Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Fall 2023

Team Majestic
RFID Smart Stethoscope

Michael Allen
Jaime Barajas
Efrain Morales Loya
Chau Nguyen
Khoi Tran

Revision History

Revision	Date	Author(s)	Description
0.1	9.11.2023	MA	Document Creation
0.2	09.23.2023	MA, CN, KT, EM	Draft Completion
0.3	09.25.2023	CN, EM, MA, JB, KT	Draft Review
1.0	12.6.2023	MA,	Final Version Completed

CONTENTS

List of Figures

1 INTRODUCTION

The RFID Smart Stethoscope (RSS) is conceived as a groundbreaking educational apparatus, aimed at amplifying diagnostic acumen within healthcare education. By harnessing the capabilities of programmable RFID chips embedded in a wearable shirt, the RSS can adeptly reproduce an array of auscultation sounds, simulating diverse health scenarios.

- **Conception and Design:** The product, originally conceptualized with advanced diagnostic education in mind, has been meticulously tailored for medical students, healthcare experts, and educators. It aligns with the goals of providing comprehensive and realistic diagnostic training for various stakeholders in medical training.
- **Purpose of This Document:** This Detailed Design Specification delves into the intricate technicalities of the RSS. It provides insights into the preliminary design ideations, foundational choices, and the overall system architecture and features.
- **Scope and Application:** The RSS is envisioned to be a cornerstone in medical education, setting a new standard in auditory diagnostic training. Its design ensures versatility, catering to a wide range of auditory scenarios and health conditions.

2 SYSTEM OVERVIEW

The RFID Smart Stethoscope system is a sophisticated blend of hardware and software designed to transform the learning experience for medical students and professionals. At its core, the system is segmented into distinct layers that collectively work to create an immersive auscultation simulation.

- **RFID Layer:** Its primary function is to manage the interactions with the RFID tags. Each of the 15 RFID tags has a unique ID, which when scanned by the Smart Stethoscope, determines the body part that has been probed.
- **Stethoscope Layer:** This acts as the central hub, comprising of the RFID scanner, sound database, and headphones. The RFID scanner picks up the unique ID from the RFID layer, and based on this ID, fetches the appropriate sound file from the sound database. This sound file then is relayed through the headphones, producing the relevant auscultation sounds for the user.
- **Application Layer:** This layer offers a user interface for a Smart Hospital employee to upload and assign specific audio files to respective body parts. Furthermore, this layer can modify sound settings based on user preferences or preset configurations. The Application Layer also houses several subsystems:
  - *Sound Files:* Responsible for storing the audio files.
  - *Sound Volume:* Manages the volume adjustments.
  - *Body Part:* Assigns specific sounds to the respective body parts.
  - *Presets:* Holds predefined configurations for various auscultation sounds.
  - *Sound Upload:* Manages the uploading and categorization of the audio files.

The meticulous coordination between these layers and subsystems ensures that when a specific part of the body (represented by an RFID tag) is scanned, the Smart Stethoscope plays the corresponding auscultation sound with the desired settings.

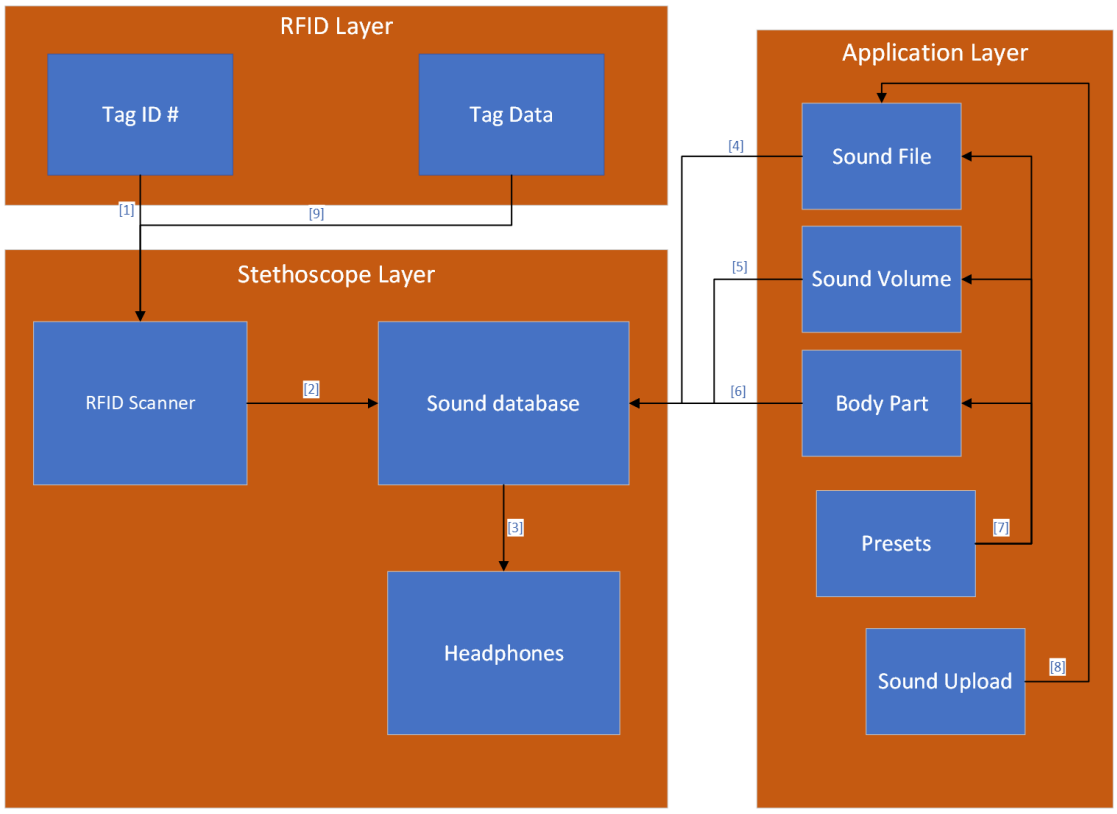

Figure 1: System architecture

3 Application Layer Subsystems

3.1 Layer Hardware

This section describes the primary hardware components that make up the core of the layer. These components have been chosen to achieve a harmonious integration of functionality, power efficiency, and compactness.

- **Raspberry Pi Zero:** Serving as the central computing unit, the Raspberry Pi Zero is known for its compact form factor combined with reasonable processing capabilities. It offers a versatile platform, supporting a range of peripherals, making it an ideal choice for embedded applications.
- **Pisugar 3:** Powering the Raspberry Pi Zero is the Pisugar 3, a battery module explicitly designed for this device. With its tailored fit and power capacity, it ensures that the Raspberry Pi Zero remains operational for extended durations without any compromise on performance.
- **RASPIAUDIO Audio DAC Hat Sound Card (Audio+V2):** Audio functionalities are crucial for our layer. The RASPIAUDIO DAC Hat is an advanced sound card that gets mounted atop the Raspberry Pi Zero. It is responsible for capturing and playback of audio with high fidelity, ensuring that sound data remains undistorted and true to its source.
- **Stemedu RC522 RFID Reader Writer Module:** This is a specialized module designed to read and write data to RFID cards and tags. Included in our setup are the S50 White Card and the RFID

Laundry Tags operating at a frequency of 13.56 MHz. Such a setup aids in unique identification and access control, a pivotal aspect of our layer.

- **RFID Laundry Tags 13.56 MHz:** These are passive tags working in conjunction with the RC522 Reader Writer Module. Given their design and form factor, they can be seamlessly integrated into a variety of objects, enabling them to be identified or tracked effortlessly.

3.2 Layer Operating System

The layer requires the use of the Raspbian operating system. Raspbian is specifically designed for Raspberry Pi devices, providing optimal performance and compatibility for our Raspberry Pi Zero. This Debian-based OS ensures efficient execution of processes, offers a stable platform for application development, and receives regular updates for security and stability enhancements.

- **Debian Foundation:** Originating from Debian, Raspbian carries with it a legacy of stability, security, and versatility. This base provides a rich software environment and ensures longevity in terms of support and updates.
- **Tailored for Raspberry Pi:** Raspbian has been fine-tuned for the Raspberry Pi architecture. This specialization ensures that the Raspberry Pi Zero operates at its optimal performance level, balancing both power efficiency and computational power.
- **Software Compatibility:** With its vast repository, Raspbian provides a myriad of software packages, tools, and utilities, simplifying the process of enhancing and expanding the functionalities of our layer.
- **Security and Stability:** Continuous updates provided to Raspbian not only aim at feature enhancement but also address security vulnerabilities. This ensures our layer remains robust against potential threats and always performs at its best.

3.3 Layer Software Dependencies

The RFID Smart Stethoscope relies heavily on various software dependencies, essential for providing an interactive, efficient, and user-friendly interface to medical students, healthcare professionals, and educators. Utilizing JavaScript as the primary development language, the following is a detailed breakdown of libraries, frameworks, and other dependencies that power the application layer of our system:

- **JavaScript:** The primary language chosen for developing the application layer. JavaScript's versatility in web development offers the advantage of creating a responsive and dynamic user interface. It also supports object-oriented programming, which aids in crafting a modular and maintainable codebase for our application.
- **Node.js:** As a runtime environment, Node.js enables server-side execution of JavaScript. Essential for building scalable web applications, it ensures smooth processing of user interactions and seamless communication with the RFID system.
- **RFID JS Library** A hypothetical library tailored for RFID interactions within JavaScript environments. This would provide functions and methods to read RFID data, process it, and associate it with relevant auscultation sounds.

3.4 Sound File Subsystem

The Sound Files subsystem is a software module within the larger application framework, specifically designed to manage the audio file assets of the application. Its core purpose is to handle all operations related to audio files,ensuring they are stored, retrieved, written to external devices, and deleted efficiently. This is a software module within the application, tailored to interact directly with both the application's internal processes.

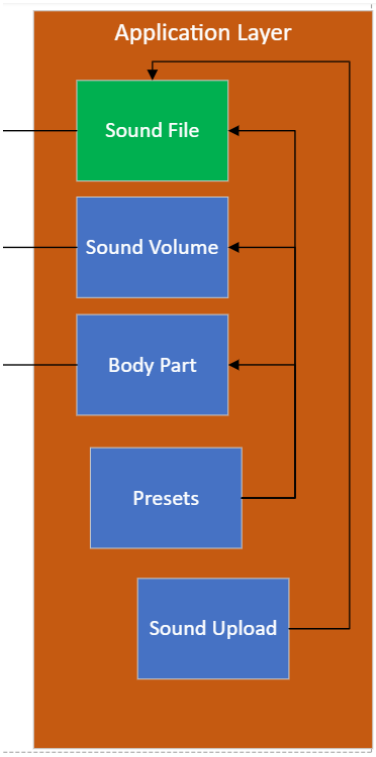

Figure 2: Sound file subsystem diagram

3.4.1 Subsystem Hardware

N/A

3.4.2 Subsystem Operating System

The RFID Smart Stethoscope's application subsystem is primarily developed and tested on the Windows operating system to ensure optimal functionality and performance. The choice of Windows aligns with the broader objective of reaching a wide user base and leveraging existing development tools and infrastructure. Specifically:

- **Windows:** Serving as the primary platform, modern Windows versions are known for their stability, versatility, and vast developer community. Given that Windows natively supports various JavaScript development environments and tools, it offers an efficient and comprehensive development platform for our application. In addition, the inclusion of features like the Windows Subsystem for Linux (WSL) enriches the development experience by allowing access to a broader range of tools if necessary.
- **Raspbian:** Our system leverages the Raspberry Pi Zero as a key hardware component. As such, Raspbian, tailored specifically for Raspberry Pi devices, becomes relevant in the broader system

context. Even if our primary application doesn't run directly on Raspbian, integration or communication protocols need to be established to ensure seamless interaction between the software and hardware layers.

3.4.3 Subsystem Software Dependencies

The RFID Smart Stethoscope application subsystem heavily relies on various software dependencies to deliver its functionality seamlessly. These dependencies encompass libraries, frameworks, and potentially design software for mechanical parts or circuits. Here's a breakdown of the key software dependencies for our system:

- **JavaScript Runtime - Node.js:** Serving as the backbone of our server-side application, Node.js provides the runtime environment to execute JavaScript code. It comes packed with npm (node package manager) that eases the management of other software libraries and frameworks.
- **Express.js:** A minimal and flexible Node.js web application framework, Express.js provides a robust set of features for web and mobile applications, aiding in setting up our server.

These software dependencies play a vital role in ensuring that our system is efficient, robust, and scalable. They collectively contribute to building a seamless integration of hardware and software, driving the innovative functionalities of the RFID Smart Stethoscope.

3.4.4 Subsystem Programming Languages

The development and functionality of the RFID Smart Stethoscope's application subsystem are contingent upon a selection of programming languages that synergize both performance and versatility. The choice of languages ensures that the system operates optimally, integrates seamlessly with various components, and allows for future scalability. Below is a description of the programming languages used:

- **JavaScript:** Serving as the cornerstone of our application development, JavaScript is employed both on the client and server sides. The language's dynamic and high-level characteristics make it ideal for building responsive interfaces and driving server-side logic. With the support of the Node.js runtime, JavaScript is the primary language driving the core functionalities of our system, including interactions with the database, handling RFID data, and managing user requests.
- **HTML/CSS:** For crafting the user interface of the application, HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are indispensable. While HTML structures the content, CSS provides the stylistic and layout properties, ensuring the application is both functional and aesthetically appealing.

The amalgamation of these programming languages ensures a robust, efficient, and user-friendly operation of the RFID Smart Stethoscope's application subsystem. They facilitate the seamless integration of software components, making the educational tool both intuitive and dynamic.

3.4.5 Subsystem Data Structures

In the Sound File subsystem of the RFID Smart Stethoscope application, specialized data structures are designed to handle the audio files and their associated attributes. These structures facilitate rapid access, modification, and transmission of sound-related data. Here are the primary structures used:

3.4.6 Subsystem Data Structures

In the Sound File subsystem of the RFID Smart Stethoscope application, specialized data structures are designed to handle the audio files and their associated attributes. These structures facilitate rapid access, modification, and transmission of sound-related data. Here are the primary structures used:

- **Audio File Metadata Structure:** Central to the Sound File subsystem, this structure holds details related to each audio file:
  - *File ID:* A unique identifier for the audio file.
  - *Tag ID Mapping:* Establishes a relationship between the audio file and its corresponding RFID tag.
  - *Duration:* Represents the length of the audio clip.
  - *Description:* Provides a brief description outlining the health scenario the audio simulates.
  - *Volume Settings:* Stores the default and user-defined volume settings for playback.
- **Audio Buffer Structure:** Designed for temporary storage during playback, it holds:
  - *Buffered Data:* Segments of audio files loaded for quick playback.
  - *Current Play Position:* Keeps track of where in the buffer the playback currently is.
- **Audio Queue Structure:** This structure is employed when there are multiple audio requests in quick succession, ensuring smooth and sequential playback:
  - *Queue:* A list of audio files awaiting playback.
  - *Current Audio:* The audio currently being played from the queue.
  - *Next Up:* The next audio file to be played.

These structures are pivotal in ensuring the Sound File subsystem functions smoothly, catering to user interactions and delivering an effective learning experience.

3.4.7 Subsystem Data Processing

The RFID Smart Stethoscope's application subsystem employs a combination of standard and custom algorithms to ensure the seamless identification of RFID scans and subsequent retrieval of associated auscultation sounds. Here are some of the primary algorithms and processing strategies in use:

- **RFID Scan Detection Algorithm:**
  - *Step 1:* Initialize RFID reader and listen for scans.
  - *Step 2:* On tag detection, read and validate the data packet structure, ensuring the checksum matches for data integrity.
  - *Step 3:* Extract the Tag ID from the validated data packet.
  - *Step 4:* Send the Tag ID to the application layer for further processing.
- **Audio Retrieval Algorithm:**
  - *Step 1:* Receive the Tag ID from the RFID scan detection algorithm.
  - *Step 2:* Match the Tag ID against the Audio File Metadata Structure to find the corresponding audio file.

- *Step 3:* Load the audio file into memory and prepare it for playback.
- *Step 4:* Send a signal to the user interface, indicating the audio is ready for playback.

- **Dynamic Volume Adjustment:** For certain conditions or auscultation sounds, the volume might need dynamic adjustment based on user preferences or the inherent volume of the recorded sound.
  - *Step 1:* Analyze the audio file's decibel levels.
  - *Step 2:* Compare with user's set preferences or use default settings.
  - *Step 3:* Adjust the volume dynamically as the sound plays, ensuring the user hears clear and distinct sounds.

These algorithms and processing strategies are central to the operation of the RFID Smart Stethoscope. While some of these processes employ standard techniques, their integration within the system, especially the dynamic volume adjustment, offers a unique blend tailored to the specific requirements of our educational tool.

3.5 Sound Volume Subsystem

The Sound Volume subsystem is a dedicated software module within the application, designed to provide precise volume control for individual audio files. At its core, this module ensures that sounds are played according to user-defined volume preferences, enabling a personalized audio experience within the application. This is a software module of the application, tailored to interact with the audio playback mechanisms to adjust and regulate volume settings.

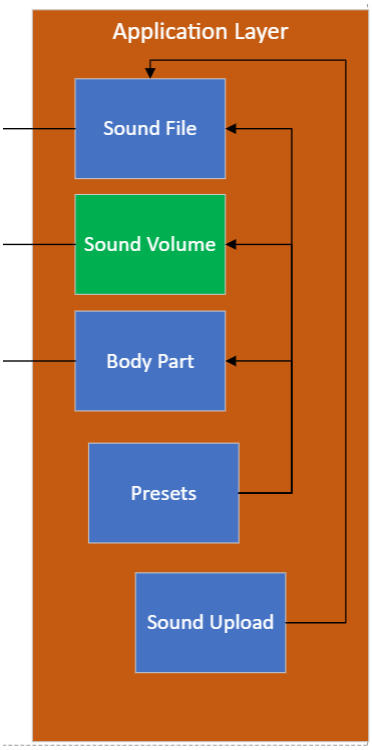

Figure 3: Sound Volume subsystem diagram

3.5.1 Subsystem Hardware

While the Sound Volume subsystem is primarily software-oriented, it does interface with several key hardware components to ensure precise and user-specific volume control:

- **Audio Output Device:** The subsystem directs audio to hardware output devices, such as headphones or speakers. The quality and capabilities of these devices can influence the perceived audio volume and clarity.
- **Volume Control Interface:** Some devices, particularly wearable tech or dedicated medical equipment, may have physical volume control buttons or dials. The subsystem is designed to interpret and respond to such hardware-based volume adjustments, synchronizing them with the software settings.
- **Audio Processing Chip:** In certain designs, a dedicated audio processing chip or Digital-to-Analog Converter (DAC) may be utilized to ensure high-quality sound output. The Sound Volume subsystem would interface with this chip to ensure that volume adjustments do not degrade audio quality.

Though the emphasis of the Sound Volume subsystem is on software, its effective operation is closely tied to its harmonious interaction with these hardware components.

3.5.2 Subsystem Operating System

The Sound Volume subsystem, being a software module dedicated to audio file volume control, operates within the broader context of the application's chosen operating system. Given its design and the technologies in play, the following operating system considerations have been made:

- **Windows:** As the primary operating system for the RFID Smart Stethoscope application development, Windows offers a stable and versatile platform for the Sound Volume subsystem. The broad user base and extensive developer support make it suitable for ensuring consistent audio performance and easy integration with various audio hardware components.
- **Cross-Platform Considerations:** While Windows is the primary OS, efforts have been made to ensure that the Sound Volume subsystem is modular and adaptable. With potential future expansions in mind, it has been designed to be compatible with other popular operating systems like macOS and Linux, especially in terms of audio processing and volume control mechanisms.

The choice of operating system not only ensures smooth operation but also compatibility with a wide range of audio output devices and hardware volume control interfaces.

3.5.3 Subsystem Software Dependencies

The Sound Volume subsystem's primary function is to provide precise control over the volume of individual audio files. To achieve this, several software dependencies are integrated to ensure smooth functionality and interaction with the broader application. Here are the main software dependencies for this subsystem:

- **Audio Processing Libraries:** Libraries such as Web Audio API or Howler.js might be used to manage and process audio in a web environment. These provide extensive tools for manipulating volume, amongst other audio properties.

- **JavaScript Frameworks:** Given the subsystem's implementation in JavaScript, certain frameworks like Node.js may be employed, especially if there's a need for server-side volume controls or settings storage.
- **UI/UX Libraries:** Libraries like jQuery UI or Materialize might be used to implement volume sliders, dials, or other user interfaces that allow users to adjust the volume to their preference.
- **Device Compatibility Layers:** To ensure that volume controls are consistent across a wide range of devices, libraries like Modernizr can be used to detect and manage device-specific features.
- **Database Connection Libraries:** If user-specific volume settings are to be saved and retrieved, libraries connecting the subsystem to databases, such as Sequelize for Node.js, might be integrated.

It's essential that these software dependencies are regularly updated and maintained to ensure compatibility with the latest devices and operating systems, as well as to address any security concerns.

3.5.4 Subsystem Programming Languages

The Sound Volume subsystem, focusing on delivering precise volume control for individual audio files within the application, primarily utilizes the following programming languages:

- **JavaScript:** At the heart of the Sound Volume subsystem is JavaScript, a versatile and widely-used language suitable for web applications. JavaScript is responsible for real-time volume adjustments, interaction with UI components like volume sliders, and handling user inputs. The asynchronous nature of JavaScript, along with its integration capabilities with various audio processing libraries, makes it ideal for this module.
- **HTML5 and CSS3:** While not programming languages in the traditional sense, HTML5 and CSS3 play pivotal roles in structuring and styling the user interface components associated with the volume controls. The audio elements of HTML5, in particular, provide native tools for audio playback, which can be further enhanced and controlled using JavaScript.

While the core functionality relies on JavaScript, the integration of HTML5 and CSS3 ensures a seamless and user-friendly interface for volume adjustments, providing users with an intuitive experience. It's essential to maintain a consistent coding standard and stay updated with the latest features and best practices in these languages to ensure the subsystem's optimal performance.

3.5.5 Subsystem Data Structures

In the Sound Volume subsystem, data organization is of paramount importance to ensure efficient volume control and real-time adjustments. Specific classes and data structures have been designed to facilitate these tasks, offering a smooth user experience. Some of these data structures include:

- **AudioFile:** Represents individual audio files in the system.
  - *fileID:* A unique identifier for the audio file.
  - *filePath:* Path to the physical location of the audio file.
  - *volumeLevel:* Stores the current volume level for the specific audio file.
  - *metadata:* Contains details like duration, bitrate, and other relevant audio file specifications.
- **VolumeSettings:** Captures user-defined volume preferences.
  - *userID:* Identifier for the user.

  - *audioFileID:* Identifier of the associated audio file.
  - *preferredVolume:* User-defined preferred volume level for the audio file.

- **VolumeControlPacket:** For any external hardware components or integrations, this data packet structure can be used to transmit volume control instructions.
  - *Header:* Indicates the start of the packet.
  - *Command:* Specifies the action (e.g., volume up, volume down, mute).
  - *Value:* Specifies the volume level or adjustment value.
  - *Footer:* Marks the end of the packet.

These data structures ensure that the Sound Volume subsystem operates effectively, providing users with precise control over the volume of individual audio files. A consistent structure facilitates easier integration, debugging, and enhancement of the subsystem in the future.

3.5.6 Subsystem Data Processing

The Sound Volume subsystem employs a combination of established algorithms and bespoke processing strategies to offer the most refined and responsive volume control experience for individual audio files.

- **Volume Normalization Algorithm:** A crucial step before adjusting the volume is to normalize the audio file. This ensures that the audio has a consistent amplitude throughout, offering a uniform listening experience. The subsystem uses the Root Mean Square (RMS) normalization, which adjusts the audio signals based on their RMS value. It enables the system to maintain audio clarity while altering volume levels.

- **User Preference Retention:** When a user adjusts the volume for a particular audio file, the change is stored using a linear regression model. Over time, this model identifies and predicts user preferences for different audio files based on their historical adjustments. It's a self-learning system that tailors the volume settings to each individual's preference.

- **Real-time Volume Control:** A custom algorithm has been developed to handle on-the-fly volume adjustments. When a user interacts with the volume control, this algorithm processes the current audio data buffer, applying volume multiplication factors in real-time without causing lag or distortion.

- **Volume Transition Smoothing:** To prevent abrupt volume changes which can be jarring for users, a smoothing algorithm is implemented. When transitioning between different volume levels, especially during fast adjustments, this algorithm ensures a gradual transition, enhancing the listening experience.

These data processing techniques work in tandem to ensure that the Sound Volume subsystem not only meets but exceeds user expectations in terms of responsiveness, accuracy, and customization. While some methodologies, like RMS normalization, are grounded in established audio processing practices, others, like the user preference retention model, are tailored specifically for the unique requirements of the RFID Smart Stethoscope system.

3.6 Body Part Subsystem

The Body Parts subsystem is an essential software module nested within the application layer, dedicated to mapping specific sound files to distinct parts of the body. Integrating with the broader system, it allows users to dictate which sounds are played when a certain part of the body RFID tags on the simulation shirt, is scanned or interacted with. This ensures an intuitive and immersive user experience, merging the the tactile sensation of the body with the auditory feedback from the application. This is a software module within the application that designed to interface with both Sound File subsystem and Sound Data base on hardware layer.

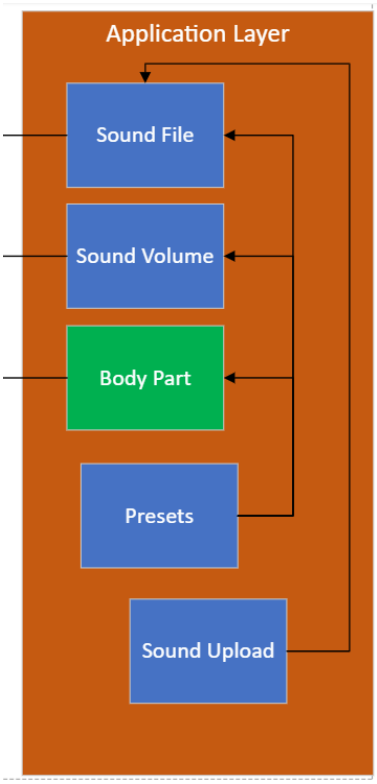

Figure 4: Body Part subsystem diagram

3.6.1 Subsystem Hardware

The Body Parts subsystem, while primarily being a software component, does rely on certain hardware elements to realize its functions efficiently. Here's a breakdown of the involved hardware components:

- **RFID Scanner:** The primary interaction between the user and the system begins at the RFID scanner level. Each distinct body part on the simulation shirt is embedded with a unique RFID tag. When users scan or interact with a specific part of the shirt, the RFID scanner detects the corresponding tag, which then triggers the software component of the Body Parts subsystem to fetch and play the appropriate sound file.
- **Simulation Shirt:** Crafted with the intent to provide a realistic experience, the simulation shirt acts as the tangible interface. It houses the RFID tags, strategically placed, each correlating to a different body part, ensuring precise sound playback corresponding to the location of interaction.

- **Connection Interfaces:** To ensure seamless data transmission between the RFID scanner and the application layer, the system utilizes standardized connection interfaces. Depending on the specifics of the hardware design, this might range from USB connections to wireless protocols.
- **Sound Database Hardware:** Residing at the hardware layer, the sound database hardware is a repository storing all the sound files. When the Body Parts subsystem receives a signal from the RFID scanner indicating a particular body part's interaction, it queries the sound database to retrieve and play the appropriate sound.

The successful operation of the Body Parts subsystem is contingent on the smooth integration and functioning of these hardware components. Their combined efforts culminate in an intuitive and immersive auditory feedback experience as users interact with different parts of the simulation shirt.

3.6.2 Subsystem Operating System

The Body Parts subsystem, being a software component, requires an operating system (OS) environment to function efficiently. The OS provides the necessary platform for the subsystem to operate, manage its tasks, and communicate with other subsystems. Here's a breakdown of the operating system considerations for the Body Parts subsystem:

- **Windows:** Given its widespread use and familiarity, the subsystem is primarily developed and tested on the Windows OS. This ensures broad compatibility and ease of deployment, especially in academic or medical training facilities that commonly use Windows-based systems.
- **Linux (Debian-based distributions):** Debian-based distributions like Ubuntu are also considered due to their stability and extensive repositories. Being open-source, they provide flexibility for further development and modifications, ensuring the Body Parts subsystem can be tailored to specific needs if required.
- **Cross-Platform Considerations:** To ensure versatility, the Body Parts subsystem is designed with portability in mind. Utilizing platform-agnostic programming practices ensures that the subsystem can be deployed across different OS environments with minimal adjustments.

3.6.3 Subsystem Software Dependencies

The Body Parts subsystem relies on various software dependencies to ensure smooth operation, accurate mapping of audio files to specific body parts, and seamless integration with other subsystems. The following list highlights some of these key software dependencies:

- **RFID Reader Library:** To read and interpret signals from the RFID tags embedded in the simulation shirt, a dedicated library optimized for RFID communication is necessary. This library enables the subsystem to detect which body part RFID tag is being scanned and thus determine the appropriate sound file to play.
- **Audio Playback Framework:** Given the auditory nature of the feedback, a robust audio playback framework or library is essential. This ensures that sound files are played without lag, distortion, or other audio artifacts, providing users with a realistic experience.
- **Database Management System (DBMS):** The subsystem requires a DBMS to store, retrieve, and manage the mapping data between different body parts and their corresponding audio files. Such a system enables easy updates and modifications to the audio files and their associations.

- **Intercommunication Middleware:** To seamlessly interact with the Sound File subsystem and the Sound Database on the hardware layer, middleware that facilitates inter-subsystem communication is essential. This ensures real-time data exchange and coordination among various components.
- **User Interface Toolkit:** To visualize and, if necessary, adjust the mapping between body parts and sound files, a user interface toolkit is beneficial. This provides developers or trainers with a graphical interface to make adjustments or updates.

While the aforementioned dependencies are core to the subsystem's functionality, there may be additional software tools, libraries, or frameworks incorporated based on evolving requirements, optimization needs, or feedback from end-users.

3.6.4 Subsystem Programming Languages

The Body Part Subsystem, being a pivotal module within the application layer, necessitates a blend of programming languages to achieve its functionalities, particularly for audio file mapping, RFID tag reading, and integration with other subsystems. Here are the primary programming languages employed:

- **JavaScript:** As a versatile and widely adopted language, JavaScript plays a dominant role in the development of the Body Part Subsystem. Its asynchronous capabilities make it ideal for real-time RFID tag reading and triggering the appropriate audio file playback. Additionally, several of the aforementioned software dependencies and libraries offer JavaScript interfaces, ensuring seamless integration.
- **SQL (Structured Query Language):** For managing the mapping data between body parts and audio files stored in the database, SQL is used. It provides the tools to query, update, and manage relational data, facilitating quick retrieval of the correct audio file based on the RFID tag scan.
- **Python or C:** Depending on the specific RFID reader library or the middleware used for inter-subsystem communication, either Python or C might be incorporated. These languages offer efficient low-level operations, potentially optimizing the system's response time and ensuring smooth communication with hardware components.

3.6.5 Subsystem Data Structures

The Body Part Subsystem, while managing audio mapping to respective body parts, relies on well-defined data structures to ensure accuracy and efficiency. Key data structures include:

- **BodyPartMapping Class:**
  - *RFIDTagID:* A unique identifier associated with each RFID tag on the simulation shirt.
  - *BodyPartName:* Descriptive name of the body part (e.g., "left lung", "right ventricle").
  - *AudioFileReference:* A pointer or link to the corresponding audio file in the database.
- **RFIDDataPacket Structure:**
  - *Header:* Signifies the commencement of the data packet.
  - *RFIDTagID:* Transmits the ID of the scanned RFID tag.
  - *Timestamp:* Notes the exact time the tag was read.
  - *Checksum:* Guarantees the integrity of data during transmission.

- *Footer:* Concludes the data packet.

These structures facilitate seamless interaction between the RFID tags, the application, and the sound database.

3.6.6 Subsystem Data Processing

The Body Part Subsystem integrates several data processing algorithms and strategies to enhance its performance:

- **RFID Tag Decoding:** As RFID tags are scanned, a decoding algorithm is employed to translate raw data into meaningful tag IDs, which then get mapped to the relevant body part and its associated audio.
- **Linear Search with Optimization:** While it might seem elementary, the mapping list might not be large enough to warrant complex search algorithms. However, frequently accessed body parts and sounds can be cached or indexed for quicker retrieval.
- **Checksum Validation:** Before processing any RFID data packet, a validation check is performed using the checksum to ensure data consistency. This ensures that corrupted or incomplete packets don't adversely impact the system's operation.
- **Audio Playback Synchronization:** Once the correct audio file is identified, a synchronization algorithm ensures that the playback is smooth and in sync with any previously playing sounds or user interactions.

The chosen algorithms and processing strategies aim to optimize response time, ensuring a real-time and immersive user experience.

3.7 Presets Subsystem

The Presets subsystem, seamlessly integrated within the overarching software environment, is pivotal in facilitating the rapid deployment of predefined configurations within the system. By allowing users to tap into previously saved settings and scenarios, it ensures efficiency and reduces the redundancy of manual re-configurations. This is a software module within the application. Its primary function is to manage and apply predefined configurations or scenarios within the application's environment, streamlining operations and interactions with other subsystems.

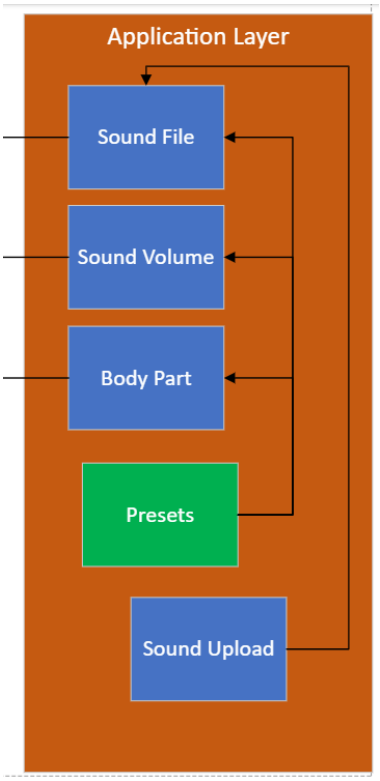

Figure 5: Presets subsystem diagram

3.7.1 Subsystem Hardware

The Presets subsystem, being primarily a software module, does not directly interface with specific hardware components. Instead, it relies on the broader system's hardware infrastructure to store, manage, and apply the saved configurations. Any hardware dependencies are abstracted by the software layer, ensuring smooth operations without direct hardware interactions. However, it's worth noting that the subsystem will utilize the system's storage hardware, such as SSDs or HDDs, to persistently store the preset configurations.

3.7.2 Subsystem Operating System

The Presets subsystem inherits its operating system requirements from the overarching application environment. Since it is a software module integrated within the main application, it doesn't have unique OS needs. Thus, if the primary system operates on Windows, for instance, then the Presets subsystem will naturally function within that Windows environment. The subsystem is designed to be OS-agnostic to ensure maximum compatibility and integration with the primary application platform.

3.7.3 Subsystem Software Dependencies

The following are software dependencies for the Presets subsystem:

- **Configuration Management Library:** This library will aid in the creation, management, and deployment of various preset configurations.
- **Database ORM (Object-Relational Mapping):** Given that the presets might be stored in a database, an ORM would help interface with the database without writing SQL queries directly.

- **File I/O Libraries:** Essential for reading and writing preset configurations to and from storage, ensuring that settings can be saved and retrieved as needed.
- **JSON or XML Parsers:** These are critical if the configurations are stored in JSON or XML format, facilitating the reading and modification of preset data.

The above libraries and frameworks are integral to the subsystem's operations, enabling efficient management of preset configurations.

3.7.4 Subsystem Programming Languages

The Presets subsystem predominantly uses JavaScript, considering its seamless integration capabilities and versatility in web and desktop application development. Given that it's a software module nested within a larger application, JavaScript ensures both interoperability and performance, particularly when managing and deploying preset configurations. Libraries or frameworks related to JavaScript, like Node.js or Electron, may also be utilized.

3.7.5 Subsystem Data Structures

The Presets subsystem employs several critical data structures to manage the preset configurations effectively. A Preset Configuration Structure is used to store each predefined setting, including the following components:

- *Preset ID:* A unique identifier for each preset configuration.
- *Configuration Data:* The actual settings and parameters that define the configuration.
- *Metadata:* Additional information such as creation date, last modified date, and user who created the preset.

This structured approach ensures that preset configurations are stored, retrieved, and applied efficiently and accurately, reducing the risk of errors during the configuration application process.

3.7.6 Subsystem Data Processing

The Presets subsystem employs several algorithms and processing strategies to manage the preset configurations effectively. A specific configuration Application Algorithm is used to apply the saved settings to the current system state, which involves the following steps:

1. Retrieve the selected preset configuration from storage.
2. Parse the configuration data and extract the individual settings.
3. Apply each setting to the corresponding component or subsystem within the application.
4. Validate the applied configuration to ensure it has been set correctly.

This algorithm ensures that the preset configurations are applied accurately and efficiently, allowing users to switch between different scenarios or settings rapidly.

3.8 Sound Upload Presets

The Sound Upload subsystem functions as the gateway for introducing and managing new sound files within the system. Its design is meticulously laid out in the architectural block diagram, illustrating its pivotal role in the system's data flow dynamics and shedding light on its interconnections and communication patterns with adjacent subsystems. This subsystem is a software module within the application

structure. It specifically caters to the sound management domain of the system, focusing primarily on the intake and preliminary processing of sound files.

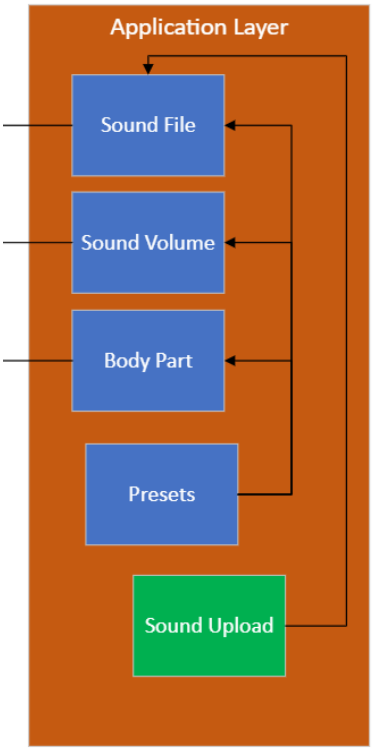

Figure 6: Sound Upload subsystem diagram

3.8.1 Subsystem Hardware

While the Sound Upload subsystem primarily operates within the software domain, its performance and efficiency can be influenced by specific hardware components, such as:

- **Storage Drives:** High-speed storage solutions like SSDs ensure rapid read/write operations when importing or exporting sound files.
- **Sound Card:** An advanced sound card or audio interface can ensure that the uploaded sound files are accurately processed and reproduced without distortions.
- **Microphone Input:** In scenarios where live recording or real-time feedback is required, a dedicated microphone input port and possibly an ADC (Analog-to-Digital Converter) might be crucial.

3.8.2 Subsystem Operating System

Given the nature of the application and its broader integration requirements, the Sound Upload subsystem is designed to operate efficiently on modern versions of:

- **Linux (Debian-based distributions):** Debian-based OSes like Ubuntu are favorable due to their stability and extensive software repositories.
- **Windows:** With native support for numerous sound management libraries and tools, Windows remains a viable choice.

- **Raspbian:** If a Raspberry Pi-based solution is employed elsewhere in the system, ensuring compatibility with Raspbian becomes relevant.

3.8.3 Subsystem Software Dependencies

To efficiently handle the upload and preliminary processing of sound files, including formats like MP4, the Sound Upload subsystem relies on various software libraries, tools, and frameworks spanning multiple programming languages:

- **FFmpeg:** A comprehensive multimedia handling suite used for decoding, encoding, transcode, mux, demux, and stream audio and video files, including MP4.
- **libsndfile (C):** A C library for reading and writing sound files in various formats.
- **Web Audio API (JavaScript):** A high-level web API for processing and synthesizing audio in web applications.
- **Audacity Libraries:** For advanced sound editing or waveform visualization.
- **HTML5 File API (JavaScript):** Enables the web-based part of the application to interact with files stored on the user's computer.

3.8.4 Subsystem Programming Languages

The Sound Upload subsystem leverages multiple programming languages to ensure versatility, efficient processing, and broad compatibility:

- **C:** Utilized for low-level sound file handling, performance optimization, and to interface with certain libraries like 'libsndfile'.
- **JavaScript:** Employs Web Audio API for web-based functionalities, enabling user interactions for sound file uploads, and real-time processing within a browser environment.

3.8.5 Subsystem Data Structures

Efficient data management is crucial for the seamless operation of the Sound Upload subsystem. Several data structures have been devised for this purpose:

- **SoundFile Object:**
  - *FileID:* Unique identifier for each uploaded sound file.
  - *FileName:* Name of the sound file.
  - *FileFormat:* Format type (e.g., MP4, WAV).
  - *FileData:* Binary data of the sound file.
  - *Metadata:* Additional information like duration, bitrate, etc.
- **UploadPacket:** For transmitting sound file data to the application:
  - *Header:* Indicates the start of a new data packet.
  - *FileInformation:* Contains the SoundFile Object.
  - *Checksum:* Ensures data integrity during transmission.
  - *Footer:* Marks the conclusion of the data packet.

3.8.6 Subsystem Data Processing

The Sound Upload subsystem implements a combination of standard and specialized algorithms:

- **Audio File Format Detection:** Uses signature-based detection to identify and validate the format of uploaded sound files.
- **Checksum Verification:** Implemented to ensure the integrity of sound files during transmission and storage. Common algorithms like CRC32 or MD5 might be employed.
- **Metadata Extraction:** Algorithms that parse the sound file to extract useful metadata (e.g., duration, channels, bit-rate).
- **File Conversion (if needed):** If the system requires a specific format for sound files, the subsystem incorporates algorithms to convert uploaded files to the desired format.
- **Sound Normalization:** Unique to this project, it processes uploaded sounds to ensure a consistent volume across all files, enhancing user experience.

These data processing strategies ensure that the subsystem can efficiently intake, verify, and prepare sound files for usage within the broader application environment.

4 Stethoscope Layer Subsystems

The Stethoscope Layer is the nerve center for the system's tactile-to-audio interactions. Comprising a series of interlinked subsystems, this layer meticulously converts RFID scans into immersive audio feedback. Each subsystem, from the RFID scanner to the headphones, is intricately designed to work in unison, thus ensuring an uninterrupted and enriching user experience. This section delves deep into the fabric of each subsystem, highlighting the pivotal hardware components, software dependencies, and architectural nuances that drive their collective performance.

4.1 Layer Hardware

The Stethoscope Layer primarily incorporates a seamless blend of hardware components, ensuring the conversion of tactile interactions into auditory feedback. At its core, it employs an embedded computing system that coordinates and manages the operations of each individual subsystem. This embedded system boasts a high-speed processor, optimized for real-time processing, and is coupled with an adequate memory unit. Apart from this centralized computing unit, the hardware components specific to each subsystem, such as the RFID scanner and headphones, will be discussed in their respective sections.

4.2 Layer Operating System

The embedded computing system of the Stethoscope Layer runs on a custom lightweight operating system, tailored for real-time operations. This operating system is optimized for efficient task scheduling, ensuring that each subsystem gets the required computational resources without latency. It also supports a variety of drivers and interfaces that facilitate seamless communication between the subsystems, enhancing the fluidity and responsiveness of the overall system.

4.3 Layer Software Dependencies

For the Stethoscope Layer to function cohesively, it leverages a series of software libraries and frameworks. These include:

- **RFID Communication Library:** This facilitates the communication between the embedded system and the RFID scanner, ensuring efficient data transfer and processing.
- **Audio Processing Framework:** To ensure that sound files are correctly interpreted, modified, and transmitted to the headphones.
- **Real-Time OS Scheduling Library:** A library that ensures the real-time requirements of the layer are met, effectively scheduling processes and managing resources.

Any specific software dependencies related to individual subsystems will be further elaborated in their respective sections.

4.4 RFID Scanner

The RFID Scanner subsystem operates as the primary interface with the RFID layer, functioning as the bridge that transforms physical RFID scans into actionable digital data. By accurately capturing the unique ID numbers from RFID tags, it ensures that these are relayed to the sound database, serving as the foundation for subsequent operations and interactions within the system. This subsystem is a hardware integrated component. While the physical hardware handles the actual scanning of RFID tags, an accompanying software component ensures that the gathered data is processed and relayed appropriately within the system.

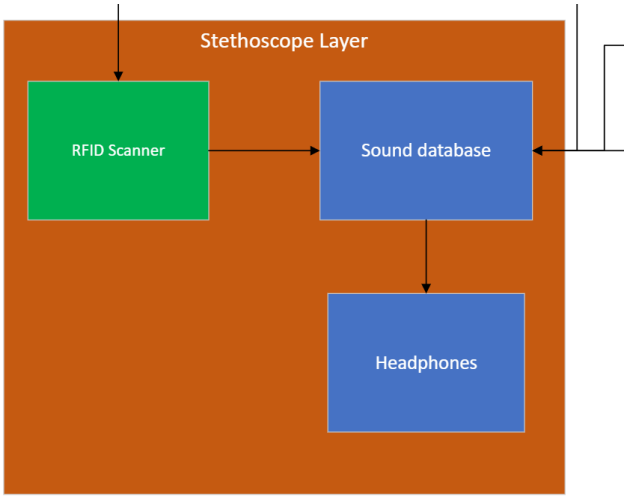

Figure 7: RFID Scanner subsystem diagram

4.4.1 Subsystem Hardware

The RFID Scanner subsystem boasts a state-of-the-art RFID scanning module, designed for high-accuracy and efficient scanning of RFID tags. This module is capable of capturing both low-frequency and high-frequency tags, ensuring a broad range of compatibility. The scanner typically consists of an antenna, a transceiver, and a decoder which then translates the RFID data into a format suitable for processing. To ensure optimal performance, the scanner is equipped with advanced noise-cancellation capabilities to minimize interference.

4.4.2 Subsystem Operating System

Given the specialized nature of the RFID Scanner subsystem, it employs a lightweight, real-time operating system (RTOS). This RTOS is optimized for quick response times, ensuring that RFID scans are promptly captured and processed. The focus is primarily on efficiently handling incoming data streams and communicating with the overarching Stethoscope Layer system.

4.4.3 Subsystem Software Dependencies

For the seamless operation of the RFID Scanner subsystem, the following software dependencies are essential:

- **RFID Communication Library:** A specialized library that provides functions to initiate scanning, capture data, and communicate with other parts of the system.
- **Data Formatting Framework:** Ensures that the raw RFID data is converted into a standardized format suitable for further processing within the Stethoscope Layer.

4.4.4 Subsystem Programming Languages

The RFID Scanner subsystem predominantly uses C and C++ for its programming needs. C is chosen for its close-to-hardware capabilities, allowing for real-time operations, while C++ facilitates object-oriented design, especially useful for the software component that manages and processes the scanned data. Both languages provide the necessary speed and efficiency crucial for real-time RFID data processing.

4.4.5 Subsystem Data Structures

For effective management of the scanned RFID data, the subsystem employs specific data structures:

- **RFIDPacket:** A structured data type that encapsulates the raw data obtained from an RFID scan. This may include attributes such as:
  - *tagID*: The unique identifier of the RFID tag.
  - *timestamp*: The time the tag was scanned.
  - *signalStrength*: An optional attribute indicating the strength of the signal received, which can be used for proximity-related features.
- **Queue:** A first-in-first-out (FIFO) structure to temporarily store RFID packets before they're processed or relayed to other parts of the system. This ensures that no data is lost during high-frequency scanning events.

4.4.6 Subsystem Data Processing

The primary algorithm employed by the RFID Scanner subsystem is the "RFID Data Filtering Algorithm". Given the possibility of redundant or erroneous scans, this algorithm ensures that only valid and unique tag reads are processed further:

1. Capture raw RFID data into an RFIDPacket.
2. Check the validity of the data (e.g., correct format, reasonable timestamp).
3. Compare the newly scanned tagID with previously scanned IDs within a specific timeframe to prevent duplicates.
4. If the tag is deemed unique and valid, relay it to the sound database; otherwise, discard it.

This filtering mechanism is essential to provide a smooth user experience, preventing the system from being overwhelmed by superfluous data and ensuring accuracy in sound playback corresponding to RFID scans.

4.5 Sound database

The Sound Database subsystem is unequivocally the central nerve of the Smart Stethoscope, seamlessly interweaving various data streams to produce harmonized audio output. As the system's cerebral component, it adeptly discerns which sounds to play, informed by an array of inputs it constantly receives and processes. This subsystem is a software module, intricately designed to interface with both hardware components, such as the RFID Scanner, and other software subsystems like the Application layer.

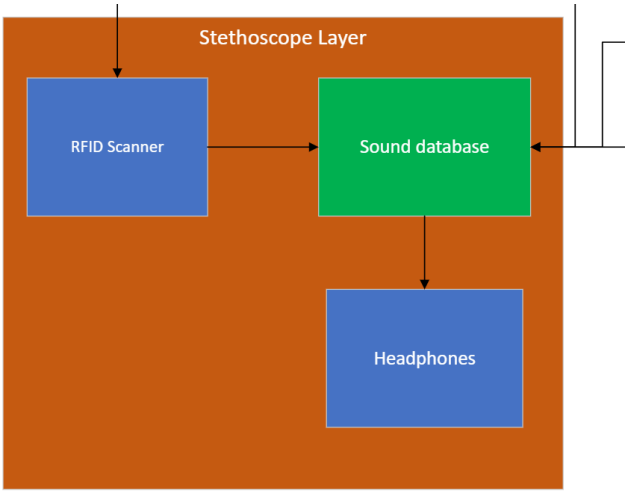

Figure 8: Sound database subsystem diagram

4.5.1 Subsystem Hardware

The Sound Database primarily operates as a software component. However, it requires a sufficiently fast storage medium, likely an SSD (Solid State Drive), to ensure that sound files can be accessed and played back in real-time upon request. Moreover, it benefits from being hosted on a server with ample RAM for caching frequently used sounds, optimizing playback latency.

4.5.2 Subsystem Operating System

The Sound Database operates on a Linux-based server OS, chosen for its stability, security, and the extensive library support it offers for database management and sound processing.

4.5.3 Subsystem Software Dependencies

Dependencies for the Sound Database include:

- Relational Database Management Systems (RDBMS) such as MySQL or PostgreSQL to manage and index the sound files.
- Sound processing libraries like FFmpeg for format conversion and sound manipulation.
- Server-side frameworks to facilitate communication with other subsystems.

4.5.4 Subsystem Programming Languages

The predominant languages used for the Sound Database are:

- SQL for database queries.
- C,C++,Java Script and its associated libraries for back-end logic, sound processing, and integration tasks.

4.5.5 Subsystem Data Structures

Sound data management employs the following data structures:

- **SoundFileRecord:** A class or structure capturing metadata about each sound file. Attributes may include:

- *fileID*: A unique identifier for each sound file.
- *filename*: The actual file's name and path.
- *associatedRFID*: The RFID tag which, when scanned, should trigger this sound.
- *duration*: The sound file's length in seconds.

- **SoundQueue:** A dynamic queue structure that lines up sounds to be played based on RFID scans and application requests.

4.5.6 Subsystem Data Processing

The core algorithm within the Sound Database is the "Sound Retrieval and Playback Algorithm":

1. On receiving an RFID scan or application request, look up the 'SoundFileRecord' associated with the given identifier.
2. If the sound file exists, load it into the SoundQueue.
3. Prioritize sounds in the queue based on time of request and application logic (e.g., certain urgent sounds may be prioritized over others).
4. Play back sounds from the queue to the headphone or other output device.

This algorithm ensures seamless sound retrieval and playback, maximizing the user's auditory experience.

4.6 Headphones

The Headphones subsystem plays a pivotal role as the final output interface of the Smart Stethoscope. It is essentially a hardware component, specifically designed to provide an audible rendition of the processed sound data. The subsystem's architecture ensures optimal sound clarity, user comfort, and compatibility with the preceding layers of the stethoscope system. Its interaction with the software component is largely in the realm of receiving processed sound data and rendering it audible to the user.

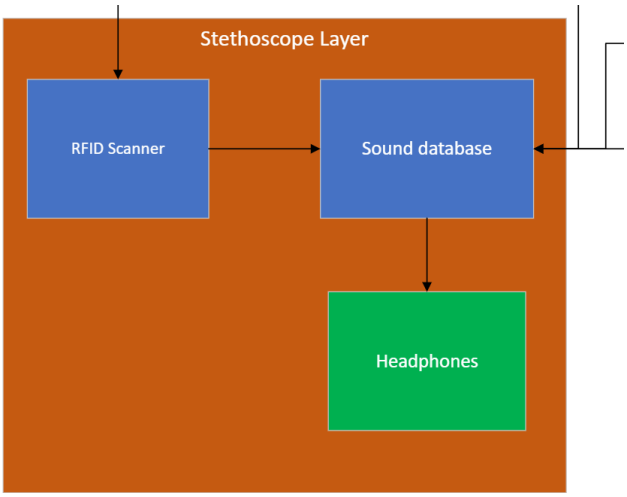

Figure 9: Headphones subsystem diagram

4.6.1 Subsystem Hardware

The primary hardware component for this subsystem is the headphone set, complete with speakers, connecting wires, and potential control buttons for volume adjustment or track selection. The headphones might also possess noise-cancellation features, ensuring clear sound reproduction in various environments.

4.6.2 Subsystem Operating System

The headphones, being primarily a hardware component, do not require a specific operating system. However, any embedded software, if present for features like active noise cancellation or Bluetooth connectivity, would run on a proprietary firmware.

4.6.3 Subsystem Software Dependencies

N/A

4.6.4 Subsystem Programming Languages

N/A

4.6.5 Subsystem Data Structures

For basic headphones, data structures are not relevant. However, if the headphones possess smart features (like saving equalizer settings), they may use basic data structures to store and retrieve these settings. Any data transmitted to the headphones, for example via Bluetooth, would be in the form of audio packets following standard audio transmission protocols.

4.6.6 Subsystem Data Processing

Headphones primarily receive and render audio data. Any processing would be limited to equalization, amplification, and, in some cases, noise cancellation. Algorithms such as Fast Fourier Transforms (FFT) could be employed in active noise-cancelling headphones to generate anti-noise signals.

5 RFID Layer Subsystems

The RFID (Radio Frequency Identification) Layer serves as the foundation of the Smart Stethoscope's identification capabilities. This layer is crucial in capturing and decoding signals from RFID tags, thereby enabling the system to identify and fetch corresponding sound data. This section provides an in-depth insight into the technical intricacies of the RFID Layer, elucidating on its hardware specifics, software dependencies, and the operating system.

5.1 Layer Hardware

The RFID Layer predominantly features the RFID Laundry Tags operating at 13.56 MHz HF. These tags have been specially designed for embedding into linens, towels, and medical scrubs, thereby necessitating a robust yet small diameter design. Crafted from PPS and sealed with epoxy, these tags are equipped to endure high temperatures and rough handling.

5.2 Layer Operating System

The RFID Laundry Tags, given their passive nature, do not utilize an operating system. Their operations are hard-wired and they function when they come into the proximity of an RFID reader that operates at their frequency.

5.3 Layer Software Dependencies

While the RFID Laundry Tags themselves don't have direct software dependencies, the systems interfacing with them, such as RFID readers, might necessitate specific drivers, SDKs, or middleware to decode and interpret the data efficiently.

5.4 Tag Data

The tag data works with the tag ID number to help confirm the identity of the tag. The data on the tag stores the name of the programmed audio file so that the stethoscope knows which sound to play when the tag is scanned.

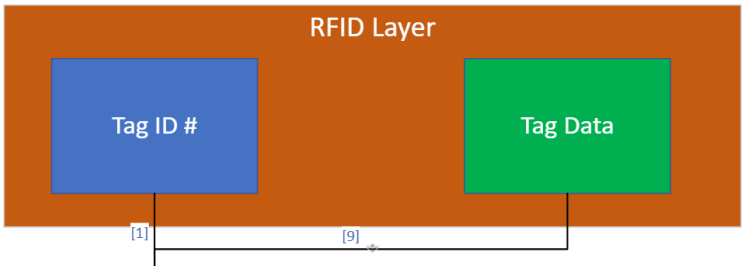

Figure 10: RFID Chip subsystem diagram

5.4.1 Subsystem Hardware

N/A

5.4.2 Subsystem Operating System

The RFID Laundry Tag, operating as a passive RFID system, lacks an operating system. It relies on hard-wired logic to transmit its tag data when queried by a compatible RFID reader.

5.4.3 Subsystem Software Dependencies

No direct software dependencies exist for the tag data. However, systems, especially readers, which interact with these tags may require specialized software for precise data interpretation.

5.4.4 Subsystem Programming Languages

Given the passive nature of the RFID Laundry Tag, it doesn't employ any programmable logic akin to microcontrollers or computers. Therefore, no programming languages are associated directly with the tag. However, there is a wide variety of Python libraries to help communicate with the RIFD tags and pull its tag data.

5.4.5 Subsystem Data Structures

The primary data on the RFID Laundry Tag is its unique identifier. Depending on the tag's storage capability, there might be additional information stored. This identifier, and any supplementary data, adheres to either the ISO 15693 or ISO 14443 NFC protocols.

5.4.6 Subsystem Data Processing

On the tag itself, data processing is minimal. The tag merely responds with its data when queried by an RFID reader. Significant data interpretation or further processing is the responsibility of the reader or downstream systems.

5.5 Tag ID Number

Functioning as the bedrock of the RFID Layer, the Tag ID Number is used for the main control of the system as a whole. When the tag is scanned it sends it ID number, and that is used in part with the tag data to help determine what part of the body was scanned.

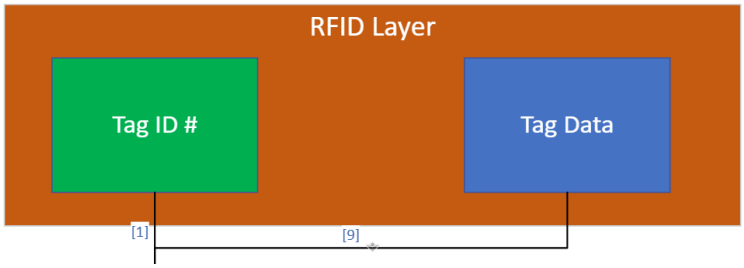

Figure 11: Tag ID Number subsystem diagram

5.5.1 Subsystem Hardware

N/A

5.5.2 Subsystem Operating System

The RFID Laundry Tag, operating as a passive RFID system, lacks an operating system. It relies on hard-wired logic to transmit its ID number when queried by a compatible RFID reader.

5.5.3 Subsystem Software Dependencies

No direct software dependencies exist for the ID number. However, systems, especially readers, which interact with these tags may require specialized software for precise data interpretation.

5.5.4 Subsystem Programming Languages

Given the passive nature of the RFID Laundry Tag, it doesn't employ any programmable logic akin to microcontrollers or computers. Therefore, no programming languages are associated directly with the tag. However, there is a wide variety of Python libraries to help communicate with the RIFD tags and pull its tag ID.

5.5.5 Subsystem Data Structures

The primary data on the RFID Laundry Tag is its unique identifier. Depending on the tag's storage capability, there might be additional information stored. This identifier, and any supplementary data, adheres to either the ISO 15693 or ISO 14443 NFC protocols.

5.5.6 Subsystem Data Processing

On the tag itself, data processing is minimal. The tag merely responds with its data when queried by an RFID reader. Significant data interpretation or further processing is the responsibility of the reader or downstream systems.

6 APPENDIX A

Include any additional documents (CAD design, circuit schematics, etc) as an appendix as necessary.

[1–5]

REFERENCES

[1] "Pisugar 3: Battery for raspberry pi zero," 2021, hardware component.

[2] R. P. Foundation, "Raspberry pi zero," 2015, hardware component.

[3] RASPIAUDIO, "Audio dac hat sound card (audio+v2) for raspberry pi4 all models pi zero / pi3 / pi3b / pi3b+ / pi2," 2019, better Quality Than USB. Hardware component.

[4] "Rfid laundry tags 13.56 mhz hf," 2019, designed to be imbedded or attached to bed linens, towels, and primary medical scrubs or uniforms. PPS + epoxy design. Available with both ISO 15693 and 14443 NFC memory chips.

[5] Stemedu, "Rc522 rfid reader writer module rf ic card sensor with s50 white card + writable key ring for arduino for raspberry pi nano nodemcu (pack of 2sets)," 2018, hardware component.