**Live Streaming Strategy**

Firebase is not suitable for live streaming due to latency and storage costs. Instead, the Flutter app will display the ESP32-CAM stream using an MJPEG video stream over HTTP.

**How Flutter Displays ESP32-CAM Stream?**

ESP32-CAM hosts an MJPEG stream on http://<ESP\_IP>:81/stream.

Flutter fetches and displays the stream using an Image.network() widget or a WebView.

Method 1: Using Image.network() (Simplest)

This method loads the MJPEG stream like a normal image.

Note: This works but may not be smooth.

Method 2: Using flutter\_inappwebview (Recommended)

Use an embedded WebView to display the ESP32-CAM's MJPEG stream.

**Why WebView?**

It directly plays the stream with lower latency and better performance.

**How to Get the ESP32-CAM Stream URL?**

Upload this ESP32-CAM code in Arduino IDE or PlatformIO:

After flashing, check the Serial Monitor for the ESP32 IP address.

Example output: Camera Ready! Use 'http://192.168.1.100:81/stream' to view

Replace YOUR\_ESP32\_IP in Flutter code with the actual IP.

**Final Thoughts**

For best results, use WebView in Flutter.

Ensure ESP32-CAM and phone are on the same WiFi.

Use a static IP or mDNS (http://esp32cam.local/stream) for consistency.

**Query:**

What if, i get the IP, send it to firebase, flutter gets it from there, and displays the camera stream on screen without having to leave the app to get IP from ESP32-CAM. Or modifying code in flutter to directly communicate with ESP32-CAM (complex as we already have a strategy of using firebase as a relay)

**Response:**

That’s actually a solid and practical strategy — you’re leveraging Firebase as a "middleman" to relay the ESP32-CAM's IP to Flutter, avoiding hardcoding, manual IP entry, or mDNS issues.

Let’s break it down:

## ✅ **Proposed Workflow (IP via Firebase)**

### 1. **ESP32-CAM connects to WiFi, gets its IP.**

* Immediately uploads its IP to Firebase Realtime Database under a known path, e.g., /esp32cam/ip.

### 2. **Flutter app reads this IP from Firebase** when needed.

* Builds the stream URL dynamically:  
  http://<retrieved\_IP>:81/stream

### 3. **Flutter displays stream in WebView** → Done.

## 🔧 **Advantages of This Approach**

* Dynamic IP handling: No need to hardcode or manage static IPs.
* No mDNS limitations (some devices can’t resolve .local).
* No user input needed → Fully automatic.
* Keeps your existing Firebase infrastructure useful.
* ESP32-CAM stays standalone, only pushing IP once.

## 🔻 **Considerations**

* The phone and ESP32-CAM must be on the same network (for the stream to load), since local IP is not globally routable.
* Flutter → ESP32-CAM stream is direct (not via Firebase), just using Firebase to get the IP.
* Minimal data use from Firebase → Just small IP string fetch.

## 🔥 **ESP32-CAM Sample Code to Upload IP to Firebase**

You’ll need Firebase\_ESP\_Client library.

#include <WiFi.h>

#include <Firebase\_ESP\_Client.h>

// WiFi and Firebase config

const char\* ssid = "YourSSID";

const char\* password = "YourPassword";

// Firebase credentials

#define API\_KEY "your-firebase-api-key"

#define DATABASE\_URL "https://your-project.firebaseio.com/"

#define USER\_EMAIL "user@example.com"

#define USER\_PASSWORD "password"

FirebaseData fbdo;

FirebaseAuth auth;

FirebaseConfig config;

void setup() {

Serial.begin(115200);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) delay(500);

Serial.println(WiFi.localIP()); // For debug

config.api\_key = API\_KEY;

config.database\_url = DATABASE\_URL;

auth.user.email = USER\_EMAIL;

auth.user.password = USER\_PASSWORD;

Firebase.begin(&config, &auth);

Firebase.reconnectWiFi(true);

String ipStr = WiFi.localIP().toString();

Firebase.RTDB.setString(&fbdo, "/esp32cam/ip", ipStr);

}

void loop() {

// Nothing needed in loop for this purpose

}

## 🧩 **Flutter (Simplified Approach)**

### Read IP from Firebase:

final DatabaseReference ref = FirebaseDatabase.instance.ref('esp32cam/ip');

final DataSnapshot snapshot = await ref.get();

final ip = snapshot.value as String;

### Show Stream:

WebView(

initialUrl: 'http://$ip:81/stream',

javascriptMode: JavascriptMode.unrestricted,

)

## 🎯 Conclusion

Yes, this Firebase IP relay approach is perfect for:

* Simplicity
* Flexibility
* No network configuration pain
* Keeping your app professional and seamless

**Query:**

So, if im using a webserver with FOSS code (that mostly people use), how would i just fetch the image and not the controls from the web server.

**Response:**

Great question! If your ESP32-CAM is serving a full web page (with controls, buttons, etc.) and you only want to fetch the MJPEG video stream — not the whole HTML page — here’s how to do it:

## 🎯 **ESP32-CAM Has Two Key URLs:**

1. Full Web Interface (with buttons)  
   http://<ESP\_IP>/  
   → Loads the full HTML page with controls and embedded stream.
2. Raw MJPEG Stream (video only)  
   http://<ESP\_IP>:81/stream  
   → Sends only the video, no HTML, no buttons.

## ✅ **Solution: Use Only the MJPEG Stream URL**

In your Flutter app, or any external viewer, ignore the full web page and load only this:

http://<ESP\_IP>:81/stream

This URL directly serves the camera frames (as MJPEG) — no buttons, no red controls, just pure video.

## ⚙️ **What Happens Technically?**

* The ESP32-CAM typically hosts a web server with two routes:
  + / → Serves HTML with buttons.
  + /stream or :81/stream → Serves raw MJPEG stream via HTTP.

## 🚨 **Important**

Make sure the ESP32-CAM firmware includes the stream server (startCameraServer() with MJPEG) and it’s accessible on port 81.

## 🧪 **Test It Manually First**

1. Enter in browser:  
   http://<ESP\_IP>/ → See buttons + video.
2. Then enter:  
   http://<ESP\_IP>:81/stream → Only video.

## 🔧 **Conclusion**

To only fetch the image/video, point your app to http://<ESP\_IP>:81/stream.  
That URL does not serve HTML or controls, just the raw video stream.