ESP32 + Firebase Realtime Database + Expo Go

Complete Guide: Control LED & Display Sensor Reading from Your Phone

1. Overview & Goals

In this guide, you'll build a complete IoT system connecting an ESP32 microcontroller to a mobile app via Firebase Realtime Database. By the end, you will have:

• A mobile app with a button that turns an LED on/off on the ESP32

• Real-time sensor readings from the ESP32 displayed on your phone

• Understanding of Firebase Realtime Database for IoT projects

2. Firestore vs Realtime Database

Firebase offers two database solutions. Understanding the differences helps you choose the right one:

2.1 Cloud Firestore

• **Data Model:** Document-based (collections containing documents with fields)

• **Queries:** Advanced queries with compound filtering, sorting, and indexing

• **Scaling:** Automatically scales to handle millions of users

• **Offline:** Robust offline support with automatic sync

• **Pricing:** Pay per document read/write/delete

• **Best For:** Complex apps, mobile/web apps, structured data

2.2 Realtime Database (Used in This Guide)

• **Data Model:** JSON tree (one large JSON object)

• **Queries:** Basic filtering and sorting

• **Scaling:** Manual sharding for very large datasets

• **Latency:** Extremely low latency (ideal for IoT!)

• **Pricing:** Pay for bandwidth and storage (not per-operation)

• **Best For:** IoT devices, real-time sync, simple data structures

2.3 Why Realtime Database for ESP32?

For IoT projects with microcontrollers like ESP32, Realtime Database is often preferred because:

• Lower latency for real-time sensor data

• Simpler JSON structure matches embedded programming patterns

• Bandwidth-based pricing is better for frequent small updates

• Well-established Arduino libraries (FirebaseClient by Mobitz)

> **Key Difference:** Firestore uses collections/documents (like folders/files), while Realtime Database is one big JSON tree (like nested JavaScript objects).

3. Prerequisites

Before starting, ensure you have:

For Expo App:

• Node.js (v18 or newer)

• npm or yarn package manager

• Expo Go app on your iOS or Android phone

• A Google account (for Firebase)

For ESP32:

• Arduino IDE (2.0+ recommended)

• ESP32 board support installed in Arduino IDE

• ESP32 development board

• LED, 220-330 ohm resistor, potentiometer (or any analog sensor)

• Breadboard and jumper wires

4. Create a New Expo Project

Open your terminal and run the following commands:

```
# Create a new Expo app
npx create-expo-app@latest esp32-controller

# Navigate into the project directory
cd esp32-controller

# Start the development server (we'll do this later)
# npx expo start
```

5. Connecting with Expo Go

5.1 Same WiFi Network

IMPORTANT: Your development computer and your phone must be on the same WiFi network for Expo Go to connect properly.

5.2 Scan the QR Code

When you run **npx expo start**, a QR code appears in your terminal:

• **iOS:** Open your Camera app and scan the QR code. Tap the notification to open in Expo Go.

• **Android:** Open the Expo Go app and use the 'Scan QR Code' option.

5.3 Access the Debug Menu

Shake your phone while the app is running to open the Expo developer menu. This gives you access to reload, developer tools, and debugging options.

> **Tip:** If the QR code doesn't work, try pressing 's' in the terminal to switch to Expo Go mode.

6. Firebase Project Setup

6.1 Create a Firebase Project

1. Go to **https://console.firebase.google.com**

2. Click **Add project**

3. Enter a project name (e.g., 'esp32-controller')

4. Disable Google Analytics (optional, not needed for this project)

5. Click **Create project** and wait for it to complete

6. Click **Continue** when ready

6.2 Register a Web App

We need to register a Web App to get the Firebase configuration for our Expo app:

1. In your project console, click the **Web icon (</>)**
2. Enter an app nickname (e.g., 'expo-app')
3. **Do NOT** check 'Firebase Hosting'
4. Click **Register app**
5. **Copy the entire firebaseConfig object** - you'll need this!
6. Click **Continue to console**

6.3 Set Up Authentication (Required for ESP32)

The ESP32 needs to authenticate with Firebase:

1. In the left sidebar, click **Build** > **Authentication**
2. Click **Get started**
3. Click **Email/Password**
4. Toggle **Enable** for Email/Password
5. Click **Save**
6. Go to the **Users** tab
7. Click **Add user**
8. Enter an email (e.g., esp32@yourproject.com) and password
9. Click **Add user**

Save this email and password - you'll need them for the ESP32 code.

6.4 Create the Realtime Database

1. In the left sidebar, click **Build** > **Realtime Database**
2. Click **Create Database**
3. Select a location closest to you (e.g., United States, Belgium)
4. Select **Start in test mode**
5. Click **Enable**

6.5 Copy Your Database URL

After creation, your database URL is shown at the top of the page. It looks like:

```
https://your-project-id-default-rtdb.firebaseio.com/
```

Copy this URL - you'll need it for both the ESP32 and Expo app.

6.6 Get Your API Key

1. Click the **gear icon** next to 'Project Overview'
2. Select **Project settings**
3. Scroll down and copy your **Web API Key**

> **Security Note:** Test mode rules expire after 30 days and allow anyone to read/write. For production, configure proper security rules in the Rules tab.

7. Install Firebase in Your Expo Project

Navigate to your project folder and run:

```
cd esp32-controller
npx expo install firebase
```

8. Create Configuration Files

8.1 Create firebaseConfig.ts

Create a new file called **firebaseConfig.ts** in your project root:

```
// firebaseConfig.ts
import { initializeApp } from "firebase/app";
import { getDatabase } from "firebase/database";

// Replace with YOUR config from Firebase Console!
const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_PROJECT.firebaseapp.com",
  projectId: "YOUR_PROJECT_ID",
  storageBucket: "YOUR_PROJECT.appspot.com",
```

```
  messagingSenderId: "YOUR_SENDER_ID",
  appId: "YOUR_APP_ID",
  // IMPORTANT: Add your Realtime Database URL
  databaseURL: "https://YOUR_PROJECT-default-rtdb.firebaseio.com/"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

// Initialize Realtime Database
export const database = getDatabase(app);
export { app };
```

8.2 Create metro.config.js

Create a file called **metro.config.js** in your project root:

```
// metro.config.js
const { getDefaultConfig } = require('expo/metro-config');
const config = getDefaultConfig(__dirname);
config.resolver.sourceExts.push('cjs');
module.exports = config;
```

9. Project Structure

```
esp32-controller/
  |- app/
  |   |- (tabs)/
  |   |   |- index.tsx      <-- We will replace this file
  |   |   |- explore.tsx
  |   |   |- _layout.tsx
  |   |- _layout.tsx
  |- firebaseConfig.ts      <-- Create this file
  |- metro.config.js        <-- Create this file
  |- package.json
  |- ...other files
```

10. Create the Expo App Interface

Replace the contents of **app/(tabs)/index.tsx** with:

```
// app/(tabs)/index.tsx
import React, { useState, useEffect } from 'react';
import {
  StyleSheet,
  Text,
  View,
  TouchableOpacity,
  Alert,
} from 'react-native';
import { database } from '../../firebaseConfig';
import { ref, onValue, set } from 'firebase/database';

export default function HomeScreen() {
  const [ledOn, setLedOn] = useState(false);
  const [sensorValue, setSensorValue] = useState<number | null>(null);
  const [connected, setConnected] = useState(false);

  useEffect(() => {
    // Listen to LED state from Firebase
    const ledRef = ref(database, '/device/led');
    const unsubscribeLed = onValue(ledRef, (snapshot) => {
      const value = snapshot.val();

      // Initialize to false if no value exists in database
      if (value === null) {
        set(ledRef, false);
        return;
      }

      setLedOn(value === true);
      setConnected(true);
    }, (error) => {
      console.error('LED listener error:', error);
      Alert.alert('Error', 'Failed to connect to database');
    });

    // Listen to sensor value from Firebase
    const sensorRef = ref(database, '/device/sensor');
    const unsubscribeSensor = onValue(sensorRef, (snapshot) => {
      const value = snapshot.val();
```

```
    if (value !== null) {
      setSensorValue(value);
    }
  }, (error) => {
    console.error('Sensor listener error:', error);
  });

  // Cleanup listeners on unmount
  return () => {
    unsubscribeLed();
    unsubscribeSensor();
  };
}, []);

// Toggle LED state in Firebase
const toggleLed = async () => {
  try {
    const ledRef = ref(database, '/device/led');
    await set(ledRef, !ledOn);
  } catch (error) {
    console.error('Error toggling LED:', error);
    Alert.alert('Error', 'Failed to toggle LED');
  }
};

return (
  <View style={styles.container}>
    <Text style={styles.title}>ESP32 Controller</Text>

    {/* Connection Status */}
    <View style={styles.statusContainer}>
      <View style={[styles.statusDot,
        { backgroundColor: connected ? '#22c55e' : '#ef4444' }]} />
      <Text style={styles.statusText}>
        {connected ? 'Connected' : 'Connecting...'}
      </Text>
    </View>

    {/* LED Control Card */}
    <View style={styles.card}>
      <Text style={styles.cardTitle}>LED Control</Text>
      <TouchableOpacity
        style={[styles.ledButton,
          { backgroundColor: ledOn ? '#eab308' : '#374151' }]}
        onPress={toggleLed}
      >
        <Text style={styles.ledButtonText}>
          {ledOn ? '■ ON' : '■ OFF'}
        </Text>
      </TouchableOpacity>
      <Text style={styles.hint}>Tap to toggle the LED</Text>
    </View>

    {/* Sensor Reading Card */}
    <View style={styles.card}>
      <Text style={styles.cardTitle}>Sensor Reading</Text>
      <Text style={styles.sensorValue}>
        {sensorValue !== null ? sensorValue : '---'}
      </Text>
      <View style={styles.progressBar}>
        <View style={[styles.progressFill,
```

```
            { width: `${((sensorValue || 0) / 4095) * 100}%` }]} />
          </View>
          <Text style={styles.hint}>Range: 0 - 4095 (12-bit ADC)</Text>
        </View>
      </View>
    );
}
```

Add these styles at the bottom of index.tsx:

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#f5f5f5',
    paddingHorizontal: 20,
    paddingTop: 20,
  },
  title: {
    fontSize: 28,
    fontWeight: 'bold',
    textAlign: 'center',
    marginBottom: 10,
    color: '#1a1a1a',
  },
  statusContainer: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'center',
    marginBottom: 20,
  },
  statusDot: {
    width: 12,
    height: 12,
    borderRadius: 6,
    marginRight: 8,
  },
  statusText: {
    fontSize: 14,
    color: '#6b7280',
  },
  card: {
    backgroundColor: '#fff',
    borderRadius: 16,
    padding: 20,
    marginBottom: 20,
    shadowColor: '#000',
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.1,
    shadowRadius: 4,
    elevation: 3,
  },
  cardTitle: {
    fontSize: 18,
    fontWeight: '600',
    color: '#374151',
    marginBottom: 15,
    textAlign: 'center',
  },
  ledButton: {
    padding: 30,
    borderRadius: 100,
    alignItems: 'center',
    justifyContent: 'center',
    alignSelf: 'center',
    width: 150,
    height: 150,
  },
  ledButtonText: {
    fontSize: 24,
```

```
    fontWeight: 'bold',
    color: '#fff',
  },
  hint: {
    fontSize: 12,
    color: '#9ca3af',
    textAlign: 'center',
    marginTop: 10,
  },
  sensorValue: {
    fontSize: 48,
    fontWeight: 'bold',
    textAlign: 'center',
    color: '#3b82f6',
    marginBottom: 15,
  },
  progressBar: {
    height: 8,
    backgroundColor: '#e5e7eb',
    borderRadius: 4,
    overflow: 'hidden',
  },
  progressFill: {
    height: '100%',
    backgroundColor: '#3b82f6',
    borderRadius: 4,
  },
});
```

11. ESP32 Hardware Setup

11.1 Components Needed

• ESP32 development board (any variant)

• LED (any color)

• 220-330 ohm resistor

• Potentiometer (10K) OR any analog sensor (temperature, light, etc.)

• Breadboard and jumper wires

11.2 Wiring Diagram

```
ESP32 Wiring:

LED Circuit:
  ESP32 GPIO 2  ---[220 ohm resistor]---[LED +]---[LED -]--- GND
  (GPIO 2 is also the built-in LED on most ESP32 boards)

Potentiometer (as analog sensor):
  Left Pin   ---- 3.3V
  Middle Pin ---- ESP32 GPIO 34 (analog input)
  Right Pin  ---- GND

Note: GPIO 34 is input-only on ESP32, perfect for analog reads.
```

You can substitute any analog sensor for the potentiometer.

12. ESP32 Arduino Code

12.1 Install Required Library

In Arduino IDE:

1. Go to **Sketch** > **Include Library** > **Manage Libraries**

2. Search for **FirebaseClient**

3. Install **FirebaseClient** by Mobitz

12.2 Complete ESP32 Code

Create a new sketch and paste the following:

```
// ESP32 Firebase Realtime Database - LED Control + Sensor
#define ENABLE_USER_AUTH
#define ENABLE_DATABASE

#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <FirebaseClient.h>

// ===== UPDATE THESE VALUES! =====
#define WIFI_SSID "YOUR_WIFI_SSID"
#define WIFI_PASSWORD "YOUR_WIFI_PASSWORD"

#define API_KEY "YOUR_FIREBASE_API_KEY"
#define DATABASE_URL "https://your-project-default-rtdb.firebaseio.com/"
#define USER_EMAIL "esp32@yourproject.com"
#define USER_PASSWORD "your_esp32_password"

// ===== PIN DEFINITIONS =====
#define LED_PIN 2        // Built-in LED on most ESP32 boards
#define SENSOR_PIN 34    // Analog input pin

// ===== FIREBASE OBJECTS =====
void processData(AsyncResult &aResult);

UserAuth user_auth(API_KEY, USER_EMAIL, USER_PASSWORD);
FirebaseApp app;
WiFiClientSecure ssl_client;
using AsyncClient = AsyncClientClass;
AsyncClient aClient(ssl_client);
RealtimeDatabase Database;

// ===== TIMING VARIABLES =====
unsigned long lastSensorUpdate = 0;
const unsigned long sensorInterval = 2000; // 2 seconds

unsigned long lastLedCheck = 0;
const unsigned long ledCheckInterval = 500; // 500ms

// ===== STATE VARIABLES =====
int sensorValue = 0;
bool ledState = false;
```

```
void setup() {
  Serial.begin(115200);

  // Initialize pins
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);

  // Connect to WiFi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected! IP: ");
  Serial.println(WiFi.localIP());

  // Configure SSL client
  ssl_client.setInsecure();
  ssl_client.setConnectionTimeout(1000);
  ssl_client.setHandshakeTimeout(5);

  // Initialize Firebase
  initializeApp(aClient, app, getAuth(user_auth), processData, "authTask");
  app.getApp<RealtimeDatabase>(Database);
  Database.url(DATABASE_URL);

  Serial.println("Firebase initialized!");
}

void loop() {
  app.loop();

  if (app.ready()) {
    unsigned long now = millis();

    // === CHECK LED STATE FROM FIREBASE ===
    if (now - lastLedCheck >= ledCheckInterval) {
      lastLedCheck = now;
      Database.get(aClient, "/device/led", processData, false, "getLed");
    }

    // === SEND SENSOR VALUE TO FIREBASE ===
    if (now - lastSensorUpdate >= sensorInterval) {
      lastSensorUpdate = now;
      sensorValue = analogRead(SENSOR_PIN);
      Database.set<int>(aClient, "/device/sensor", sensorValue,
                        processData, "setSensor");
      Serial.print("Sensor: ");
      Serial.println(sensorValue);
    }
  }
}

void processData(AsyncResult &aResult) {
  if (!aResult.isResult()) return;

  if (aResult.isError()) {
    Serial.print("Error: ");
    Serial.println(aResult.error().message().c_str());
```

```
    return;
  }

  if (aResult.available()) {
    String taskId = aResult.uid().c_str();
    String payload = aResult.c_str();

    if (taskId == "getLed") {
      // Debug output - see what Firebase returns
      Serial.print("LED payload: [");
      Serial.print(payload);
      Serial.println("]");

      // Determine new state from payload
      bool newState = (payload == "true" || payload == "1");

      // ONLY update LED if state actually changed
      if (newState != ledState) {
        ledState = newState;
        digitalWrite(LED_PIN, ledState ? HIGH : LOW);
        Serial.print("LED changed to: ");
        Serial.println(ledState ? "ON" : "OFF");
      }
    }
  }
}
```

12.3 Update Configuration Values

Replace these placeholders in the code:

• **YOUR_WIFI_SSID** - Your WiFi network name

• **YOUR_WIFI_PASSWORD** - Your WiFi password

• **YOUR_FIREBASE_API_KEY** - From Firebase Project Settings

• **DATABASE_URL** - Your Realtime Database URL (include https://)

• **USER_EMAIL** - The email you created in Firebase Authentication

• **USER_PASSWORD** - The password for that user

12.4 Upload to ESP32

1. Select your ESP32 board: **Tools** > **Board** > **ESP32 Dev Module**

2. Select the correct port: **Tools** > **Port**

3. Click **Upload**

4. Open **Serial Monitor** (115200 baud) to verify connection

> **Important:** The ESP32 code only updates the LED when the state *actually changes*. This prevents flickering. Watch the Serial Monitor for 'LED changed to: ON/OFF' messages to confirm state changes.

13. Testing the Complete System

13.1 Start the ESP32

1. Power on ESP32 via USB

2. Open Serial Monitor - verify WiFi and Firebase connection

3. You should see 'Firebase initialized!' and sensor readings

13.2 Start the Expo App

```
cd esp32-controller
npx expo start
```

Scan the QR code with Expo Go on your phone.

13.3 Test LED Control

1. Tap the LED button in the app

2. The LED on your ESP32 should turn on/off

3. The button color changes to indicate state (yellow=ON, gray=OFF)

4. Serial Monitor should show 'LED changed to: ON' or 'LED changed to: OFF'

13.4 Test Sensor Reading

1. Turn the potentiometer (or change your sensor)

2. Watch the value update in the app (every 2 seconds)

3. The progress bar shows the relative value

13.5 Verify in Firebase Console

Go to your Firebase Realtime Database. You should see:

```
{
  "device": {
    "led": false,    // Changes when you tap the button
    "sensor": 2048   // Updates every 2 seconds from ESP32
  }
}
```

14. Troubleshooting

ESP32 won't connect to WiFi:

• Double-check SSID and password (case-sensitive)

• Ensure your WiFi is 2.4GHz (ESP32 doesn't support 5GHz)

ESP32 won't connect to Firebase:

• Verify DATABASE_URL includes https:// and ends with /

• Check that API_KEY matches Project Settings

• Ensure user email/password match what you created in Authentication

App shows 'Connecting...' forever:

• Make sure databaseURL is in your firebaseConfig.ts

• Verify you're importing from 'firebase/database'

• Check terminal/console for error messages

LED flickering or not responding:

• Check Serial Monitor for 'LED payload: [...]' messages

• Ensure the app initializes /device/led to false if null

• Verify ESP32 code only updates LED when state changes

• Manually set /device/led to true in Firebase Console to test

Sensor value stuck at 0:

• Verify potentiometer middle pin connects to GPIO 34

• Ensure 3.3V and GND are connected to outer pins

• Try different ADC pins: 32, 33, 35, 36, 39

15. Files Summary

```
Files Created:
  esp32-controller/
    |- firebaseConfig.ts   # Firebase configuration
    |- metro.config.js     # Metro bundler config
    |- app/(tabs)/index.tsx # Main app interface (replaced)

  Arduino/
    |- esp32_controller.ino # ESP32 sketch

Firebase Database Structure:
  /device/led     <- App writes (true/false), ESP32 reads
  /device/sensor  <- ESP32 writes (0-4095), App reads

Key Points:
  - App initializes /device/led to false if it doesn't exist
  - ESP32 only updates LED when state actually changes
  - Both use real-time listeners for instant updates
```

Resources

• ESP32 Firebase Tutorial: https://randomnerdtutorials.com/esp32-firebase-realtime-database/

• Firebase Realtime Database: https://firebase.google.com/docs/database

• FirebaseClient Library: https://github.com/mobizt/FirebaseClient

• Expo Firebase Guide: https://docs.expo.dev/guides/using-firebase/