**Section 40 – Firebase Cloud Messaging (FCM) App**

**1. Key Concepts Taught**

* **Firebase Cloud Messaging (FCM)**:  
  A free cross-platform service to send push notifications or messages to Android, iOS, and web clients.
* **Use cases**:
  + Notify users about updates (new content, subscriptions, offers).
  + Increase engagement and retention.
  + Deliver targeted messages via **topics** or **direct tokens**.
* **Messaging Types**:
  + **Notification messages** – Displayed by the system tray automatically; easy for basic alerts.
  + **Data messages** – Delivered to the app code for custom handling; can be processed in the background.
* **Topics**:  
  Like subscribing to a "channel" (e.g., weather). Users only get notifications for subscribed topics.
* **Workers in Android**:  
  Background tasks (e.g., WorkManager) used to listen for incoming FCM messages even when the app isn’t active.
* **Notification Channels** (Android 8+):  
  Required for grouping notifications; allows user control over channel settings.
* **FCM Token**:  
  Unique identifier for a device/app instance, used for sending targeted messages.

**2. Step-by-Step Implementation**

**Step 1: Create Firebase Project**

1. Go to [**Firebase Console**](https://console.firebase.google.com/).
2. Create a new project (e.g., FCMApp).
3. Register Android app with:
   * **Package Name** (must match Android project).
4. Download google-services.json.
5. Place it in:
6. app/ (Android view → Project view → app folder)

**Step 2: Add Firebase & FCM Dependencies**

**project-level build.gradle**

buildscript {

repositories {

google() // Google's Maven repository

}

dependencies {

classpath 'com.google.gms:google-services:4.3.13'

}

}

allprojects {

repositories {

google()

}

}

**app-level build.gradle**

plugins {

id 'com.android.application'

id 'com.google.gms.google-services' // Google Services Plugin

}

dependencies {

// Firebase BOM (Bill of Materials) for version alignment

implementation platform('com.google.firebase:firebase-bom:32.1.0')

// Firebase Cloud Messaging

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

// Firebase Analytics (optional but useful)

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

// WorkManager for background processing

implementation "androidx.work:work-runtime:2.7.1"

}

**Step 3: Add Permissions in Manifest**

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

<uses-permission android:name="android.permission.POST\_NOTIFICATIONS" /> <!-- Android 13+ -->

**Step 4: Configure Manifest Metadata and Service**

<application

...>

<!-- Default Notification Icon and Color -->

<meta-data

android:name="com.google.firebase.messaging.default\_notification\_icon"

android:resource="@drawable/ic\_notification"/>

<meta-data

android:name="com.google.firebase.messaging.default\_notification\_color"

android:resource="@color/black"/>

<!-- Default Notification Channel -->

<meta-data

android:name="com.google.firebase.messaging.default\_notification\_channel\_id"

android:value="@string/fcm\_default\_channel"/>

<!-- Firebase Messaging Service -->

<service

android:name=".MyFirebaseMessagingService"

android:exported="false">

<intent-filter>

<action android:name="com.google.firebase.MESSAGING\_EVENT"/>

</intent-filter>

</service>

</application>

**Step 5: Create Firebase Messaging Service**

**MyFirebaseMessagingService.java**

public class MyFirebaseMessagingService extends FirebaseMessagingService {

private static final String TAG = "MyFirebaseMsgService";

// Called when a message is received

@Override

public void onMessageReceived(RemoteMessage remoteMessage) {

// Log sender

Log.v(TAG, "From: " + remoteMessage.getFrom());

// Check for data payload

if (remoteMessage.getData().size() > 0) {

Log.v(TAG, "Message data payload: " + remoteMessage.getData());

scheduleJob(); // Run long tasks in background

}

// Check for notification payload

if (remoteMessage.getNotification() != null) {

Log.v(TAG, "Message Notification Body: " +

remoteMessage.getNotification().getBody());

sendNotification(remoteMessage.getNotification().getBody());

}

}

// Called when token is refreshed

@Override

public void onNewToken(String token) {

Log.v(TAG, "Refreshed token: " + token);

sendRegistrationToServer(token); // Optional for backend use

}

// Schedule background job

private void scheduleJob() {

OneTimeWorkRequest work = new OneTimeWorkRequest.Builder(MyWorker.class).build();

WorkManager.getInstance(this).enqueue(work);

}

private void sendRegistrationToServer(String token) {

// For backend server integration

}

// Create and show a notification

private void sendNotification(String messageBody) {

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

intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP);

PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,

PendingIntent.FLAG\_ONE\_SHOT | PendingIntent.FLAG\_IMMUTABLE);

String channelId = getString(R.string.fcm\_default\_channel);

Uri defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE\_NOTIFICATION);

NotificationCompat.Builder notificationBuilder =

new NotificationCompat.Builder(this, channelId)

.setSmallIcon(R.drawable.ic\_notification)

.setContentTitle(getString(R.string.fcm\_message))

.setContentText(messageBody)

.setAutoCancel(true)

.setSound(defaultSoundUri)

.setContentIntent(pendingIntent);

NotificationManager notificationManager =

(NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);

// Android O+ requires channel

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

NotificationChannel channel = new NotificationChannel(channelId,

"Channel Human Readable Title",

NotificationManager.IMPORTANCE\_DEFAULT);

notificationManager.createNotificationChannel(channel);

}

notificationManager.notify(0, notificationBuilder.build());

}

}

**Step 6: Create Worker Class**

public class MyWorker extends Worker {

private static final String TAG = "MyWorker";

public MyWorker(@NonNull Context context, @NonNull WorkerParameters params) {

super(context, params);

}

@NonNull

@Override

public Result doWork() {

Log.v(TAG, "Performing long-running task...");

return Result.success();

}

}

**Step 7: MainActivity – Subscribe to Topics & Get Token**

public class MainActivity extends AppCompatActivity {

Button btnSubscribe, btnGetToken;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

btnSubscribe = findViewById(R.id.button\_subscribe);

btnGetToken = findViewById(R.id.button\_get\_token);

// Subscribe to "weather" topic

btnSubscribe.setOnClickListener(v ->

FirebaseMessaging.getInstance().subscribeToTopic("weather")

.addOnCompleteListener(task -> {

String msg = task.isSuccessful() ? "Subscribed" : "Subscription failed";

Toast.makeText(MainActivity.this, msg, Toast.LENGTH\_SHORT).show();

})

);

// Get current FCM registration token

btnGetToken.setOnClickListener(v ->

FirebaseMessaging.getInstance().getToken()

.addOnCompleteListener(task -> {

if (!task.isSuccessful()) {

Log.w("FCM", "Fetching FCM registration token failed", task.getException());

return;

}

String token = task.getResult();

Log.v("FCM", "Token: " + token);

Toast.makeText(MainActivity.this, "Token: " + token, Toast.LENGTH\_SHORT).show();

})

); } }

**3. Tools, Libraries, and APIs Used**

* **Firebase Console** – Project creation, FCM setup, message sending.
* **Firebase Cloud Messaging API** (com.google.firebase:firebase-messaging)
* **Firebase BOM** – Version alignment.
* **Firebase Analytics** (optional but integrated).
* **WorkManager API** (androidx.work) – Background processing.
* **NotificationCompat** (androidx.core.app.NotificationCompat) – Backward-compatible notifications.
* **PendingIntent** – Handles actions when user taps notification.
* **Notification Channels** – Required for Android 8+.

**4. Best Practices & Latest Industry Approaches**

* Always use **BOM** to manage Firebase versions.
* On **Android 13+**, request POST\_NOTIFICATIONS runtime permission.
* For sensitive or high-priority notifications, use PRIORITY\_HIGH and IMPORTANCE\_HIGH.
* Use **topic messaging** for grouped audiences and **token-based messaging** for individual devices.
* **Never hardcode API keys or sensitive info** in the app.
* Implement a backend server if you need:
  + **User segmentation**
  + **Advanced targeting**
  + **Custom payload logic**
* Use **data payloads** for silent background updates.
* Use **WorkManager** instead of IntentService for compatibility with Doze/App Standby modes.
* Allow user control over notifications in app settings.

**Part B – Missing but Important Topics**

* **Push notification analytics**: Use Firebase Analytics or custom events to measure engagement.
* **Localization**: Send different notification texts per user language.
* **Image-rich notifications**: Use setStyle(new NotificationCompat.BigPictureStyle()).
* **Click actions**: Deep-link to specific screens instead of always MainActivity.
* **Security**:
  + Verify tokens on server before sending messages.
  + Don’t expose token publicly.
* **Testing**: Use Firebase Console **Test Device** feature.
* **FCM message priorities**:
  + **High** – Deliver immediately, wakes device.
  + **Normal** – Delivered when device is active.
* **Handling notification taps**: Read intent extras in MainActivity to perform navigation.
* **Unsubscribe from topics** when user opts out.