**Section 35 – Firebase: Introduction to Realtime Database**

**1. Key Concepts**

1. **Firebase Overview**
   * Google’s mobile & web application development platform.
   * Provides **backend services** without needing your own server.
   * Designed for:
     + Fast development
     + Scalability
     + Easy integration with Google Cloud services
2. **Realtime Database**
   * **NoSQL**, **cloud-hosted** database.
   * Stores data as **JSON**.
   * Provides **real-time synchronization** — changes reflect instantly across all connected clients.
   * Best for **live, collaborative, low-latency** applications.
3. **Other Firebase Services (High-level)**
   * **Cloud Firestore** → More flexible queries, offline support, better scalability.
   * **Authentication** → Email/password, Google, Facebook, Twitter, etc.
   * **Cloud Functions** → Serverless backend logic.
   * **ML Kit** → Pre-built machine learning models.
   * **Hosting** → Static site hosting.
   * **Cloud Storage** → File storage (images, audio, video).
   * **Crashlytics** → Crash reporting.
   * **Performance Monitoring** → App performance analytics.
   * **Test Lab** → Cloud-based device testing.
   * **In-App Messaging, Analytics, Predictions, A/B Testing, Cloud Messaging, Remote Config, Dynamic Links**.

**2. Implementation Steps – Adding Firebase to Android Project**

**Step 1 – Create Firebase Project**

1. Go to [**Firebase Console**](https://firebase.google.com/).
2. **Sign in** with Google account.
3. Click **Add Project** → Enter project name.
4. Enable **Google Analytics** (optional but recommended).
5. Select an Analytics account → Click **Create Project** → Wait until ready.

**Step 2 – Register Android App**

1. Select **Android** platform.
2. Enter:
   * **Package name** (from app/src/main/AndroidManifest.xml)
   * App nickname (optional)
   * **SHA-1 certificate** (optional, needed for Google Sign-In, Phone Auth, Dynamic Links)
3. Click **Register App**.

**Step 3 – Add google-services.json**

1. Download **google-services.json** from Firebase console.
2. Place it inside:
3. app/ <-- app module root
4. **Naming is case-sensitive** — must be exactly google-services.json.

**Step 4 – Add Firebase Dependencies**

**build.gradle (project)**

plugins {

id 'com.android.application'

id 'com.google.gms.google-services' version 'X.X.X' apply false // Plugin for Firebase

}

dependencies {

// Firebase Bill of Materials (BoM) - keeps versions consistent

implementation platform('com.google.firebase:firebase-bom:32.7.0')

}

**build.gradle (app)**

plugins {

id 'com.google.gms.google-services' // Apply Firebase plugin

}

dependencies {

implementation platform('com.google.firebase:firebase-bom:32.7.0') // BoM

implementation 'com.google.firebase:firebase-analytics' // Analytics

}

**BoM** allows you to skip specifying individual Firebase versions.

**3. Enabling & Using Realtime Database**

**Step 1 – Enable in Firebase Console**

1. In Firebase Console → **Build > Realtime Database**.
2. Click **Create Database**.
3. Choose **location** (default).
4. Choose **Test Mode** (open read/write for 30 days, for learning only).
5. Click **Enable**.

**Step 2 – Add Dependency**

**build.gradle (app)**

implementation 'com.google.firebase:firebase-database'

(Skip version if using BoM)

**Step 3 – Database Rules**

Default test mode rules:

{

"rules": {

".read": "true",

".write": "true"

}

}

⚠️ Test mode is insecure — use authentication & fine-grained rules in production.

**4. Writing Data**

**MainActivity.java**

// 1. Get database instance

FirebaseDatabase database = FirebaseDatabase.getInstance();

// 2. Get reference to a node (like a table in SQL)

DatabaseReference myRef = database.getReference("messages");

// 3. Write value to this node

myRef.setValue("Hello from our course!");

**Manifest Permission**

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

✅ This creates:

{

"messages": "Hello from our course!"

}

**5. Reading Data in Real-time**

DatabaseReference myRef = FirebaseDatabase.getInstance().getReference("messages");

myRef.addValueEventListener(new ValueEventListener() {

@Override

public void onDataChange(DataSnapshot snapshot) {

String value = snapshot.getValue(String.class); // Read as String

textView.setText(value); // Update UI

}

@Override

public void onCancelled(DatabaseError error) {

Log.w("Firebase", "Failed to read value.", error.toException());

}

});

* **onDataChange** → Called whenever data changes at the reference.
* **onCancelled** → Called on read error.

**6. Writing & Reading Custom Objects**

**User.java**

public class User {

private String name;

private String email;

// \*\*Empty constructor required for Firebase\*\*

public User() {}

public User(String name, String email) {

this.name = name;

this.email = email;

}

// Getters & setters

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

}

**Writing Object**

DatabaseReference userRef = FirebaseDatabase.getInstance().getReference("users");

User user1 = new User("Jack", "jack@gmail.com");

userRef.setValue(user1);

Result in Firebase:

{

"users": { "name": "Jack", "email": "jack@gmail.com" }

}

**Reading Object**

userRef.addValueEventListener(new ValueEventListener() {

@Override

public void onDataChange(DataSnapshot snapshot) {

User user = snapshot.getValue(User.class); // Deserialize into User object

textView.setText(user.getEmail()); // Display email

}

@Override

public void onCancelled(DatabaseError error) {

Log.w("Firebase", "Error reading user", error.toException());

}

});

Without an empty constructor in User, Firebase will **throw an error**.

**7. Tools, Libraries, APIs Used**

* **Firebase Console** → Project creation, DB setup.
* **Firebase Realtime Database SDK** (com.google.firebase:firebase-database).
* **Firebase BoM** → Version management.
* **ValueEventListener** → Real-time data sync.
* **DataSnapshot** → Data wrapper for reading values.

**8. Best Practices**

* Use **BoM** to avoid version conflicts.
* Use **child()** to structure data hierarchically:
* myRef.child("user1").setValue(user1);
* Avoid test mode in production → use secure **rules**.
* Always keep **empty constructors** in model classes.
* Avoid deep nesting → makes queries & updates harder.
* Use **push()** to generate unique keys instead of overwriting:
* userRef.push().setValue(user1);
* Use **SingleValueEventListener** for one-time reads instead of real-time listening.
* Handle **onCancelled()** for proper error logging.

**Part B – Extra Knowledge Not in Video**

1. **Offline Support**
   * Enable local persistence:
   * FirebaseDatabase.getInstance().setPersistenceEnabled(true);
   * Data is cached & synced when back online.
2. **Querying**
3. myRef.orderByChild("name").equalTo("Jack");
   * You can filter with:
     + orderByChild()
     + orderByKey()
     + startAt(), endAt(), limitToFirst()
4. **Batch Updates**
   * Update multiple nodes atomically:
   * Map<String, Object> updates = new HashMap<>();
   * updates.put("users/user1/name", "Jack Updated");
   * updates.put("users/user1/email", "newjack@gmail.com");
   * database.getReference().updateChildren(updates);
5. **Security Rules for Auth**
6. {
7. "rules": {
8. "users": {
9. "$uid": {
10. ".read": "$uid === auth.uid",
11. ".write": "$uid === auth.uid"
12. }
13. }
14. }
15. }
16. **Difference: Realtime Database vs Firestore**
    * Realtime DB: JSON tree, lower latency, simpler queries.
    * Firestore: Document/collection model, complex queries, better scalability.