Presentation Script: IntelliGaze: A Wearable AI Camera System

(Slide 1: Title Slide)

- **Mim:** "Hello everyone, and welcome to our presentation on IntelliGaze: A Wearable AI Camera System. We are a team from the Microprocessor Lab at BGC Trust University Bangladesh. My name is Mim, and I'm here with my teammates [List names: Touhidul Alam Seyam, Eftakar Uddin, Shafiul Azam Mahin, and Muntasir Rahman]. Our project focuses on providing Real-time Environmental Description via AI and TTS. We are grateful for the guidance of our supervisor, Junior Lecturer Radiathun Tasnia. Our presentation today is on May 2nd, 2025."

(Slide 2: Outline)

- **Mim:** "Here is the outline for our presentation today. We'll start with an introduction to IntelliGaze, cover the system's workflow and architecture, discuss the core logic and the technologies we've chosen. Then, we'll touch upon important security aspects, highlight our individual team contributions, look at the potential future benefits of this project, and finally, conclude our presentation."

(Slide 3: What is IntelliGaze?)

- **Mim:** "So, what is IntelliGaze? The core idea behind our project is to create an innovative, wearable system that integrates an ESP32-CAM, potentially mounted on goggles, to capture and document moments using AI. This system aims to provide real-time auditory descriptions of the user's surroundings, effectively giving them a hands-free way to understand their environment. While not the primary focus, it implicitly allows users to stream live video. The scene descriptions are generated using AI, specifically Google Gemini. And the overall system aims for hands-free operation, primarily controlled via a mobile app. In terms of applications, IntelliGaze is designed first and foremost as an assistive technology for the visually impaired. It can also serve as a contextual awareness tool, and even for personal memory archival, allowing users to revisit visual moments accompanied by AI-generated descriptions."

(Slide 4: Challenges Addressed)

- **Mim:** "IntelliGaze was conceived to address several significant challenges. Firstly, the daily difficulties faced by visually impaired individuals in navigating and fully understanding their dynamic environments in real-time. There's a clear need for immediate, hands-free contextual information about their surroundings. Many people also desire an accessible way to document and recall visual experiences without relying solely on traditional methods. Lastly, we observed that existing assistive technologies sometimes lack real-time, AI-powered descriptive capabilities or seamless, wearable integration into daily life."

(Slide 5: How IntelliGaze Helps)

- **Mim:** "IntelliGaze provides solutions to these challenges in several ways. It delivers immediate auditory feedback about the surroundings, powered by AI analysis and Text-to-Speech. It offers a comfortable, wearable, and hands-free experience via the goggles and mobile app interface. We leverage the sophisticated capabilities of Google Gemini for intelligent and detailed scene descriptions. This system creates a potential tool for significantly enhanced independence and richer environmental interaction for the visually impaired. Additionally, it serves as a novel personal memory archival system, capturing moments and automatically generating descriptive context."
- **Mim:** "Now, Mahin will discuss why IntelliGaze is needed and the system's structure."

(Slide 6: Why IntelliGaze is Needed)

- **Mahin:** "Building on Mim's points, IntelliGaze is particularly needed because it directly addresses the limitations of many existing assistive tools by providing *real-time*, *AI-driven* environmental descriptions, which isn't always standard. We achieve this using accessible hardware like the ESP32-CAM and powerful, yet cost-effective AI like Google Gemini, making it a capable solution without prohibitive cost. The hands-free, wearable form factor offers a seamless integration into a user's daily life. This combination provides significant value, particularly for visually impaired users, enhancing their navigation, safety, and contextual understanding. It also opens doors for broader applications as Mim mentioned."

(Slide 7: System Workflow)

- **Mahin:** "Let's look at the overall workflow of IntelliGaze. The process starts with the ESP32-CAM continuously streaming video over the local WiFi network. This stream is consumed by our Python Backend Server. The server is the central processing unit. It fetches the video stream and runs a secure WebSocket Server to communicate with the mobile app. It manages the state of connected clients – whether they are active or inactive, and their preferred description interval. In the 'Automatic Mode,' the server periodically checks the video stream for significant scene changes. If a change is detected, it sends the current frame to the AI API for analysis and then broadcasts the resulting description to all active mobile app clients. The server also handles direct commands received from the mobile app, such as 'start'/'stop' descriptions, 'set_interval', or requesting an immediate 'describe_now'."

(Slide 8: Overall Component Interaction)

- **Mahin:** "This diagram illustrates the high-level interaction between the main components. We have the Hardware, the ESP32-CAM, streaming video over the Trusted Local WiFi Network. The Python Backend Server connects to this network to fetch the stream. It uses HTTPS requests to interact with the Google Generative AI API, sending images and receiving descriptions. The Mobile App, a React Native application, connects to the Python Backend Server using Secure WebSockets (WSS). The app sends commands and receives descriptions, status updates, and error messages. Within the mobile app, the received descriptions are processed for UI display, added to history, and converted into auditory feedback via TTS, accompanied by Haptics for tactile confirmation."
- **Mahin:** "Next, Seyam will dive deeper into the internal design of the server and the user interaction flow."

(Slide 9: Modular Design)

- **Seyam:** "Thank you, Mahin. Moving into the internal workings of our Python Backend Server, we designed it with modularity in mind. This diagram shows the key modules and their interactions. We have a Config Loader to manage settings like API keys and the ESP32 URL. The Stream Fetcher is responsible for connecting to the ESP32's HTTP stream, handling reconnections, and providing the latest frame. The Description Task is crucial; it includes a Scene Change Detector that compares frames using a method like SSIM, and an API Caller that interacts with the Google API and handles potential errors. The WebSocket Server manages all client connections, handles authentication, parses commands, enforces rate limiting, and broadcasts information. The Client Manager tracks each connected client, their active state, and their requested description interval. Finally, the Timer Manager orchestrates the periodic Description Task for each active client based on their interval settings. All these modules communicate internally to deliver the core functionality."

(Slide 10: User Interaction States)

- **Seyam:** "This flowchart simplifies the state transitions within the React Native mobile application. When the app starts, it's in a 'Disconnected' state, typically showing a connect button or attempting auto-retry. Upon a successful WebSocket connection, it transitions to a 'Connected' state, initially 'Inactive' (meaning not automatically describing). In the Inactive state, the user sees the main controls. From here, they can tap 'Describe Now' for a single, immediate description, or tap 'Start' to move to the 'Active' state. In the 'Active' state, the app receives descriptions periodically from the server, triggering Text-to-Speech and Haptic feedback. The user can adjust the description interval or tap 'Stop' to return to the Inactive state. The app can receive error messages from the server in any connected state and display them accordingly."

(Slide 11: Optimizing API Calls)

- **Seyam:** "One critical challenge is managing the cost and latency associated with repeated API calls to the AI model. To optimize this, our Python server employs a scene change detection logic. As shown in this flowchart, during a periodic timer tick for an active client, we fetch the latest frame from the ESP32 stream. We then compare this current frame with the 'Last Analyzed Frame' (the one that was previously sent to the AI). We calculate a similarity score, for example, using SSIM. If the similarity score is high (above a defined threshold), it indicates the scene hasn't changed significantly, and we skip the API call.

Only if the scene has changed (similarity is low) do we send the current frame to the Google API. The current frame is then stored as the 'Last Analyzed Frame' for the next comparison. This process drastically reduces the number of API calls compared to analyzing every single frame."

(Slide 12: Key Technologies Used)

- **Seyam:** "To implement IntelliGaze, we've utilized a stack of technologies. The Hardware is the ESP32-CAM WiFi Module, a cost-effective camera board. Video Streaming from the ESP32 is done using MJPEG over HTTP. Our Backend is written in Python 3, leveraging libraries like `asyncio` for asynchronous operations, `websockets` and `aiohttp` / `requests` for networking, `opencv-python` and `scikit-image` for image processing including SSIM, `google-generativeai` for the AI interaction, and `python-dotenv` for managing environment variables. The Frontend is built using Expo React Native with TypeScript, enabling cross-platform mobile development. Communication between the backend and frontend is secured using Secure WebSockets (WSS). The core AI functionality relies on the Google Generative AI, specifically the Gemini API. The Output is provided as Text-to-Speech using the Expo TTS library, and we integrate tactile Feedback using the Expo Haptics API."
- **Seyam:** "Eftekhar will now walk us through the security aspects and our team's specific contributions."

(Slide 13: Connecting Securely)

- **Seyam:** "Ensuring secure communication between the mobile app and the server is important. This diagram illustrates the sequence we follow to establish a secure WebSocket connection with token-based authentication. The process begins with the Mobile App initiating a WSS connection, which involves a standard TLS handshake. This handshake sets up an encrypted tunnel. Once the secure tunnel is established, the Mobile App sends an authentication message containing a predefined secret token. The Python Server's WebSocket Handler validates this token. If the token is valid, the server responds with an 'auth_ack' message indicating success ('status: ok'). The client is now authenticated and authorized to send operational commands. If the token is invalid, the server responds with a 'fail' status and may disconnect the client. This token authentication adds a layer of security on top of the WSS encryption."
- **Seyam:** "Over to you, Eftekhar."

(Slide 14: Risks and Mitigations (Trusted LAN Focus))

- **Eftekhar:** "Thank you, Seyam. While IntelliGaze is designed with a focus on operation within a trusted local WiFi network, we've considered potential risks and implemented mitigations.
  - **Network:** We assume a trusted WiFi network using modern encryption like WPA2 or WPA3. Operating on public networks is inherently risky.
  - **ESP32 Stream (HTTP):** The video stream from the ESP32 is unencrypted HTTP. We mitigate this by relying on the security of the WiFi network itself. Implementing HTTPS directly on the ESP32 is challenging within the scope of this project setup.
  - **WebSocket (WSS):** For the crucial app-server communication, we use Secure WebSockets ('wss://') with TLS certificates to encrypt all data exchanged.
  - **Authentication:** We implement token-based authentication for WebSocket clients. Clients must provide a valid token upon connection to prevent unauthorized access or command execution.
  - **API Key:** The sensitive Google API key is stored securely on the server host machine, ideally in an environment variable or a `.env` file, never hardcoded or sent to the app. Securing the host machine is key here.
  - **DoS:** To mitigate potential Denial of Service issues, we plan to implement rate limiting on WebSocket connections and the frequency of commands processed by the server.
  - **Input Validation:** All commands and data received from the mobile app on the server side are sanitized and validated to prevent processing malicious or malformed input.
  - **Resource Management:** The server is designed to properly clean up resources associated with a client when they disconnect, preventing resource leaks."

(Slide 15: Individual Contributions)

- **Eftekhar:** "This project was a collaborative effort, and each team member played a vital role.
  - **Touhidul Alam Seyam:** Led the overall project design and integration. Developed the core Python backend server, including asynchronous stream handling, WSS communication, scene change detection, and Gemini API integration. Implemented core React Native app logic and features. Designed the system architecture, workflow, diagrams, and presentation structure.
  - **Eftakar Uddin (Myself):** Configured and flashed the ESP32-CAM firmware. Conducted initial video stream testing and debugging. Assisted with setting up the Python server's frame fetching module.
  - **Tasmim Akther Mim:** Designed and implemented the user interface (UI) and user experience (UX) for the Expo React Native mobile application. Integrated the Text-to-Speech (TTS) and Haptics feedback components.
  - **Shafiul Azam Mahin:** Performed integration testing between the Python server, mobile app, and the ESP32-CAM stream. Researched suitable WebSocket libraries and contributed significantly to debugging network communication issues.
  - **Muntasir Rahman:** Contributed to the project documentation. Researched Google Generative AI API capabilities and usage patterns relevant to the project. Assisted with overall system testing and validation."

(Slide 16: Potential Impact and Next Steps)

- **Eftekhar:** "Looking towards the future, IntelliGaze has significant potential impact.
  - **Enhanced Accessibility:** This is the most immediate impact, offering the potential for significant improvement in independence and quality of life for visually impaired individuals by providing crucial real-time environmental information.
  - **Broader Applications:** The core technology is adaptable. It can be applied in augmented reality overlays, professional training simulations requiring contextual awareness, or automated documentation in various fields.
  - **Technological Advancement:** IntelliGaze serves as a platform for integrating more sophisticated AI models in the future, such as dedicated object recognition, activity detection, or face recognition. We could also integrate sensor fusion, adding input from audio or GPS, or explore cloud integration for more powerful processing or data storage.
  - **Personalization:** Future versions could learn user preferences – for example, preferring descriptions of people over objects in a crowded room – or adapt descriptions based on context or the user's location."

(Slide 17: IntelliGaze System Overview / Conclusion)

- **Eftekhar:** "To summarize, IntelliGaze is a wearable system leveraging an ESP32-CAM for continuous video input. A Python backend processes this video, interacts with AI for descriptions, and manages secure mobile app communication via WSS. A key optimization is the scene change detection to reduce API calls. The React Native app provides comprehensive user control – allowing starting/stopping, setting intervals, requesting on-demand descriptions, displaying history, and delivering auditory (TTS) and tactile (Haptic) feedback. We've incorporated security measures focusing on operation on trusted networks. Overall, IntelliGaze provides a robust framework for real-time, AI-driven environmental description. This presentation has outlined the system design and architecture, and our next steps involve the detailed implementation and rigorous testing of all the components described. We are excited about the potential of this project."
- **Eftekhar:** "Thank you all for your time and attention. We are now ready to answer any questions you may have."