Manual del Desarrollador – Sistema de Seguridad IoT

1. Estructura del Repositorio

Se recomienda una estructura monorepo con subdirectorios para cada servicio:

• backend/ – API REST (Django/Flask) • worker/ – Servicio MQTT → BD • ai/ – Servicio de IA con FastAPI • frontend/ – Dashboard web • infra/ – Archivos Docker Compose, configuración de Mosquitto, etc. • docs/ – Documentación técnica y diagramas

2. Convenciones de Código

• Lenguaje principal: Python 3.10. • Estilo: PEP8, utilizando herramientas como black y isort. • Tests: pytest para cada servicio, con tests unitarios y de integración básicos. • Logs: usar logging estándar de Python, configurado vía dictConfig.

3. Backend REST

El backend expone endpoints para gestionar dispositivos, zonas, eventos y evidencias. Se utiliza un ORM (Django ORM o SQLAlchemy) para interactuar con PostgreSQL. Se recomienda organizar el código en módulos: routers, models, schemas, services.

4. Worker MQTT

El worker se encarga de: • Suscribirse a topics MQTT de dispositivos ('devices/#' y 'security/#'). • Parsear los tópicos para determinar tipo de mensaje (telemetría, evento, estado). • Insertar datos en las tablas 'measurements' y 'events'. • Implementar la lógica 'PIR → luces + cámaras' consultando la base de datos.

5. Servicio IA

El servicio IA recibe rutas de evidencias (imágenes/vídeos) y devuelve metadatos de detección. Se puede implementar como un microservicio FastAPI con un endpoint '/analyze' que recibe un ID de evidencia, carga el archivo desde disco, ejecuta el modelo TFLite y actualiza la columna 'ai_metadata' en la tabla 'evidences'.

6. Flujo de Desarrollo

1. Crear ramas por feature. 2. Agregar tests unitarios. 3. Ejecutar los tests localmente. 4. Levantar el entorno Docker Compose en modo desarrollo. 5. Probar interacciones end-to-end (PIR simulado → worker → backend → DB).