產品需求文件 (PRD) - 內部開發版

AI Hub - 智慧行為辨識監控系統

1. 產品概述

1.1 產品名稱

AI Hub - 智慧行為辨識監控系統 (Action Recognition Server Stack)

1.2 產品定位

一套以**影像串流處理為核心**的智慧視覺平台。系統提供完整的多路 RTSP/WebRTC/HLS 串流管理、轉碼、分發能力，並基於這個串流基礎設施，支援靈活擴展各種應用功能，包括 AI 物件偵測、虛擬圍欄告警、事件錄影、行為分析等。採用微服務架構與 Docker 容器化部署，讓開發者可以根據實際需求自由組合與擴展功能模組。

1.3 目標用戶

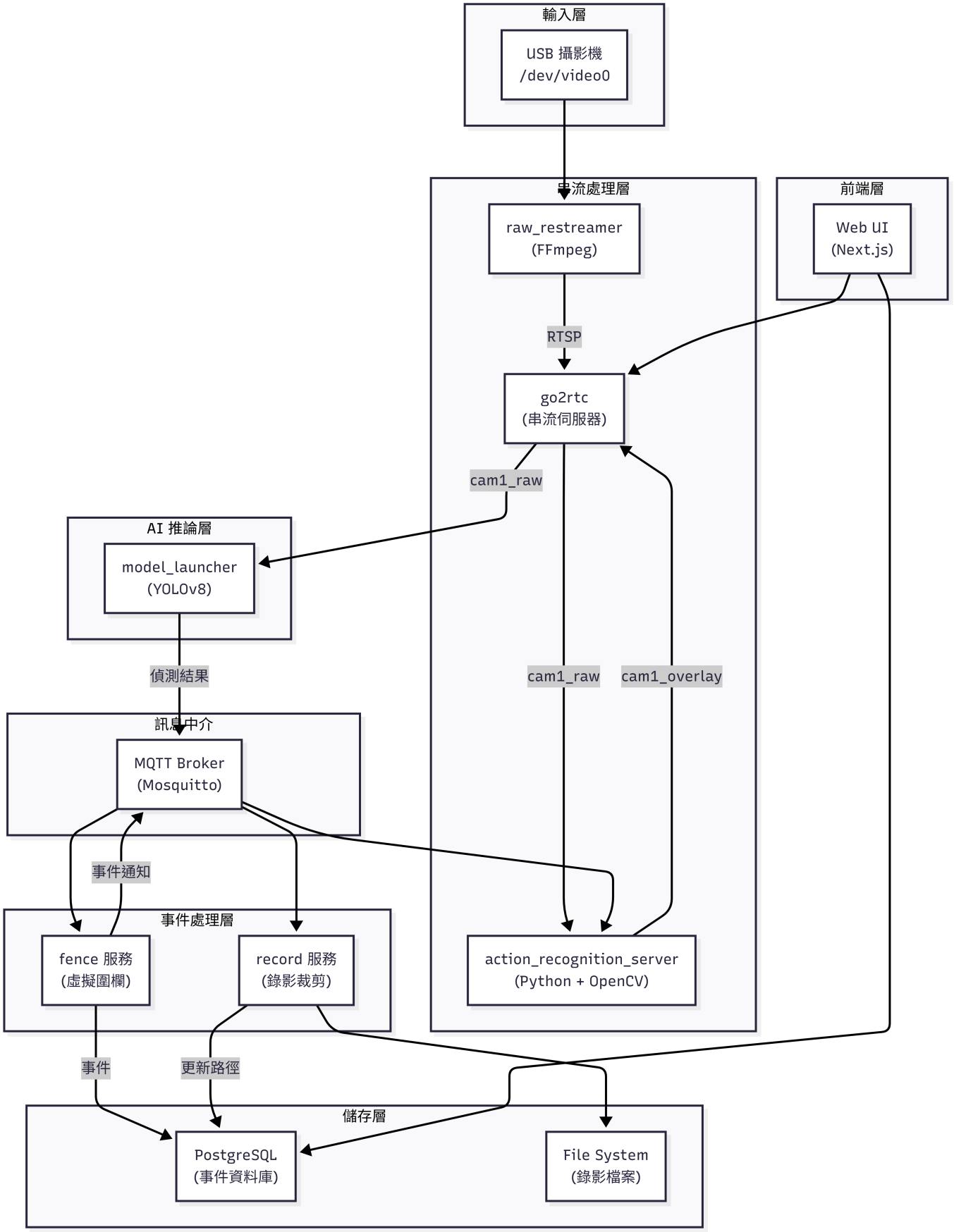
- 需要智慧監控解決方案的企業或機構
- 安全監控系統整合商
- 需要行為辨識與異常偵測的場域管理者
- 開發者與研究人員進行視覺辨識系統的原型開發

1.4 核心價值

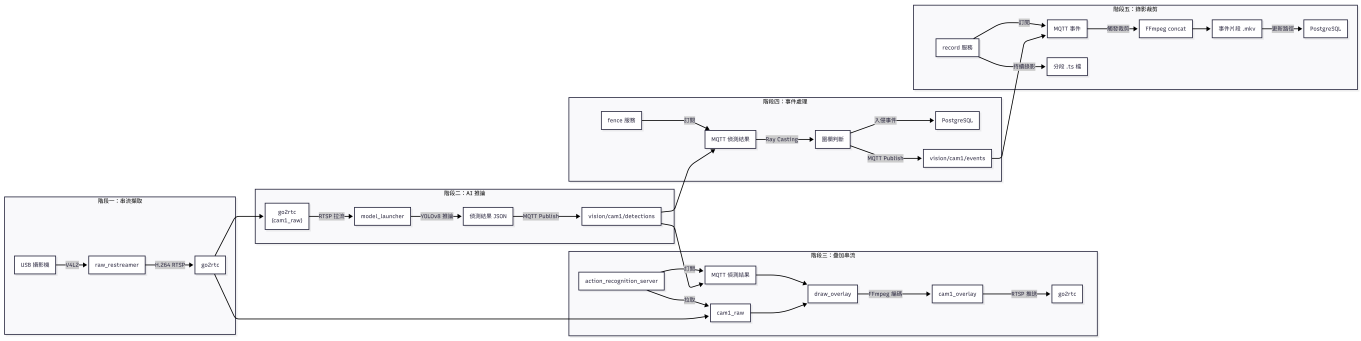
- **完整的串流基礎設施**：支援多協議串流輸入輸出 (RTSP/WebRTC/HLS)，提供低延遲轉碼與分發能力
- **功能模組化架構**：各服務獨立運行，可根據需求選擇性部署 AI、錄影、告警等功能
- **AI 能力可擴展**：支援熱插拔多種 AI 模型，從物件偵測到行為辨識皆可靈活配置
- **開發者友善**：基於 MQTT 事件驅動架構，易於整合自定義功能與第三方服務
- **硬體加速支援：**Intel VAAPI 硬體編解碼，降低 CPU 負載並提升串流處理效能

2. 系統架構

2.1 整體系統架構圖



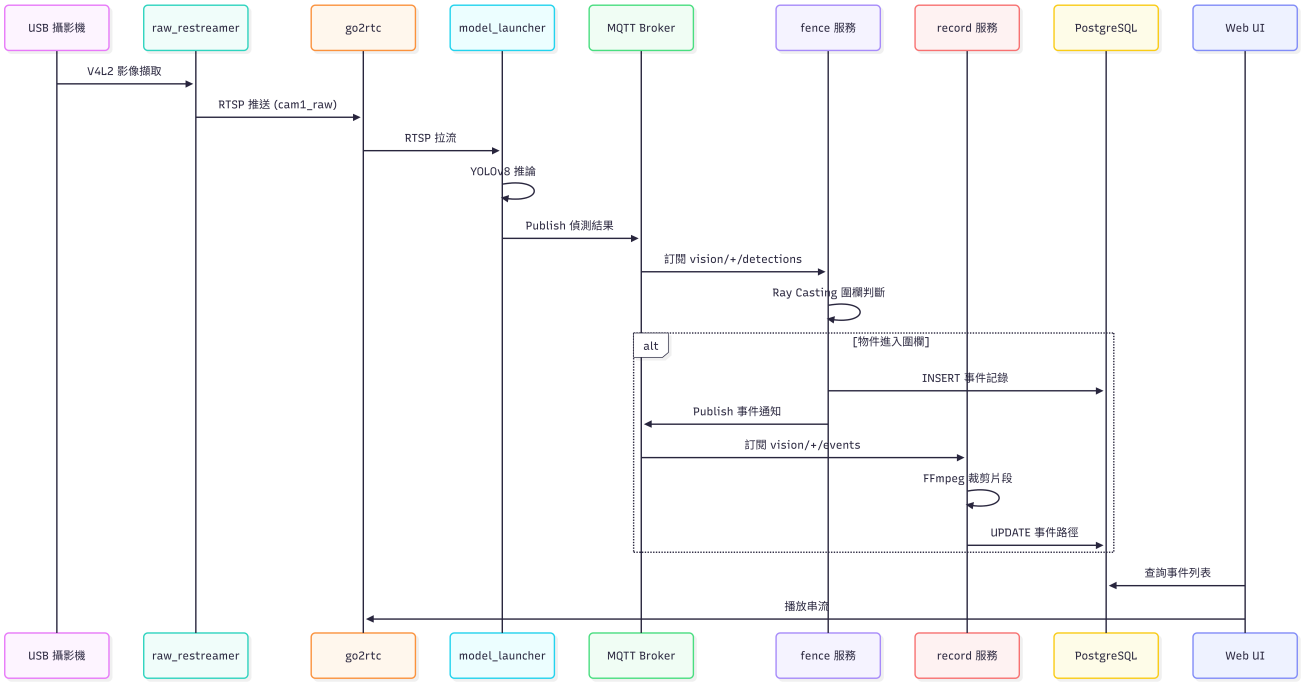
2.2 資料處理流程圖



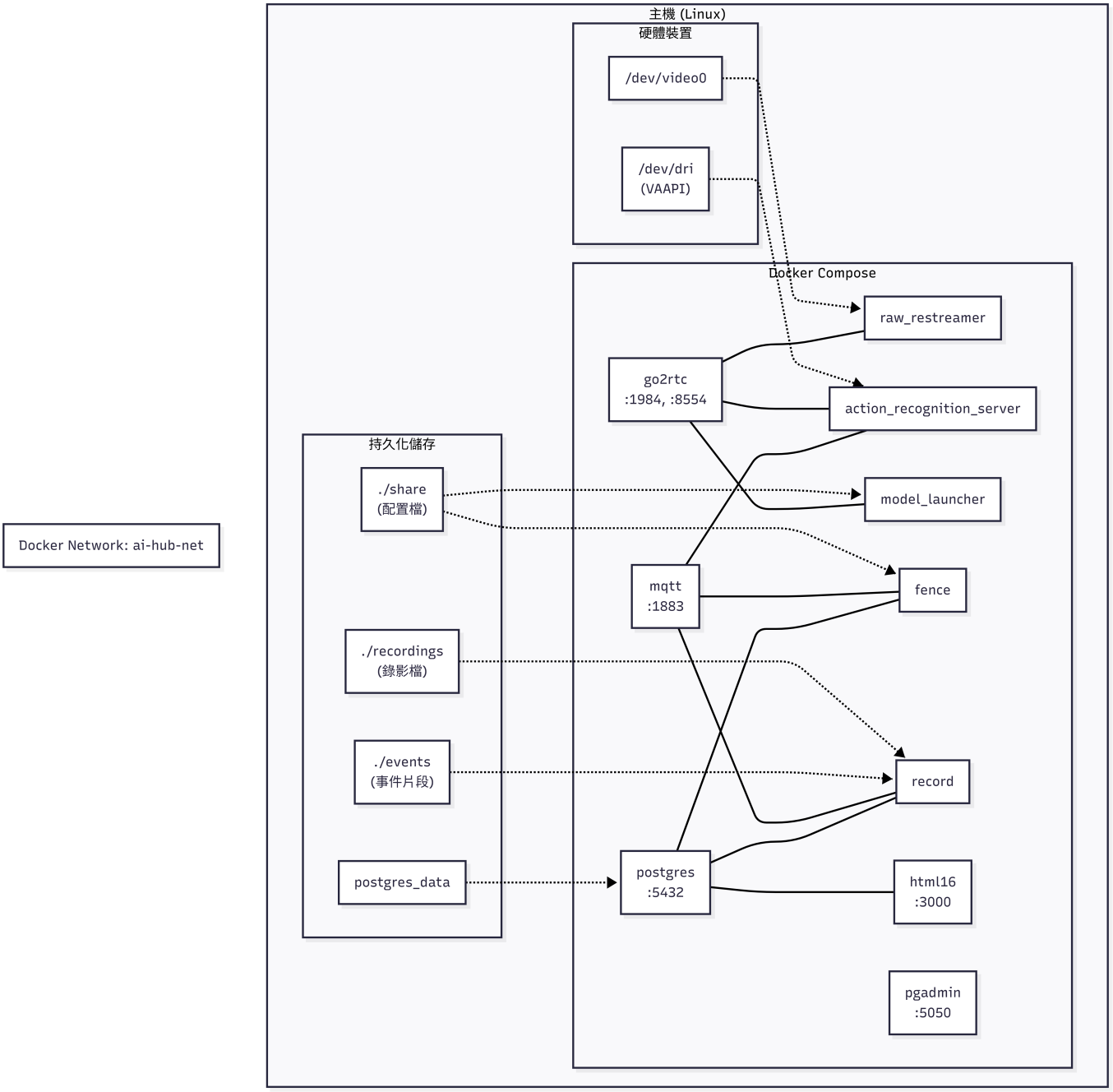
2.3 服務組成

服務名稱	技術棧	主要職責
raw_restreamer	FFmpeg 6.1	從 USB 攝影機擷取原始影像並轉推到 go2rtc
action_recognition_server	Python + OpenCV	拉取原始串流、疊加辨識結果、重新編碼推流
model_launcher	Python + YOLOv8	動態啟動模型推論進程、發布偵測結果至 MQTT
fence	Python + Paho MQTT	監聽偵測結果、判斷虛擬圍欄入侵、產生事件
record	Python + FFmpeg	持續錄影、監聽事件 MQTT、裁剪事件片段
go2rtc	Go	多協議串流伺服器 (RTSP/WebRTC/HLS)
mqtt	Eclipse Mosquitto 2	MQTT Broker 提供訊息中介
postgres	PostgreSQL 16	事件與錄影資料持久化
pgadmin	pgAdmin 4	資料庫管理介面
htmlv2 (html16)	Next.js 16 + React	Web 儀表板與配置管理

2.4 事件處理時序圖


USB 攝影機
raw_restreamer
go2rtc
model_launcher
MQTT Broker
fence 服務
record 服務
PostgreSQL
Web UI
V4L2 影像擷取
RTSP 推送 (cam1_raw)
RTSP 拉流
YOLOv8 推論
Publish 偵測結果
訂閱 vision/+/detections
Ray Casting 圍欄判斷
alt
[物件進入圍欄]
INSERT 事件記錄
Publish 事件通知
訂閱 vision/+/events
FFmpeg 裁剪片段
UPDATE 事件路徑
查詢事件列表
播放串流
USB 攝影機
raw_restreamer
go2rtc
model_launcher
MQTT Broker
fence 服務
record 服務
PostgreSQL
Web UI

2.5 部署架構圖


主機 (Linux)
硬體裝置
/dev/video0
/dev/dri
(VAAPI)
Docker Compose
raw_restreamer
go2rtc
:1984, :8554
action_recognition_server
持久化儲存
./share
(配置檔)
model_launcher
Docker Network: ai-hub-net
mqtt
:1883
fence
./recordings
(錄影檔)
./events
(事件片段)
record
postgres_data
postgres
:5432
html16
:3000
pgadmin
:5050

3. 功能需求

3.1 視訊串流處理

3.1.1 原始串流擷取

- **功能描述**：目前支援 USB 攝影機 (`/dev/video0`)；架構保留 RTSP 來源擴充能力
- **輸入格式**：YUYV422 (USB) / H.264 (RTSP)
- **輸出**：RTSP 串流推送至 go2rtc
- **技術參數**：
  - 解析度：640×480 (可配置)
  - 幀率：10 FPS (可配置)
  - 編碼：H.264 (libx264) 或硬體編碼
  - 優化參數：固定位元率與 Zerolatency 調優
  - 傳輸協議：RTSP over TCP

3.1.2 行為辨識疊加

- **功能描述**：拉取原始串流、執行推論、疊加邊界框與標籤、輸出新串流
- **處理流程**：
  1. 從 go2rtc 拉取 `cam1_raw` 串流
  2. 使用 OpenCV 解碼為影格
  3. 調用 `draw_overlay()` 函式疊加辨識結果
  4. 透過 FFmpeg 子程序重新編碼為 RTSP 串流
- **硬體加速**：
  - 自動偵測 `/dev/dri/renderD128` (Intel VAAPI)
  - 支援環境變數 `ACTION_HWACCEL` 控制：`auto` / `vaapi` / `none`
- **可配置參數**：
  - 輸入/輸出 RTSP URL
  - 解析度 (預設 1280×720)
  - 幀率 (預設 15 FPS)
  - 重連策略與失敗次數上限

3.1.3 多協議串流輸出

- **功能描述**：go2rtc 提供多種協議存取同一串流
- **支援協議**：RTSP、HLS、MSE (Web 播放)
- **埠口配置**：
  - RTSP: 8554
  - HTTP API: 1984
  - WebRTC: 8555 (保留但不作為主要 Web 播放手段)

3.2 AI 物件偵測與推論

3.2.1 模型管理

- **功能描述**：透過 `models.json` 配置多個 AI 模型
- **模型資訊**：
  - 模型名稱、類型 (目前僅支援 YOLOv8)
  - 權重檔案路徑 (`models/`)
  - 類別檔案 (`class/`)
  - 輸入尺寸 (預設 640×640)
  - 推論裝置 (CPU/GPU)
  - Runner 腳本路徑

3.2.2 動態啟動推論

- **功能描述**：`model_launcher` 根據相機配置自動啟動對應模型
- **匹配邏輯**：
  - 從 `cameras.json` 讀取每個相機的 `modelId`
  - 在 `models.json` 中匹配對應模型
  - 支援模糊匹配 (slugify、partial match)
- **多攝影機支援**：同一模型可服務多個攝影機

3.2.3 偵測結果發布

- **功能描述**：推論結果透過 MQTT 發布
- **訊息格式**：

```
{
  "cameraId": "cam1",
  "timestamp": "2025-12-26T12:00:00.000Z",
  "detections": [
    {
      "class_name": "person",
      "bbox": [0.1, 0.2, 0.5, 0.8],
      "score": 0.95
    }
  ]
}
```

- **MQTT Topic**：`vision/<cameraId>/detections`
- **QoS**：可配置 (預設 0)

3.3 虛擬圍欄與事件偵測

3.3.1 虛擬圍欄配置

- **功能描述**：在相機畫面上定義多邊形區域，偵測特定物件進入
- **配置方式**：
  - 在 `cameras.json` 中為每個相機配置 `virtualFences` 陣列
  - 每個圍欄包含：
    - `name`：圍欄名稱
    - `points`：多邊形頂點座標 (支援正規化或像素座標)
    - `detectObjects`：觸發物件類別清單 (如 `["person", "car"]`)
    - `enabled`：是否啟用
- **座標正規化**：自動將像素座標轉換為 0-1 正規化座標

3.3.2 入侵偵測邏輯

- **演算法**：Ray Casting (光線投射法) 判斷點是否在多邊形內
- **觸發條件**：
  1. 偵測框中心點位於圍欄內
  2. 物件類別在 `detectObjects` 清單中
  3. 圍欄處於啟用狀態
- **冷卻機制**：
  - `FENCE_COOLDOWN_SEC`：同一圍欄/物件組合的最小觸發間隔 (預設 30 秒)
  - `FENCE_LEAVE_SEC`：物件離開後多久重新啟用 (預設 5 秒)

3.3.3 事件產生與通知

- **資料庫紀錄**：
  - 插入事件至 PostgreSQL `events` 表
  - 欄位：`id`, `camera_id`, `class_name`, `ts`, `thumbnail`, `score`
- **MQTT 發布**：
  - Topic: `vision/<cameraId>/events`
  - Payload: 事件 JSON (包含 `id`, `camera_id`, `class_name`, `ts`, `score`)

3.4 錄影與事件裁剪

3.4.1 持續錄影

- **功能描述**：全時錄製每個相機串流，按時間分段儲存
- **分段策略**：
  - 分段長度：`SEGMENT_SECONDS` (預設 300 秒 = 5 分鐘)
  - 對齊時鐘：`-segment_atclocktime` 確保整點對齊
  - 檔案格式：`.ts` (MPEG-TS) → 後處理轉為 `.mkv`
- **目錄結構**：

```
recordings/
  <cameraId>/
    <YYYY-MM>/
      <DD>/
        HH-MM-SS.mkv
```

- **後處理**：
  - 自動將 `.ts` 轉檔為 `.mkv` (Matroska)
  - 等候檔案穩定後執行 (預設 2 秒)

3.4.2 緩衝錄影 (Buffer Recording)

- **功能描述**：維護短期循環緩衝，用於事件發生時快速提取「事前」畫面
- **配置**：
  - `EVENT_BUFFER_ENABLED`：是否啟用 (預設 1)
  - `EVENT_BUFFER_SECONDS`：緩衝時長 (預設 10 秒)
  - `EVENT_BUFFER_SEGMENT_SECONDS`：緩衝分段長度 (預設 1 秒)
  - `EVENT_BUFFER_REENCODE`：是否重新編碼以確保固定 GOP
  - `EVENT_BUFFER_GOP`：GOP 大小 (預設 10)
- **目錄結構**：

```
recordings_buffer/
  <cameraId>/
    <YYYY-MM>/<DD>/HH-MM-SS.ts
```

- **清理機制**：自動刪除超過緩衝時長的舊檔案

3.4.3 事件片段裁剪

- **功能描述**：監聽事件 MQTT，裁剪事前 N 秒與事後 M 秒的影片
- **裁剪模式**：
  - **Mode 1 - 從持續錄影裁剪**：
    - 適用於長分段錄影 (如 5 分鐘)
    - 尋找事件時間所在的分段檔案

        - 使用 FFmpeg concat 協議串接多個分段
        - 精確裁剪出 `[ts - PRE, ts + POST]` 區間
    - **Mode 2 - 從緩衝裁剪 (優先)**：
        - 適用於短緩衝分段 (如 1 秒)
        - 事件發生時立即從緩衝目錄提取「事前」片段
        - 即時錄製「事後」片段
        - 串接後裁剪為最終事件影片
- **參數配置**：
    - `EVENT_PRE_SECONDS`：事前秒數 (預設 10)
    - `EVENT_POST_SECONDS`：事後秒數 (預設 10)
- **輸出**：
    - 檔案名稱：`<eventId>.mkv`
    - 儲存路徑：`share/events/`
    - 自動更新資料庫 `events.thumbnail` 欄位

3.4.4 分段穩定等候

- **功能描述**：避免裁剪正在寫入的檔案
- **等候邏輯**：
    - `SEGMENT_READY_GRACE`：分段結束後額外等候時間 (預設 2 秒)
    - `SEGMENT_MAX_WAIT`：最大等候時間 (預設 15 秒)
    - 超過最大等候時間仍未就緒則放棄裁剪

3.5 Web 儀表板

3.5.1 首頁儀表板

- **功能描述**：展示所有相機的即時串流與狀態
- **資訊顯示**：
    - 相機名稱、位置、狀態 (線上/離線)
    - 即時 WebRTC 播放器
    - 最近事件提醒

3.5.2 相機管理

- **功能描述**：新增、編輯、刪除相機配置
- **可配置項目**：
    - 相機 ID、名稱、位置、群組
    - RTSP URL、串流 ID
    - 解析度、幀率
    - 使用的模型 ID (`modelId`)
    - 虛擬圍欄配置 (`virtualFences`)
- **配置持久化**：儲存至 `share/cameras.json`

3.5.3 事件瀏覽

- **功能描述**：查看所有觸發的事件記錄
- **展示內容**：

    - 事件時間、相機、偵測類別、信心度
    - 事件縮圖 (從影片中提取)
    - 事件影片播放
- **篩選與搜尋：**
    - 按相機篩選
    - 按時間範圍篩選
    - 按偵測類別篩選

3.5.4 錄影管理

- **功能描述**：瀏覽與下載持續錄影片段
- **目錄樹展示：**
    - 按相機/日期分層
    - 顯示檔案大小與時長
- **操作：**
    - 線上播放
    - 下載
    - 刪除

3.5.5 系統設定

- **功能描述**：調整系統參數
- **設定項目：**
    - go2rtc 連線設定
    - MQTT Broker 設定
    - 錄影保留政策
    - 使用者權限管理 (未來擴充)

---

4. 非功能需求

4.1 效能需求

- **串流延遲：**< 500ms (在區域網路環境)
- **推論速度：**≥ 10 FPS (取決於硬體與模型)
- **事件反應時間：**< 2 秒 (從偵測到 MQTT 發布)
- **並發支援**：單機至少支援 4 路 1080p 實時處理

4.2 可靠性需求

- **自動重連**：所有串流與 MQTT 連線支援斷線重連
- **錯誤容忍**：單一服務故障不影響其他服務
- **資料持久化**：事件與錄影資料不因系統重啟遺失

4.3 可擴展性需求

- **水平擴展**：支援多節點部署 (MQTT 與 DB 共享)
- **模型熱插拔**：新增模型無需重啟整個系統
- **API 開放性**：提供 RESTful API 供第三方整合

4.4 安全性需求

- **RTSP 認證**：支援 `publisher:secret` 認證機制
- **MQTT 認證**：支援 username/password (可選)
- **資料庫安全**：預設密碼應在生產環境更換
- **網路隔離**：敏感服務不暴露於公網

4.5 可維護性需求

- **容器化部署**：所有服務 Docker 化，一鍵啟動
- **日誌記錄**：統一日誌格式，支援集中收集
- **配置管理**：環境變數與 JSON 配置檔分離
- **版本控制**：配置檔與程式碼納入 Git

---

5. 資料模型

5.1 cameras.json

```
{
  "cameras": [
    {
      "id": "cam1",
      "name": "前門攝影機",
      "location": "1F 大廳",
      "group": "入口",
      "enabled": true,
      "rtspUrl": "rtsp://127.0.0.1:8554/cam1_raw",
      "streamUrl": "http://localhost:1984/stream.html?src=cam1_overlay",
      "streamId": "cam1_overlay",
      "resolution": "1280x720",
      "fps": 15,
      "modelId": "yolov8n",
      "virtualFences": [
        {
          "name": "入口區域",
          "enabled": true,
          "points": [
            {"x": 0.2, "y": 0.3},
            {"x": 0.8, "y": 0.3},
            {"x": 0.8, "y": 0.7},
            {"x": 0.2, "y": 0.7}
          ],
          "detectObjects": ["person"]
        }
      ]
    }
  ]
}
```

5.2 models.json

```
{
  "models": [
    {
      "name": "YOLOv8 Nano",
      "type": "yolov8n",
      "weights": "/models/yolov8n.pt",
      "class_file": "/class/coco.txt",
      "inputSize": [640, 640],
      "device": "cpu",
      "runner": "/app/yolov8_inference.py"
    }
  ]
}
```

5.3 PostgreSQL Schema

events 表

```
CREATE TABLE events (
    id VARCHAR(32) PRIMARY KEY,
    camera_id VARCHAR(64) NOT NULL,
    class_name VARCHAR(128),
    ts TIMESTAMP WITH TIME ZONE NOT NULL,
    thumbnail VARCHAR(256),
    score REAL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
CREATE INDEX idx_events_camera_ts ON events(camera_id, ts DESC);
CREATE INDEX idx_events_class ON events(class_name);
```

recordings 表 (可選，未來擴充)

```
CREATE TABLE recordings (
    id VARCHAR(32) PRIMARY KEY,
    camera_id VARCHAR(64) NOT NULL,
    start_ts TIMESTAMP WITH TIME ZONE NOT NULL,
    end_ts TIMESTAMP WITH TIME ZONE NOT NULL,
    file_path VARCHAR(512) NOT NULL,
    file_size BIGINT,
    duration REAL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

---

6. 介面定義

6.1 MQTT Topics

Topic Pattern	方向	說明	Payload 範例
`vision/<cameraId>/detections`	Publish	推論結果	`{"cameraId":"cam1", "timestamp":"...", "detections":[...]}`
`vision/<cameraId>/events`	Publish	圍欄事件	`{"id":"evt_abc", "camera_id":"cam1", "ts":"...", "class_name":"person"}`

6.2 go2rtc HTTP API

- **GET /api/streams**：列出所有串流
- **GET /api/stream?src=**：取得指定串流資訊
- **POST /api/stream?dst=&src=**：新增串流來源

6.3 Next.js Server Actions

- `getEvents(filter)`：查詢事件列表
- `deleteEvent(eventId)`：刪除事件
- `getCameras()`：取得相機列表
- `updateCamera(cameraId, config)`：更新相機配置

---

7. 部署需求

7.1 環境需求

- **作業系統**：Linux (Ubuntu 20.04+ / Debian 11+)
- **Docker**：24.0+
- **Docker Compose**：v2.0+
- **硬體**：
  - CPU: 4 核心以上 (建議 Intel with Quick Sync)
  - RAM: 8GB 以上
  - 儲存: 至少 100GB (視錄影保留需求)
  - GPU: 可選 (NVIDIA / Intel iGPU)

7.2 網路需求

- **內網環境**：所有服務在同一 Docker 網路
- **外部存取**：
  - go2rtc Web UI: 1984
  - Web 儀表板: 3000

- MQTT Broker: 1883 (可選對外)
- PostgreSQL: 5432 (可選對外)

7.3 啟動流程

```
# 1. 設定環境變數
cp app/.env.example app/.env
vi app/.env

# 2. 調整相機與模型配置
vi share/cameras.json
vi share/models.json

# 3. 啟動所有服務
docker compose up -d --build

# 4. 檢查日誌
docker compose logs -f action_recognition_server
docker compose logs -f fence
docker compose logs -f record

# 5. 存取 Web UI
open http://localhost:3000
```

---

8. 附錄

8.1 參考文件

- go2rtc 官方文檔
- YOLOv8 文檔
- FFmpeg 官方文檔
- Mosquitto MQTT Broker

8.2 專案結構說明

```
action_recognition_server/
├── app/                        # 行為辨識服務
│   ├── action_recognition_server.py
│   ├── Dockerfile
│   └── requirements.txt
├── models_classify/            # 模型推論服務
│   ├── launcher.py
│   ├── yolov8_inference.py
│   ├── models/                # 模型權重檔
│   └── class/                 # 類別定義
├── fence/                      # 虛擬圍欄服務
│   ├── main.py
│   └── Dockerfile
├── record/                     # 錄影服務
│   ├── main.py
│   └── Dockerfile
├── html16/                     # Next.js Web UI
│   ├── app/
│   ├── components/
│   └── Dockerfile
├── go2rtc/                     # 串流伺服器配置
│   └── go2rtc-config.yaml
├── MQTT/                       # MQTT Broker 配置
│   └── mosquitto.conf
├── postgres/                   # 資料庫初始化腳本
│   └── init/
├── share/                      # 共享資料目錄
│   ├── cameras.json
│   ├── models.json
│   ├── events/                # 事件影片
│   └── recordings/            # 持續錄影
└── docker-compose.yml          # 服務編排
```

8.3 環境變數清單

詳見各服務的 `.env` 檔案與 `docker-compose.yml` 中的 `environment` 區段。

文件版本: v1.0 **最後更新**: 2025-12-27