🧪 Android + OpenCV-C++ + OpenGL Assessment + Web - RnD Intern

📆 Duration: 3 **Days**

This is a **time-bound technical assessment** to evaluate your practical skills in **Android development, OpenCV (C++), OpenGL ES, JNI (NDK), and TypeScript (Web)**. The focus is on **integration and rendering**, not on perfect UI or advanced features.

⚠️ **IMPORTANT:**

Proper use of **GitHub or GitLab** for version control is **mandatory**.

If your project is **not properly committed and pushed** to a public (or shareable private) repository, **your submission will not be evaluated**.

🔧 Tech Stack Requirements

- **Android SDK** (Java/Kotlin)
- **NDK** (Native Development Kit)
- **OpenGL ES 2.0+**
- **OpenCV (C++)**
- **JNI** (Java ↔ C++ communication)
- **TypeScript** (for a minimal web-based viewer / debug tool)
- Optional: GLSL shaders, Android CameraX, or OpenCV Camera Bridge (Java side)

🚀 Challenge: Real-Time Edge Detection Viewer

You're building a **minimal Android app** that captures camera frames, processes them using OpenCV in C++ (via JNI), and displays the processed output using OpenGL ES.

Additionally, create a **small TypeScript-based web page** that can receive a dummy processed frame (static image or base64) and display it — to demonstrate ability to bridge native processing results to a simple web layer.

🧩 Key Features (Must-Have)

1. 📸 Camera Feed Integration (Android)

- Use `TextureView` or `SurfaceTexture` to capture frames from the camera.
- Set up a **repeating image capture stream** (Camera1 or Camera2 API).

2. 🔁 Frame Processing via OpenCV (C++)

- Send each frame to native code using JNI.
- Apply a **Canny Edge Detection** or **Grayscale filter** using OpenCV (C++).
- Return the processed image (or pass directly to OpenGL texture).

3. 🎨 Render Output with OpenGL ES

- Render the processed image using **OpenGL ES 2.0** (as a texture).
- Ensure smooth real-time performance (minimum 10–15 FPS).

4. 🌐 Web Viewer (TypeScript)

- A minimal web page (TypeScript + HTML) that displays:
    - A static sample processed frame (can be saved from Android run).
    - Basic text overlay for frame stats (FPS, resolution).
- Demonstrates comfort with TypeScript project setup and DOM updates.

⚙️ Architecture Guidelines

- **Modular project structure**, with at least:

    ```
    /app (Java/Kotlin code)
    /jni (C++ OpenCV processing)
    /gl (OpenGL renderer classes)
    /web (TypeScript web viewer)
    ```

- Use **native C++** for all OpenCV logic.
- Keep Java/Kotlin focused on camera access and UI setup.
- Keep TypeScript clean, modular, and buildable via `tsc`.
- Use **proper Git commits** (meaningful messages, modular changes, not one giant dump at the end).

⭐ Bonus (Optional)

- Button to toggle between:
  - Raw camera feed
  - Edge-detected output
- Add an FPS counter or log frame processing time.
- Use OpenGL shaders to apply visual effects (grayscale, invert).
- Add a simple WebSocket or HTTP endpoint (mock) for the web viewer.

📦 Submission Instructions

- Push your **entire project** to a **public GitHub or GitLab repo** (or share a private repo with access granted).
- Your commit history should clearly reflect your development process (no single “final commit” uploads).
- Add a `README.md` with:
  - ✅ Features implemented (Android + Web)
  - 📷 Screenshots or GIF of the working app
  - ⚙️ Setup instructions (NDK, OpenCV dependencies)
  - 🧠 Quick explanation of architecture (JNI, frame flow, TypeScript part)

⚠️ **Submissions without a proper Git repository will not be evaluated.**

✅ Evaluation Criteria

Area	Weight
Native-C++ integration (JNI)	25%

Area	Weight
OpenCV usage (correct & efficient)	20%
OpenGL rendering	20%
TypeScript web viewer	20%
Project structure, documentation, and commit history	15%