SmartCity AI System

Complete Project Documentation

Threat & Weapon Detection · Fight Detection · Abandoned Bag · Crowd · ANPR

AI Pipeline · Real-time Analytics Dashboard

YOLOv8 · EasyOCR · Flask · NestJS · Vue 3 · MySQL

Generated: 2026-05-08

1. Project Overview

The **SmartCity AI System** is an end-to-end intelligent video surveillance platform that ingests live or recorded camera feeds and automatically detects threats, fights, abandoned objects, suspicious loitering, crowd density, and unregistered vehicles in real time. It pairs a YOLOv8-based computer-vision pipeline with rule-based behavioural heuristics and EasyOCR for plate recognition, surfacing every event on a live operations dashboard.

The system is split into three independently deployable tiers — a Python Flask AI microservice, a NestJS gateway/aggregator, and a Vue 3 single-page dashboard — communicating over HTTP with MJPEG video streaming and Server-Sent Events for low-latency telemetry.

Key Capabilities

- **Threat & Weapon Detection** — guns, knives, explosives, grenades.
- **Fight / Violence Detection** — distance-oscillation heuristic over tracked persons.
- **Abandoned Bag Detection** — bags persisting without an owner nearby.
- **Suspicious Loitering** — same person occupying a region across many frames.
- **Crowd Density Detection** — count-based threshold alert.
- **ANPR** — vehicle detection + plate OCR + MySQL registry lookup.
- **Real-time MJPEG streaming** with annotated bounding boxes.
- **Server-Sent Events (SSE)** for live stats and alerts with snapshots.
- **Live dashboard** — totals, vehicle breakdown, alerts feed, charts.

Project Information

Project Name	SmartCity AI System
Workspace Root	/Users/hikmatullah/Documents/SmartCity
Modules	safe-city-ai (Python AI), backend (NestJS), frontend (Vue 3)
AI Frameworks	Ultralytics YOLOv8, EasyOCR, OpenCV, PyTorch
Backend Frameworks	Flask 3.1, NestJS 11
Frontend	Vue 3, Vite, TailwindCSS 4, Chart.js, vue-router
Database	MySQL (PyMySQL driver)
Default Ports	Flask AI: 5000 · NestJS: 3000 · Vite Dev: 5173

2. AI Models

The system relies on YOLOv8 for real-time object detection and EasyOCR for reading text, supplemented by smart algorithmic tracking and rule-based behaviour analysis.

2.1 Threat & Weapon Detection Model

- **Model:** YOLOv8n — *Subh775/Threat-Detection-YOLOv8n* (Hugging Face)
- **Weights file:** safe-city-ai/models/threat_detection.pt (~6 MB)
- **Role:** Scans frames to detect dangerous objects instantaneously.
- **Detects:** Guns, Knives, Explosives, and Grenades.
- **Cadence:** Runs every Nth frame (default every 5) — set via `WEAPON_EVERY_N_FRAMES`.
- **Confidence threshold:** 0.45 default.

2.2 Fight & Violence Detection

- **Model:** YOLOv8n (person tracking) + custom motion algorithm.
- **Role:** Tracks people across frames; identifies fights using a distance-oscillation heuristic.
- **Core insight:** Pedestrians crossing paths show a single drop-then-rise in distance; fighters show repeated oscillation (push apart / pull together).
- **Guards:** both must be in contact range, both must be moving, must persist for N consecutive frames.
- **Tunables:** FIGHT_CLOSE_PX=60, FIGHT_MIN_SPEED=8, FIGHT_MIN_OSCILLATIONS=3, FIGHT_OSCILLATION_WINDOW=16, FIGHT_CONFIRM_FRAMES=8.

2.3 Abnormal Behaviour & Abandoned Bags

- **Model:** YOLOv8n (person/bag tracking) + temporal tracker (rules).
- **Crowd Density:** alert when person count > 10 in a frame.
- **Abandoned Bag:** backpack/handbag detected with no person within 120 px → flagged red as ABANDONED.
- **Loitering:** when ≥ 75% of the last 150 frames show a person in the same 60×60-px region.

2.4 Automatic Number Plate Recognition (ANPR)

- **Vehicle model:** YOLOv8n COCO pre-trained — detects cars, motorcycles, buses, trucks (class IDs 2/3/5/7).
- **OCR:** EasyOCR (English) on the lower 60% ROI of each vehicle box.
- **Plate validation:** regex/heuristic rejects brand words (TOYOTA, HONDA, …); requires mix of letters and digits, 3–12 chars.
- **Stabilisation:** PlateVoteCache confirms a reading once it appears in ≥ 2 of the last 5 frames for the same vehicle slot.
- **Slot tracking:** IoU ≥ 0.30 between frames keeps the slot identity; a confirmed plate is cached for 15 frames before a re-scan.
- **DB lookup:** cleaned plate matched against `registered_vehicles` in MySQL.

- **Alert cooldown:** 10 s per unregistered plate to avoid spam.

3. System Flow & Notification Pipeline

1. Camera Captures. The camera (or uploaded video) streams to the Python AI service.

2. Pipeline Execution. Each frame is simultaneously checked for:

- **Weapons** — YOLOv8 threat model detects guns/knives/explosives.
- **Fights** — person positions tracked for erratic, high-speed proximity.
- **Abandoned bags** — bags tracked to see if their owner leaves them.
- **Crowd density** — count-based alert when threshold exceeded.
- **Loitering** — same person in a region across many frames.
- **Vehicles & Plates** — YOLO finds vehicles → EasyOCR reads plates.

3. Database Lookup. Plate text is cleaned and looked up in the MySQL `registered_vehicles` table.

4. Alerts & Visualisation. The system dynamically draws bounding boxes (red for threats / fights / unregistered plates, green for people, cyan for vehicles, amber for bags) and pushes events to the Vue dashboard via MJPEG + Server-Sent Events.

Flow Diagram (from project brief)

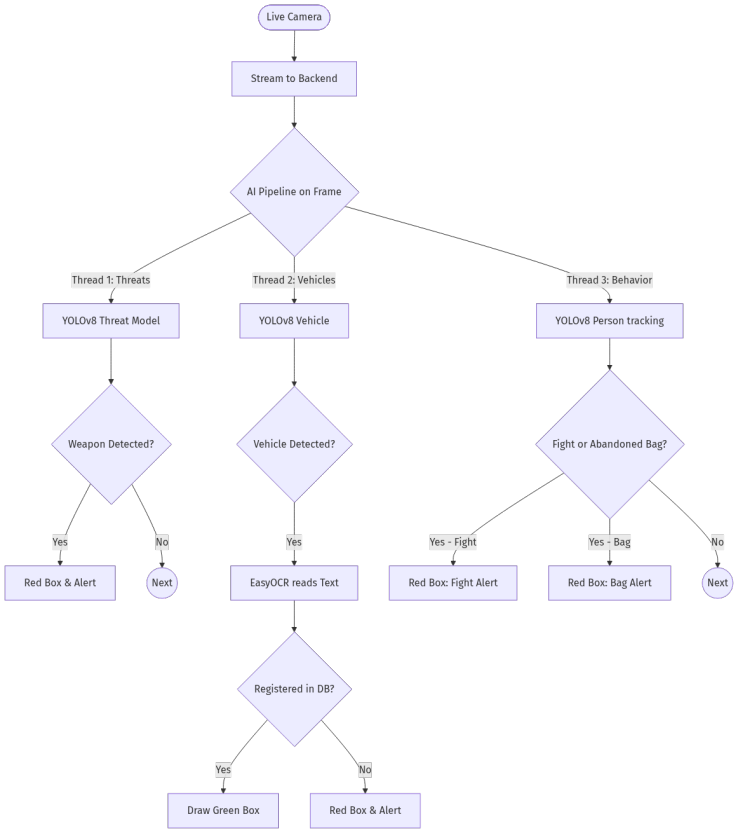
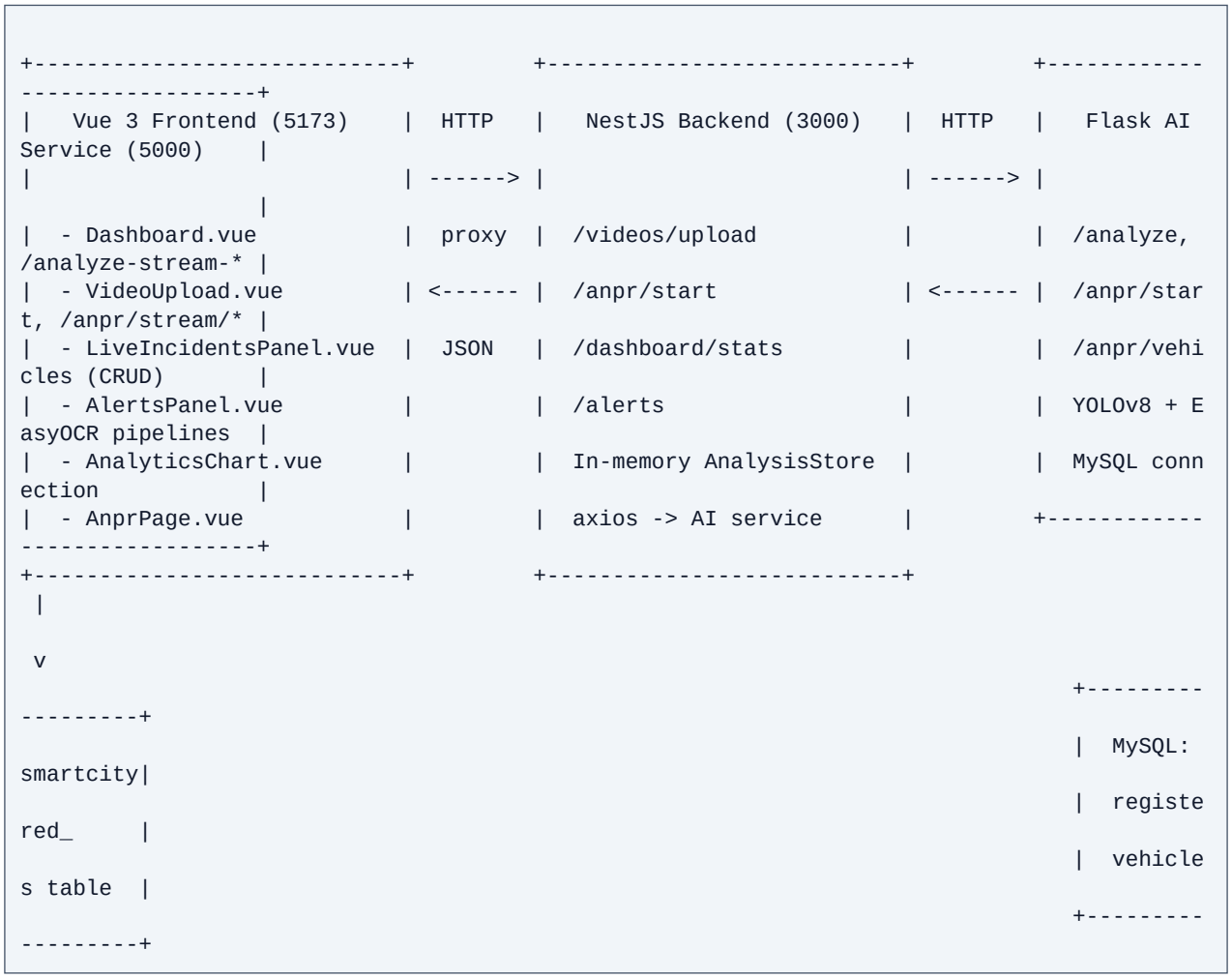


Figure 1 — End-to-end pipeline: camera → AI models → alerts dashboard.

4. System Architecture

The application is split into three tiers, each in its own subdirectory, communicating over HTTP/SSE/MJPEG:



Tier 1 — Python AI Service (Flask)

Runs all heavy ML: YOLOv8 (general + threat-specific), EasyOCR, OpenCV video I/O. Streams annotated MJPEG frames, pushes alert + stats events via SSE, and persists processed videos to `outputs/`. Owns the MySQL connection used by ANPR.

Tier 2 — NestJS Backend Gateway

Acts as the public API for the dashboard. Accepts uploads, forwards them to the AI service, aggregates per-video analysis into an in-memory `AnalysisStore`, and exposes `/dashboard/*` + `/alerts` endpoints.

Tier 3 — Vue 3 Frontend (Vite)

Single-page dashboard. Uses Vite proxy to mount NestJS at `/api` and the AI service at `/ai-video`. Displays live MJPEG, stats counters, alert feed with snapshot thumbnails, and analytics chart. Includes an ANPR page with vehicle CRUD.

5. Directory Layout

```
SmartCity/
■■■ SmartCity_Project_Details.docx       Project brief (source of this PDF)
■■■ requirements.txt                     Workspace-level Python deps
■■■ .gitignore
■■■ safe-city-ai/                        ■■ Tier 1: Python AI service ■■
■   ■■■ app.py                           Flask routes (analysis + ANPR)
■   ■■■ detect.py                        YOLOv8 video pipeline (batch + stream)
■   ■■■ anpr.py                          ANPR pipeline + DB layer + vote cache
■   ■■■ abnormal_detection.py            Crowd / abandoned-bag / loitering rules
■   ■■■ fight_detection.py               Distance-oscillation fight detector
■   ■■■ weapon_detection.py              Threat-model (gun/knife/grenade) wrapper
■   ■■■ requirements.txt                 AI service Python deps
■   ■■■ .env / .env.example              DB + Flask env config
■   ■■■ sql/init.sql                     MySQL schema
■   ■■■ models/threat_detection.pt       Weapon YOLO weights (~6 MB)
■   ■■■ weights/                         License-plate YOLO weights
■   ■   ■■■ LP-detection.pt
■   ■   ■■■ yolov8n-license-plate.pt
■   ■■■ yolov8n.pt / yolov8m.pt          COCO pre-trained YOLO
■   ■■■ outputs/                         Processed annotated videos
■   ■■■ temp/                            Upload staging area
■■■ backend/                             ■■ Tier 2: NestJS gateway ■■
■   ■■■ package.json                     NestJS 11 + axios + form-data
■   ■■■ nest-cli.json
■   ■■■ tsconfig.json / tsconfig.build.json
■   ■■■ src/
■   ■   ■■■ main.ts                      Bootstrap, CORS, uploads dir
■   ■   ■■■ app.module.ts                Wires all modules
■   ■   ■■■ app.controller.ts / app.service.ts
■   ■   ■■■ analysis/analysis.store.ts   In-memory aggregate stats + alerts
■   ■   ■■■ dashboard/                   /dashboard/stats|analytics|vehicles
■   ■   ■■■ alerts/                      /alerts
■   ■   ■■■ videos/                      /videos/upload + stream-result
■   ■   ■■■ anpr/                        /anpr/start (proxy)
■   ■■■ uploads/                         Multer disk storage for uploads
■   ■■■ dist/                            tsc build output
■■■ frontend/                            ■■ Tier 3: Vue 3 dashboard ■■
    ■■■ package.json                     Vue 3 + Vite + Tailwind 4 + Chart.js
    ■■■ vite.config.js                   Proxies /api → 3000, /ai-video → 5000
    ■■■ index.html
    ■■■ src/
        ■■■ main.js / App.vue / style.css
        ■■■ router/index.js              Routes: / · /vehicles · /anpr
        ■■■ services/api.js              All HTTP client calls
        ■■■ pages/
        ■   ■■■ Dashboard.vue            Main dashboard (stats + live feed)
        ■   ■■■ VehiclesDetail.vue       Vehicle breakdown drill-down
        ■   ■■■ AnprPage.vue             ANPR live + registered-vehicle CRUD
        ■■■ components/
        ■   ■■■ VideoUpload.vue          MJPEG viewer + SSE alerts panel
        ■   ■■■ LiveIncidentsPanel.vue   Real-time alert cards w/ snapshots
        ■   ■■■ AlertsPanel.vue          Alert list (history)
        ■   ■■■ AnalyticsChart.vue       Chart.js line chart
        ■   ■■■ StatCard.vue             KPI cards
        ■■■ assets/
```

6. Python AI Service (Flask)

Single Flask app (`app.py`) exposing two pipelines: general video analysis and ANPR. Each upload spawns a background thread; the client receives a session ID and connects to MJPEG + SSE endpoints to receive frames, stats, and alerts in real time.

6.1 Endpoints — Video Analysis

Method	Path	Description
GET	`/health`	Liveness probe.
GET	`/outputs/<file>`	Serve processed MP4 with HTTP range support.
POST	`/analyze`	Batch — process whole video, return final stats.
POST	`/analyze-stream-start`	Start streaming session, returns sessionId + streamUrl.
GET	`/stream/<sid>`	MJPEG stream of annotated frames (multipart/x-mixed-replace).
GET	`/stream/<sid>/stats/latest`	Poll latest stats: people, vehicles breakdown, alerts, frames.
GET	`/stream/<sid>/stats`	SSE — high-frequency stats updates.
GET	`/stream/<sid>/alerts`	SSE — alert events with base64 JPEG snapshots (interleaved with stats
GET	`/stream/<sid>/result`	Block until processing completes; return final stats.

6.2 Endpoints — ANPR

Method	Path	Description
POST	`/anpr/start`	Start ANPR session for uploaded video → sessionId + streamUrl.
GET	`/anpr/stream/<sid>`	MJPEG stream of annotated frames with plate boxes.
GET	`/anpr/stream/<sid>/stats/latest`	Latest counts: frames, detected, unregistered, unique.
GET	`/anpr/stream/<sid>/alerts`	SSE — unregistered-plate alerts with base64 plate-crop snapshots.
GET	`/anpr/stream/<sid>/result`	Final ANPR stats.
GET	`/anpr/vehicles`	List all registered vehicles.
POST	`/anpr/vehicles`	Register a vehicle (JSON: plate, owner, type).
DELETE	`/anpr/vehicles/<plate>`	Remove a registered vehicle.

6.3 Detection Pipeline (detect.py)

- Uses `yolov8n.pt` for streaming (≈3–4× faster than yolov8m).
- Class IDs from COCO: Person=0, Car=2, Motorcycle=3, Bus=5, Truck=7, Backpack=24, Handbag=26.
- Colour scheme — Person: green; Vehicle: cyan; Bag: amber; Alert/Fight/Weapon/Abandoned: red.
- **Frame skipping:** YOLO runs every `DETECT_EVERY_N_FRAMES` (default 2); skipped frames reuse the last annotation.
- **Inference resize:** long edge `YOLO_INFER_SIZE` (default 640).
- **JPEG quality:** `STREAM_JPEG_QUALITY` (default 60).
- Maintains a per-frame history (cap 150) for loitering analysis.

- Drops MJPEG frames if the queue exceeds 30 to prevent unbounded buffering.

6.4 ANPR Pipeline (anpr.py)

Per-frame stages:

- **Stage 1:** YOLO detects vehicle bounding boxes; filter by COCO IDs {2,3,5,7} and confidence ≥ 0.35.
- **Stage 2:** Build ROI = lower 60% of vehicle box (where plates live).
- **Stage 3:** EasyOCR reads the ROI; allowlist A–Z/0–9 only; characters scaled up so they're ≥ 32 px tall.
- **Stage 4:** looks_like_plate() validates length 3–12, requires both letters and digits, rejects brand-name false positives.
- **Stage 5:** PlateVoteCache requires the same reading (cosine ≥ 0.75) in ≥ 2 of the last 5 frames per vehicle slot.
- **Stage 6:** MySQL lookup → green box if registered, red box + alert if not, cyan if DB unavailable.
- **Stage 7:** Confirmed plate cached for 15 frames before re-scanning (saves OCR cost while the same car remains in view).
- **Slot tracking:** IoU ≥ 0.30 between current vehicle box and stored slot box keeps cached plate; below threshold, a fresh scan starts.
- **Alert cooldown:** 10 s per plate before another alert fires.

7. Detection Algorithm Details

7.1 Fight Detection (fight_detection.py)

Distance-oscillation heuristic. Two persons are flagged when:

- Edge-to-edge gap ≤ `FIGHT_CLOSE_PX` (60 px) — they're in contact range.
- Both have average centroid speed ≥ `FIGHT_MIN_SPEED` (8 px/frame) — neither is loitering.
- Their pairwise distance has ≥ `FIGHT_MIN_OSCILLATIONS` (3) direction reversals over the last `FIGHT_OSCILLATION_WINDOW` (16) frames.
- Pattern persists for ≥ `FIGHT_CONFIRM_FRAMES` (8) consecutive detect-frames before the alert fires.

Person tracking: greedy nearest-neighbour assignment with max distance 80 px between consecutive frames. Each track stores up to 30 historical centroids in a deque; pair distance histories are pruned when no longer active.

7.2 Abnormal Behaviour (abnormal_detection.py)

- **Crowd density:** alert if frame_people > `crowd_threshold` (default 10).
- **Abandoned bag:** for each bag, compute Euclidean distance from its centre to every person centre; if all are > 120 px, mark as abandoned.
- **Loitering:** over the last 150 frames, quantise person centres into 60×60 pixel grid cells and count occurrences per cell. If any cell sees ≥ 75 % of frames, fire 'Suspicious Loitering Behavior'.

7.3 Weapon Detection (weapon_detection.py)

- Lazy-loaded singleton holding the YOLOv8n threat model.
- Path: `safe-city-ai/models/threat_detection.pt` (override via `WEAPON_MODEL_PATH`).
- Runs every `WEAPON_EVERY_N_FRAMES` = 5 frames to amortise the cost.
- Confidence threshold 0.45.
- Maps class names to alert strings — e.g. 'gun' → 'Weapon: Gun Detected'.
- Returns both alert strings and red bounding boxes for visual overlay.

8. NestJS Backend

NestJS 11 application. Bootstrapped in `src/main.ts`: enables CORS for the Vite dev server (`http://localhost:5173`), creates `uploads/`, and listens on `$PORT` (default 3000).

8.1 Modules

- **AnalysisModule** — global `AnalysisStore` service (in-memory: stats, alerts, time-series).
- **VideosModule** — POST /videos/upload (multer disk storage, 500 MB cap), POST /videos/stream-result (frontend submits final stream stats).
- **AnprModule** — POST /anpr/start (proxies upload to Flask AI).
- **DashboardModule** — GET /dashboard/stats, /dashboard/analytics, /dashboard/vehicles.
- **AlertsModule** — GET /alerts (returns AnalysisStore.getAlerts()).

8.2 Endpoints

Method	Path	Description
POST	`/videos/upload`	Upload a video; forwards to Flask /analyze-stream-start by default.
POST	`/videos/stream-result`	Frontend submits final stream stats → recorded in AnalysisStore.
POST	`/anpr/start`	Upload a video for ANPR; forwarded to Flask /anpr/start.
GET	`/dashboard/stats`	Aggregate counts: videos processed, people, vehicles, alerts.
GET	`/dashboard/analytics`	Time-series labels + people/vehicles arrays for the chart.
GET	`/dashboard/vehicles`	Vehicle breakdown: cars, motorcycles, buses, trucks.
GET	`/alerts`	Most recent stored alerts.

8.3 Configuration

AI_SERVICE_URL	Flask AI service base (default http://localhost:5000)
AI_STREAMING	Set to 'false' to use synchronous /analyze instead of stream-start.
AI_OUTPUTS_PATH	Path where Flask writes processed videos (default ../safe-city-ai/outputs).
PORT	NestJS HTTP port (default 3000).

AnalysisStore behaviour: increments aggregate counters on every `addAnalysis`; classifies alert messages into weapon / crowd_density / abandoned_bag / loitering / fight_detected / incident; stores most-recent first; provides time-series labels by createdAt timestamp.

9. Vue 3 Frontend

Vite + Vue 3 SPA styled with TailwindCSS 4. Routes:

- `/` → **Dashboard.vue** — KPI cards, live MJPEG viewer, alerts panel, analytics chart.
- `/vehicles` → **VehiclesDetail.vue** — vehicle breakdown by type.
- `/anpr` → **AnprPage.vue** — live ANPR feed, unregistered alerts, registered-vehicle CRUD form.

9.1 Components

- **VideoUpload.vue** — file picker, MJPEG <img> bound to /ai-video/stream/<sid>, per-second polling of stats/latest.
- **LiveIncidentsPanel.vue** — opens an EventSource to /ai-video/stream/<sid>/alerts; renders alert cards with snapshots.
- **AlertsPanel.vue** — pulls /api/alerts (history view).
- **AnalyticsChart.vue** — Chart.js line chart of people vs vehicles over time.
- **StatCard.vue** — KPI card primitive used across pages.

9.2 API Client (services/api.js)

```
getDashboardStats()        GET  /api/dashboard/stats
getDashboardAnalytics()    GET  /api/dashboard/analytics
getVehicleBreakdown()      GET  /api/dashboard/vehicles
getAlerts()                GET  /api/alerts
uploadVideo(file)          POST /api/videos/upload
submitStreamResult(r)      POST /api/videos/stream-result
fetchStreamResult(sid)     GET  /ai-video/stream/<sid>/result
fetchLatestStats(sid)      GET  /ai-video/stream/<sid>/stats/latest
anprStartSession(file)     POST /api/anpr/start
anprFetchStats(sid)        GET  /ai-video/anpr/stream/<sid>/stats/latest
anprListVehicles()         GET  /ai-video/anpr/vehicles
anprAddVehicle(p,o,t)      POST /ai-video/anpr/vehicles
anprDeleteVehicle(plate)   DELETE /ai-video/anpr/vehicles/<plate>
```

9.3 Vite Proxy

Configured in `vite.config.js` — both proxies strip their prefix and forward unbuffered (timeout: 0 for `/ai-video` so MJPEG isn't cut off):

```
server: {
  proxy: {
    '/api':     { target: 'http://localhost:3000', rewrite: '' },
    '/ai-video':{ target: 'http://localhost:5000',
                  rewrite: '', timeout: 0 },
  },
}
```

10. Database (MySQL)

A single MySQL schema named `smartcity` with one table backing the ANPR registry. The Flask service `init_db()` creates the table at startup if it doesn't exist; the same schema can be bootstrapped manually:

```
mysql -u root -p < safe-city-ai/sql/init.sql
```

Schema

```
CREATE DATABASE IF NOT EXISTS smartcity
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;

USE smartcity;

CREATE TABLE IF NOT EXISTS registered_vehicles (
    id            INT AUTO_INCREMENT PRIMARY KEY,
    plate_number  VARCHAR(20)  NOT NULL UNIQUE,
    owner_name    VARCHAR(100),
    vehicle_type  VARCHAR(50),
    registered_at TIMESTAMP    DEFAULT CURRENT_TIMESTAMP
);
```

Connection Settings

DB_HOST	localhost
DB_PORT	3306
DB_USER	root
DB_PASSWORD	(set in safe-city-ai/.env)
DB_NAME	smartcity
Driver	PyMySQL 1.1.2
Charset	utf8mb4
Cursor	DictCursor
Connect timeout	5 s

Programmatic Access

bullet `init_db()` — create the table (idempotent).

bullet `is_plate_registered(plate)` → (record_dict | None, bool).

bullet `get_all_vehicles()` — list registry, newest first.

bullet `add_vehicle(plate, owner, type)` → (ok: bool, msg).

bullet `remove_vehicle(plate)` → bool.

bullet `clean_plate(text)` — normalises (uppercase, strips non-alphanumeric).

11. Configuration & Environment Variables

safe-city-ai/.env (Flask AI service)

```
# ■■ MySQL Database ■■■■■■■■■■■■■■■■■■■■■■■■■■■■
DB_HOST=localhost
DB_PORT=3306
DB_USER=root
DB_PASSWORD=your_password_here
DB_NAME=smartcity

# ■■ AI service ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
FLASK_PORT=5000
```

Detection-tuning environment variables

YOLO_MODEL	Default model file for batch /analyze (default yolov8n.pt).
YOLO_INFER_SIZE	Long-edge resize before YOLO inference (default 640).
DETECT_EVERY_N_FRAMES	Run YOLO every Nth frame for streaming (default 2).
WEAPON_EVERY_N_FRAMES	Run threat model every Nth frame (default 5).
WEAPON_MODEL_PATH	Override path to threat_detection.pt.
STREAM_JPEG_QUALITY	MJPEG JPEG quality (default 60).
FIGHT_CLOSE_PX	Edge-gap for 'contact' (default 60).
FIGHT_MIN_SPEED	Minimum avg speed per person (default 8 px/frame).
FIGHT_MIN_OSCILLATIONS	Reversals required (default 3).
FIGHT_OSCILLATION_WINDOW	Frames in oscillation window (default 16).
FIGHT_CONFIRM_FRAMES	Consecutive frames to confirm (default 8).
FIGHT_HISTORY_LEN	Track history length (default 30).

NestJS backend env

AI_SERVICE_URL	Flask AI base URL (default http://localhost:5000).
AI_STREAMING	'false' to use synchronous /analyze.
AI_OUTPUTS_PATH	Local path to processed videos.
PORT	HTTP port (default 3000).

Frontend env (Vite)

VITE_API_URL	Override NestJS base URL (defaults to /api in dev).
VITE_AI_VIDEO_URL	Override AI service base URL (defaults to /ai-video in dev).

12. Setup & Run Instructions

Prerequisites

- Python 3.10+ with venv
- Node.js 18+ and npm
- MySQL 8 server (local or remote)
- macOS / Linux (paths assume macOS in this workspace)

12.1 Database initialisation

```
mysql -u root -p < safe-city-ai/sql/init.sql
```

12.2 Python AI service

```
cd safe-city-ai
cp .env.example .env          # set DB_PASSWORD etc.
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
python app.py                 # listens on :5000
```

12.3 NestJS backend

```
cd backend
npm install
npm run start:dev             # listens on :3000
```

12.4 Vue frontend

```
cd frontend
npm install
npm run dev                   # listens on :5173 (proxies /api and /ai-video)
```

12.5 Production build (frontend)

```
cd frontend
npm run build
npm run preview
```

12.6 Verify the stack

- GET `http://localhost:5000/health` → {"status":"ok"}
- Open `http://localhost:5173`, upload a short MP4, watch annotated frames stream live.
- Open the ANPR page, register a plate, then upload a video containing it — registered plates render green, unregistered fire red alerts.

13. Dependencies

13.1 Python — safe-city-ai/requirements.txt

```
# Web framework
Flask==3.1.3
flask-cors==6.0.2
Werkzeug==3.1.6
python-dotenv==1.2.2

# Database (MySQL)
PyMySQL==1.1.2

# Computer Vision
opencv-python==4.11.0.86
opencv-contrib-python==4.10.0.84
numpy==2.4.3
pillow==12.1.1

# Deep Learning
torch==2.10.0
torchvision==0.25.0
ultralytics==8.4.21

# OCR
easyocr==1.7.2
pytesseract==0.3.13

# Utilities
huggingface-hub==0.24.7
requests==2.32.5
tqdm==4.67.3
PyYAML==6.0.2
```

13.2 NestJS — backend/package.json

```
dependencies:
  @nestjs/common         ^11.0.1
  @nestjs/core           ^11.0.1
  @nestjs/platform-express ^11.0.1
  @nestjs/serve-static   ^5.0.4
  axios                  ^1.13.6
  form-data              ^4.0.5
  reflect-metadata       ^0.2.2
  rxjs                   ^7.8.1

devDependencies:
  @nestjs/cli ^11  ·  typescript ^5.7  ·  jest ^30
  prettier ^3.4   ·  eslint ^9      ·  ts-jest ^29
  supertest ^7    ·  ts-node ^10    ·  ts-loader ^9
```

13.3 Frontend — frontend/package.json

```
dependencies:
  vue                    ^3.5.25
  vue-router             ^4.6.4
  axios                  ^1.13.6
  chart.js               ^4.5.1
  vue-chartjs            ^5.3.3
  tailwindcss            ^4.2.1
  @tailwindcss/vite      ^4.2.1
  postcss / autoprefixer
```

```
devDependencies:
  vite                    ^7.3.1
  @vitejs/plugin-vue      ^6.0.2
```

14. Alert Taxonomy

Every alert produced by the Python pipeline is a string; the NestJS `AnalysisStore` classifies it into a type for the dashboard. The mapping below covers every alert string emitted today:

Alert String	Source	Mapped Type
Crowd Density Detected	abnormal_detection.py	crowd_density
Possible Abandoned Bag	abnormal_detection.py	abandoned_bag
Suspicious Loitering Behavior	abnormal_detection.py	loitering
Fight Detected	fight_detection.py / detect.py	fight_detected
Weapon: Gun Detected	weapon_detection.py	weapon
Weapon: Knife Detected	weapon_detection.py	weapon
Weapon: Explosion Detected	weapon_detection.py	weapon
Weapon: Explosive Detected	weapon_detection.py	weapon
Weapon: Grenade Detected	weapon_detection.py	weapon
UNREGISTERED plate label	anpr.py (alert_queue)	(separate ANPR feed)

Visual Encoding (annotated frames)

Green box	Person (normal); registered plate.
Cyan box	Vehicle (car / motorcycle / bus / truck).
Amber box	Bag (backpack / handbag) — normal.
Red box	Threat: fight, weapon, abandoned bag, unregistered plate.
Yellow box	Plate-text candidate during voting (pre-confirmation).

15. Sequence — Live Video Analysis

```
User selects video in Dashboard.vue
         |
         v
[Frontend]   POST /api/videos/upload  (multipart/form-data)
         |
         v
[NestJS]     VideosService.processUpload()
             multer saves to backend/uploads/
             axios POST /analyze-stream-start to Flask AI
         |
         v
[Flask]      analyze_stream_start():
               - generates session_id (uuid[:12])
               - saves upload to safe-city-ai/temp/
               - spawns Thread running analyze_video_stream()
               - returns {sessionId, streamUrl, processedVideo}
         |
         v
[NestJS]     wraps response: streamUrl -> /ai-video/stream/<sid>
         |
         v
[Frontend]   - <img src="/ai-video/stream/<sid>"> shows MJPEG
             - new EventSource("/ai-video/stream/<sid>/alerts")
             - setInterval poll /ai-video/stream/<sid>/stats/latest

[Flask thread] per frame:
   1. cv2.VideoCapture.read()
   2. YOLO every Nth frame -> reuse otherwise
   3. abnormal_detection.detect_abnormal_behaviors(...)
   4. weapon_detection.detect_weapons(...) every 5 frames
   5. fight_detection.detect_fight(person_boxes)
   6. cv2 draw boxes / labels
   7. cv2.VideoWriter.write(annotated)
   8. JPEG encode -> frame_queue (cap=30)
   9. push stats -> stats_queue
  10. push new alerts (with snapshot b64) -> alert_queue

[Flask]      on done -> queue.put(None)
             returns final result via /stream/<sid>/result

[Frontend]   POST /api/videos/stream-result with final stats
[NestJS]     AnalysisStore.addAnalysis() -> dashboard updates
```

16. Sequence — ANPR Live Run

```
User clicks "Start ANPR" on AnprPage.vue with video file
         |
         v
[Frontend]   POST /api/anpr/start (multipart)
         |
         v
[NestJS]     AnprService.startAnprSession(): proxy POST to Flask /anpr/start
         |
         v
[Flask]      anpr_start():
               - session_id, save upload, spawn Thread(process_anpr_stream)
               - returns {sessionId, streamUrl, processedVideo}
         |
         v
[Frontend]   - <img src="/ai-video/anpr/stream/<sid>">
             - poll /ai-video/anpr/stream/<sid>/stats/latest
             - EventSource /ai-video/anpr/stream/<sid>/alerts

[Flask thread] per frame (every 2nd frame):
   1. YOLOv8 vehicles (filter classes 2/3/5/7, conf >= 0.35)
   2. dedup overlapping boxes (IoU > 0.40)
   3. for each top-2 vehicle:
        - ROI = lower 60% of vehicle box
        - match to existing slot by IoU >= 0.30
        - if confirmed: redraw cached box+label, decrement cooldown (15)
        - else: scan_roi_for_plate(roi)
                 -> EasyOCR readtext (allowlist A-Z 0-9)
                 -> looks_like_plate filter
                 -> PlateVoteCache.vote (window 5, threshold 2)
        - if confirmed plate:
             is_plate_registered() -> MySQL SELECT
             registered  -> green box
             unknown DB  -> cyan box
             unregistered -> red box + push alert (cooldown 10s)
   4. update latest_stats (frames/detected/unregistered/unique)
   5. push stats event, push frame to MJPEG queue

[Flask]      on done -> persist final stats in _anpr_results
             /anpr/stream/<sid>/result returns final stats

Vehicle CRUD routes (separate, dashboard-driven):
   GET    /anpr/vehicles
   POST   /anpr/vehicles  {plate, owner, type}
   DELETE /anpr/vehicles/<plate>
```

17. Quick Reference

Default Ports

Flask AI (Python)	5000
NestJS Backend	3000
Vite Dev Server	5173
MySQL	3306

Key File Paths

Flask entry	safe-city-ai/app.py
YOLO pipeline	safe-city-ai/detect.py
ANPR pipeline	safe-city-ai/anpr.py
Behaviour rules	safe-city-ai/abnormal_detection.py
Fight detector	safe-city-ai/fight_detection.py
Weapon wrapper	safe-city-ai/weapon_detection.py
MySQL schema	safe-city-ai/sql/init.sql
NestJS entry	backend/src/main.ts
Module wiring	backend/src/app.module.ts
Stats store	backend/src/analysis/analysis.store.ts
Vue entry	frontend/src/main.js
Routes	frontend/src/router/index.js
API client	frontend/src/services/api.js
Dashboard page	frontend/src/pages/Dashboard.vue
ANPR page	frontend/src/pages/AnprPage.vue

Common Commands

```
# Start everything (3 terminals)
cd safe-city-ai && source .venv/bin/activate && python app.py
cd backend && npm run start:dev
cd frontend && npm run dev

# Health check
curl http://localhost:5000/health
curl http://localhost:3000/dashboard/stats

# Register a vehicle
curl -X POST http://localhost:5000/anpr/vehicles \
  -H 'Content-Type: application/json' \
  -d '{"plate":"ABC1234","owner":"John Doe","type":"Car"}'
```

```
# List registered vehicles
curl http://localhost:5000/anpr/vehicles

# Production build
cd frontend && npm run build
cd backend  && npm run build && npm run start:prod
```

End of document.