Comprehensive Implementation Guide: Resource-Aware Bird Feeder Camera

This guide outlines the three phases of development for the solar-powered IoT bird feeder camera. The core design conserves computational resources by only activating power-intensive features (Live Feed, Real-time Settings Listener) when the mobile app's dedicated Live Feed screen is actively in use.

Phase 1: Cloud & Storage Setup (Infrastructure)

Step	Goal	Action / Key Concept	Note
1.	**Firebase Project**	Create a new Firebase project and enable necessary services.	Enable Authentication (Anonymous, Email/Password), Firestore, and Storage.
2.	**Firestore Structure**	Define the core data paths (Collections/Documents).	**config/settings** (Camera parameters), **logs/energy_data** (Battery/Solar history), **logs/sightings** (Motion history), **status/streaming_enabled** (Crucial ON/OFF flag for camera server).
3.	**Storage Buckets**	Set up Firebase Storage.	Create two buckets: one for high-resolution images/videos (/media/sightings/) and one for temporary or

			low-res thumbnails.
4.	**Pi Authentication**	Configure Firebase Admin SDK credentials.	Generate a service account key file (JSON) for the Raspberry Pi to authenticate and read/write data securely.

Phase 2: Raspberry Pi (Edge) Features Development (Power-Optimized)

The Raspberry Pi runs three services: Motion Detector/Uploader (Always On, Low Power), Data Logger (Scheduled, Low Power), and the **Control Handler (On-Demand, High Power)**.

2.1. Image Uploading and Sighting Metadata (Always Active - Low Power)

Step	Goal	Action / Key Concept	Note
5.	**Motion Detection**	Implement a simple Python script using picamera2 or similar, listening for motion (PIR sensor or simple computer vision).	The script uses minimal CPU until motion is detected. **Optional:** Implement basic object detection (e.g., using TinyML) to filter non-bird events.
6.	**Capture & Upload**	On motion, capture a high-resolution image and upload it to /media/sightings/.	Optimize capture settings to minimize processing time. This is burst activity, not continuous consumption.

7.	**Log Event**	Write a document to **logs/sightings** containing the image URL, timestamp, and metadata (e.g., Bird Species if identified, or "Unknown").	Firestore document addition is fast and lightweight.
8.	**Reliability & Retry**	Implement a local retry mechanism (e.g., save failed uploads locally to a "queue" directory).	A separate cron job can scan the local queue periodically to attempt re-uploading network failure.

2.2. Battery and Solar Data Logging (Scheduled - Low Power)

Step	Goal	Action / Key Concept	Note
9.	**Read Sensors**	Python script reads voltage, current, and temperature data from the attached solar/battery management hardware (e.g., using I2C/ADS1115).	Sensor reading is fast and lightweight. Convert raw voltage to battery percentage.
10.	**Log Data**	Append the sensor data (timestamp, voltage, percent) to the dedicated **logs/energy_data** collection.	Scheduled task (e.g., via cron every 10 minutes) ensures minimal, intermittent resource use.
11.	**System Health**	Log CPU temperature and	Crucial for diagnosing stability

		memory usage periodically.	and overheating issues remotely in a weather-exposed enclosure.

2.3. Power Conservation Strategy (On-Demand Activation)

This section details the mechanism that turns the high-power features on and off based on the app's status.

Step	Goal	Action / Key Concept	Note
12. Listener Script	The Pi must know when the mobile app's Live Feed is open.	Create a tiny, dedicated Python script running 24/7 that listens **ONLY** to the status/streaming_enabled boolean document via the Firebase Admin SDK.	This listener is extremely lightweight and consumes minimal power.
13. Service Activation	Start the high-power features.	When streaming_enabled: true, use Python's subprocess.Popen to **START** the Uvicorn server (FastAPI, Step 15) and the dedicated **Cloud Settings Listener** (Step 16).	Save the PID of the running process for later termination.
14. Service Deactivation	Stop the high-power features.	When streaming_enabled: false, use the stored PID to send a termination signal (SIGTERM or os.kill)	Immediately shuts down the high-power camera server to save battery.

		to the Uvicorn/FastAPI process.	

2.4. Configurable Settings and Live Server (Active Only When Live Feed is Open)

This module is a high-power service that is **only run when status/streaming_enabled is true**. It manages the real-time camera feed and applies settings changes from the cloud.

Step	Goal	Action / Key Concept	Research Recommendation
15. Modular FastAPI	Serve real-time data over the local network.	Define the FastAPI application with two endpoints: /live (M-JPEG streaming) and /snapshot (a single JPEG capture command).	**"FastAPI streaming camera frames MJPEG"**
16. Real-time Settings Listener	Listen for changes to the config/settings document.	This *separate* thread/process only runs while the FastAPI server is active. It uses the Firebase Admin SDK to listen for settings changes and apply them to the live camera module (e.g., exposure, focus).	This is the source of the **real-time changes**.
17. Local IP Discovery	Ensure the Pi's local IP is known to the app.	The FastAPI start-up script can write its local IP address to a temporary Firebase document	**"Python get local IP address"**

		(status/ip_address) so the Flutter app knows where to point its HTTP requests.	

Phase 3: Flutter App Development (Status Management and UI)

The app's Live Feed screen is the *master controller* for the Pi's power state.

Step	Goal	Action / Key Concept	Note
18. Sighting Feed UI	Display the motion history feed.	Use a StreamBuilder to display the contents of the **logs/sightings** collection, showing the thumbnail images and event details.	This is the default low-power gallery view.
19. Settings Feature	Allow user configuration.	Create a settings screen that updates the **config/settings** document.	Updates only matter when the Pi's listener (Step 16) is active.
20. Data Visualization	Show battery health.	Read the **logs/energy_data** periodically and display line charts for voltage, charge, and temperature.	Uses a chart library (e.g., fl_chart) for visualization.
21. Live Feed Toggle (Activation)	Tell the Pi to enable streaming.	When the user navigates *into* the Live Feed screen, in	This triggers the Pi to start the camera server.

		the screen's initState(), set **status/streaming_enabled: true**.	
22. Live Feed Toggle (Deactivation)	Tell the Pi to disable streaming.	When the user navigates *out of* the Live Feed screen, in the screen's dispose(), set **status/streaming_enabled: false**.	This immediately shuts down the high-power camera server.
23. Live Feed UI	Connect to the local FastAPI server.	Use the IP from status/ip_address to connect a stream or video player widget to the Pi's **/live** M-JPEG endpoint.	Requires handling network connectivity and error states.
24. Snapshot Button	Trigger an on-demand image capture.	On the live feed screen, send an HTTP request to the Pi's local /snapshot endpoint. The sighting feed updates automatically via the Firebase listener.	The Pi's FastAPI server handles the actual capture and Firebase upload.