

# Skylark Labs: Full Stack Engineer Coding Test

**Project:** Real-Time Multi-Camera Face Detection Dashboard (WebRTC)

**Submission:** Email deliverables to [rocio.novo@skylarklabs.ai](mailto:rocio.novo@skylarklabs.ai) by September 29, 2025.

---

## Overview

Develop a microservices-based dashboard application where users can register cameras by providing RTSP URLs, view live camera feeds via WebRTC directly in the browser, and receive real-time face-detection overlays on those streams. Face detections trigger alerts, instantly sent to the frontend via WebSockets.

---

## Requirements (detailed)

### 1) Frontend

- Login page (username/password) — JWT authentication from backend.
- Dashboard:
  - Camera management (Create / Read / Update / Delete): add RTSP URL, camera name, and location.
  - View camera tiles in a responsive grid.
  - Per-camera tile shows:
    - Live WebRTC video (with overlays coming from worker frames)
    - Alerts list (recent face detections for that camera)
    - Start / Stop button for stream
- Realtime: Subscribe to the backend websocket for alerts.

**UX expectations:** responsive MUI layout, clear feedback on stream states, and error messages.

---

### 2) Backend API

- **Authentication**
  - Implement a login flow that issues a JWT.
  - API's should be protected.
- **Camera Management**
  - Provide functionality for users to manage their cameras (add, list, update, delete).
  - Cameras should include metadata like name, RTSP URL, and whether they are enabled/disabled.
  - Support operations to start/stop camera processing.
- **Alerts / Events**
  - Expose a way to fetch recent detection alerts (e.g., face detections).
  - Ensure filtering and pagination are possible.
- **Real-time Communication**

- Implement a WebSocket (or similar) mechanism for pushing live alerts and per-camera stats to clients.
  - Use **Prisma** with PostgreSQL for the database.
- 

### 3) Worker

- The worker should handle camera start requests (triggered via API calls from backend) and be capable of managing multiple camera streams concurrently, with a target of at least four simultaneous streams.
- For each camera:
  - Read RTSP stream (FFmpeg or GStreamer, or OpenCV; pipe them for further processing).
  - Decode frames and run face detection (suggested: [go-face](#) or equivalent).
  - Draw bounding boxes and overlay text on frames: **Camera ID**, **FPS**, use OpenCV drawing utilities.
  - Send processed frames into MediaMTX (or direct WebRTC)
  - On each face detection:
    - Build an alert object with a frame.
    - POST alert to backend API to persist (Bonus: Using Message Queue)
    - Optionally write snapshot image to storage and include URL in alert payload
- Reliability:
  - Reconnect on stream failures with exponential backoff
  - Implement a frame-dropping/skipping mechanism to ensure the stream always remains real-time
- Example helper libs:
  - [github.com/Kagami/go-face](#) (face recognition/detection) — acceptable
  - OpenCV binding (gocv) for drawing ([gocv.io/x/gocv](#))
  - [github.com/blueviron/mediamtx](#) for stream publishing.

---

### Extras (explicit)

- Camera stream updates should automatically reflect on the frontend. Face detection must be toggleable from the frontend, and FPS settings should be managed from the frontend, with all changes reflected in real time.
- **Responsive UI:** Desktop + mobile layout.
- **Tests:** at least a few unit/integration tests for API and components (bonus).

---

## Programming Stack

- Frontend: **React (TypeScript) + Vite + MUI/Tailwind**
- Backend API: **TypeScript + Hono + Prisma (Postgres)**
- Worker: **Golang + Gin + FFmpeg + OpenCV + go-face** (or equivalent face detection lib)

Optional / recommended: **MediaMTX** to handle WebRTC signalling & media proxying (worker pushes processed RTSP into MediaMTX, which serves WebRTC to browsers)

---

## Deliverables

1. Public GitHub repo(s) with full source (frontend, backend, worker, infra/docker compose).
2. Submit live demo URLs (preferred). If a live demo isn't possible, provide a short, narrated video walkthrough explaining the application's functionality.
3. Properly documented README.md file.
4. Dockerize the microservices and Docker Compose to run everything.

---

## Evaluation criteria & rubric

- Functionality
- Architecture & Design
- Code Quality & Tests
- Documentation
- UI/UX & Responsiveness
- Resilience & Performance

---

## Timeline

**5 days** recommended. If live hosting is not possible within that time, the candidate should deliver a video walkthrough.

---

## Submission

- GitHub repository link
- Live demo URL(s) (or video file/link)
- Optionally, Short Video + Explanation of the app.
- Any credentials required for demo (test user)
- Short note describing known limitations & what they'd improve with more time