# **Gemma-Kavach**

## **Day 1: Setup up backend server**

I built a **production-ready backend server** using **FastAPI** to serve the **multimodal Gemma 3n model** from Google. This server supports:

* 🔤 **Text-based queries** via /generate
* 🖼️ **Image + prompt reasoning** via /ask\_image
* 🎤 **Voice/audio transcription + reasoning** via /ask

The server enables secure, real-time, and **offline-first multimodal inference**, designed to support use cases like **crisis response** and **privacy-preserving assistants**.

**🧠 Key Learnings**

1. **Stick to Official Google Docs**  
   Trying to outsmart the official guidance led to unnecessary errors. Following Google’s documentation for Gemma 3n (especially around loading AutoProcessor and AutoModelForImageTextToText) ensured compatibility and feature completeness.
2. **Correct Library Installation is Crucial**
   * Mismatched or outdated transformers, accelerate, or torch versions caused slowdowns and even runtime errors.
   * Up-to-date, version-pinned installations prevent debugging nightmares.
3. **Model Parameters Directly Impact Speed**
   * Failing to disable torch.compile, or setting the wrong attn\_implementation, can **drastically slow down token generation**.
   * Using torch\_dtype=torch.bfloat16, disable\_compile=True, and attn\_implementation="eager" gives you a huge speed boost for inference.
4. **Every Developer Should Know How to Serve Models Locally**  
   Hosting your own inference backend is critical for:
   * 💼 **Enterprise deployments** with data security needs
   * 🔐 **Offline/private use** (like in crisis zones or field missions)
   * 🛠️ Learning how open-source LLMs actually integrate with production apps

## **Day 1: Feature Planning: Gemma Kavach Vision**

Simple Objective: Let people upload a video and get analysis and otherwise I real time stream.

**This is an AI-powered real-time crowd safety monitoring system designed to prevent stampede disasters at large events.** The system continuously captures video frames from a webcam, analyzes each frame using computer vision AI to detect signs of crowd panic (pushing, falling, overcrowding), and immediately alerts security personnel through multiple channels when dangerous situations are identified. It operates like an automated safety watchdog that never gets tired or distracted - constantly scanning crowds and providing instant warnings with sound alerts, email notifications, and visual indicators on the live feed. The system is particularly valuable for festivals, religious gatherings, concerts, and any large public events where crowd management is critical, as it can detect early warning signs of stampedes before they become fatal incidents and provide documented evidence with timestamped alerts, risk scores, and visual summaries for emergency response teams.

**Key Value:** Transforms any standard camera into an intelligent crowd safety monitor that can potentially save lives by providing early detection and rapid alerting of dangerous crowd conditions, making it an essential tool for event organizers and security teams managing large gatherings.

## **Day 2 Feature Planning and implementation: Gemma Kavach Vision**

So we are fine for demo of feature 1 i.e Gemma Kavach Vision but it is like a script in my laptop so no one apart from me can use it we need to migrate this to a backend server and then make a ui so that any one can use it like that is very important from myend.

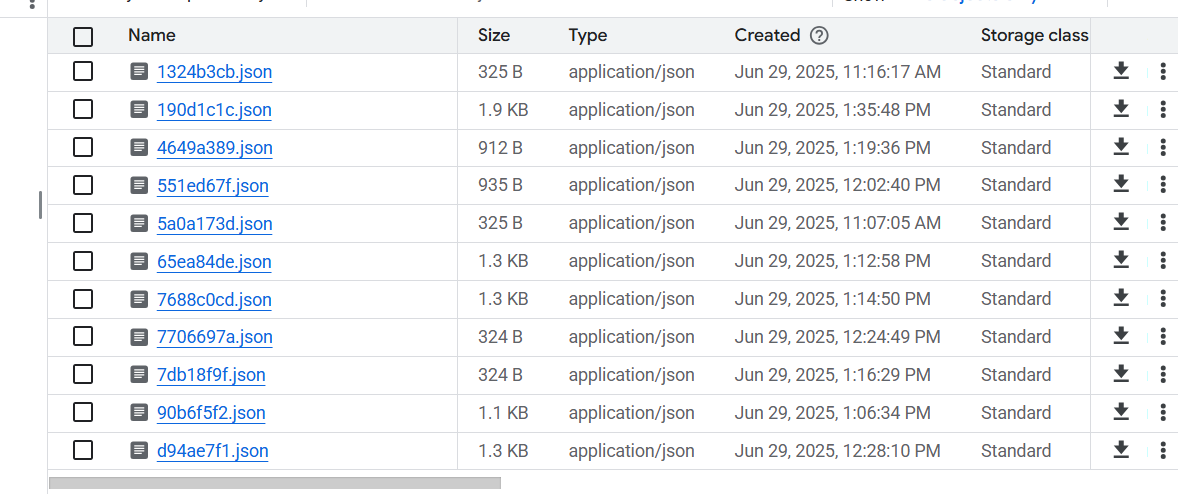
**Planning Backend for this feature.**

Gemman Kavach Vision will be a seprate backend, running so some port, the ui will seprate so we do design a full backend gemma kavach vision.

Also we need a object storage location for now using gemma3n-raw a goole coud storage engine

**1.@router.post("/session/create", response\_model=SessionResponse)**

Create Session on every new run of the app we will create session



File storage is not the best place to store this type of information but fine fow done.



2. **@router.post("/session/{session\_id}/frame", response\_model=FrameAnalysisResponse)**

The frontend will keep sending images in here

1.Get the session context

2.Make a call to gemma to see if the frame is risky or not.

3.Save the flagged image to google storage

4.Append session data for that frame

5.Send email via background tasks if we are good i.e

# Alert thresholds

MIN\_FRAMES\_FOR\_ALERT = 3

RISK\_THRESHOLD\_FOR\_ALERT = 75.0

**3. @router.get("/session/{session\_id}", response\_model=SessionStatusResponse)**

Get simple session status works fine