

UNIT 7

Advanced Android Concepts

(10 Hours) — Mobile Programming (CAC351)

7.1 Local Database with SQLite

SQLite is a lightweight, serverless, file-based relational database built into Android. Each app has a private database stored in `/data/data/<package>/databases/`. It supports standard SQL, needs no configuration and is suited to structured offline data (notes, contacts, tasks).

Establishing Connection: SQLiteOpenHelper

Android does not need a username/password connection. Instead, extend **SQLiteOpenHelper**: it creates the database on first use (*onCreate*), upgrades it when the version number changes (*onUpgrade*), and returns a connection through *getWritableDatabase()* or *getReadableDatabase()*.

Creating Database and Tables

```
public class DBHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "school.db";
    private static final int DB_VERSION = 1;
    public static final String TABLE = "student";

    public DBHelper(Context ctx) { super(ctx, DB_NAME, null, DB_VERSION); }

    @Override public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + TABLE + " (" +
            "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "name TEXT NOT NULL, " +
            "age INTEGER)");
    }
    @Override public void onUpgrade(SQLiteDatabase db, int oldV, int newV) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE);
        onCreate(db);
    }
}
```

Listing 7.1: DBHelper – creating the database and table

Data Manipulation (CRUD)

Operation	SQL	Android method
Create	INSERT	db.insert(table, null, ContentValues)
Read	SELECT	db.query(...) or db.rawQuery(sql, args) → Cursor
Update	UPDATE	db.update(table, values, whereClause, whereArgs)

Operation	SQL	Android method
Delete	DELETE	db.delete(table, whereClause, whereArgs)

```
DBHelper helper = new DBHelper(this);

// INSERT
SQLiteDatabase db = helper.getWritableDatabase();
ContentValues cv = new ContentValues();
cv.put("name", "Asha"); cv.put("age", 20);
long rowId = db.insert(DBHelper.TABLE, null, cv); // -1 means failure

// SELECT
Cursor cur = db.rawQuery("SELECT id, name, age FROM student WHERE age > ?",
    new String[]{"18"});
while (cur.moveToNext()) {
    int id = cur.getInt(cur.getColumnIndexOrThrow("id"));
    String name = cur.getString(cur.getColumnIndexOrThrow("name"));
    Log.d("DB", id + " " + name);
}
cur.close();

// UPDATE
ContentValues up = new ContentValues(); up.put("age", 21);
db.update(DBHelper.TABLE, up, "name = ?", new String[]{"Asha"});

// DELETE
db.delete(DBHelper.TABLE, "id = ?", new String[]{"1"});
db.close();
```

Listing 7.2: Insert, query, update and delete

*Note: Always use ? placeholders with whereArgs (parameterised queries) to prevent SQL injection. Close Cursors and databases. Google recommends the **Room** library (an abstraction layer over SQLite with annotations @Entity, @Dao, @Database) for new projects.*

7.2 Introduction to API

An **API (Application Programming Interface)** is a set of rules that lets one software component talk to another. In mobile apps, a **web API** lets the app send requests to a remote server and receive data (e.g., weather, login, products) over HTTP(S).

API Types

Type	Description
REST	Resource-based over HTTP using GET, POST, PUT, DELETE; returns JSON/XML. Most common
SOAP	XML-based protocol with strict contracts (WSDL); heavier, used in enterprise
GraphQL	Client specifies exactly the fields needed in a single endpoint
gRPC / WebSocket	High-speed binary RPC / real-time two-way communication (chat)
Local (library) API	Android SDK APIs itself, e.g., Camera, Location

Type	Description
Public / Private / Partner	Classification by who may access the API

HTTP method	Purpose	Example
GET	Read data	GET /students/5
POST	Create data	POST /students
PUT / PATCH	Update data	PUT /students/5
DELETE	Remove data	DELETE /students/5

Common status codes: **200** OK, **201** Created, **400** Bad request, **401** Unauthorized, **404** Not found, **500** Server error.

7.3 Introduction to JSON

JSON (JavaScript Object Notation) is a lightweight text format for exchanging data. Objects use `{ }` with key-value pairs; arrays use `[ ]`. Keys are strings; values may be string, number, boolean, null, object or array.

```
{
  "status": "ok",
  "student": { "id": 1, "name": "Asha", "passed": true },
  "subjects": ["Java", "Android", "DBMS"]
}
```

Listing 7.3: A JSON document

Android provides *org.json* (JSONObject, JSONArray) and libraries such as **Gson** or **Moshi** for converting JSON to Java objects.

```
JSONObject root = new JSONObject(jsonString);
String status = root.getString("status");
JSONObject st = root.getJSONObject("student");
String name = st.getString("name");
JSONArray subs = root.getJSONArray("subjects");
for (int i = 0; i < subs.length(); i++) Log.d("JSON", subs.getString(i));

// Gson
Student s = new Gson().fromJson(studentJson, Student.class);
String out = new Gson().toJson(s);
```

Listing 7.4: Parsing JSON with org.json and Gson

7.4 Retrieving Contents from a Remote Server

Add internet permission in the manifest: `<uses-permission android:name="android.permission.INTERNET" />`. Network calls must **never** run on the main (UI) thread – Android throws *NetworkOnMainThreadException*. Use background threads (Executor), or libraries that handle threading: **Retrofit**, **Volley**, **OkHttp**. (*AsyncTask* is deprecated.)

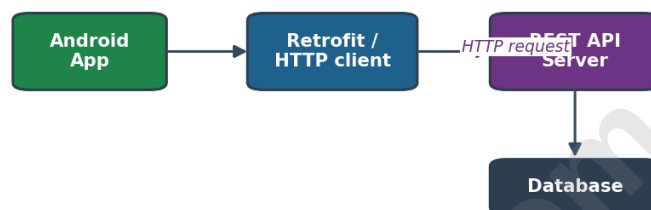


Figure 7.1: App-to-server communication using a REST API

```
// Gradle: implementation 'com.squareup.retrofit2:retrofit:2.9.0'
//         implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

public interface ApiService {
    @GET("students")      Call<List<Student>> getStudents();
    @GET("students/{id}") Call<Student>   getStudent(@Path("id") int id);
}

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.example.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();
ApiService api = retrofit.create(ApiService.class);

api.getStudents().enqueue(new Callback<List<Student>>() { // runs in background
    @Override public void onResponse(Call<List<Student>> c, Response<List<Student>> r) {
        if (r.isSuccessful()) adapter.setData(r.body()); // update RecyclerView
    }
    @Override public void onFailure(Call<List<Student>> c, Throwable t) {
        Toast.makeText(MainActivity.this, t.getMessage(), Toast.LENGTH_SHORT).show();
    }
});
```

Listing 7.5: GET request with Retrofit

7.5 Sending Contents to a Remote Server

```
// POST a new student as JSON
@POST("students")
Call<Student> addStudent(@Body Student student);

api.addStudent(new Student("Rita", 22)).enqueue(new Callback<Student>() {
    @Override public void onResponse(Call<Student> c, Response<Student> r) {
        if (r.code() == 201) Toast.makeText(ctx, "Saved", Toast.LENGTH_SHORT).show();
    }
    @Override public void onFailure(Call<Student> c, Throwable t) { /* show error */ }
});

// Form data:
@FormUrlEncoded
@POST("login")
Call<LoginResult> login(@Field("email") String e, @Field("password") String p);
```

Listing 7.6: POST request with Retrofit

Note: Use HTTPS only (cleartext HTTP is blocked by default since Android 9). Never hard-code secrets or API keys in the app source; validate input and handle errors and timeouts.

Example 1 — Choosing storage

A notes app must work offline and later sync with a server. Which technologies?

Solution

Store notes locally in SQLite/Room for offline use; use Retrofit to POST unsynced notes and GET updates when the network returns. JSON is the exchange format.

7.6 Implementing Google Maps in an Android Application

- **Prerequisites:** Android Studio with Google Play services; a Google account.
- **Create project** using the *Google Maps Views Activity* template (or add the dependency manually: *com.google.android.gms:play-services-maps*).
- **Get an API key:** Google Cloud Console → create project → enable *Maps SDK for Android* → Credentials → Create API key; restrict it to your package name and SHA-1 fingerprint.
- **Add the key** and permissions in *AndroidManifest.xml*.
- **Add a map fragment** to the layout and implement *OnMapReadyCallback*.

```

<!-- AndroidManifest.xml -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<application ...>
    <meta-data android:name="com.google.android.geo.API_KEY"
        android:value="YOUR_API_KEY" />
</application>

<!-- activity_maps.xml -->
<fragment android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent" android:layout_height="match_parent" />

```

Listing 7.7: Manifest and layout for Google Maps

```

public class MapsActivity extends AppCompatActivity implements OnMapReadyCallback {
    @Override protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_maps);
        ((SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map)).getMapAsync(this);
    }
    @Override public void onMapReady(GoogleMap map) {
        LatLng ktm = new LatLng(27.7172, 85.3240); // Kathmandu
        map.addMarker(new MarkerOptions().position(ktm).title("Kathmandu"));
        map.moveCamera(CameraUpdateFactory.newLatLngZoom(ktm, 12f));
        map.getUiSettings().setZoomControlsEnabled(true);
    }
}

```

Listing 7.8: Showing a marker on the map

*Note: Location permissions are **runtime permissions**; request them with the Activity Result API (`RequestPermission`) before enabling `map.setMyLocationEnabled(true)`.*

7.7 Publishing an Application on Google Play Store

- **Prepare the app:** test thoroughly, remove debug code and logs, set versionCode/versionName, choose a unique applicationId, set the target API level required by Google Play, add icon and screenshots.
- **Enable code shrinking** (R8/ProGuard, `minifyEnabled true`) to reduce size and obfuscate.
- **Generate a signed bundle:** Build → Generate Signed Bundle/APK → create or choose a keystore (.jks) and key alias → choose **Android App Bundle (.aab)** → release build. Keep the keystore and passwords safe; losing them prevents updates. Play App Signing can manage the final key.
- **Create a Google Play Developer account** (one-time registration fee of US\$25, identity verification).
- **Create the app** in Play Console: name, language, free/paid, declarations.
- **Store listing:** short and full description, icon (512×512), feature graphic (1024×500), screenshots, category, contact details, privacy policy URL.
- **Content rating, Data safety form, target audience** questionnaires.

- **Upload the AAB** to a testing track (internal/closed/open) or Production, add release notes.
- **Review and rollout:** Google reviews the app (hours to days); after approval it is published and can be rolled out by percentage. Monitor crashes, ratings and reviews and release updates by increasing versionCode.

Note: New personal developer accounts may have to complete extra closed-testing requirements before production release; check the current Play Console policy.

Key Takeaways

- ✓ SQLite is Android's built-in database: SQLiteOpenHelper handles creation/upgrade; CRUD via insert/query/update/delete with ContentValues and Cursor; Room is the modern wrapper.
- ✓ APIs (REST, SOAP, GraphQL) allow apps to exchange data with servers; JSON is the standard format (JSONObject, JSONArray, Gson).
- ✓ Network calls need INTERNET permission and must run off the main thread (Retrofit/Volley/OkHttp).
- ✓ Google Maps needs an API key, a SupportMapFragment and OnMapReadyCallback; markers use LatLng.
- ✓ Publishing: sign an AAB, create a Play Console account, complete the store listing and policies, then release.

— *End of Unit 7* —