# Flagging Unauthorized Billboards — Q/A Summary + Software Requirements Specification (SRS)

This document contains the final Q/A decisions we made while scoping the project, followed by a clear SRS. Use this .md file to feed GitHub Copilot, or as submission documentation.

## Table of Contents

1. Q/A Summary (decisions)
2. Software Requirements Specification (SRS)
   * Introduction
   * Functional Requirements
   * Non-Functional Requirements
   * Constraints & Assumptions
   * Deliverables
3. System Architecture (high level)
4. Recommended Project Folder Structure (for GitHub)
5. Recommended Tech Stack & Packages
6. Implementation Notes (AI model, offline-first, Firebase)
7. How to use this doc with GitHub Copilot / Next steps

# 1. Q/A Summary (decisions)

These are the answers to the step-by-step questions we asked while planning the prototype. Use these as the single source of truth for design and implementation.

* **Detection approach:** Hybrid — on-device AI suggestion (TensorFlow Lite) + manual user confirmation/correction.
* **Input mode:** Photos only (user takes a picture within the app). No live video for MVP.
* **Data & rules:** Hybrid — for big cities use a sample/mock database of authorized billboards; otherwise rely on AI + user input.
* **Mobile app platform:** Flutter (cross-platform).
* **Storage & reporting:** Offline-first: store reports locally (JSON/SQLite) when no internet; when online, sync to Firebase backend (Firestore / Storage).
* **Privacy:** Anonymous by default (photo + geotag + violation type). Optional user login for rewards/leaderboard; personal ID stored privately in backend and NOT shown on public dashboard.
* **Public dashboard:** Yes — web dashboard with Google Maps API showing pins/heatmap of flagged billboards (public-facing, no personal info).
* **Rewards/Gamification:** Optional; only for users who opt-in with login.

# 2. Software Requirements Specification (SRS)

## 2.1 Project Title

**Flagging Unauthorized Billboards — Prototype**

## 2.2 Purpose

Build a Flutter mobile app to detect and report unauthorized/non-compliant billboards using a hybrid approach — TensorFlow Lite inference + manual user confirmation. Reports should work offline and sync online. A public dashboard will display flagged billboards.

## 2.3 Scope

* Take a photo from the mobile app and run on-device inference.
* Provide AI suggestions for violations and let the user confirm or correct.
* Detect violations such as size/dimensions, placement/geolocation issues, structural hazard, and objectionable content.
* Cross-check with a mock authorized-billboard database for select cities.
* Allow anonymous reporting by default; optional login for rewards.
* Store data offline and sync to Firebase when online.
* Provide a public web dashboard with map/heatmap view.

## 2.4 Functional Requirements

### Mobile App (Flutter)

* **Camera screen**: capture photo and automatically attach timestamp + geolocation (if user allows). Provide a privacy disclaimer before first use.
* **AI inference**: run a TensorFlow Lite model on the captured image to locate billboards and predict possible violation types.
* **Manual confirmation UI**: show AI suggestions and provide checkboxes/buttons for the user to confirm or correct (e.g., Size, Placement, Structure, Content)
* **Local persistence**: save reports locally (JSON or SQLite). Provide an offline queue.
* **Sync**: when online, upload report data (photos to Firebase Storage, metadata to Firestore) and mark as synced.
* **Optional login**: email-based sign-in for rewards/leaderboard.

### Backend & Dashboard

* **Backend storage**: Firestore for metadata, Firebase Storage for photos.
* **Mock DB**: a Firestore collection (or local JSON) with authorized billboard entries (id, lat/lng, dimensions, city).
* **Public dashboard**: web page that reads public reports from Firestore and displays them as markers or heatmap using Google Maps JavaScript API.

## 2.5 Non-Functional Requirements

* **Performance**: on-device inference < 2–3 seconds for a single photo on a modest smartphone.
* **Scalability**: Firestore can scale to thousands of reports; dashboard must handle pagination or clustering.
* **Privacy**: default anonymous reports; user consent before storing personal identifiers.
* **Reliability**: offline capability with later sync.

## 2.6 Constraints

* Prototype supports **photos only** (no video).
* Mock city DB is limited (few sample entries for demo only).
* Do not use direct code cloning from public GitHub repos (per contest rules).
* Keep data collection ethical — avoid facial recognition or broad public surveillance.

## 2.7 Assumptions

* Users will permit location access for geotagging (opt-in).
* Device has camera and supports Flutter and TF Lite.

## 2.8 Deliverables

* Flutter mobile app (prototype).
* TensorFlow Lite model or placeholder for inference (trained model optional — can use pre-existing object detection architecture trained on billboard-like data).
* Firebase backend (Firestore + Storage) and mock authorized billboard DB.
* Public web dashboard with map/heatmap.
* Architecture diagram, pitch deck (≤10 slides), documentation, and demo video.

# 3. System Architecture (high level)

* **User (mobile)** → **Flutter App** (camera + TF Lite + manual UI) → Local Storage (offline) & Firebase (online)
* **Firebase Backend** → Firestore (reports + mock DB), Firebase Storage (photos)
* **Public Dashboard** → reads Firestore, shows pins/heatmap via Google Maps API

(Refer to architecture diagram image created separately.)

# 4. Recommended Project Folder Structure (for GitHub)

flagging-billboards/  
├── README.md  
├── mobile/  
│ ├── flutter\_app/  
│ │ ├── lib/  
│ │ │ ├── main.dart  
│ │ │ ├── screens/  
│ │ │ │ ├── camera\_screen.dart  
│ │ │ │ ├── review\_screen.dart  
│ │ │ │ └── settings\_screen.dart  
│ │ │ ├── models/  
│ │ │ ├── services/  
│ │ │ │ ├── tfservice.dart  
│ │ │ │ ├── local\_storage.dart  
│ │ │ │ └── firebase\_sync.dart  
│ │ ├── assets/  
│ │ │ └── tflite\_model.tflite  
│ │ └── pubspec.yaml  
│ └── README.md  
├── backend/  
│ ├── firebase\_rules.txt  
│ └── mock\_db/  
│ └── authorized\_billboards.json  
├── dashboard/  
│ ├── index.html  
│ ├── src/  
│ └── package.json  
├── docs/  
│ ├── SRS\_and\_QA.md <-- (this file)  
│ └── architecture.png  
└── demo\_video/  
 └── walkthrough.mp4

**Notes:** Put the .tflite model under assets/ and reference in pubspec.yaml.

# 5. Recommended Tech Stack & Packages

## Mobile (Flutter)

* Flutter (stable)
* Dart
* Packages (examples):
  + camera or image\_picker (capture photo)
  + geolocator (get location)
  + permission\_handler (ask permissions)
  + tflite\_flutter or tflite (TF Lite inference)
  + path\_provider & sqflite (local storage) or hive
  + firebase\_core, cloud\_firestore, firebase\_storage, firebase\_auth (if using Firebase)

## Backend & Dashboard

* Firebase (Firestore + Storage) for rapid prototype
* Public Dashboard: simple HTML/JS app using Google Maps JavaScript API (or React + Leaflet)

## AI Model

* TensorFlow -> export to TensorFlow Lite.
* Base model: MobileNet-SSD / YOLOv5 or efficient object detector converted to TF Lite.
* Labels: billboard bounding box + optional attributes (size estimate, textual content classification via OCR+classifier).

# 6. Implementation Notes

## TF Lite & Detection

* If training a custom model is hard, use a general object detector to detect large rectangular advertisement-like objects as a placeholder. The model can detect bounding boxes; then use bounding box aspect ratio and image pixel area + optional geolocation rules to guess size/placement violations.
* For content checks, you can perform a separate content classifier or explicit text OCR + rule checks for explicit words.
* Keep a fallback: if model confidence < threshold, prompt user to verify manually.

## Offline-first strategy

* Save captured reports as local JSON records (fields: local\_id, timestamp, lat, lng, ai\_suggestion, user\_choice, image\_path, synced=false)
* Background service or on-app-open syncs unsynced records when internet is available.

## Firebase schema (suggested)

* reports collection
  + report\_id (auto)
  + timestamp
  + lat, lng
  + ai\_suggestion (array)
  + user\_selection (array)
  + image\_url (Storage path)
  + synced\_by\_user\_id (optional)
  + synced (bool)
* authorized\_billboards collection (mock)
  + billboard\_id
  + lat, lng, width\_m, height\_m, city, operator\_id

# 7. How to use this doc with GitHub Copilot / Next steps

1. **Create the repository** with the folder structure above.
2. **Add this SRS\_and\_QA.md** in docs/ (already present once you push this file).
3. **Open the project in your IDE** (VS Code + Flutter extension). Use GitHub Copilot to scaffold Flutter screens and basic services.
4. **Start with camera + local saving** flow first. Then integrate TF Lite inference and finally Firebase sync + dashboard.
5. **Testing**: Create a few mock authorized billboard entries for two cities to test DB cross-check.

## Contact / Notes

* Keep the model and data ethical: do not perform face recognition or continuous public surveillance. Always show a privacy disclaimer before capturing images.
* If you want, I can also generate:
  + README.md for the repo,
  + pubspec.yaml starter snippet,
  + Example Flutter screen templates (camera\_screen.dart, review\_screen.dart),
  + Sample Firestore rules and mock DB JSON.

*End of document.*