Absolument. Voici le document de cadrage complet de votre projet, entièrement réécrit pour intégrer la nouvelle architecture plus professionnelle utilisant un téléphone comme caméra IP distante.

### **Titre du Projet : VisionStream Sentinel**

* **Sous-titre :** Une plateforme de surveillance intelligente en temps réel avec FastAPI, YOLOv8 et une source vidéo distante.
* **Pourquoi ce titre ?** "VisionStream" combine les deux concepts clés : la vision par ordinateur et le traitement de flux (streaming). "Sentinel" (Sentinelle) évoque la surveillance intelligente et proactive, ce qui donne une image professionnelle et claire de la finalité du projet.

### **Scénario Complet du Projet**

**Contexte (Le Problème) :**Une entreprise de logistique gère un entrepôt où certaines zones sont strictement réservées au personnel autorisé pour des raisons de sécurité. Les systèmes de vidéosurveillance traditionnels enregistrent des heures de vidéo, mais la détection d'une intrusion n'a lieu que *post-mortem*. Le temps de réaction est donc nul.

**Solution (Votre Projet) :**"VisionStream Sentinel" est une plateforme qui se connecte au flux d'une caméra IP (simulée par un téléphone) placée dans l'entrepôt. Le serveur central analyse la vidéo en temps réel pour détecter la présence humaine et diffuse les résultats, y compris les alertes d'intrusion, à un ou plusieurs postes de supervision.

**Fonctionnement (La Démonstration) :**

1. **Mise en Place du Capteur :** Vous activez une application de type "IP Webcam" sur votre téléphone et le positionnez pour filmer une zone (ex: une porte). Le téléphone est maintenant un capteur vidéo sur le réseau Wi-Fi.
2. **Lancement du Serveur :** Vous lancez le backend FastAPI sur votre ordinateur. Celui-ci se connecte automatiquement au flux vidéo du téléphone, commence à en extraire les images et à les analyser en continu avec YOLOv8.
3. **Interface de Supervision :** Un superviseur de la sécurité se connecte à une simple page web (votre frontend) depuis n'importe quel appareil sur le même réseau.
4. **Diffusion en Temps Réel :** La page web établit une connexion WebSocket avec le backend. Le backend commence immédiatement à lui *pousser* un flux de résultats : chaque message contient l'image vidéo actuelle et les données de détection associées. L'interface affiche ce flux, donnant l'impression d'une vidéo en direct, mais enrichie.
5. **Retour d'Information Visuel :** L'interface dessine en temps réel les boîtes de détection autour des personnes, directement sur la vidéo reçue du backend.
6. **Déclenchement d'Alerte :** Dans le backend, si les coordonnées d'une personne détectée se superposent à une "zone d'alerte" prédéfinie, le message WebSocket envoyé au superviseur contient un statut spécial ("event": "INTRUSION\_DETECTED"). L'interface réagit en affichant un avertissement visuel clair, comme un cadre rouge clignotant.

**Valeur Ajoutée :**Ce projet transforme une surveillance passive en un **système de sécurité distribué, intelligent et actif**. Il démontre une architecture réaliste et évolutive, capable de réagir en quelques millisecondes.

### **Structure Complète du Projet**

L'arborescence des fichiers reste logiquement organisée, mais le rôle de plusieurs fichiers a été profondément modifié pour s'adapter à la nouvelle architecture.

visionstream\_sentinel/

│

├── app/ # Contient tout le code source de l'application backend

│ │

│ ├── \_\_init\_\_.py

│ │

│ ├── main.py # Point d'entrée de l'application FastAPI. Gère le démarrage et le routeur.

│ │

│ ├── api/ # Contient la logique de l'API de diffusion

│ │ ├── \_\_init\_\_.py

│ │ └── streaming.py # Définit le routeur WebSocket qui gère les connexions des superviseurs.

│ │

│ ├── core/ # Contient la configuration et les modules principaux

│ │ ├── \_\_init\_\_.py

│ │ └── config.py # Gère la config (URL de la caméra IP, chemin du modèle, etc.)

│ │

│ ├── schemas/ # Définit les modèles de données Pydantic

│ │ ├── \_\_init\_\_.py

│ │ └── detection.py # Schémas pour les boîtes de détection et le message WebSocket complet.

│ │

│ └── vision/ # Contient la logique métier de la vision par ordinateur

│ ├── \_\_init\_\_.py

│ └── processor.py # NOUVEAU RÔLE : Gère la boucle de capture et de traitement du flux vidéo.

│

├── models/ # Dossier pour stocker les fichiers de modèles IA (ex: .pt pour YOLO)

│ └── yolov8n.pt

│

├── static/ # Contient tous les fichiers du frontend de supervision

│ │

│ ├── index.html # La page web principale avec une balise <canvas> ou <img>.

│ │

│ ├── css/

│ │ └── style.css

│ │

│ └── js/

│ └── main.js # RÔLE SIMPLIFIÉ : Gère la connexion WebSocket et l'affichage des données reçues.

│

├── .env # Fichier pour vos variables d'environnement locales (ex: VIDEO\_STREAM\_URL)

├── .env.example

├── .gitignore

├── README.md # Description complète, schéma d'architecture, instructions de lancement.

└── requirements.txt

### **Rôle Détaillé de Chaque Fichier Clé (Version Mise à Jour)**

* **app/main.py :** Le chef d'orchestre. Il initialise FastAPI, inclut le routeur de streaming.py, et peut être responsable du lancement de la tâche de traitement vidéo en arrière-plan au démarrage de l'application.
* **app/api/streaming.py :** Le diffuseur. Son rôle n'est plus de recevoir des données. Il gère les connexions WebSocket entrantes des clients (superviseurs) et les ajoute à une liste de "spectateurs". Il diffuse ensuite les résultats produits par processor.py à tous les spectateurs connectés.
* **app/vision/processor.py :** Le **cœur du système**. Ce module est maintenant responsable de la boucle infinie :

1. Se connecter au flux vidéo de la caméra IP (via OpenCV).
2. Lire une image.
3. La passer au modèle YOLOv8 pour analyse.
4. Vérifier la logique d'intrusion.
5. Mettre le résultat (image + données) à disposition du module de diffusion.  
   Il est complètement indépendant de FastAPI.

* **app/schemas/detection.py :** Le contrat de données. Il définit la structure du message JSON qui sera envoyé via WebSocket, contenant par exemple l'image encodée en base64, une liste de détections, et le statut de l'alerte.
* **app/core/config.py :** Le panneau de contrôle. Il charge des configurations cruciales depuis les variables d'environnement, notamment la plus importante : l'**URL du flux vidéo de la caméra IP**.
* **static/js/main.js :** Le client **passif** et léger. Son rôle est considérablement simplifié :

1. Se connecter au WebSocket du serveur.
2. Attendre de recevoir des messages.
3. À la réception d'un message, décoder l'image et la dessiner sur le canvas.
4. Dessiner les boîtes de détection par-dessus, en utilisant la couleur appropriée en fonction du statut d'alerte.

* **README.md :** Votre meilleur vendeur. Ce document doit maintenant inclure :
* Un **schéma d'architecture** simple montrant le téléphone, le backend et le client.
* Des instructions claires pour configurer l'application "IP Webcam" sur le téléphone.
* Un GIF de la démo montrant le téléphone filmant une scène et le résultat apparaissant sur l'écran du PC.