VisionVault: A Wearable AI Camera System

Real-time Environmental Description via AI and TTS

Touhidul Alam Seyam
Eftekhar Uddin
Mim Chowdhury
Mahin Ullah
Bajpakhi Muntasir

Microprocessor Lab
Professor Radiathun Tasnia, Junior Lecturer

May 2, 2025

Outline

What is VisionVault?

Core Idea: An innovative project integrating an ESP32-CAM on goggles to capture and document moments using AI.

- Provides real-time auditory descriptions of the surroundings.
- Allows users to stream live video (implicitly).
- Generates scene descriptions via AI (Google Gemini).
- Aims for hands-free operation via a mobile app.

Applications:

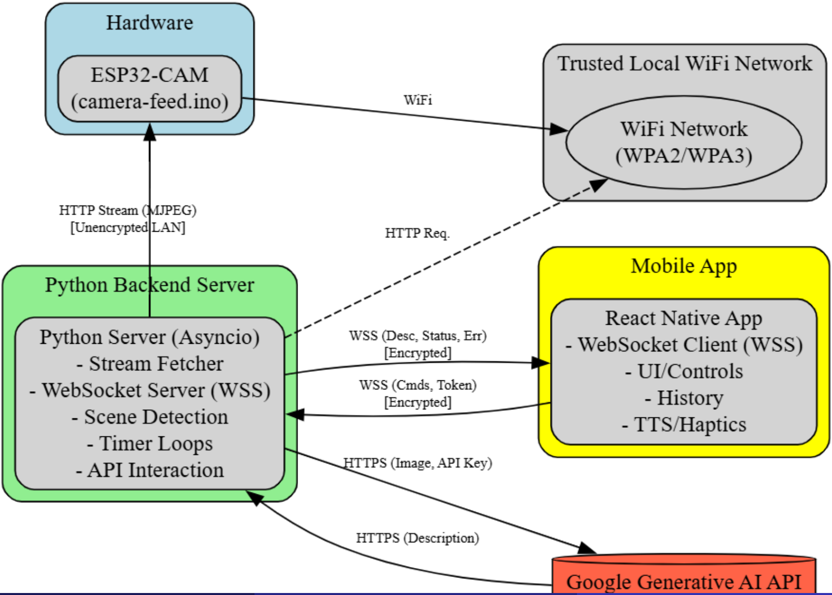
- Assistive technology for the visually impaired.

Demo Image Placeholder

ESP32 $\rightarrow$Python Server $\rightarrow$Mobile App $\rightarrow$TTS

- **ESP32-CAM:** Streams video continuously over local WiFi.
- **Python Backend Server:**
  - Fetches stream (with reconnection).
  - Runs WebSocket Server (WSS) for secure mobile app communication.
  - Manages client state (active/inactive, description interval).
  - **Automatic Mode:** Periodically checks for scene changes. If changed, sends frame to API, broadcasts description to active clients.
  - Handles commands from app (‘start‘, ‘stop‘, ‘set_interval‘, ‘describe_now‘).
- **React Native Mobile App:**
  - Connects securely to Python Server (WSS, with reconnection).
  - UI: Start/Stop, Interval config, Describe Now, History, Status indicators.
  - Sends commands to server.
  - Receives descriptions/errors.
  - Provides TTS output and Haptic feedback.

Overall Component Interaction



Modular Design

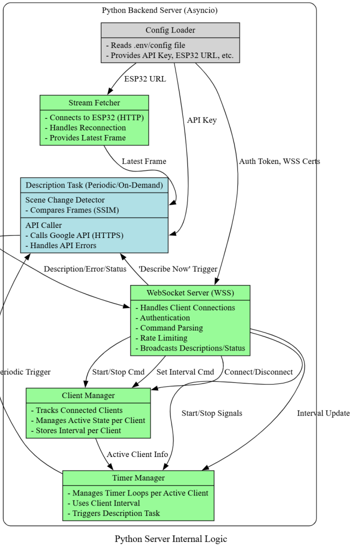


Figure: Key internal modules and data flow within the Python server.

User Interaction States

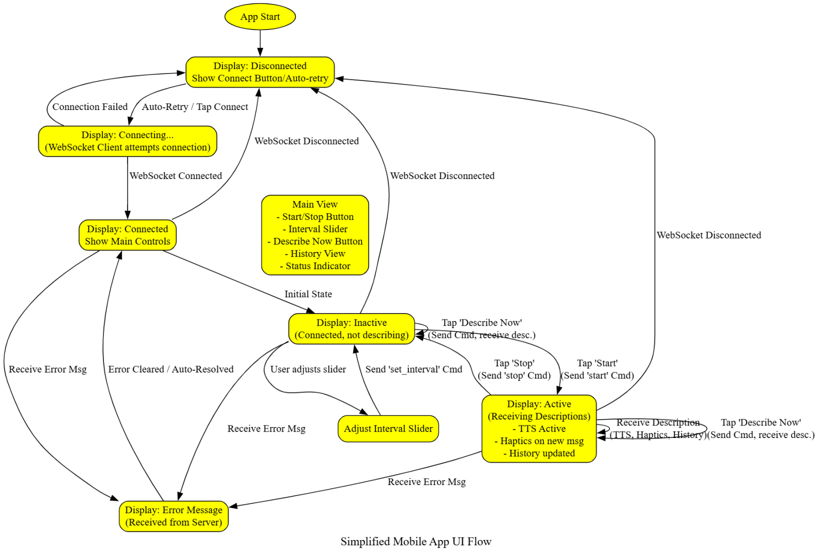
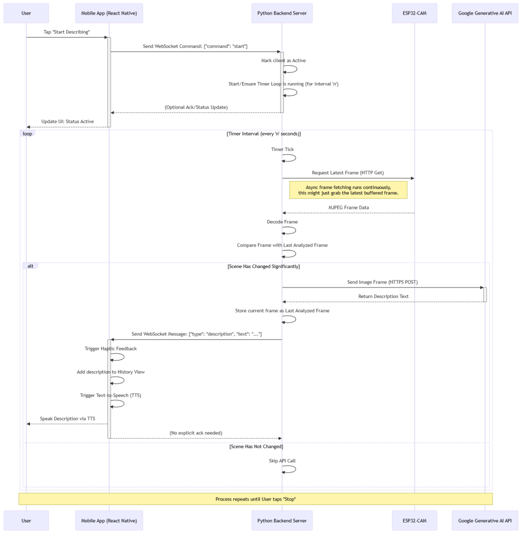
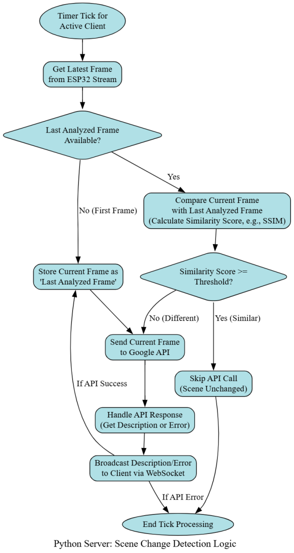


Figure: Simplified state transitions within the React Native application.

How Continuous Description Works


User
Mobile App (React Native)
Python Backend Server
ESP32-CAM
Google Generative AI API
Tap "Start Describing"
Send WebSocket Command: {"command": "start"}
Mark client as Active
Start/Ensure Timer Loop is running (for interval 'n')
(Optional Ack/Status Update)
Update UI: Status Active
loop
[Timer Interval (every 'n' seconds)]
Timer Tick
Request Latest Frame (HTTP Get)
Async frame fetching runs continuously,
this might just grab the latest buffered frame.
MJPEG Frame Data
Decode Frame
Compare Frame with Last Analyzed Frame
alt
[Scene Has Changed Significantly]
Send Image Frame (HTTPS POST)
Return Description Text
Store current frame as Last Analyzed Frame
Send WebSocket Message: {"type": "description", "text": "..."}
Trigger Haptic Feedback
Add description to History View
Trigger Text-to-Speech (TTS)
Speak Description via TTS
(No explicit ack needed)
[Scene Has Not Changed]
Skip API Call
Process repeats until User taps "Stop"
User
Mobile App (React Native)
Python Backend Server
ESP32-CAM
Google Generative AI API

Optimizing API Calls



Timer Tick for Active Client
Get Latest Frame from ESP32 Stream
Last Analyzed Frame Available?
Yes
No (First Frame)
Compare Current Frame with Last Analyzed Frame (Calculate Similarity Score, e.g., SSIM)
Store Current Frame as 'Last Analyzed Frame'
Similarity Score >= Threshold?
No (Different)
Yes (Similar)
Send Current Frame to Google API
If API Success
Skip API Call (Scene Unchanged)
Handle API Response (Get Description or Error)
Broadcast Description/Error to Client via WebSocket
If API Error
End Tick Processing
Python Server: Scene Change Detection Logic

Figure: Flowchart for deciding whether to call the AI API based on frame

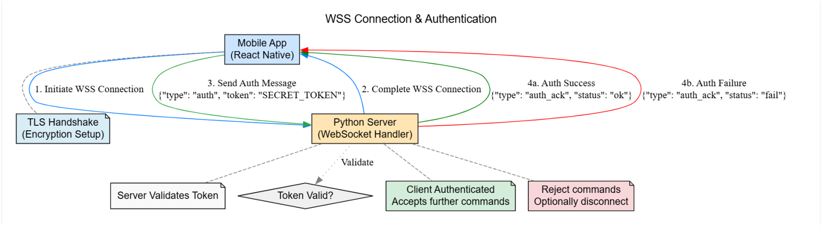
Key Technologies Used

- **Hardware:** ESP32-CAM
- **Video Streaming:** MJPEG over HTTP
- **Backend:** Python 3 with ‘asyncio‘, ‘websockets‘, ‘aiohttp‘/‘requests‘, ‘opencv-python‘, ‘scikit-image‘ (for SSIM), ‘google-generativeai‘, ‘python-dotenv‘.
- **Frontend:** React Native (JavaScript/TypeScript)
- **Communication:** Secure WebSockets (WSS)
- **AI:** Google Generative AI (Gemini API)
- **Output:** Text-to-Speech (TTS) Library (React Native)
- **Feedback:** Haptics API (React Native)

Risks and Mitigations (Trusted LAN Focus)

- **Network:** Assumes trusted WiFi (WPA2/WPA3). Public networks are risky.
- **ESP32 Stream (HTTP):** Unencrypted. Rely on WiFi security. HTTPS on ESP32 is challenging.
- **WebSocket (WSS):** *Mitigation:* Use Secure WebSockets (‘wss://‘) with TLS certificates to encrypt app-server communication.
- **Authentication:** *Mitigation:* Implement token-based authentication for WebSocket clients to prevent unauthorized access/commands.
- **API Key:** *Mitigation:* Store securely (e.g., ‘.env‘ file, environment variables). Secure the host machine.
- **DoS:** *Mitigation:* Implement rate limiting on WebSocket commands/connections on the server.
- **Input Validation:** *Mitigation:* Sanitize all commands/data received from the mobile app on the server.
- **Resource Management:** Ensure server cleans up resources on client disconnect.

Connecting Securely



WSS Connection & Authentication
Mobile App (React Native)
1. Initiate WSS Connection
3. Send Auth Message {"type": "auth", "token": "SECRET_TOKEN"}
2. Complete WSS Connection
4a. Auth Success {"type": "auth_ack", "status": "ok"}
4b. Auth Failure {"type": "auth_ack", "status": "fail"}
TLS Handshake (Encryption Setup)
Python Server (WebSocket Handler)
Validate
Server Validates Token
Token Valid?
Client Authenticated Accepts further commands
Reject commands Optionally disconnect

Figure: Sequence for establishing a secure WebSocket connection with token authentication.

Individual Contributions

- **Touhidul Alam Seyam:** Led project design and integration. Developed the core Python backend server including asynchronous stream handling, WebSocket communication (WSS), scene change detection, and Google Gemini API integration. Implemented core React Native application logic and features. Designed system architecture and workflow. Created diagrams and presentation structure.
- **Eftekhar Uddin:** Configured and flashed the ESP32-CAM firmware. Conducted initial video stream testing and debugging. Assisted with Python server frame fetching setup.
- **Mim Chowdhury:** Designed and implemented the user interface (UI) and user experience (UX) for the React Native mobile application. Integrated Text-to-Speech (TTS) and Haptics feedback components.
- **Mahin Ullah:** Performed integration testing between the Python server, mobile app, and ESP32-CAM stream. Researched WebSocket libraries and contributed to debugging network communication issues.
- **Bajpakhi Muntasir:** Contributed to project documentation,

VisionVault System Overview

- A wearable system leveraging ESP32-CAM for continuous video input.
- Python backend processes the video, interacts with AI, and manages secure mobile app communication (WSS).
- Optimizes AI calls using scene change detection.
- React Native app provides user control (Start/Stop, Interval, On-Demand), displays history, and delivers auditory (TTS) / tactile (Haptic) feedback.
- Incorporates security measures for operation on trusted networks.
- Provides a robust framework for real-time, AI-driven environmental description.

This design outlines the system; next steps involve implementation.