistics using embedded SQLite3 queries. Users execute queries that are translated to plain English in order to generate common database information graphics using third-party libraries (Pandas for executing queries, Folium to map query data, and Matplotlib to graph query data).

Github Link: [https://github.com/ApluUalberta/Crime-Statistics-Database-Program/blob/master/src/a4\\_specific\\_utils.py](https://github.com/ApluUalberta/Crime-Statistics-Database-Program/blob/master/src/a4_specific_utils.py)

Code that creates new map data upon user activating query that creates map data:

```

65 def create_new_edmonton_map():
66     """
67     Creates a new map object centered on Edmonton
68     """
69     return folium.Map(location=_FOLIUM_EDMONTON_MAP_COORDS, zoom_start=12)
70
71
72 def add_markers_to_map(map, markers, avg_val):
73     """ Plots the given markers on a map
74
75     :param map: The map to add markers to
76     :markers: A list of FolioMarkers that describe info for each maker to place on the map
77     :avg_val: The avg value to use for the markers passed in. Any marker vals less/greater than the average will scaled.
78     """
79
80     for marker in markers:
81         calc_radius = (marker.val / avg_val) * _FOLIUM_AVG_RAD_SIZE + _FOLIUM_ADDITIONAL_RAD_SIZE
82
83         folium.Circle(
84             location=marker.coords,
85             popup=marker.str,
86             radius=calc_radius,
87             color=marker.colour,
88             fill=True,
89             fill_color=marker.colour
90         ).add_to(map)
91
92
93 def write_map_to_file(map, q_num):
94     """
95     Writes the given map to file named appropriately by the question number
96     """
97     map_file_name = generate_filename_for_question_file(q_num, "html")
98     map.save(map_file_name)
99
100     print("Wrote \"{}\" to disk.".format(map_file_name))
101
102
103 def get_num_neighborhoods():
104     return _NUM_NEIGHBORHOODS
105
106
107 """

```

Code that uses embedded SQLite3 query to generate map data for the top neighbourhoods for a specific crime:

```

126 def menu_map_of_top_neighborhoods_for_a_given_crime():
127     #print("TODO: map_of_top_neighborhoods_for_a_given_crime")
128
129     connection = sqlite3.connect(_DATABASE_PATH)
130     cur = connection.cursor()
131
132     lower_limit = utils.input_int_and_validate_with_predicate("Enter start year: ", check_if_int_is_year_format)
133     upper_limit = utils.input_int_and_validate_with_predicate("Enter end year: ", check_if_int_is_year_format)
134     crime_type = input("Enter crime type: ")
135     num_neighborhood = utils.input_int_and_validate_with_predicate("Enter number of neighborhoods: ", check_if_int_is_non_negative_and_handl
136
137     cur.execute("SELECT sum(i.Incidents_Count) as counts, i.Neighbourhood_Name, c.Latitude, c.Longitude \
138 FROM coordinates c \
139 LEFT JOIN crime_incidents i on c.Neighbourhood_Name = i.Neighbourhood_Name \
140 WHERE i.Year >= ? AND \
141         i.Year <= ? AND \
142         i.Crime_Type = ? \
143         GROUP BY i.Neighbourhood_Name \
144         ORDER BY counts DESC \
145         LIMIT ? ", (str(lower_limit), str(upper_limit), crime_type, str(num_neighborhood)) )
146
147     nList = cur.fetchall()
148     bot_of_top = nList[num_neighborhood-1][0]
149     #print(nList)
150
151
152     cur.execute("SELECT sum(i.Incidents_Count) as counts, i.Neighbourhood_Name, c.Latitude, c.Longitude \
153 FROM coordinates c \
154 LEFT JOIN crime_incidents i on c.Neighbourhood_Name = i.Neighbourhood_Name \
155 WHERE i.Year >= ? AND \
156         i.Year <= ? AND \
157         i.Crime_Type = ? \
158         GROUP BY i.Neighbourhood_Name \
159         HAVING counts >= ? \
160         ORDER BY counts DESC", (str(lower_limit), str(upper_limit), crime_type, bot_of_top) )
161
162     newList = cur.fetchall()
163     markers = []
164     avg_val = 0
165
166     edmonton_map = a4_specific_utils.create_new_edmonton_map()
167
168     for n in newList:
169         nPopup = "%s <br> %s" % (n[1], n[0])
170         markers.append(FolioMarker([n[2], n[3]], nPopup, 'crimson', n[0]))
171         avg_val += n[0]
172
173     avg_val /= len(newList)
174     a4_specific_utils.add_markers_to_map(edmonton_map, markers, avg_val)
175     a4_specific_utils.write_map_to_file(edmonton_map, "Q3")

```