VisionVault: An Enhanced Wearable Camera System

From Manual Capture to Automatic Auditory Descriptions

Analysis based on Project Files

Project Overview

May 2, 2025

Outline

What is VisionVault?

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

- Transforms how moments are captured/documented.
- Allows users to stream live video.
- Capture images and generate descriptions via AI.
- Aims for hands-free operation.

Applications:

- Personal memory archival.
- Remote surveillance.
- Assistive technology.

Demo Image Placeholder

Original Workflow (ESP32 →Python →Console)

- **ESP32-CAM (‘camera-feed.ino‘):** Connects to WiFi, configures camera, streams MJPEG video via HTTP.
- **Python Script (‘take-pic.py‘):** Runs on host PC, connects to ESP32 stream, displays feed using OpenCV.
- **Manual Capture:** User presses 'c' key on the host PC.
- **AI Description:** Captured image (‘captured_image.jpg‘) sent to Google Gemini API.
- **Output:** Description printed to the host PC console.

Trigger Example (Original ‘take-pic.py‘):

```python
        # Check for key presses
        key = cv2.waitKey(1) & 0xFF

        # Save the image when 'c' is pressed
        if key == ord('c'):
            filename = f"captured_image.jpg"
            cv2.imwrite(filename, frame)
            print(f"Image saved as {filename}")
            # Create a separate thread to handle content generation
            threading.Thread(target=generate_content).start()
```

Limitations of Initial Concept

- Requires user interaction on the host PC (keyboard press).
- Not truly hands-free or mobile.
- Output is only text on the console.
- **Accessibility Challenge:** How does a visually impaired user know *when* to press 'c'?

Goals for Enhancement:

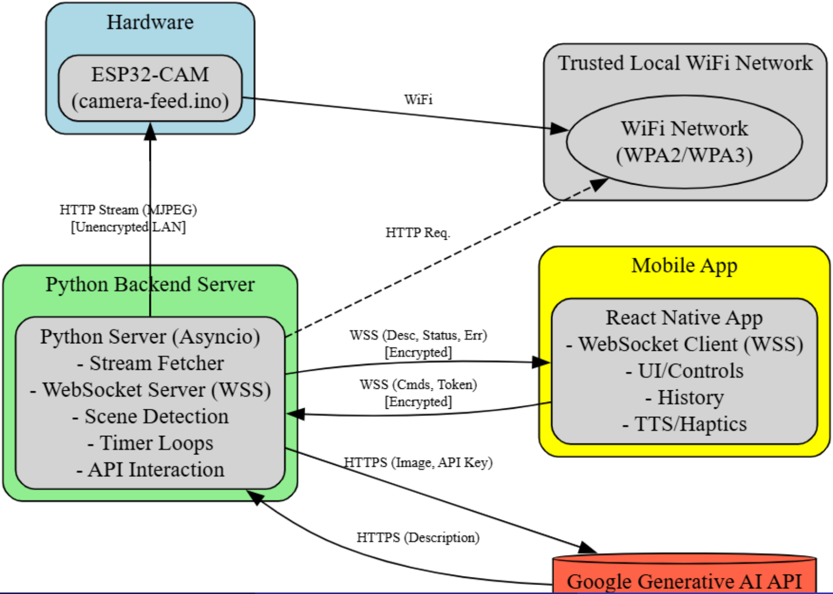
- Create a mobile interface (React Native).
- Provide auditory feedback (Text-to-Speech).
- Enable automatic, periodic descriptions.
- Add user controls via the mobile app.
- Improve robustness and add security.
- Optimize API usage (Scene Change Detection).

Mobile App →Python Server →TTS

Based on 'WORKFLOW.md':

- **ESP32-CAM:** Unchanged (streams video).
- **Python Backend Server:**
  - Fetches stream continuously (with reconnection).
  - Runs WebSocket Server (WSS) for mobile app communication.
  - Manages client state (active/inactive, 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 to Python Server (WSS, with reconnection).
  - UI: Start/Stop, Interval config, Describe Now, History, Status.
  - Sends commands to server.
  - Receives descriptions/errors.
  - Provides TTS output and Haptic feedback.

Overall Component Interaction



Modular Design

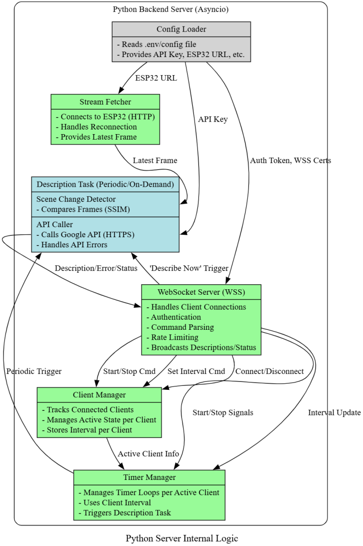


Figure: Key internal modules and data flow within the Python server.

User Interaction States

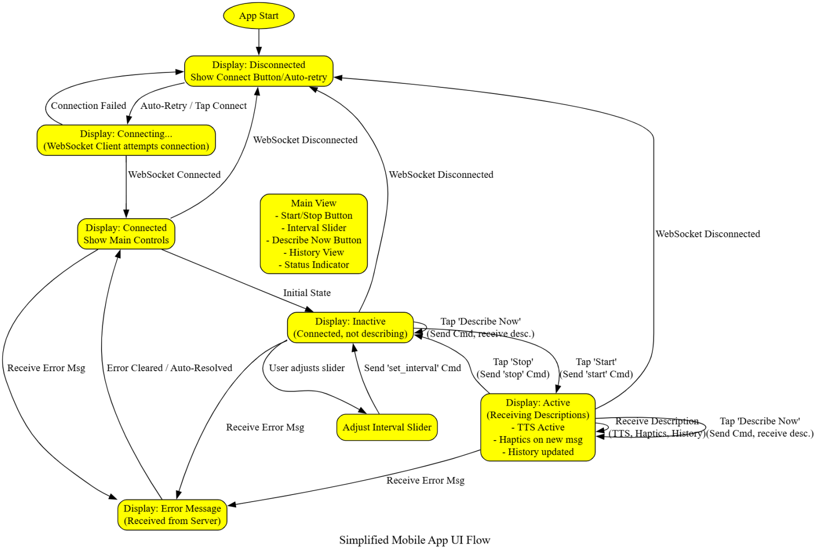
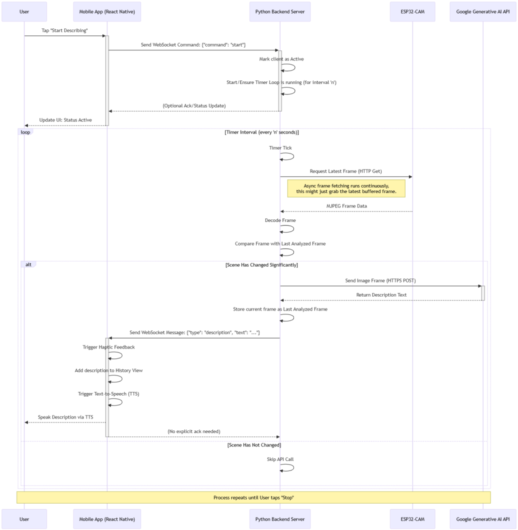


Figure: Simplified state transitions within the React Native application.

How Continuous Description Works



Optimizing API Calls

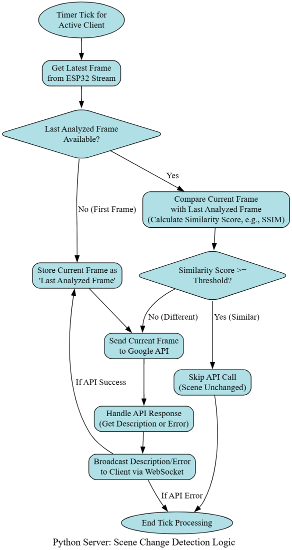


Figure: Flowchart for deciding whether to call the AI API based on frame

Key Technologies Used

Based on ‘WORKFLOW.md‘:

- **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)

From ‘camera-feed.ino‘ (Camera Configuration)

```
// ... (Pin definitions) ...

camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
// ... (Assign all camera pins) ...
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG; // Important for MJPEG
    stream

// Frame size and quality based on PSRAM
if(psramFound()){
  config.frame_size = FRAMESIZE_VGA; // Or other size
  config.jpeg_quality = 10; // 0-63 lower means higher quality
  config.fb_count = 2;
```

Risks and Mitigations (Trusted LAN Focus)

Highlights from ‘WORKFLOW.md‘:

- **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

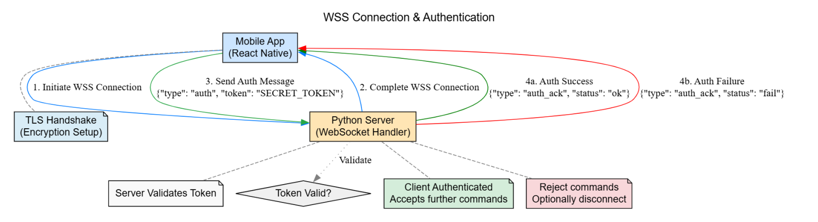


Figure: Sequence for establishing a secure WebSocket connection with token authentication.

VisionVault - Enhanced

- Evolved from a simple capture tool to an assistive technology concept.
- Utilizes ESP32-CAM for video streaming.
- Python backend manages stream, AI interaction, and app communication via WSS.
- React Native app provides user control, history, and auditory (TTS) / tactile (Haptic) feedback.
- Incorporates efficiency (scene change detection) and security (WSS, Auth) measures.
- Provides a robust framework for real-time environmental description.

Next Steps would involve actual implementation based on these designs.