# AdGuard-360: Technical Documentation

### **1. Executive Summary**

AdGuard-360 is a smart, mobility-based solution designed to tackle the pervasive issue of unauthorized billboards and urban visual clutter. By empowering citizens to report non-compliant advertisements through a user-friendly mobile application, we provide municipal authorities with a centralized, data-driven platform to enforce regulations efficiently. The system leverages artificial intelligence for automated violation detection, a real-time dashboard for administrative oversight, and gamification to foster sustained citizen engagement, creating a collaborative ecosystem for cleaner, more organized urban environments.

### **2. System Architecture**

AdGuard-360 is built on a scalable, service-oriented architecture designed for real-time data processing and high availability. The system is decoupled into distinct components, allowing for independent development, deployment, and scaling.

**Core Components:**

* **Mobile Client (React Native):** A cross-platform mobile application that serves as the primary interface for citizens. It's designed for ease of use, enabling users to quickly capture images of billboards, which are then automatically tagged with GPS coordinates.
* **Web Dashboard (Next.js / React):** A comprehensive administrative portal for government authorities. It features an interactive map to visualize report locations, a detailed table for managing cases, and analytics to track progress and identify hotspots.
* **Backend API (Node.js & Express.js):** The central hub of the system. This RESTful API handles all business logic, including user authentication, data validation, report processing, and communication between the clients, database, and AI service.
* **AI/ML Service (Python):** A dedicated microservice for intelligent image analysis. It uses computer vision models to detect billboards within an image and Optical Character Recognition (OCR) to extract text, which aids in identifying the advertiser.
* **Databases:**
  + **MongoDB:** The primary NoSQL database used for its flexible schema. It stores all persistent data, including user profiles, report details, image metadata, and billboard information.
  + **Redis:** An in-memory data store used for high-performance tasks such as caching frequently accessed data, managing user sessions, and powering the real-time leaderboard for gamification.

**Architectural Diagram:**



### **3. Data and Process Flow**

The system operates on a clear, event-driven data flow from initial reporting to final resolution.

#### **Flow 1: Citizen Reporting**

1. **Capture & Submit:** A citizen opens the AdGuard-360 mobile app, takes a photo of a potential illegal billboard, and hits submit. The app automatically bundles the image with the device's current GPS coordinates.
2. **API Ingestion:** The app sends a secure API request to the **Node.js Backend**.
3. **Initial Record:** The backend creates a new report document in **MongoDB** with a status of PENDING\_ANALYSIS and stores the image (e.g., in a cloud storage bucket, linking the URL in the document).
4. **AI Processing:** The backend pushes a job to a queue, signaling the **Python AI/ML Service** to process the new image.
5. **Analysis & Update:** The AI service fetches the image, performs billboard detection and OCR, and sends the results (e.g., { is\_billboard: true, confidence: 0.95, text: "..." }) back to the backend.
6. **Finalize Report:** The backend updates the report in MongoDB with the AI analysis and changes the status to PENDING\_REVIEW.

#### **Flow 2: Authority Review and Action**

1. **Dashboard View:** An official logs into the **Web Dashboard**. The dashboard fetches all reports with the PENDING\_REVIEW status from the backend.
2. **Case Management:** Reports are displayed on a map and in a sortable list. The official can click on a report to view the image, location, AI-extracted text, and the reporting citizen's details.
3. **Decision & Update:** The official verifies the report and updates its status (e.g., ACTION\_TAKEN, RESOLVED, INVALID). This action is a PUT request to the backend API, which updates the report in **MongoDB**.
4. **Citizen Notification:** Upon status change (especially to RESOLVED), the backend triggers a push notification to the original reporting citizen.
5. **Gamification:** If the report is valid, the backend updates the citizen's score in the **Redis** leaderboard, promoting continued engagement.

### **4. Technology Stack**

| **Component** | **Technology/Framework** | **Purpose** |
| --- | --- | --- |
| **Mobile App** | React Native | Cross-platform development (iOS/Android) |
| **Web Dashboard** | Next.js, React, Tailwind CSS | Modern UI, server-side rendering, responsive design |
| **Backend** | Node.js, Express.js | API development, handling business logic |
| **AI/ML** | Python, TensorFlow/PyTorch, OpenCV | Computer vision, OCR |
| **Database** | MongoDB, Mongoose | Primary data storage, flexible schema |
| **Cache** | Redis | Caching, session management, leaderboards |
| **Deployment** | Docker, Nginx | Containerization, reverse proxy, load balancing |