NAIT

Edmonton, Alberta

**Geek Goggles**

Submitted To

Mr. Gary Munro, Instructor

English and Communication Department

&

Mr. Aj Armstrong, Instructor

Computer Engineering Technology

Submitted By

Hayden Seivewright, Student

&

Joshua Akinmoluwa, Student

CMPE 2965

Computer Engineering Technology

March 28, 2025

**STATEMENT OF CONTRIBUTION**

This document was co-authored by Hayden Seivewright and Joshua Akinmoluwa. The list of contributions in the table below were reviewed for accuracy by both participants, and we view the division of effort to be equivalent.

|  |  |  |
| --- | --- | --- |
| Section | Author | Reviewer |
| Title Page, Letter of Transmittal | Hayden Seivewright | Joshua Akinmoluwa |
| Abstract | Hayden Seivewright | Joshua Akinmoluwa |
| Introduction | Hayden Seivewright | Joshua Akinmoluwa |
| Overview | Hayden Seivewright | Joshua Akinmoluwa |
| Physical Design | Hayden Seivewright | Joshua Akinmoluwa |
| Software Design | Joshua Akinmoluwa | Hayden Seivewright |
| Web Application Design | Joshua Akinmoluwa | Hayden Seivewright |
| Database Design | Joshua Akinmoluwa | Hayden Seivewright |
| Embedded System Design | Hayden Seivewright | Joshua Akinmoluwa |
| Implementation | Hayden Seivewright | Joshua Akinmoluwa |
| Project Results | Hayden Seivewright | Joshua Akinmoluwa |
| Conclusion | Joshua Akinmoluwa | Hayden Seivewright |
| User Manual | Hayden Seivewright | Joshua Akinmoluwa |
| Bill of Materials | Hayden Seivewright | Joshua Akinmoluwa |

13823 138 Ave NW

Edmonton, AB

T6V0M1

March 28, 2025

Mr. Gary Munro and Mr. AJ Armstrong

Instructors

NAIT

11762 – 106 Street NW

Edmonton, AB T5G 2R1

Dear Mr. Munro and Mr. Armstrong:

As per the requirements of CMPE2960, we are submitting the report titled Geek Goggles for evaluation.

This report provides the details of the research, design and final implementation of this project. It reviews the challenges and triumphs encountered doing the development of the project. This report also examines the technical aspects for the software and hardware components included in the project.

We are proud of this project and all the tools and techniques we learned along the way. We hope others can find this project as interesting and useful as we have.

We would like to like to extend a massive thank you to all the instructors at NAIT who have taught us so much over the years and help us grow into the technicians we now are. We would also like to thank our peers for the support, camaraderie and motivation they have provided. Finally, we would like to thank our family and friends for supporting us through our educational journey and for putting up with all stress and late nights.

Sincerely,

A black background with a black square

AI-generated content may be incorrect.

A black background with a black square

AI-generated content may be incorrect.

Hayden Seivewright & Joshua Akinmoluwa

CNT Students

**TABLE OF CONTENTS**

[**STATEMENT OF CONTRIBUTION** ii](#_Toc195567337)

[**LIST OF FIGURES AND TABLE** v](#_Toc195567338)

[**ABSTRACT** vii](#_Toc195567339)

[**1** **INTRODUCTION** 1](#_Toc195567340)

[**2** **OVERVIEW** 2](#_Toc195567341)

[**3** **DESIGN** 3](#_Toc195567342)

[**3.1** **Physical Design** 3](#_Toc195567343)

[**3.1.1** **Heads Up Display Research** 3](#_Toc195567344)

[**3.1.2** **Microprocessor Research** 6](#_Toc195567345)

[**3.1.3** **Sensor Research** 6](#_Toc195567346)

[**3.2** **Software Design** 7](#_Toc195567347)

[**3.2.1** **Web Server** 8](#_Toc195567348)

[**3.2.2** **Database Access Layer** 8](#_Toc195567349)

[**3.2.3** **WebSocket Server** 8](#_Toc195567350)

[**3.2.4** **MQTT Broker** 9](#_Toc195567351)

[**3.2.5** **ESP32 Firmware** 10](#_Toc195567352)

[**3.3** **Web Application Design** 10](#_Toc195567353)

[3.3.1 **User Interface** 11](#_Toc195567354)

[**3.3.2** **System Functionality** 11](#_Toc195567355)

[**3.3.3** **Communication with Backend** 12](#_Toc195567356)

[**3.4** **Database Design** 13](#_Toc195567357)

[**3.4.1** **Entity Framework** 14](#_Toc195567358)

[**3.4.2** **Database Schema Design** 14](#_Toc195567359)

[**3.5** **Embedded System Design** 16](#_Toc195567360)

[**3.5.1** **Language Choices** 16](#_Toc195567361)

[**3.5.2** **UI Design** 17](#_Toc195567362)

[**3.5.3** **Voice Recognition** 22](#_Toc195567363)

[**4** **IMPLEMENTATION** 24](#_Toc195567364)

[**4.1** **Communication** 24](#_Toc195567365)

[**4.1.1** **Server and Pi Communication** 24](#_Toc195567366)

[**4.1.2** **Pi and Peripheral Communication** 25](#_Toc195567367)

[**4.2** **Physical Construction** 25](#_Toc195567368)

[**5** **PROJECT RESULTS** 26](#_Toc195567369)

[**5.1** **Successes** 26](#_Toc195567370)

[**5.2** **Challenges and Changes** 27](#_Toc195567371)

[**6** **CONCLUSION** 27](#_Toc195567372)

[**7** **REFERENCES** 29](#_Toc195567373)

[**8** **APPENDICES** 32](#_Toc195567374)

[**Appendix A: Glossary** 32](#_Toc195567375)

[**Appendix B: Schematics** 33](#_Toc195567376)

[**Raspberry Pi Embedded System** 33](#_Toc195567377)

[**ESP32 Voltmeter Peripheral** 34](#_Toc195567378)

[**Appendix C: Bill of Materials** 35](#_Toc195567379)

# **LIST OF FIGURES AND TABLE**

[Figure 1 - Heads up display optics 5](#_Toc195481994)

[Figure 2: Project Data Sending Over WebSocket 9](#_Toc195481995)

[Figure 3: Projects Webpage 11](#_Toc195481996)

[Figure 4: Sample AJAX Function 12](#_Toc195481997)

[Figure 5: Login using AJAX-CRUD 13](#_Toc195481998)

[Figure 6: Database Schema 15](#_Toc195481999)

[Figure 7 - Note Table Model 16](#_Toc195482000)

[Figure 8 - State Machine 18](#_Toc195482001)

[Figure 9 - Decibel Calculation 20](#_Toc195482002)

[Figure 10 - Air Quality Calculation 21](#_Toc195482003)

[Figure 11 - Speech-to-intent function 23](#_Toc195482004)

# **ABSTRACT**

When working on an electronics project, a technician, hobbyist or engineer may need to reference manuals or datasheets multiple times through-out a project. They may also need to read data from multiple instruments, or they may need to record and document the projects progress. All of these activates can pull a user out of a focused state and become distractions. Geek Goggles aims to provide a hands-free solution to all these distractions so a user can stay focused on the current task. To achieve this solution, Geek Goggles has three main components, a web app, head up display safety glasses, and a peripheral instrument to read voltages. The web app allows users to create and track projects and stores all info in a MySQL database. The current projects info is then uploaded to the Raspberry Pi powered safety glasses via web sockets which allow the user to view and record data. Finally, a ESP32 is used as a proof-of-concept voltmeter peripheral that streams voltage readings to the glasses via MQTT.

This project required multiple components and technologies to come together to be successful. There were multiple technologies to research and many challenges to overcome from HUD optics, to wireless communication, and even AI assisted voice-to-text. In the end we were able to create a prototype that achieved our goals but going forward we would love to enhance the optics so the user can more clearly see the display, update the physical design to be more aesthetically pleasing and develop a native phone app so the device can communicate via Bluetooth.

# **INTRODUCTION**

It is frustrating while working with electronics to constantly be needing to double check datasheets and device pinouts, a more technological solution is required. Most people solve this problem by printing out datasheets or pulling them up on a computer monitor, these solutions are less than optimal since desk space is usually at a premium while working and looking up at a monitor can be a distraction. Geek Goggles aims to modernize this approach by combining a heads-up display, voice commands, photography and electronic tools into one device.

This project has three main elements. First, the user must be able to upload various documents and be able to view them in a heads-up display. Secondly, the user must be able to attach a peripheral device to get live data from. Finaly, the user must be able to record a project via pictures and notes. This projected faced many challenges and some major pivots in both scope and design, from failed delivery of parts to ditching Bluetooth as the primary form of wireless communication. In the end, Geek Goggles achieved the goal of provided a convenient, hands-free solution to recording a project and viewing documents.

The purpose of this report is to discuss the decisions made, the challenges faced, and how Geek Goggles were created. This report will first explore a general overview of the project. It will discuss the design of the web app, then both the software and physical design of the safety glasses, and finally the Voltmeter. This report will then discuss the implementation of all three parts into a single, unified system. Finally, it will review the successes and failures of the project and our overall conclusions.

# **OVERVIEW**

Geek Googles consist of three elements, the web app, the glasses and the peripheral voltmeter. Each one of these elements interact with each other and can discussed separately for easier understanding.

The first element is the web app. The web app is an application that can run on either a phone or a computer and is the primary location where a user can interact with their project. The web app consists of a front-end UI designed with JavaScript and CSS, a back-end server designed with ASP.NET and a MySQL database. The web-app is where a user can login, create a new project and upload any important documents or notes to the glasses. These project documents are then transmitted to the glasses on start up through a web socket so they can be viewed. Once the user is finished working on their current project and shut down the glasses, any new notes, pictures or audio files recording during their session are uploaded back to the web app.

The second element is the glasses themselves. The glasses are powered by a Raspberry Pi Zero 2 W, which provides both processing power as well as wireless communication. The Pi is then connected to two primary sensors, a MEMS microphone for recording and sensing sound and a BME680 air sensor, which tracks temperature, air quality, humidity and pressure. These sensors are used to relay safety warnings such as excessive noise and poor air quality, as well as relay important data to the user. The Pi also is connected to a camera allowing the user to take pictures and document their project. Finally, all this data is displayed with a 0.2-inch micro display which is magnified and collimated through a series of lens and then reflected off a combiner to provide a Heads-up Display (HUD) that the user can view at a comfortable viewing distance. The user can interact with the device via 2 physical buttons or by using voice commands.

The final element is a proof-of-concept peripheral instrument that records data and live streams it to the glasses. Currently, this peripheral is a voltmeter that allows the user to get voltage readings from the circuit or electrical project they are working on. These readings are then published to a MQTT server on the web app to which the glasses receive the data from, allowing the user to view them real time without needed to look up from their current work.

# **DESIGN**

## **Physical Design**

The physical design section will discuss the design, research, and sourcing of the physical components the make up the Geek Goggles device. The schematics and bill of materials used in the physical design can be found in the appendices.

### **Heads Up Display Research**

The first step of creating the physical design was to begin researching and ordering the required parts. There were four main components that needed to be researched, the optics, the CPU/MCU, the air sensor and the microphone. One major component that required research was the optics needed to create the heads-up display (HUD). There are roughly four types of optics used for HUDs, but it is difficult to categorize since most designs use some form of hybrid of each. The four main styles considered for Geek Goggles were birdbath designs, prism based, combiner based and waveguides. The two main considerations were cost and time to acquire as both time and funding were very limited resources. Using a Waveguide would lead to the highest quality and many modern AR and HUD glasses such as Meta Orion, Vuvix blade, and Magic leap all use this technology (Simard, 2024). The problem with waveguides is they are very expensive, with some options ranging from $800 to $4000 (Displaymodule, n.d.). Based off the project’s constraints, either a combiner-based approach or a prism based one were the best options due their price. After some more research, it appeared that many hobbyists gravitate towards a combiner-based approach and there were many examples online of what a combiner design would look like, including a video series breaking down various designs (Mañolo, 2025). A combiner is a piece of semi-reflective material that can reflect light while also letting some light through provided a semi-transparent mirror like surface perfect for a HUD (Guttag, 2016). After deciding on a hybrid style combiner that would contain a micro-display which gets magnified through a Fresnel lens and then made clearer with a collimating lens before reflecting off the combiner. These multi lens system require extreme precision and some in-depth knowledge of optical design and physics which were outside of the scope of this project and still very costly. The diagram in figure 1 describes how this hybrid system projects an image.

Figure 1 - Heads up display optics

Diagram of a diagram of a human head

AI-generated content may be incorrect.

Source 1: Blanche (N.D)

Figure 1 shows an example of how a collimating and Fresnel lens can produce a holographic image at a comfortable viewing distance. After having a conversation with Mañolo Mancelli (Mancelli, 2025), from the Mañolo youtube channel he pointed us in the direction of ordering a deconstructed viewfinder from a digital camera off AliExpress. These view finders already have the three parts of the combiner system required, a micro-display, Fresnel lens and collimating lens all already designed to work with each other and at a fraction of the cost of developing one from scratch. Shortly after this conversation, there was a suitable deconstructed viewfinder found on AliExpress, and it was ordered.

### **Microprocessor Research**

The next system that needed research was the microprocessor needed to process data and render the display. Based on previous research, size constraints and the fact that the project would require some form of wireless communication there were two options, the Raspberry Pi Zero or a ESP32 based controller. As the project was also going to require a lightweight AI Voice recognition model to allow for voice commands, the processer would have to be powerful enough to run this model. This requirement narrowed the choice to the Raspberry Pi Zero 2 W and the ESP32-S3. The pre-built heads-up display optics used a composite video input, which the Pi would natively support but the ESP32-S3 would require extra modification or external modules to run. Due to there still being some unknowns, both the Raspberry Pi and the ESP32-S3 were ordered and the ESP32 would eventually be used to run the peripheral Voltmeter device.

### **Sensor Research**

The final system needing to be researched was the sensors required to give extra functionality to the glasses. There were two main functions required, an air quality sensor to give air quality warnings and a microphone to enable voice commands and sound level warnings. For the air quality sensor, after a quick search on Digikey (Digikey, n.d.), the BME680 was selected since it provided even more functionality than originally expected. It provided temperature, humidity, air quality and pressure all in one which allowed extra features to be added to the project with no increase in cost.

As for the Microphone, the project required something that could clearly record voice audio as well as sense environmental noise levels. There are two main kinds of microphones that were considered, a micro-electro-mechanical system (MEMS) style or an electret condenser microphones (ECM) style. The MEMS emerged as the clear choice due to its price, size and availability (Rose, 2021). The first mic ordered was a Adafruit I2S SPH0645 as it was the first one to appear in a MEMS I2S search but unfortunately it arrived broken, after looking for a different mic the INMP441 was found on Amazon and appeared to be the higher quality version of the original choice.

## **Software Design**

The software component of this project is responsible for the enabling seamless communication and interaction between the user, the Raspberry Pi, the database and the ESP32 microcontroller. This project allows the user to monitor, record and display voltage readings, project notes, and upload pictures through the web user interface, while also supporting real time picture capture, voice controls and a display via the Geek goggles.

The major components of the system are:

* Web Server
* Database Access Layer
* WebSocket Server
* MQTT Broker
* ESP32 Firmware

### **Web Server**

The web server powering this project is built using ASP.NET Core and this serves as the brain for the entire system. This technology efficiently handles all the HTTP requests from the website and serves as the central coordination point for all operations. The server implements a RESTful API framework with proper endpoints for communication between the frontend and the backend. The server also maintains database connections through Entity Framework Core, implementing efficient data access.

### **Database Access Layer**

The database access layer manages all the interactions with the MySQL database. Entity Framework Core serves as the object relational mapper (ORM), creating a level of abstraction between the application code and their respective database operations. This framework helps transform complex database domain models into properly database tables. Performance optimization techniques like eager loading, lazy loading, and query projection are strategically implemented to minimize database roundtrips and reduce response times for data-intensive operations (Microsoft, n.d.).

### **WebSocket Server**

The WebSocket server establishes a persistent, low-latency communication channel between the web server and connected clients, such as the Raspberry Pi. This enables the server to instantly push real-time updates like login confirmations or current project details directly to the devices without requiring repeated polling.

A screenshot of a computer program

AI-generated content may be incorrect.For the project, the web socket server is pushing critical data to the client devices. Upon successful login and project selection, all the project data and its related user data is sent to the Pi over the web socket on the click event of ‘sync’ button in the web page. The code snippet in figure 2 shows how this is accomplished. Figure 2 shows a data frame being created and converted to JSON which is then UTF8 encoded and broadcast via the connected socket. This real-time synchronization helps the Pi to keep track of the project currently worked on and all its related information.

Source 2 - (Seivewright & Akinmoluwa, 2025)

Figure 2 - Project Data Sending Over WebSocket

### **MQTT Broker**

The MQTT Broker serves as a lightweight and efficient messaging medium between the ESP32, the Server, and the Raspberry Pi. Designed for low-bandwidth and high-latency environments, MQTT (Message Queuing Telemetry Transport) is well-suited for IoT applications where power efficiency and minimal overhead are critical (MQTT, n.d.).

In this project, the broker acts a middleman, receiving the voltage readings from the ESP32 and passing the readings over to the Pi. This model does not need the ESP32 to maintain a constant connection to the broker or be aware of which clients are subscribed. It simply publishes the reading to ‘esp32’ topic, and the message is received by the broker and any client connected can retrieve the message.

### **ESP32 Firmware**

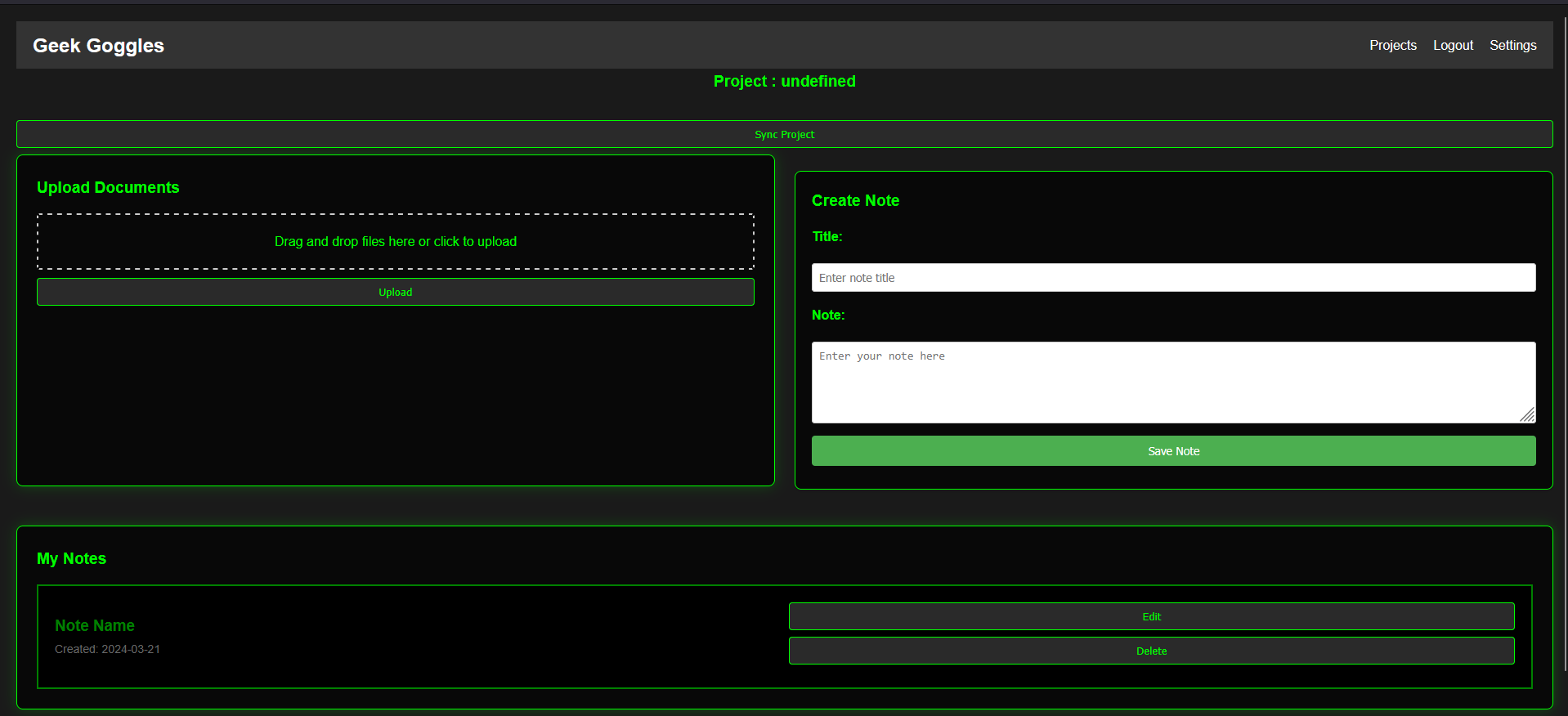
The ESP32 microcontroller serves as the project’s voltmeter. This embedded device utilizes its built-in Analog-to-Digital-Converter (ADC) to continuously sample voltage readings from connected sensors. The wiring schematic for the voltmeter can be viewed in appendix B. These readings are processed for transmission through Wi-Fi.

Once the ESP32 is connected to the local network, the ESP32 establishes connection to the MQTT broker where it publishes the readings to the ‘esp32’ topic. In this case the ESP32 only acts as a data source and does not need to receive any incoming data. Since the microcontroller operations are handled locally, latency is reduced. This design enhances the system’s responsiveness, reliability, and overall flexibility in real-world deployment scenarios.

## **Web Application Design**

The web application serves as the medium that enables the user to interact with the project, built with HTML, CSS, Javascript and JQuery. The user can interact with the project by performing actions like logging in, managing projects, taking notes and uploading pictures. It is designed to be responsive across different devices.

### **User Interface**

The user interface is how the user interacts with the web application and is designed to be responsive and accessible across devices. As seen in the Figure 3, the User interface is organized into different modules such as the project dashboard, note-taking and management sections and picture upload interface. In addition, it also provides visual feedback to guide the users whenever possible.

Source 3 - (Seivewright & Akinmoluwa, 2025)

Figure 3 - Projects Webpage

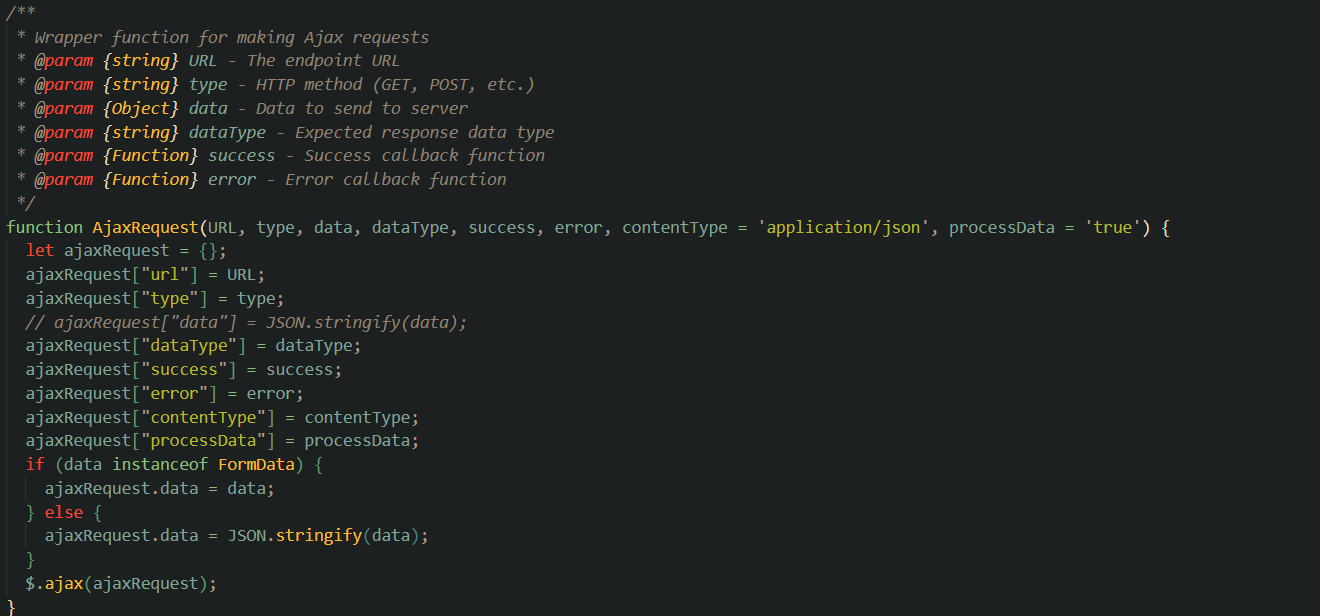
### **System Functionality**

The system employs a proper client-side storage approach, utilizing both localStorage and sessionStorage to efficiently store and keep track of user and project information. Upon successful login, the session data is securely stored as a session variable ensuring proper data management across page navigations. Whenever a user logs out, the stored data is completely cleared. This facilitates proper user data handling and clearing while preventing unauthorized access in shared devices scenarios and reducing potential webpage vulnerabilities.

### **Communication with Backend**

The web application uses AJAX calls for CRUD operations such as:

* Logging in / out
* Creating or switching projects
* Uploading images
* Managing notes

After selecting a project, the user is presented with a button at the top of the screen to ‘sync’ the project to the Raspberry Pi. Upon clicking this, an ajax call is made which in turn sends all the project data to the Pi. An example of the ajax call function can be seen in figure 4 along with how it is called once a user clicks login in figure 5.

Source 4 - (Seivewright & Akinmoluwa, 2025)

Figure 4 - Sample AJAX Function

Figure 5 shows the function, “loginUser” that is called when a user clicks the “login” button, this function grabs the user’s inputted username and password and sends it to the server using the “AjaxResquest” function seen in figure 4.

Figure 5 - Login using AJAX-CRUD

Source 5 - (Seivewright & Akinmoluwa, 2025)

## **Database Design**

The project required a database to store and manage its users and their projects data. When researching databases systems for the project, MySQL stood out as the one with the most examples and with its proven performance with high-end applications like Facebook and Twitter (MySQL, n.d.). MySQL was chosen as the database system and Entity Framework (EF) was used in code-first approach for easy integration with C# and allows for easy database schema changes through migrations.

### **Entity Framework**

The Entity Framework is an open-source framework developed by Microsoft that is available to the .NET platform, it uses a code-first approach to map and interact with a database (Patel, 2024). The code-first approach was selected for the project to define the database models directly with C# Models. This approach helps simplifies the database management by enabling easy updates through EF Migrations. In the approach the database creation and modification are handled by C# classes thereby reducing dependency on SQL Scripting.

### **Database Schema Design**

The database consists of several key tables that store the different aspects of the projects functionality. The major tables and their relationships are shown in figure 6.

Figure 6: Database Schema

A screenshot of a computer

AI-generated content may be incorrect.

Source 6 - (Seivewright & Akinmoluwa)

The Users table contains the user’s data while Projects table contains information about the user’s current project. Readings tables capture real-time data gotten by the Geek Goggles while MyFiles and Notes tables tracks and records documents and pictures either taken by the device or saved in the device for the current project being worked on. The code snippet in figure 7 shows the class which represents the model for the Notes table and its relationship with Projects table.

A computer screen shot of a program code

AI-generated content may be incorrect.In figure 7 you can see how a class is used to create the note table as well as the fields inside of it. A project can have multiple notes which is represented by the Project Member in the Model. The Id is used as a unique identifier for each Project Note.

Source 7 - (Seivewright & Akinmoluwa, 2025)

Figure 7 - Note Table Model

## **Embedded System Design**

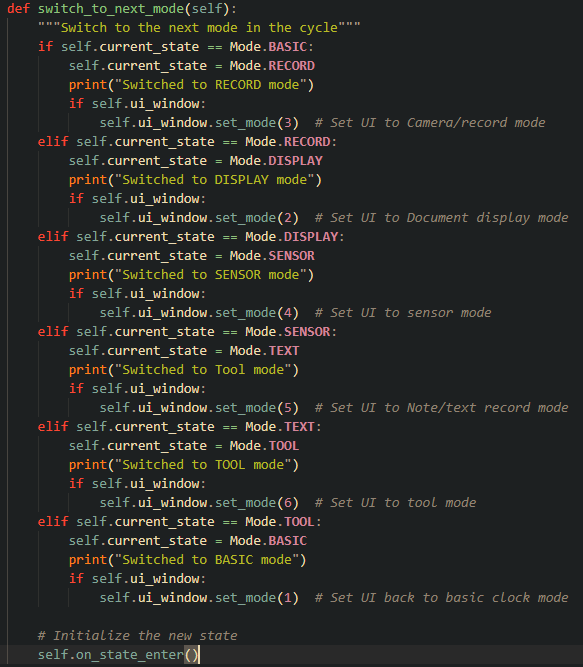
The embedded system design section will discuss the technologies, libraries and code design that were used in the development of the Raspberry Pi Zero 2 w microprocessor found in the Geek Goggles.

### **Language Choices**

The primary programming language used to develop the software found in the Geek Goggles embedded system is Python. Originally, the C language was used in the early stages of development but due to an unfortunate sensor failure was dropped. Early on when doing building a simple proof-of-life program for the SPH0645 microphone progress came to a halt after the microphone was powered and its address was accessible by the microprocessor, but no audio data was able to be read. After hours of debugging and rewiring with no success it was found that most of the information and tutorials online for SPH0645 was written in python, so the choice was made to rewrite the existing code in python to see if that could get it to work. This rewrite included the existing BME680 sensor, input/output (IO), and basic UI code. Even after making these changes and multiple online tutorials, the microphone was still not working (lady ada, 2017; Hrisko, 2020). Finally, it was decided that the microphone was faulty and a new, higher quality INMP441 microphone was ordered. This new microphone worked right away with the existing Python code and at this point the entire code base was already rewritten so it was decided that going forward that Python would be the language of choice.

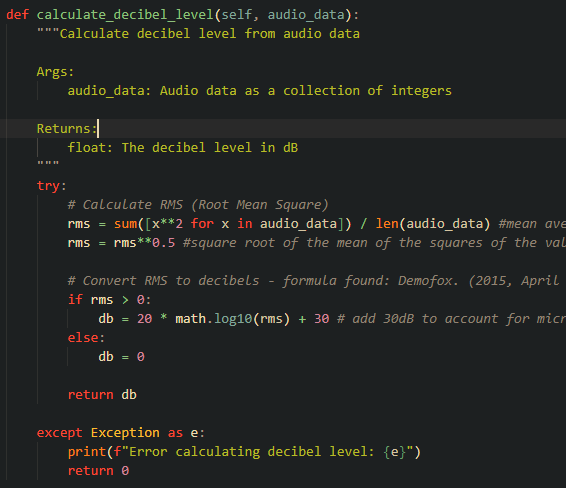
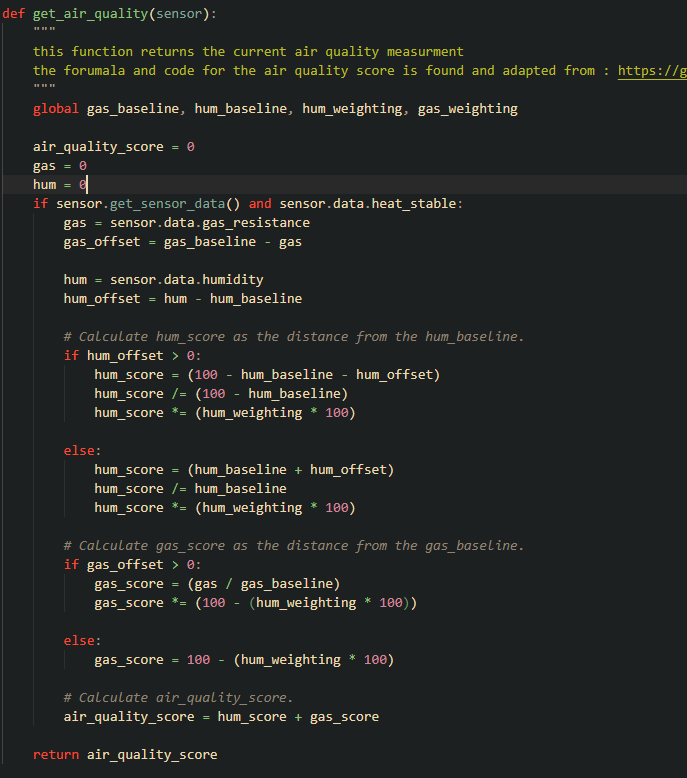
### **UI Design**

After a language was settled on, the next decision was what technology or framework was going to be used to render the Graphical User Interface (GUI) that will display information to the user through the HUD. The three frameworks that were looked were Kivy, Tkinter, and PyQT. Kivy is a popular GUI framework, but its focus is mainly on touch style applications on mobile devices which is not the kind of design that was needed (Kivy, n.d.). The next framework looked at was Tkinter, which is the default GUI built into Python. Tkinter was used for a simple test program but there were troubles having multiple screens display for all the different modes that the Geek Goggles required. After the short test with Tkinter, PyQt was given the same simple test. There were plenty of tutorials and well developed documentation found online and after following a simple beginner tutorial (Fitzpatrick, 2025), PyQT achieved exactly what was needed and was used going forward to design the GUI.

The Geek Goggles user interface required multiple modes so the user can select the different functions of the Goggles. This requirement lent itself to a state machine model where each state would be a mode in the UI. The state machine was design with six states that reflect the 6 modes that the user could select. These modes are the clock mode, camera mode, document display mode, sensor mode, note record mode, and finally the tool mode. The function that allows the states to switch can be reviewed in the figure 8. The clock mode, referred to as “BASIC” in the code, is the initial state that the goggles enter on start up. This is the recommended mode that users stay in while working on a project as it only displays the current time and is visually unintrusive. The second mode is the camera mode, referred to as “RECORD” mode in the code, which displays the view from the camera and allows the user to take a photo by with clicking the “action button” or by using a voice command. The third mode is the document display mode, referred to as “DISPLAY” mode in the code, this is where the use can view any uploaded documents, images or notes that were previously added to the project via the web application. The fourth mode is the sensor mode which displays the current temperature and humidity data from the BME680 sensor. The fifth mode is the note recording mode, referred to as “TEXT” mode in the code, which allow the user to record a short ten second note that will be transcribed to a text note on the project file. The sixth and final mode is the tool mode. The tool mode displays live data from the connect peripheral device, which in the prototype is a voltmeter the sends the data via MQTT to the goggles.

Source 8 -(Seivewright & Akinmoluwa, 2025)

Figure 8 - State Machine

The final addition to the UI is the safety alerts. The safety alerts are semi-transparent overlays the pop-up over-top of the current display. There are two safety alerts, the first activates when the decibel level of the environment exceeds 90db which the World Health Organization (WHO) states can damage hearing after 4 hours per week (WHO, n.d.). To produce this alert the microphone takes a sample the environments noise every 30 seconds in a separate thread. To calculate the decibel level the root mean square (RMS) of the samples is required, so the samples are squared then the average of the squared samples is squared rooted to get the RMS. Then the decibels are found by using the standard formula for converting amplitude to decibels, the code snippet in figure 9 shows this in action (demofox, 2015). In figure 9 you can see that the squares of the audio samples recorded are summed up and then the rms is calculated by finding the square root of the sum which then gets converted to decibels. The second alert activates when the air quality index (AQI) exceeds an index level of 150 which the United States Government deems an unhealthy level (AirNow, n.d.). To produce this alert the BME680 sensor takes a sample of the air quality every 30 seconds and based off the baseline gas reading, which is calculated on start up, the current humidity, and the gas reading, produces an air quality rating. The code snippet in figure 10 shows how this calculation was implemented. The calculation creates a humidity score based off the deviation from the baseline and a gas score based off the deviation from the baseline then adds them together with a 25% and 75% weighting respectively (Primoroni, 2024).

Source 9 - (Seivewright & Akinmoluwa)

Figure 9 - Decibel Calculation

Source 10 - (Seivewright & Akinmoluwa)

Figure 10 - Air Quality Calculation

### **Voice Recognition**

The final technology to be chosen was the AI model to be used for voice commands and voice-to-text. The two main constraints were that it needed to be lightweight enough to run locally on the Pi and it needed to be free to use. The first model looked at was Goggles voice model, which unfortunately recently required a credit card to activate but would still be relatively cheap for the usage the Geek Goggles would require but this fact disqualified it as an option (Google, n.d.). The next option looked at was OpenAI’s whisper text-to-speech model. This model is free for non-commercial use but does not run well locally on a raspberry pi and would require some extra programming to get things like wake words or voice commands working (Heer, 2024; kkielhofner, 2023). Finally, Picovoice was found and it fit the required use case perfectly. Picovoice is a AI voice company that runs various light-weight models for different use cases. They have a “wake word” model called Porcupine used to trigger events when certain words are spoken, a voice-to-intent model called Rhino and a speech to text model called Leopard the filled the exact three uses that Geek Goggles required. Best of all, these models, except for Leopard, are designed to run on the Pi zero and they are free to use for non-commercial projects up until a certain threshold of tokens. Picovoice also provided documentation with examples on how to get these models up and running on the Raspberry Pi and Leopard running on the server (Picovoice, n.d.).

The wake word system works by using the Porcupine model to constantly run in a separate thread listening for the wake word that was trained, “Hello Geek”. Once the wake word is detected the system then triggers the Rhino voice-to-intent model to pick up the next words spoken. The Rhino model allows the training of specific intent words which when picked up by the microphone can be used to trigger certain actions in code. These “intents” and how they trigger functions can be viewed in figure 11. In figure 11 you can see that when a specific “intent” such as “next\_mode” is detected, it falls into the if-else statement and calls a specific function or block of code.

Figure 11 - Speech-to-intent function

Source 11 - (Seivewright & Akinmoluwa, 2025)

The speech-to-text model Leopard proved the most challenging as it requires much more processing power than the Raspberry Pi running the Geek Goggles could offer. The model was used to transcribe an audio recording from the user and upload it to their project file on the web application. Since the Pi could not process the data itself, the raw audio file had to be sent over a web socket to the server where the server could run the model effectively and process the audio into text.

# **IMPLEMENTATION**

The implementation section will discuss the communication between the web application, the Geek Goggles and the peripheral tool. This section will also detail the physical construction of the Geek Goggles device.

## **Communication**

The Geek Goggles system uses multiple forms of wireless communication to send data between the Raspberry Pi and the server, as well as the Raspberry Pi and the peripheral voltmeter. This section will review these forms of communication between components.

### **Server and Pi Communication**

To send data between the server and the Raspberry Pi embedded system, web sockets were used. Originally, the system was developed using Bluetooth to send data directly from the user’s phone or computer running the web application, but this ran into challenges as browsers only have access to a devices Bluetooth Low Energy (BLE) and not the full Bluetooth capabilities. This didn’t work as BLE has challenges sending large amounts of data such as audio files and images which was a large majority of the data needing to be sent. Instead, the project pivoted to web sockets which can easily handle larger data transfers as well allowing the data to be sent directly to the server rather than the front-end web application.

### **Pi and Peripheral Communication**

To send data between the Raspberry Pi and the peripheral voltmeter, MQTT was used. MQTT allows fast transfers of text-based data between devices and since a simple stream of volt readings was all that needed to be sent it fit the use case. The server runs as the MQTT broker to which the voltmeter publishes data, and the Pi subscribes to the broker to receive the data. Since both the Pi and the voltmeter run on the same local network this system allowed for nearly instantaneous data transfers.

## **Physical Construction**

The physical construction of the Geek Goggles needed to be simple enough to complete with the limited resources and limited time available but also still provide the benefits of safety glasses and comfortable support the various sensors, the camera and the Raspberry Pi. These constraints lent to the use of 3D printing smaller components that could then be attached to an existing set of safety glasses so the user could still be properly protected while working on a project. With the support of the NAIT CNT instructors, three main components were designed and 3D printed. A left-side compartment which holds the Raspberry Pi, the camera and the micro-display, a right-side compartment the holds the sensors and the buttons for user controls and three strap points so a simple head strap could be attached to the glasses and provide support. The head strap was sourced from an old virtual reality helmet as the weight and design were similar enough to provide support for the glasses. These various components were then attached to the glasses using epoxy providing a strong, durable bond.

# **PROJECT RESULTS**

The project results section will discuss the ways in which the project was successful, the aspects that were only partially completed and finally any changes that would have been made.

## **Successes**

The project ended up successfully completing the majority of verification test and achieved some of the long shot goals set aside at the start, such as voice commands. The most challenging success was the micro-display module as the original part ordered did not arrive and a secondary display had to be ordered late into the project. The second display was not optimal as it lacked documentation and its design was slightly different that what was originally designed for, but it was the only option that would work in the time frame remaining. The display ended up arriving on time and after a few attempts to directly solder it to the Raspberry Pi’s composite video test pads it successfully displayed the GUI.

The project successfully allowed the user to control the device by the two-button interface on the glasses and by using voice commands. There was also a challenge with the web application and the Raspberry Pi properly communicating data back and forth as the user uploaded documents on the web application or added notes or pictures from the Geek Goggles device but in the end, it was able to seamlessly transfer the data.

The project successfully implemented communication between the peripheral voltmeter and the Raspberry Pi device and transfer speeds were at an acceptable level.

## **Challenges and Changes**

Due to time and resource constraints some of the goals laid out at the beginning of the project were only partially completed. Even though the micro-display properly displayed the GUI, the combination of the size of the display and the cheaper combiner material that was selected. The projection of the display was not as clear as was originally hoped for, but this issue could have been solved with more time and resources to try different material and set-ups. One of the features listed in the verification tests was the ability to set a timer on the web application and have an alert go off on the glasses once the timer ran out. Unfortunately, at the time of writing this report this timer feature was not implemented due to time constraints although many of the components required are already implemented in the code such as the ability to track time and send alerts, so it could easily be added in the future. The physical appearance and comfort of the Geek Goggles were not the focus of the project but could be improved with more design time and resources.

# **CONCLUSION**

Overall working on the Geek Goggles project has been a very challenging yet rewarding experience that brought together numerous different aspects of computer engineering ranging from embedded systems to full-stack web development, real-time data handling and IoT communication.

The project set out to build a system that could collect voltage readings, support voice commands, and allow users to take notes and upload images all through a connected pair of smart Geek goggles. Along the way, a lot was learned such as how to manage real-time messaging using MQTT and Web Sockets, how to design a responsive web interface with database-backed project tracking, and how to integrate hardware components like the ESP32-S3 and Raspberry Pi to work as a unified system.

There were plenty of technical hurdles, like synchronizing data between devices, ensuring consistent communication across platforms, and making sure everything stayed responsive under load. Ultimately, through testing and debugging and redirecting when needed, the project successfully managed to reach its core goals.

Looking ahead, this project has a lot of potential. Upgraded sensors, an AR overlay, video recording feature or improved physical design could make the project even better. But as it stands, this project already reflects how modern technologies, electronics, and software can work together to solve real world problems and help people work more efficiently.

# **REFERENCES**

AirNow. (n.d.) *Air Quality Index (AQI) Basics.*  AirNow. https://www.airnow.gov/aqi/aqi-basics/

Blanche, Pierre-Alexandre. (n.d). *Holographic Combiners Improve Head-Up Displays.* Photonics. <https://www.photonics.com/Articles/Holographic_Combiners_Improve_Head-Up_Displays/a64487>

Demofox. (2015, April 14). *Decibels (dB) and Amplitude.* The blog at the bottom of the sea. <https://blog.demofox.org/2015/04/14/decibels-db-and-amplitude/>

Digikey. (n.d.). Digikey. <https://www.digikey.ca/>

Displaymodule. (n.d.). Displaymodule. https://www.displaymodule.com/

Fitzpatrick, Martin. (March 19, 2025).  *Creating your first app with PyQt5.* Pythonguis. <https://www.pythonguis.com/tutorials/creating-your-first-pyqt-window/>

Google. (n.d.). *Text-to-speech pricing.* [*https://cloud.google.com/text-to-speech/pricing*](https://cloud.google.com/text-to-speech/pricing)

Guttag, Karl. (2016, October 21). *AR/MR Optics for Combining Light for a See-Through Display (Part 1).* KGOnTech. <https://kguttag.com/2016/10/21/armr-optics-for-combining-light-for-a-see-through-display-part-1/>

Heer, Dominique. (2024, August 21). *Controlling your Computer with Coice Commands by using OpenAI Whisper.* Medium. https://medium.com/@dominique.heer/controlling-your-computer-with-voice-commands-by-using-openai-whisper-09c867c635b2

Hrisko, Joshua. (2020, November 22). *Recording Stereo Audio on a Raspberry Pi.* MakerPortal. <https://makersportal.com/blog/recording-stereo-audio-on-a-raspberry-pi>

Kivy. (n.d.). *Welcome to Kivy.*  <https://kivy.org/doc/stable/>

Kkielhofner. (2023, March 7). *Amazon’s big dreams for Alexa fall short (ft.com)* [Online forum post]. Hackernews. https://news.ycombinator.com/item?id=35051418

lady ada. (2017, February 22). *Adafruit I2S MEMS Microphone Breakout > Raspberry Pi Wiring & Test.* Adafruit. https://learn.adafruit.com/adafruit-i2s-mems-microphone-breakout/raspberry-pi-wiring-test

Mañolo. (n.d.). *Home* [YouTube channel]. Retrieved February 02, 2025, from <https://www.youtube.com/@ManolloMancelli>

Mancelli, Mañolo (2025, January 27). C*onversation regarding display modules.* [Discord].

Microsoft (n.d.). *Code First to New Database* <https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database>

Microsoft. (n.d.). *Efficient querying.* <https://learn.microsoft.com/en-us/ef/core/performance/efficient-querying>​[Microsoft Learn](https://learn.microsoft.com/en-us/ef/core/performance/efficient-querying?utm_source=chatgpt.com)

MQTT. (n.d.). *Getting Started with MQTT*  <https://mqtt.org/getting-started/>

MySQL. (n.d.). *Why mySQL.* <https://www.mysql.com/why-mysql/>

Patel, Ravi. (2024, October 4). *A Beginner’s Guide to Entity Framework Core (EF Core).* Medium. https://medium.com/@ravipatel.it/a-beginners-guide-to-entity-framework-core-ef-core-5cde48fc7f7a

Picovoice. (n.d.) *Docs*. [https://picovoice.ai/docs/](%20https:/picovoice.ai/docs/)

Pimoroni. (2024, February 9). *Indoor-air-quality.py* [Source code]. Pimoroni. <https://github.com/pimoroni/bme680-python/blob/main/examples/indoor-air-quality.py>

Rose, Brice. (2019, February 21). *MEMS versus ECM: Comparing Microphone Technologies.* DigiKey. <https://www.digikey.ca/en/articles/mems-vs-ecm-comparing-microphone-technologies>

Simard, F. (2024, January 12). *Introduction to Smart Glasses — 2024 Tech Review.* Medium. https://medium.com/antaeus-ar/introduction-to-smart-glasses-2024-tech-review-d871e7d95cb8

World Health Organization. (n.d.) *Deafness and hearing loss: Safe listening.*  World Health Organization. https://www.who.int/news-room/questions-and-answers/item/deafness-and-hearing-loss-safe-listening

# **APPENDICES**

## **Appendix A: Glossary**

1. HUD - Heads up display
2. UI - User interface
3. CSS - Cascading Style Sheet
4. MEMS - Micro-Electro-Mechanical System
5. MQTT - Message Queuing Telemetry Transport
6. EF - Entity Framework
7. IAQ - Indoor Air Quality
8. BLE - Bluetooth low energy

## **Appendix B: Schematics**

### **Raspberry Pi Embedded System**

A screenshot of a computer

AI-generated content may be incorrect.

### **ESP32 Voltmeter Peripheral**

A computer diagram of a circuit board

AI-generated content may be incorrect.

## **Appendix C: Bill of Materials**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Manufacturer Part Number | Description | Quantity | Reference Designator | Cost | Extended Cost |
| SC1176 | Raspberry Pi Zero 2 W | 1 | J1 | $21.50 | $21.50 |
| ESP32-S3-DEVKITC-1U-N8R8 | ESP32-S3-DevKit | 1 | U2 | $21.43 | $21.43 |
| INMP411ACEZ-R7 | INMP411 Microphone | 1 | MK1 | $1.69 | $1.69 |
| SEN-16466 | BME680 Breakout | 1 | U1 | $28.64 | $28.64 |
| TGV01-100 | Safety Glasses | 1 | N/A | $5.25 | $5.25 |
| A1268 | Battery Power Bank | 1 | BT1 | $39.99 | $39.99 |
| D6L90 F1 LFS | Push Button Switch | 3 | SW1, SW2, SW3 | $2.13 | $6.39 |
| CF14JT30K0 | 30k Resistor | 1 | R1 | $0.16 | $0.16 |
| CF14JT10K0 | 10k Resistor | 1 | R2 | $0.16 | $0.16 |
| 1N4728A | Zenor Diode 1N4728 | 1 | D1 | $0.24 | $0.24 |
| N/A | SON-LC02A Display | 1 | DS1 | $76.89 | $76.89 |
| V0049 | Head Strap | 1 | N/A | $25.58 | $25.58 |
| N/A | Raspberry Pi V1.3 mini Camera | 1 | N/A | $24.95 | $24.95 |
| N/A | 32gb Micro SD | 1 | N/A | $12.99 | $12.99 |