Module Guide for Audio360

Team #6, Six Sense
Omar Alam
Sathurshan Arulmohan
Nirmal Chaudhari
Kalp Shah
Jay Sharma

November 17, 2025

1 Revision History

Date	**Version**	**Notes**
2025-11-13	1.0	Initial write-up

2 Reference Material

This section records information for easy reference.

2.1 Symbols, Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
Audio360	360 Audio analysis system on smart glasses
DAG	Directed Acyclic Graph
DOA	Direction of Arrival
FR	Functional Requirement
HUD	Heads Up Display
ICA	Independent Component Analysis
M	Module
MG	Module Guide
NFR	Non-functional Requirement
OS	Operating System
R	Requirement
SC	Scientific Computing
SPI	Serial Peripheral Interface
SRS	Software Requirements Specification
UC	Unlikely Change
USART	Universal Synchronous/Asynchronous Receiver/Transmitter

Table 1: Symbols, abbreviations and acronyms used in the MG document.

See SRS Documentation at Symbols, Abbreviations, and Acronyms for a complete table used in Audio360.

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team [1]. We advocate a decomposition based on the principle of information hiding [2]. This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by [1], as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed [1]. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.
Rationale: There will be a benefit to using a hardware that is smaller (better integration with glasses), cheaper (reduce costs for user), and stronger compute (allows for further advanced audio processing), if one exists in the market.

AC2: The number of microphones in microphone array.
Rationale: Depending on the performance on audio classification and directional analysis features, more or less microphones may be required than what is specified NFR9.2.

AC3: The arrangement of the microphones relative to each other.
Rationale: The system may be deployed on devices other than glasses, such as a hat. As a result, the microphone arrangement can change.

AC4: The output display hardware. At the moment, it is Rokid Glasses [3], which projects visuals onto glasses lens. Display may be changed to a simpler screen.
Rationale: Due to Rokid Glasses availability and cost, it may be swapped with a cheaper and easily available display that can be integrated easily onto a glasses frame.

AC5: The number of classification of audio sources.
Rationale: NFR5.1 specifies at least 3 distinct audio classification of audio sources required. This can be changed to detect a single audio classification if detecting one source is more critical.

AC6: The sample rate that audio is sampled from the environment from the environment.
Rationale: NFR7.1 states to sample at 16 kHz as that is the minimum to analyze audible sounds to human. However, the scope can expand to detect sounds outside the human audible range to notify users of other critical sounds that human would have never been able to process on their own.

AC7: The sample size of audio data to analyze each time step. Currently specified to be up to 4096 samples from NFR3.2.
Rationale: For higher quality of audio processing, an higher sample rate is required.

AC8: The visualization content on the output display.
Rationale: Stakeholders may require alternative information or a different presentation format that is more effective and appropriate for the display. This is related to NFR6.2

AC9: Traditional algorithms may be used over hardware acceleration.
Rationale: Hardware acceleration can consume more power than traditional software algorithms. The trade-off of slower processing over power consumption may be desirable.

AC10: Action when low confidence occurs during audio source classification.
Rationale: According to FR5.4, the system shall notify user when results from classification is low confidence. However, this may be disregarded if the accuracy of the classification module is near perfect.

AC11: The method of generating simulated microphone array input for testing and proof of concept.
Rationale: Since the use case is primarily for testing and proof of concept, the team will be prioritize methods that are easily integrable and provide sufficient functionality of testing. This can change as the team experiments with different methods at earlier stages. The final product will replace module linked to this anticipated change entirely with hardware microphone drivers. See table 4 for linked modules.

AC12: The SD card logging method for diagnostic audio data.
Rationale: The SD card diagnostic method is useful for bulk data logging during testing. In a production system it is going to be removed since no debug logs will be captured.

AC13: The implementation of Independent Component Analysis (ICA) for separating multiple simultaneous audio sources from mixed signals.
Rationale: The SRS mentions multi-source handling as a functionality goal (Section G.4), where the system should detect and track multiple simultaneous sound sources. ICA would enable the extraction of individual audio sources from mixed microphone signals, improving classification and direction estimation accuracy in complex acoustic environments. However it is complicated to implement and is defined as a stretch goal by the team.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The input devices for providing audio data from the environment are microphones.
Rationale: Cheapest and easiest method of getting real-time audio from the environment of the user. Although, AC11 mentions the potential of changing inputs for testing and proof of concept, UC1 is focused on the final product.

UC2: The processor works as a closed system. That is no external devices other than the microphones and display can be connected, and all processing occurs on the processor (processing is not offloaded to another device).
Rationale: Adheres to user Goal 7, user safety and privacy, as well as FR9.1. AC1 will be considered if additional compute power is required.

UC3: Microphones must always be synchronized at each sample.
Rationale: Microphone synchronization is a requirement for directional analysis, which a core feature. Relates to FR1.2:

UC4: The core features interact with hardware through an hardware abstraction layer.
Rationale: This allows the achieve hardware antricipated change without system redesign.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Microphone Input Module

M2: USART Communication Module

M3: SD Card Interface Module

M4: Microphone Diagnostic Module

M5: Audio Generation Module

M6: DOA (Direction of Arrival) Processor Module

M7: Audio360 Engine Module

M8: Audio Sampling Module

M9: Audio Spectral Leakage Prevention Module

M10: Audio Normalizer Module

M11: Audio Anomaly Detection Module

M12: Fast Fourier Transform Module

M13: Logging Module

M14: Fault Manager Module

M15: Mel Filter Module

M16: Discrete Cosine Transform Module

M17: Principle Component Analysis Module

M18: Linear Discriminant Analysis Module

M19: Classification Module

M20: Visualization Module

M21: Independent Component Analysis (ICA) Module

Level 1	**Level 2**
Hardware-Hiding Module	M1
	M2
	M3
	M4
Behaviour-Hiding Module	M7
	M20
Software Decision Module	M5
	M6
	M8
	M9
	M10
	M11
	M12
	M13
	M14
	M15
	M16
	M17
	M18
	M19
	M21

Table 2: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

1. NFR2.1 and FR1.4 specify that the system shall detect and propagate faults to the embedded firmware layer for appropriate handling. A software fault manager module, M14, is responsible for monitoring and reacting to software error states. In addition, the M4 module monitors the microphone array for hardware faults.

2. NFR3.2 specifies that the system shall support different audio sample sizes. Sample size directly affects both the speed and quality of audio analysis. Consequently, a dedicated module, M8, is responsible for managing this.

3. FR3.4 recommends the use of hardware acceleration to leverage hardware optimizations. However, there may be cases where traditional software algorithms perform more efficiently, such as when processing smaller datasets. Therefore, specific computational tasks are encapsulated within dedicated modules that determine whether to use hardware acceleration based on the input characteristics. This design consideration affects M9, M10, and M12.

4. FR3.5 requires the system to detect anomalies in the audio stream. To address this, a dedicated module, M11, is responsible for monitoring the audio data and identifying anomalies such as clipping or dropouts.

5. FR4.5 requires that the software be portable across different hardware platforms. This directly influenced the design decision to abstract the Audio360 Engine from the hardware layer. Inputs to the Audio360 Engine are maintained in a generic format, enabling deployment on various hardware targets, including simulation environments.

6. FR5.1, FR5.4 requires the frequency analysis component to classify audio sources. The classification will be done by performing Principle Component Analysis (PCA) on extracted features from the Mel-Frequency domain and performing Linear Discriminant Analysis (LDA) on components to predict the classification. Since this is a multi-step process, specific computational tasks are encapsulated within de-coupled modules that will be called individually. This design affects M15, M16, M17, M18 and M19.

7. FR5.2, NFR5.3 requires determining direction of sound source in radians (FR5.3) within a specific error threshold. As a result, DOA processor module M6 is responsible for implementing these requirements. This modular design facilitates testing to ensure that the specified error threshold is satisfied.

8. FR6.1, FR6.2, FR8.1, and FR8.2 specify that the system shall display sound classification and direction information to users. NFR6.1 requires prioritization of safety-critical information, while NFR6.2 mandates non-intrusive presentation. NFR8.1 requires a minimum of 30 display updates per second for low latency. To satisfy these requirements, a dedicated visualization module is responsible for receiving classification and direction data from the Audio360 Engine, formatting the information according to display constraints, and managing the visual presentation on the smart glasses display. The module implements prioritization logic to ensure safety-critical alerts are displayed prominently while maintaining a non-obstructive HUD-style interface. This design decision separates visualization concerns from audio processing, allowing the display format and content to be modified independently (supporting AC: Visualization Content) without affecting core audio analysis functionality.

9. The SRS mentions multi-source handling as a functionality goal (Section G.4), where the system should detect and track multiple simultaneous sound sources when feasible. To support this capability, an Independent Component Analysis (ICA) module, M21, is designed to separate mixed audio signals into individual source components. This module would enable improved classification and direction estimation accuracy when multiple sound sources are present simultaneously by separating them before processing. The module is designed to be conditionally invoked by the Audio360 Engine when multiple sources are detected, allowing the system to operate with or without this capability.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by [1]. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *Driver Layer*, this means that the module is provided by the STM32 or by standard programming language libraries. *Audio360* means the module will be implemented by the Audio360 software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules

7.1.1 Microphone Input Module M1

Secrets: The configuration and initialization of the hardware peripherals required to read data from the microphone array.

Services: Provides a driver interface to read audio sample data from the microphone array as a buffered stream.

Implemented By: Driver Layer

Type of Module: Library

7.1.2 USART Communication Module M2

Secrets: The configuration and initialization of the hardware peripherals required for USART serial communication with external devices.

Services: Provides an interface for sending and receiving serial data using the USART protocol on the serial pins of the microcontroller.

Implemented By: Driver Layer

Type of Module: Library

7.1.3 SD Card Interface Module M3

Secrets: The configuration and initialization of the SPI hardware peripherals and clocks required for SD card communication. This module also hides the SD card file system management and driver used to send commands to the SD card.

Services: Provides an interface for reading and writing files to an SD card module connected to the microcontroller via SPI.

Implemented By: Driver Layer

Type of Module: Library

7.1.4 Microphone Diagnostic Module M4

Secrets: The internal logic used to perform diagnostic tests on the microphone array hardware using hardware ADC converters.

Services: Provides diagnostic state information about the microphone array hardware and its connections to the microcontroller.

Implemented By: Driver Layer

Type of Module: Abstract Data Type

7.2 Behaviour-Hiding Module

7.2.1 Audio360 Engine Module (M7)

Secrets: The internal coordination logic that determines the execution order and timing of core audio processing features. Also, the mechanisms that manage data flow between hardware interfaces and the software modules.

Services: Orchestrates the overall audio processing by receiving raw input data and managing module communication.

Implemented By: Audio360

Type of Module: Abstract Data Type

7.2.2 Visualization Module (M20)

Secrets: The algorithms and data structures used to format and render visual information on the smart glasses display. This includes the layout strategy for HUD elements, prioritization logic for safety-critical information, and the rendering pipeline for directional indicators and classification labels. The module also hides the specific display hardware interface details, allowing the visualization format to be modified independently of the underlying display technology.

Services: Accepts classification labels and direction angles (in radians) from the Audio360 Engine, formats this information according to display constraints and prioritization rules, and renders visual indicators on the smart glasses display. The module ensures non-intrusive presentation while maintaining a minimum update rate of 30 frames per second. It also handles display of error states and safety feature failure alerts.

Implemented By: Audio360

Type of Module: Abstract Data Type

7.3 Software Decision Module

7.3.1 Audio Generation Module (M5)

Secrets: The algorithm for simulating room acoustics and generating synthetic microphone array data from a given audio source and position. This includes room response calculations and spatial audio propagation models.

Services: Provides simulated four-channel microphone array output for testing and proof-of-concept development. Takes as input an audio source signal and 3D position coordinates, and outputs four synchronized audio streams representing what each microphone in the array would capture given the room acoustics and source location.

Implemented By: Python using pyroomacoustics library

Type of Module: Library

Notes: This module is temporary and intended only for proof-of-concept development. It will be removed in the final product once physical hardware microphones are integrated. The module enables algorithm development and testing without requiring the complete hardware setup.

7.3.2 DOA (Directional of Arrival) Processor Module (M6)

Secrets: The specific direction of arrival estimation algorithm and its implementation details, including signal processing techniques (MUSIC, SRP-PHAT, or FRIDA), frequency analysis methods, and coordinate system transformations.

Services: Analyzes four synchronized microphone audio streams to estimate the direction of arrival of a sound source. Takes as input four time-domain or frequency-domain audio signals and outputs an angle (in degrees) representing the estimated direction of the sound source relative to the microphone array's reference axis. Provides real-time directional audio analysis with target accuracy of 45 degrees as specified in the SRS (NFR5.3).

Implemented By: Audio360 (Python for prototyping, C/C++ for embedded implementation)

Type of Module: Abstract Data Type

Notes: This module is core to the system's primary functionality of providing directional awareness. It implements one or more DOA algorithms and may include algorithm selection logic based on environmental conditions or computational constraints. Ideally, there should be no changes to this module for the final product aside from the internal implementation details (programming language for efficiency and combatilibility).

7.3.3 Audio Sampling Module (M8)

Secrets: The data structure used to store sampled audio data, the scheduling of sampling times, and the memory optimization strategy for acquiring and storing samples.

Services: Provides sampling of audio data from a given source while preserving the order of individual samples.

Implemented By: Audio360

Type of Module: Abstract Data Type

7.3.4 Audio Spectral Leakage Prevention Module (M9)

Secrets: Windowing function strategy (Windowing is the method to minimize spectral leakage).

Services: Applying windowing on audio signal to reduce spectral leakage before frequency domain processing.

Implemented By: Audio360

Type of Module: Abstract Object

7.3.5 Audio Normalizer Module (M10)

Secrets: The normalization algorithm.

Services: Processes audio signals to maintain consistent amplitude across different sources.

Implemented By: Audio360

Type of Module: Abstract Object

7.3.6 Audio Anomaly Detection Module (M11)

Secrets: The detection algorithms used to identify anomalies in the audio stream. Also includes the list of dependent modules that must be notified upon specific anomaly occurrences.

Services: Detects anomalies in the audio signal, such as clipping, dropouts, or empty buffer conditions, and issues appropriate notifications.

Implemented By: Audio360

Type of Module: Abstract Object

7.3.7 Fast Fourier Transform Module (M12)

Secrets: The specific algorithm to compute the fourier transform, potentially including hardware acceleraion provided by the STM32 platform.

Services: Computes the discrete Fourier transform of the input signal to obtain its frequency domain representation.

Implemented By: Audio360

Type of Module: Abstract Object

7.3.8 Logging Module (M13)

Secrets: The method used to route character streams to output sources, and the internal log formatting strategy, including time-stamping and file source information.

Services: Provides centralized logging of data streams to designated output destinations for debugging and monitoring.

Implemented By: Audio360

Type of Module: Abstract Object

7.3.9 Software Fault Manager Module (M14)

Secrets: The internal mechanisms and decision logic used to detect, manage, and recover from software system faults.

Services: Monitors system error states and manages critical faults to maintain core functionality when possible.

Implemented By: Audio360

Type of Module: Abstract Data Type

7.3.10 Mel Filter Module (M15)

Secrets: The specific algorithm for converting a signal from the spectral domain to the mel spectral domain.

Services: Accepts a time-frequency domain signal and outputs the corresponding time-mel-frequency signal based on the desired number of mel-bands.

Implemented By: Audio360

Type of Module: Abstract Object

7.3.11 Discrete Cosine Transform Module (M16)

Secrets: The specific algorithm for computing the Discrete Cosine Transform (DCT) from a signal in the mel spectral domain.

Services: Accepts a time-mel-frequency signal, and runs the algorithm that computes the DCT of the signal. This will output the Mel Frequency Ceptral Coefficients (MFCC) that will describe the main features of the input signal.

Implemented By: Audio360

Type of Module: Abstract Object

7.3.12 Principle Component Analysis Module (M17)

Secrets: The specific algorithm for transforming the MFCC feature vector into a lower-dimensional space by extracting only the most significant variance. Also includes the projection matrix and mean feature matrix learned during offline training.

Services: Accepts MFCC feature vector and produces new representation that preserves only the most significant variance from the original MFCC vector. This reduces the computational load for the LDA (M18) module.

Implemented By: Audio360

Type of Module: Abstract Data Type

7.3.13 Linear Discriminant Analysis Module (M18)

Secrets: The specific algorithm for applying Linear Discriminant Analysis (LDA) to determine which class the sound most likely belongs to and the probability. Also includes the project matrix and class-specific weight and bias values learned during offline training.

Services: Accepts feature vector extracted from the PCA (M17) module and performs linear discriminant classification to output the classification label corresponding to the detected sound category.

Implemented By: Audio360

Type of Module: Abstract Data Type

7.3.14 Classification Module (M19)

Secrets: The modules and the order to call them in to get the most probable classification using raw input audio. It also contains heuristics for determining when a classification is considered low-confidence.

Services: Accepts raw input audio, passes the audio through various modules to determine the most probable classification and it's confidence. Internally determines whether the classification is considered low confidence and outputs the classification with a flag if the confidence is low.

Implemented By: Audio360

Type of Module: Abstract Data Type

7.3.15 Independent Component Analysis (ICA) Module (M21)

Secrets: The specific ICA algorithm implementation (e.g., FastICA) and its parameters for blind source separation. This includes the statistical independence criteria, optimization methods, and convergence thresholds used to separate mixed audio signals into independent source components.

Services: Accepts multi-channel audio signals from the microphone array and separates them into individual independent audio source components. Takes as input mixed audio signals from multiple microphones and outputs separated audio streams, each representing an individual sound source. This enables improved classification and direction estimation when multiple sound sources are present simultaneously. The module may be conditionally invoked by the Audio360 Engine when multiple sources are detected.

Implemented By: Audio360

Type of Module Abstract Object

Notes: This module is a stretch goal and may or may not be implemented depending on computational constraints and development timeline. It supports the multi-source handling functionality mentioned in the SRS (Section G.4). If implemented, it would enhance the system's ability to handle complex acoustic environments with overlapping sound sources by separating them before classification and direction estimation.

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

8.1 Modules to Requirements Traceability

The following table maps each module to the functional and non-functional requirements from the SRS that it helps satisfy. Requirements are referenced using the labels defined in the SRS document.

Req.	**Modules**
FR1.2	M1
FR1.4	M14, M4
NFR1.1	M8
NFR1.2	M8
NFR1.3	M1
FR2.1	M1, M2, M3
FR2.3	M14
NFR2.1	M14
FR3.1	M12
FR3.2	M10
FR3.3	M9
FR3.4	M9, M10, M12
FR3.5	M11
NFR3.1	M12
NFR3.2	M8
FR4.1	M7
FR4.2	M7

FR4.3	M7
FR4.4	M7, M14
NFR4.1	M8
NFR4.2	M8
NFR4.3	M8
FR5.1	M15, M16, M17, M18, M19
FR5.2	M6
FR5.3	M6
FR5.4	M19
NFR5.3	M6
FR6.1	M7, M20
FR6.2	M20, M14
NFR6.1	M20
NFR6.2	M20
FR7.1	M1, M8
FR7.2	M1
NFR7.1	M1
FR8.1	M20
FR8.2	M20
NFR8.1	M20

Table 3: Trace Between Requirements and Modules

8.2 Modules to Anticipated Changes Traceability

The following table maps anticipated changes to the modules that would need to be modified if those changes occur. This supports the principle of information hiding by showing which design decisions are localized to specific modules.

AC	**Modules**
AC1	-
AC2	M1
AC3	-
AC4	M20
AC5	M19
AC6	M8
AC7	M8
AC8	M20
AC9	M12
AC10	M7
AC11	M5, M5
AC12	M3, M13
AC13	M21

Table 4: Trace Between Anticipated Changes and Modules

AC1, AC3, are tied to hardware changes, thus there is no corresponding software module that traces to them.

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. [4] said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

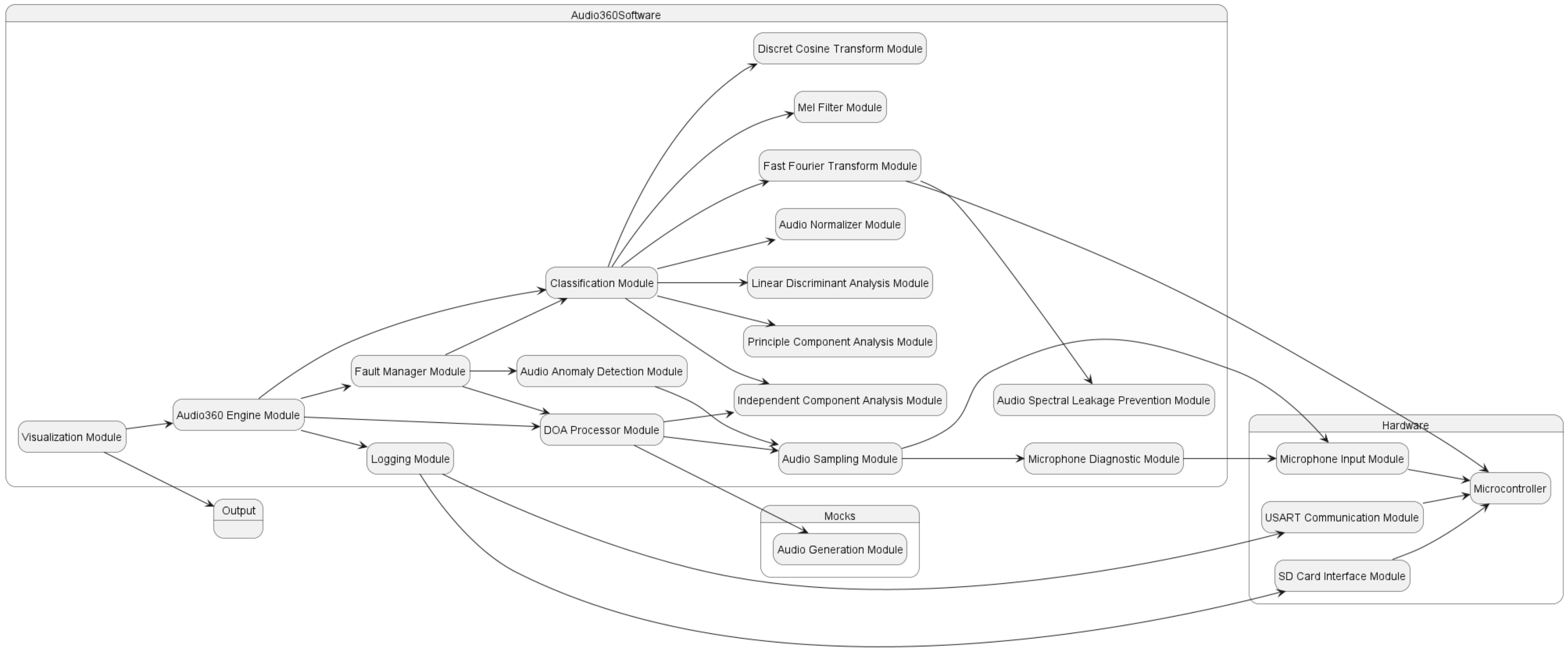

Figure 1: Use hierarchy among modules

10 User Interfaces

The user interface for Audio360is primarily visual, displayed through smart glasses in a heads-up display (HUD) format. The interface provides real-time visual indicators showing sound source classification and direction of arrival information.

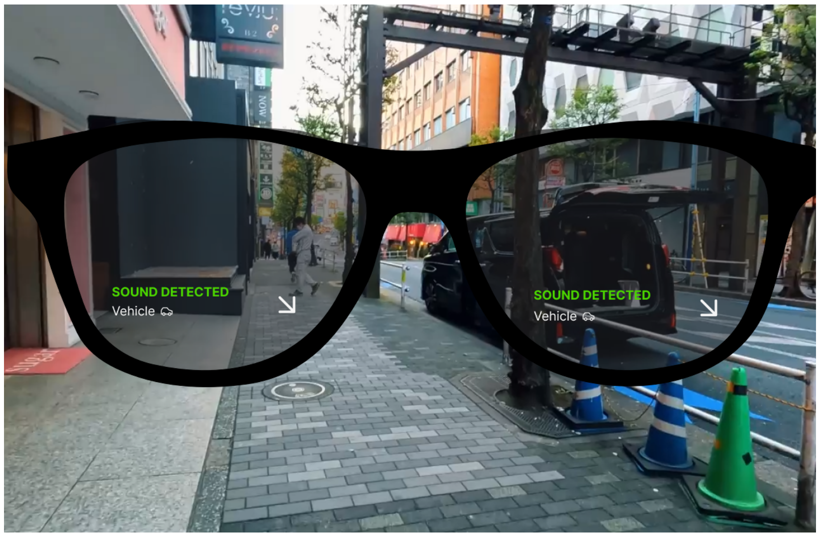

Figure 2: Smart glasses HUD visualization showing sound detection interface. The display shows a detected sound source (Vehicle) with directional information overlaid on the user's field of view.

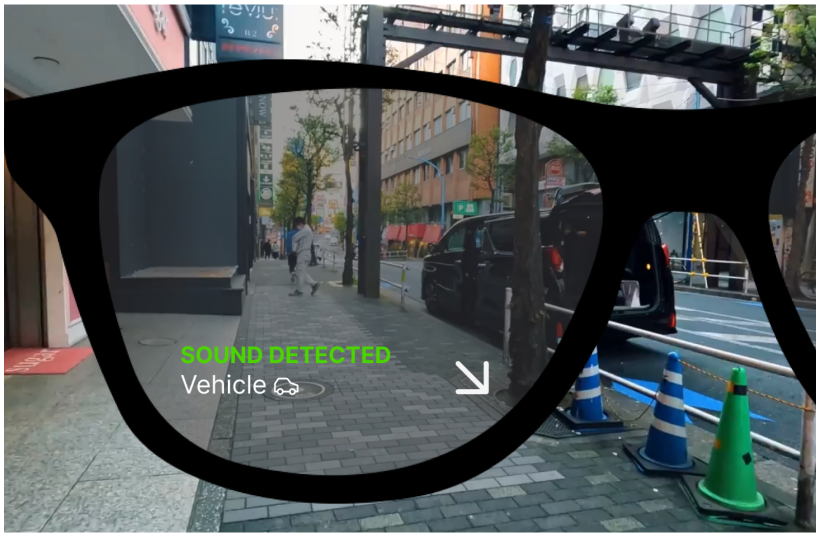

Figure 3: Close-up view of the smart glasses HUD interface. This view provides a detailed look at the AR overlay elements including the sound detection indicator, classification label, and directional arrow displayed on the left lens.

11 Design of Communication Protocols

Not applicable at this stage of the project since there is no SDK or connectivity information available for the Rokid smart glasses. Communication protocols will be designed once the hardware integration phase begins and more details about the smart glasses platform are known.

12 Timeline

A roadmap for completing the module implementations is provided in the Github project roadmap. It is organized by milestones corresponding to the major phases of the project timeline (Rev 0, Rev 1).

Major implementation for the following modules have been completed as of the writing of this document:

- Audio Generation Module (M5)
- Logging Module (M13)
- Fast Fourier Transform Module (M12)
- SD Card Interface Module (M3)
- USART Communication Module (M2)

No tasks are thus generated for these modules in the project roadmap.

References

[1] D. Parnas, P. Clement, and D. M. Weiss, "The modular structure of complex systems," in *International Conference on Software Engineering*, 1984, pp. 408–419.

[2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Comm. ACM*, vol. 15, no. 2, pp. 1053–1058, Dec. 1972.

[3] Rokid, "Rokid glasses," 2025. [Online]. Available: https://global.rokid.com/?srsltid=AfmBOoo-Mc8pQf3WkW2WnUC0L6rIWmC9SK6KKv4U_6GXD-AeR2IXig8y

[4] D. L. Parnas, "Designing software for ease of extension and contraction," in *ICSE '78: Proceedings of the 3rd international conference on Software engineering*. Piscataway, NJ, USA: IEEE Press, 1978, pp. 264–277.