Harmonic

Author name: Ben Kadaner

Table of Contents

Introduction

Initiation:

- **Initial description of the project:** Harmonic is an application that allows the user to count a short sound segment within a long recording. The user will record a sound by, for example, pressing a button on the keyboard. Then he will record a longer sound segment, for example, himself typing, and then the application will count how many times the noise of pressing one of the keyboard buttons occurred. The application will be bouth for the phone and for the PC.

  In my opinion, in modern times, automation should be done for as many things as possible. Automation will lead to the fact that we will not have to invest in the simple things, and we will be able to invest our abilities in the more important things. So when my teacher told me that he had to count how many times a certain sound was repeated, I thought it would be a good idea to make a project about it.

  The application serves various purposes, for example enhancing sports performance. For instance, individuals aiming to track their punching repetitions or rope jumps can rely on the app, streamlining the process and eliminating the necessity for manual counting. This feature simplifies their training regimen and fosters greater efficiency.

  In the development of this application, I anticipate encountering two significant challenges. Firstly, I will need to acquire the skills to code for the Android platform, an area in which I currently lack experience. Secondly, the nature of my project presents a unique challenge: I must devise a method to accurately count sounds, with very few existing code examples to guide me. Overcoming these obstacles will be crucial to the success of my effort.

- **Audience description:** Harmonic is an application that will be used by those who wish to make their lives easier in various ways. Those who want to ease their work of counting will use the app. While it often goes unnoticed, many everyday actions produce distinct sounds. Harmonic enables users to identify and associate these sounds with specific actions, allowing them to effortlessly tally the occurrences of those actions. Harmonic simplifies the process of counting sound events in recordings by offering a user-friendly solution tailored to everyone.

  Here is a short list of people who may find the application useful:

  1. **Academic and Scientific Researchers**: Researchers who need to quantify specific sound events in their audio recordings as part of their academic or scientific work. For instance, zoologists studying bird behavior might need to count the number of bird calls within a given time frame.

  2. **Media Content Creators**: Content creators, such as podcasters, video producers, and sound designers, who want to monitor and tally particular sound effects or events within their content. For instance, a music producer working on a hip-hop track might need to count the number of times a specific snare drum sound is played within a section of the song.

  3. **General Users**: Individuals interested in tracking sound events within their lives for personal, recreational, or self-improvement purposes. For example, someone who has a habit of tapping their fingers when stressed might use the application to monitor and count the occurrences of this behavior as a means of self-awareness and stress management.

- **Goals:** The essence of this application lies in simplifying daily life by empowering users to effortlessly count sounds they encounter throughout their day. Whether it's tracking footsteps, counting repetitions during exercise, or monitoring other auditory cues, this application aims to streamline the process and optimize efficiency. By providing a user-friendly solution for sound counting, the application endeavors to enhance productivity and convenience in the lives of its users.

• Problem Definition and Benefits:

The problem - The user needs to find a way to count the number of occurrences of a sound in a recording.

What I am trying to achieve - I am trying to develop an application that lets the user the ability to record a short sound and a recording. Then the application will automatically count how many times the sound appeared in the recording. Essentially, I’m trying to automate the process of detecting specific sounds in audio recordings.

Benefits -

- Efficiency: The application empowers users to swiftly and precisely count occurrences without the need for manual effort. By automating the counting process, users can accomplish their tasks with ease and accuracy, saving valuable time and effort.
- Time-saving: With the capability to count occurrences in lengthy videos in real time, the application enables users to multitask effectively. This functionality allows users to focus on other tasks while the application automatically tallies occurrences, enhancing productivity and efficiency.
- Errors reduction: Automation within the application significantly minimizes the likelihood of human error when counting sound occurrences. Unlike humans, who may lose focus, become fatigued, or inadvertently skip or double-count occurrences, the application remains consistent and precise, ensuring accurate results without the limitations associated with human counting
- Increased productivity: By automating the task of counting occurrences, the application liberates users from manual counting, allowing them to allocate their time and energy to more productive endeavors. With the application handling repetitive counting tasks efficiently and accurately, users can achieve higher levels of productivity in their daily activities.

services -

- Sound Recording: The application offers a sound recording feature, enabling users to capture audio directly within the application. This functionality ensures seamless integration of sound recording and counting capabilities within a single platform.
- Sound Playing: The application offers a sound-playing feature, enabling users to play the audio they captured directly within the application. This functionality ensures seamless integration of sound playing and sound recording within a single platform.
- Target Sound Recording: Users initiate recording by simply pressing a button, triggering the application to commence capturing audio. As the recording progresses, the application simultaneously transmits the audio data to the server for analysis while continuing to record. This real-time process enables seamless counting of sound occurrences without interrupting the recording flow. The recording process persists until the user decides to stop.
- Sound Occurrence Counting: The application employs advanced algorithms to analyze the target sound and accurately count occurrences of the reference sound, such as keyboard button presses.
- User Interface: The application features an intuitive and user-friendly interface, simplifying the process of inputting sound segments, and accessing results. Its streamlined design ensures ease of navigation and accessibility, making for a seamless user experience.
- Output: The application will display the number of occurrences of the sound within the longer recording.
- Sound Saving: The users will be able to save sounds for future use.
- Login: Allows existing users to access their personal accounts securely.
- Sign Up: Every user can create their personal account using a username and password that will be saved in the server securely.

- ## Comparison of existing apps:

  In my research, I found the app “Exercise Sound Counter” which is a free iOS app that helps you track your exercise routine by listening to the sounds of your workout. It uses the phone's microphone to listen for sounds like clapping, running, and jumping, and then counts the number of times it hears each sound. You can also set goals for yourself in terms of the number of times you want to hear a sound, and the app will track your progress toward your goals.

Feature	Exercise Sound Counter	Harmonic
Purpose	Tracks exercise routine by listening to workout sounds	Counts short sound segments within a long recording
Input method	Microphone	Microphone
Sound analysis	Counts existing of loud sound	Counts occurrences of a reference sound within a longer recording
Output	Graphs and charts of exercise progress	Count of sound occurrences
Additional features	Goal setting	sound saving

Overall, Exercise Sound Counter is an app focused on tracking exercise routines, while Harmonic is a more versatile tool that can be used for a wider range of tasks.

Here are some specific points of comparison:

  - Exercise Sound Counter is specifically designed for tracking exercise routines. It uses the phone's microphone to listen for sounds like clapping, running, and jumping, and then counts the number of times it hears each sound. This makes it a good choice for people who want a simple and easy way to track their progress.

  - Harmonic is a more versatile tool that can be used for a wider range of tasks. It can be used to count occurrences of any sound, not just sounds related to exercise. This makes it more versatile toll.

In contrast to Shazam, which specializes in music identification, Harmonic is tailored for users seeking a tool to count occurrences of specific sounds in recordings. Shazam is a popular music identification app that allows users to identify songs by simply listening to a short snippet. It compares the audio fingerprint of the snippet with its vast database of recorded music to find a match. Once identified, users can access additional information about the song, such as artist, album, and lyrics, as well as options to listen to the full track on supported platforms.

Feature	Shazam	Harmonic
Purpose	Identifies songs and music	Counts short sound segments within a long recording
Input method	Microphone	Microphone
Sound analysis	Compares the recorded sound to a database of songs and music	Counts occurrences of a reference sound within a longer recording
Output	Song title, artist, and album information	Count of sound occurrences
Additional features	Song lyrics, music videos, and artist information	sound saving

Overall, Harmonic is a more specialized tool focused on counting sound occurrences, while Shazam is a more general music identification app.

Here are some specific points of comparison:

- Harmonic is specifically designed for counting occurrences of a reference sound within a longer recording. It allows users to record a short sound segment and then record a longer audio segment that contains the sounds they want to analyze. The tool will then compare the recorded sounds to the reference sound and count how many times the reference sound occurs. This makes it a good choice for people who need to count occurrences of a specific sound, such as a how many time a door was closed.

- Shazam is a more general music identification app. It can identify a wide range of songs, including genres like pop, rock, hip-hop, and classical. It also provides additional information about the identified song, such as the song title, artist, and album information. This makes it a good choice for people who want to know what song they are listening to.

- ## Project technology review:

  New and Unfamiliar Technology -
  My project leverages a technique known as Fast Fourier Transform (FFT) for sound analysis. While not entirely new, FFT remains underutilized in many consumer sound software applications compared to more established methods like spectrogram analysis or waveform comparison.

  FFT is an algorithmic technique employed for transforming time-series data into the frequency domain, particularly useful for analyzing sound signals. In the context of sound analysis, FFT enables the breakdown of audio segments into their constituent frequencies, allowing for detailed spectral analysis. It essentially acts as a sophisticated tool for frequency analysis, ensuring accurate identification and comparison of sound characteristics despite variations in timing or duration.

  System Definition Restrictions -

  In terms of system definition, there are certain restrictions and considerations to keep in mind:

  - **Device Requirements:** Users will require a computer that can tun windows if they wish to use the computer client. Those how chose to use the Android app will need a compatible Android device, with a minimum of Android 11 version, to ensure optimal audio recording and app performance.
  - **Microphone:** To maximize the app's capabilities, users are encouraged to use a functional and high-quality microphone for optimal audio recording.
  - **Data Transmission:** A stable internet connection is essential for sending recorded audio to the server, ensuring smooth data transfer.

- ## Project Delimitation:

  Harmonic primarily focuses on counting specific sound occurrences. While it is a sound analysis application, it does not directly handle complex network-related or operating system-specific functions. Here's a breakdown of how the project delimits its scope:

  Areas Where the Project Deals with Networks:

  - Internet Connection - Harmonic functionality relies on users having access to a network connection to transfer audio data. A stable internet connection is essential for efficient data transfer.
  - Communication Protocol - The project utilizes a TCP-based protocol for secure data transmission. This protocol ensures the confidentiality and integrity of the sent data, providing a secure communication channel between the client and the server.
  - Encryption - To keep the users' information safe, we use strong encryption methods. Specifically, we employ AES and Diffie-Hellman. AES ensures data is scrambled at the start and unscrambled at the end, preventing unauthorized access. Diffie-Hellman allows us to securely create a shared key between the client and server. These measures ensure that your data stays confidential and secure during transmission.

  Areas Where the Project Deals with Operating Systems:

  - Operating Systems (Android and Windows): The project includes both an Android app and a Windows application. The Android app operates within the Android operating system environment, utilizing its features for audio recording, user interaction, and data storage. Similarly, the Windows application operates within the Windows operating system environment, leveraging its respective features for the same functions.
  - Server Operating System: On the server side, where audio analysis takes place, the Harmonic server runs on a Windows operating system. The server does not address or influence the operating system besides opening and closing files as well as managing DB.

Areas Where the Project Doesn’t Deal with:

- Human Voice Recognition: Harmonic does not attempt to recognize specific speakers or distinguish human voices. The intricacies involved in accurately identifying individuals based on voice patterns are complex and fall outside the project's intended scope.
- Sound Recognition in Noisy Environments: Harmonic does not aim to recognize sounds in noisy or challenging environments. For instance, it will not count specific musical notes in a crowded concert hall. The project focuses on simpler sound analysis tasks within controlled or moderate environments.
- 100% Accuracy: Recognizing the practical limitations, Harmonic doesn't claim to achieve 100% accuracy, acknowledging the inherent challenges in sound analysis. The project strives for high precision but acknowledges the possibility of occasional inaccuracies.
- Artificial Intelligence: Unlike many systems with similar functions, Harmonic does not utilize artificial intelligence. The system opting instead to achieve its goals through mathematical methods. This decision reflects a focus on leveraging mathematical techniques for sound analysis rather than relying on AI.

Characterization:

- ## description of the system:

  System Overview:

  The system comprises a client and a server-side component designed to assist in sound recording, analysis, and occurrence counting. The application offers users a convenient platform to record, store, and analyze sound data, enhancing user experience and aiding in sound analysis tasks.

  Client Application:

  - **User Authentication -** The app includes a secure login system, enabling users to access their personal account.
  - **Sound Recording -** Users can record short sound segments, that will be used in later analysis. Additionally, users have the option to save these sound segments within the app for future reference and analysis.
  - **Sound Comparison and Counting -** Recorded sound segments are sent to the server for similarity comparison and occurrence counting.
  - **Data Encryption -** Users' personal information information is encrypted before transmission to the server to ensure secure data handling.
  - **Server Interaction -** The application communicates with a server, transmitting sound data for analysis and receiving occurrence counts.

Server-side Functionality:

- **Receiving Sound Data -** The server reliably receives recorded sound data from the application, ensuring that all packets are successfully delivered.
- **Similarity Analysis -** Conducts analysis to identify the occurrences of specific sound segments within longer recordings.
- **Counting Occurrences -** Utilizes advanced algorithms to count the occurrences of the recorded sound segments accurately.
- **Response to application -** Sends the occurrence count back to the user's application for display.
- **Sound Saving -** The application offers users the capability to save short sound files for future use, ensuring convenient access without the need for re-recording. These saved sound files can be stored within the server for easy retrieval and analysis whenever needed
- **Sound Library Management -** The server manages a repository of audio files, including short sound files used for comparisons and segments of long recordings for analysis. Short sound files are temporarily stored during the analysis process and deleted at the end of the recording session. In contrast, segments of long recordings are saved temporarily for analysis purposes and then swiftly deleted to optimize storage space and maintain data privacy.

Communication and Security:

- **Data Transmission Protocol -** The application and server communicate using TCP-based protocol, ensuring secure data exchange between the application and the server.
- **Encryption -** User information is encrypted to prevent unauthorized access and ensure data security.

User Experience Enhancement:

- The system is designed to improve the user's ability to analyze sound occurrences easily and effectively.
- Allows users to maintain a library of regularly used sounds for future use.

This system design aims to streamline the process of sound analysis, offering a user-friendly interface for sound recording and automated occurrence counting, all while maintaining data security and user privacy.

- ## Capabilities it will grant users:

  Harminc is an application that has many uses in the user's daily life. However, I believe that there are specific groups that will benefit the most:

  - **Academic and Scientific Researchers:**
    - **Precise Sound Event Analysis -** Enables the quantification and analysis of specific sound events within academic or scientific recordings.
  - **Media Content Creators:**
    - **Effortless Sound Monitoring -** Offers a convenient method to monitor and tally particular sound effects or events within their content.
    - **Quick Event Tallying -** Rapidly counts occurrences of desired sounds in audio content, aiding in content creation.
  - **Hobbyists and Enthusiasts:**
    - **Personal Sound Event Tracking -** Provides a user-friendly solution for tracking sound events within audio recordings.
    - **Recreational Analysis -** Supports hobbyist interests by enabling the counting of preferred sound elements in recordings for personal use.
  - **General Users:**
    - **Simplified Sound Occurrence Counting -** Provides an easy-to-use solution for identifying and counting everyday sound occurrences.
    - **Effortless Task Management -** Eases the task of tallying specific sounds in daily life, reducing the need for manual counting efforts.

  Harmonic's versatile capabilities cater to specific user needs, enabling effective utilization across various fields such as professional analysis, content creation, and more, offering a user-friendly and accessible solution for individuals seeking to simplify counting tasks outside specialized categories. This provides a convenient and efficient tool for their everyday needs.

- ## Detail of tests:

Sound Recording Test:
- Record sound segments of varying lengths under different environmental conditions (quiet and noisy settings).

Sound Occurrence Counting Test:
- Record a reference sound within a longer segment, ensuring varied occurrences.
- Check if the system accurately counts the occurrences matching the reference sound.

User Interface Test:
- Have testers interact with the application's interface to create an account, log in, record, save sounds, and view results.
- Evaluate the user-friendliness, intuitiveness, and effectiveness of the interface.

Data Transmission Test:
- Send recorded sound files and evaluate if they were received properly.

Error Handling Test:
- Introduce incorrect file formats, non-functional internet connections, and other error conditions.
- Assess how the system responds to and manages error scenarios.

- **Time management:**

Topic	Date
Searching for a topic for the project	October
Project proposal	31/10/2023
Characterization	30/11/2023
Implementation	25/12/2023
Progress report 20 percent code	20/1/2024
Progress report 50 percent code	1/3/2024
100 percent code progress report - and the start of tests	31/3/2024
Pre-test	Start of May
Project file submission	1/5/2024

- ## Risks:

  Not Being Able to Recognize Sounds:
  - **Planned Actions-**
    - Thorough Testing: Implement extensive testing to refine the voice recognition algorithm continually.
    - Step-by-Step Improvements: Make small changes to the voice recognition feature bit by bit based on how it worked in testing.
  - **Actual Implementation-**
    - Implemented continuous testing and feedback loops during development.
    - Iteratively improved the voice recognition algorithm based on many tests.

  User Usability Challenges:
  - **Planned Actions-**
    - User-Friendly Interface: Develop an intuitive applycation interface for simple use.
    - Feedback Handling: Implement changes requested by different users.
  - **Actual Implementation-**
    - Designed a user-friendly interface with clear use instructions.
    - Changed many things about the applycation appearances. Reduced two screens, increased font size, and more.

  Learning Android:
  - **Planned Actions-**
    - Training and Resources: Invest time in learning Android development through tutorials and documentation.
    - Consultation: Seek guidance from experienced Android developers when facing challenges.
  - **Actual Implementation-**
    - Dedicated time to study Android development materials.
    - Collaborated with my friend and my father for mentorship.

Sound Recording:

- **Planned Actions-**
    - Online Code Adoption: Utilize online resources to integrate desired functionalities into the project.
    - Consultation: Seek assistance from experienced programmers to resolve integration issues.
- **Actual Implementation-**
    - Attempted to incorporate code snippets found online into the project, but encountered compatibility issues. Resorted to analyzing and modifying the source code of the libraries used to address the integration challenges.
    - I got assistens from my father who is familiar with some of the libraries I used.

Math Understanding:

- **Planned Actions-**
    - Educational Resources: Engage in focused learning of required mathematical concepts.
    - Collaboration: Collaborate with more knowledgeable people to overcome math-related challenges.
- **Actual Implementation-**
    - Engaged in targeted learning sessions to enhance mathematical understanding of the topic.
    - Collaborated with my mother and grandmother (wich both have master's degrees in mathematics) to address specific mathematical aspects.

Field of knowledge

Analysis:

- **Capabilities:**

server:

The Name of the Ability: Sign up user
Essence of the Ability:
Registration of a new user.
Collection of Required Abilities:
  - Communication
  - Encryption
  - Decryption
  - packet integrity check
  - DB check
  - Enter credential into DB
  - Return result

Necessary Objects:
  - Encryption/Decryption
  - Communication
  - DB
  - multithreading

The Name of the Ability: Log in user
Essence of the Ability:
Connect users to their accounts.
Collection of Required Abilities:
  - Communication
  - Encryption
  - Decryption
  - packet integrity check
  - DB check
  - Comper credential to DB info
  - Return result

Necessary Objects:
  - Encryption/Decryption
  - Communication
  - DB
  - multithreading

The Name of the Ability: Sound-saving
Essence of the Ability:
Save users sounds for future use
Collection of Required Abilities:
- Communication
- Sound reciving
- packet integrity check
- DB check
- DB insersion
- Return result

Necessary Objects:
- Communication
- File system
- DB
- multithreading

The Name of the Ability: Sound reciving
Essence of the Ability:
Receive sound file from user
Collection of Required Abilities:
- Communication
- packet integrity check
- File management

Necessary Objects:
- Communication
- File system

The Name of the Ability: Sound names returing
Essence of the Ability:
Return all the sound names of the user
Collection of Required Abilities:
- Communication
- packet integrity check
- DB check
- Return result

Necessary Objects:
- Communication
- DB

The Name of the Ability: <u>Sound counting</u>
Essence of the Ability:
Return all the sound names of the user
Collection of Required Abilities:

- File management
- Sound comparison
- Return result

Necessary Objects:

- Communication
- File system

client:

The Name of the Ability: Sign up
Essence of the Ability:
Registration of a new user.
Collection of Required Abilities:
- SignUp screen
- Credential check against the rules
- Communication
- Encryption
- Decryption
- packet integrity check
- DB check
- Return of the result
- Display result

Necessary Objects:
- User API
- Encryption/Decryption
- Communication
- DB

The Name of the Ability: Log in
Essence of the Ability:
Connect users to their accounts.
Collection of Required Abilities:
- Login screen
- Check if one of the credential values is empty
- Communication
- Encryption
- Decryption
- packet integrity check
- DB check
- Return of the result
- Display result

Necessary Objects:
- User API
- Encryption/Decryption
- Communication
- DB

The Name of the Ability: Record
Essence of the Ability:
Record sound through a microphone
Collection of Required Abilities:
- Recording screen screen
- Permissions check
- Permissions requests
- File management
- Sound listening

Necessary Objects:
- User API
- File system
- multithreading

The Name of the Ability: Sound-saving
Essence of the Ability:
Save sound for future use
Collection of Required Abilities:
- Sound saving pop-up
- Sound name legality check
- Permissions check
- Permissions requests
- File management
- Sound sending
- Display result
- DB update

Necessary Objects:
- User API
- File system
- Communication
- DB
- multithreading

The Name of the Ability: Sound-sending
Essence of the Ability:
Send sound to the server
Collection of Required Abilities:
- Permissions check
- Permissions requests
- File management
- packet integrity check
- Return of the result
- Communication
- Display result
- DB update

Necessary Objects:
- User API
- File system
- Communication
- DB
- multithreading

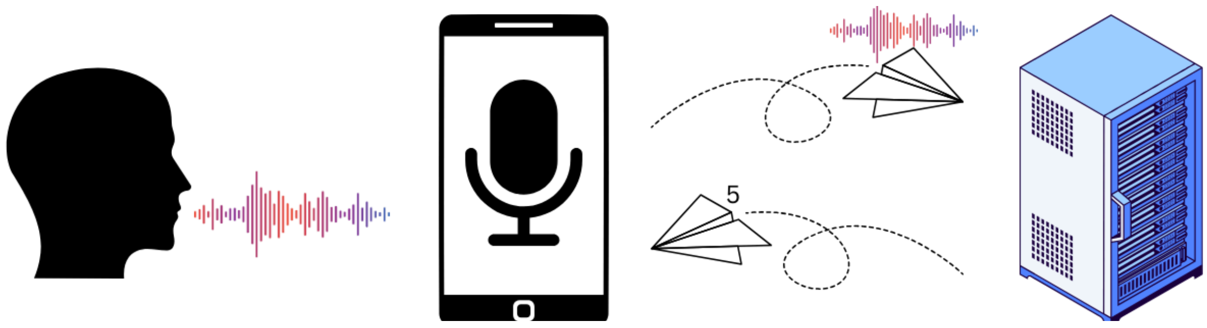
Architecture

Description of the architecture

- ### Hardware Description:

  Server-Side Hardware:
  - **Server Infrastructure -** This would consist of physical servers.
  - **Storage Devices -** Storage station for data retention and retrieval.
  - **Networking Equipment -** Routers and switches facilitating data transfer.

  Client-Side Hardware:
  - **End-User Devices: -** Smartphones, and other devices like computers can run the application with the proper emulator if you wish to use the phone app. To use the computer application you will need a computer or a proper device to run Windows 10 or later.
  - **Microphones -** Hardware integral for recording sounds.



- ## Technology:

  Programming Language:
    - **Python -** Utilized for server-side scripting due to its versatility and vast libraries in signal processing, data analysis, and server-side functionalities. Used in the computer client due to its compatibility with recording and file management.
    - **Java (Android) -** Employed for developing the client-side application on the Android platform for sound recording and user interface as wall as file management.

  Operating Platform:
    - **Android OS -** Used for running the client-side application on various Android devices or emulators.
    - **Windows -** Used for running the server side and the computer client.

  Communication:
    - **TCP Protocol -** Utilized for secure data transmission between the client and the server.

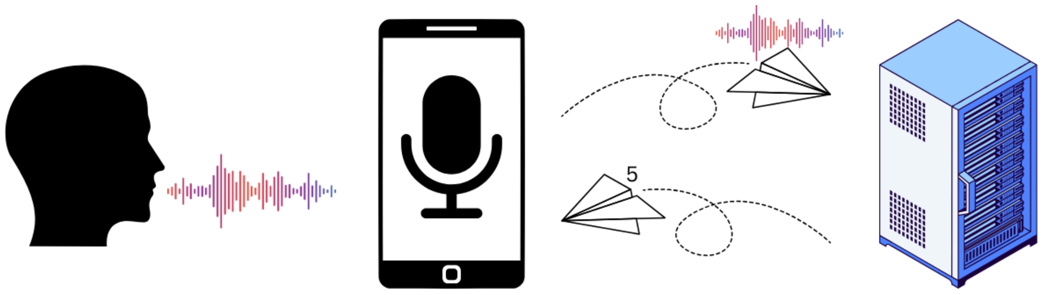
  Areas of Interest:
    - **Signal Processing -** Leveraging Dynamic Time Warping (DTW) for time series analysis and similarity comparisons in sound recognition.
    - **Data Security -** Emphasized user authentication and encryption for ensuring privacy.
    - **Graphical User Interface (GUI) -** Focused on creating a user-friendly interface for the client by designating each part of the process to create a simple flow.
    - **Recording -** Central to the functionality of the app, enabling users to record, save, and reuse sound segments for subsequent analysis.

• Flow Of Information:

***The graph shows a phone but it works the same with both types of clients**

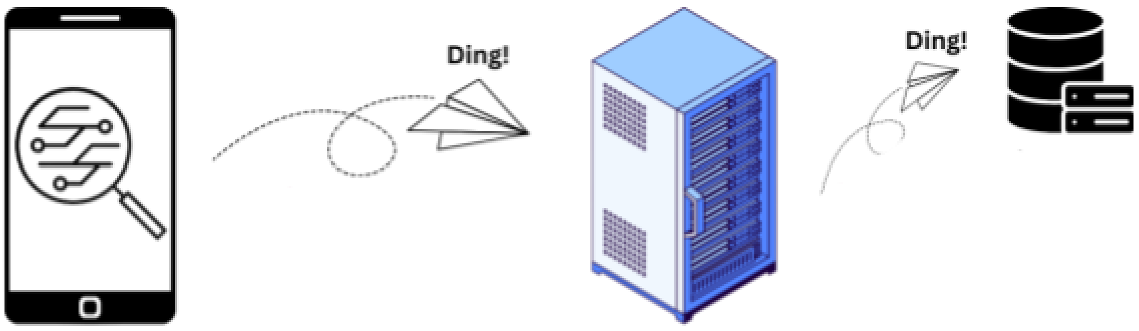
Data Processing:

- **From User Application to Server -** Recorded sound data is sent from the user to the server for analysis and processing.
- **From Server to User Application -** The number of occurrences is sent back to the user application for display. The only time the user doesn't get a number is when he sends the sound he is searching for.



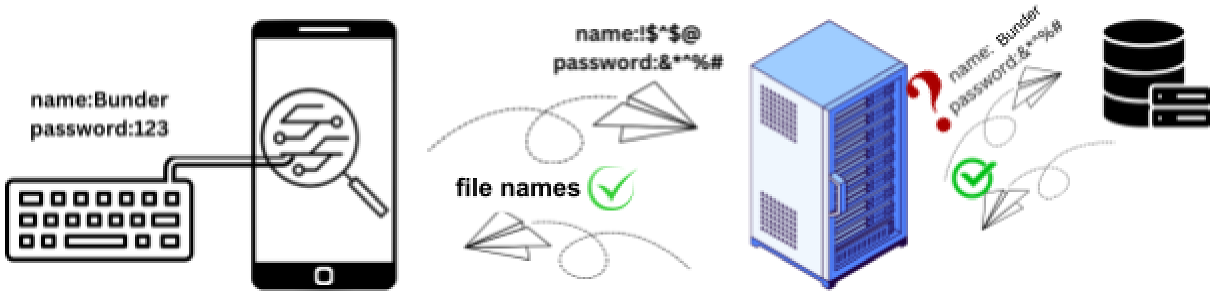
Sound Storage:

- **From User Application to Server -** Sounds that the user wishes to use later are sent to the server for storage. The sound files are saved in the server file system and the names of the files are saved in the DB.
- **From Server to User Application -** Previously saved sound names are retrieved from the server upon user connection.



User Authentication and Security:

- **From User Application to Server -** User credentials, are sent to the server for validation.
- **From Server to User Application -** Confirmation or denial of access is transmitted back to the user application. If the user information was validated they also get their sound names of the sounds they saved.



(Login and signup share a very similar information flow)

● Main Algorithms:

Formulation and analysis of the algorithmic problem:

The algorithmic problem involves analyzing and comparing two sounds that are not completely the same. Sound is natural, which means that it will not be the same. In my application, I am trying to count the number of occurrences of a sound in a recording. All of the sound elements are natural, which means that they will never be the same. I need to find a way to match two sounds even when they are different.

Similarity Calculation:

- **Fourier Transform -** The Fourier Transform, particularly its faster version known as FFT, is a mathematical tool used to convert sound data from its original form in the time domain to a representation in the frequency domain. This transformation essentially breaks down complex sound signals into their constituent frequencies, facilitating comparisons by revealing the underlying spectral content of the sound.
- **Pattern Matching -** DTW is an algorithmic technique employed for aligning and comparing sequences of data, particularly useful for time-series data like sound signals. In the context of sound analysis, DTW enables the comparison of audio segments by allowing flexible alignment even when the segments have different lengths or temporal distortions. It essentially acts as a sophisticated pattern-matching tool, ensuring accurate comparison and identification of sound occurrences despite variations in timing or duration.

Relevant Sources:

FFT

- Senin, M., & Goldberg, I. (2009). "A Comparative Study of Dynamic Time Warping, Cross-Correlation, and Waveform Similarity for Audio Signal Matching."
- Brigham, E. O., & Morrow, R. E. (1969). "Fast Fourier Transform and Its Applications."
- Bishop, C. M. (2006). "Pattern Recognition and Machine Learning."
- Smith, J. (2007). "Fourier Transforms in Audio Analysis."
- Salamon-Jespersen, S., & McGowan, R. W. (1994). "Dynamic Time Warping for Speech Recognition."

DTW

- Yurika Permanasari (2019). “Speech recognition using Dynamic Time Warping”
- Pavel Senin (2008). “Dynamic time warping algorithm review”
- Eamonn Keogh & Chotirat Ann Ratanamahatana (2004) “Exact indexing of dynamic time warping”
- Kongming Wang & Theo Gasser (1997). “ALIGNMENT OF CURVES BY DYNAMIC TIME WARPING”

Chosen Algorithm Rationale:
I chose FFT over DTW for this problem due to the following reasons:

- **Focus on frequency analysis -** FFT excels at analyzing sustained frequencies, which is ideal for identifying recurring sound patterns like finger snaps. Many sounds, although short bursts, have distinct frequency components that FFT can accurately capture and analyze.

  DTW is designed for capturing rapid changes within the signal, which might be suitable for some short events. However, it struggles with frequency-based analysis that many sounds require, making it less effective in accurately identifying and differentiating sounds.

- **Accounting for variations -** Real-world sounds exhibit natural variations in volume, pitch, and duration. FFT's ability to break down signals into their frequency components allows it to handle these variations effectively. This makes it easier to establish a consistent reference for identifying similar sounds in the recording.

  DTW's warping capabilities are better for aligning sequences with time changes, but FFT's frequency analysis provides a more robust approach for differentiating sounds amidst natural variations and background noise.

- **Flexibility for user-defined sounds -** My application doesn't limit itself to a specific sound. FFT can be adapted to analyze various sounds by focusing on their frequency characteristics. By creating reference FFT patterns for different sounds, it becomes possible to identify and count occurrences of user-defined sounds like coughs or claps accurately.

  DTW is less effective in this context as it requires precise alignment over time, which can be challenging for varied sound types and less adaptable for frequency-based analysis needed for different sounds.

● Development Environment:

Development Tools:

Client-side phone App:

- **IDE:** Android Studio
- **Programming Language:** Java (for Android)
- **Emulator or Physical Device:** Device Emulator or Physical Android Device

Client-side PC application:

- **IDE:** PyCharm
- **Programming Language:** python
- **Physical Device:** PC to run the application

Server-side:

- **IDE:** PyCharm
- **Programming Language:** Python
- **Database Management System:** SQLite for local testing and development

Testing Tools:

- **Android:** Emulator in Android Studio or Physical Android Device for application testing
- **PC-client:** An average PC
- **Server-side:** An average PC

● Communication Protocol:

My communication protocol is billed this way:

Client:

(Message size)|(the length of the message until the data)~(request code)~(additional variables if needed that are each separated with “~”)~ data

Explanation of each part:

- **Message size -** To ensure that none of the packets were not missed I am putting the size of the packet at the start to ensure the packet was received fully.
- **Length of the message until the data -** Because I am mostly sending sound files it will be inefficient to decode all the data that is transferred.
- **Request code -** each message that is transferred needs to be dealt with accordingly. The request code ensures that the server will know how to deal with the message.
- **Variable -** In this part are all the variables that will be decoded by the server. Here are all the textual messages. For example username password and more.
- **Data -** In this part the sound is saved. Unlike the other part, this is not decrypted to text to ensure simple transmission into a file on the server side.

Server:
(Message size)| data
Explanation of each part:

- **Message size -** To ensure that none of the packets were not missed I am putting the size of the packet at the start to ensure the packet was received fully.
- **Data -** The messages from the server a fairly simple. They all are nothing more than a couple of words.

Client Request codes:

- **Login -** Emphasize that the client is training to log in. With this, there are also 2 variables attached. The variables are the user credentials (username and password).
- **SignUp-** Emphasize that the client is training to sign up as a new user. With this, there are also 2 variables attached. The variables are the user credentials (username and password).
- **ShortRecordSave -** Emphasize that the client is training to save the short sound at the server files system for single time use. With this, there are also 3 variables attached. The variables are the user's username, whether the packet is the last or not (1 or 0), and finally the sound.
- **ShortRecordExist -** Emphasize that the client is training to use one of the sounds he saved. With this, there are also 2 variables attached. The variables are the user's username and the sound name.
- **LongRecord -** Emphasize that the client is sending the recording stream to the server files system. With this, there are also 2 variables attached. The variables are the user's username and finally the sound.
- **LongRecordPy -** Emphasize that the client is sending the recording stream to the server files system. With this, there are also 3 variables attached. The variables are the user's username, whether the packet is the last or not (1 or 0), and finally the sound.
- **SaveRecord-** Emphasize that the client is training to save the short sound at the server files system for future use. With this, there are also 3 variables attached. The variables are the sound name, the user's username, whether the packet is the last or not (1 or 0), and finally the sound.

Sender → Receiver	Message Name	Message example	Message Description
Client → Server	Log in	Login~Rick~ad23c~	Login - message code Rick- username ad23c - user's password
Server → Client	Logged in seccsefuly	Username and password match	Plain text that the application gets to know that it can move to the next screen
Server → Client	Logged in unseccsefuly	Username and password do not match	Plain text that the application gets to know that the username and password do not match
Client → Server	Sign up	SignUp~Rick~ad23c~	SignUp- message code Rick - username ad23c - user's password
Server → Client	Signed up seccsefuly	Sign up successful	Plain text that the application gets to know that the user credentials were saved
Server → Client	Signed up unseccsefuly	Sign up failed	Plain text that the application gets to know that there was a problem with the registration

Server → Client	Username in use	Username is in use	Plain text that the application gets to know to tell the user that the username is unavailable
Client → Server	Save short sound for single use	ShortRecordSave~Rick~0~011010100110011	ShortRecordSave- message code Rick - username 0 - the packet is not the last 0110…- the sound in binary form
Server → Client	Sound saved successfully	Saved short record	Plain text that the application gets to know that the sound is saved
Server → Client	Received sound part successfully	Got short record	Plain text that the application gets to know that the packet got to the server
Server → Client	Received sound part unsuccessfully	Error saving record	Plain text that the application gets to know that there was a problem with the saving
Client → Server	Use save sound	ShortRecordExist~Rick~bop~	ShortRecordExist- message code Rick - username bop - sound name
Server → Client	Make short record	Saved short record	Plain text that the application gets to know that the server prepared the sound
Server → Client	Error making the short record	Error making short record	Plain text that the application gets to know that the server didn’t manage to prepare

			the sound
Client → Server	Get sound stream	LongRecord~Rick~011010100110011	LongRecord- message code Rick - username 0110…- the sound in binary form
Server → Client	Occurrences count	Number of occurrences: (number)	Plain text that the application gets to know how many times the sound appeared in the stream this time
Server → Client	Received stream unsuccessfully	Error saving record	Plain text that the application gets to know that there was a problem with the saving
Client → Server	Save short sound for future use	SaveRecord~Bop~Rick~0~011010100110011	SaveRecord- message code Bop - sound’s name Rick - username 0 - the packet is not the last 0110…- the sound in binary form
Server → Client	Sound saved successfully	Saved file	Plain text that the application gets to know that the sound is saved
Server → Client	Received sound part successfully	Got file part	Plain text that the application gets to know that the packet got to the server
Server → Client	Received sound part unsuccessfully	Error saving record	Plain text that the application gets to know that there was a problem with the saving
Client → Server	Get saved sound names	GetSoundsNames~Rick	GetSoundsNames- message code Rick - username

Server → Client	Sound’s names	Bop~Zing~Bam	All the different sound names the user saved, spaced with “~”
Server → Client	Unexpected code	unidentified code	The server sends the application that the message code doesn’t exist for help in bug fixing
Server → Client	Unexpected error	Error: {error}	The server sends the application that there was an error on the server side for help in bug fixing doesn’t exist for help in bug fixing
Client → Server	Save long sound for single use from computer	LongRecordPy~Rick~0~011010100110011	LongRecordPy- message code Rick - username 0 - the packet is not the last 0110…- the sound in binary form
Server → Client	Received recording part successfully from the computer	Got long record	Plain text that the application gets to know that the packet got to the server

*I created two separate methods and sets of code to capture the long recording from the computer and the phone, due to their different recording processes. All other methods and codes are used for both devices.

- ## System Screens:

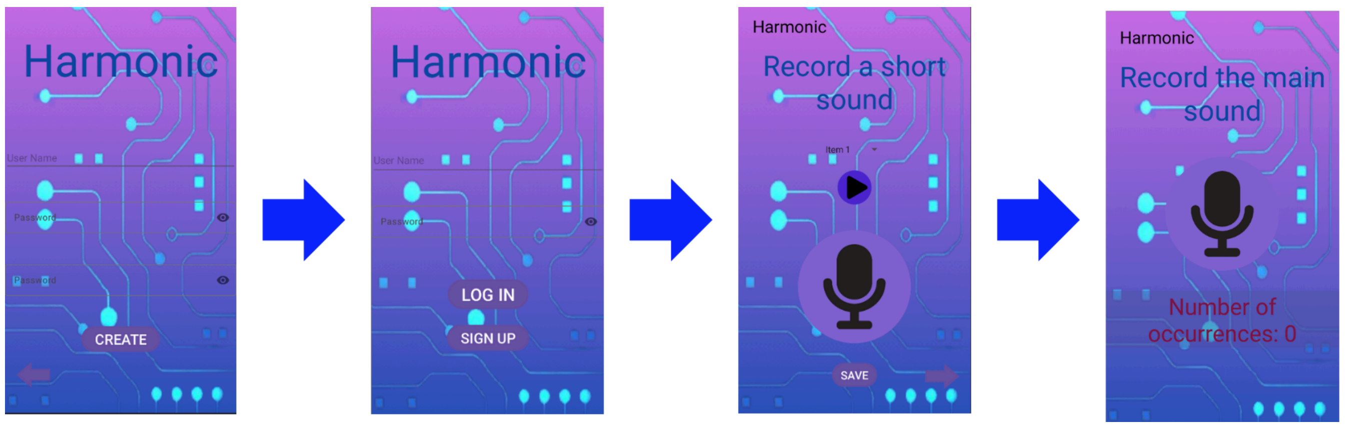
  Login Screen:
  - **Description:** Allows users to input their username and password for identification.
  - **Function:** Authentication and user login.

  Sign Up Screen:
  - **Description:** Allows users to input their username and password for registration.
  - **Function:** Signing up.

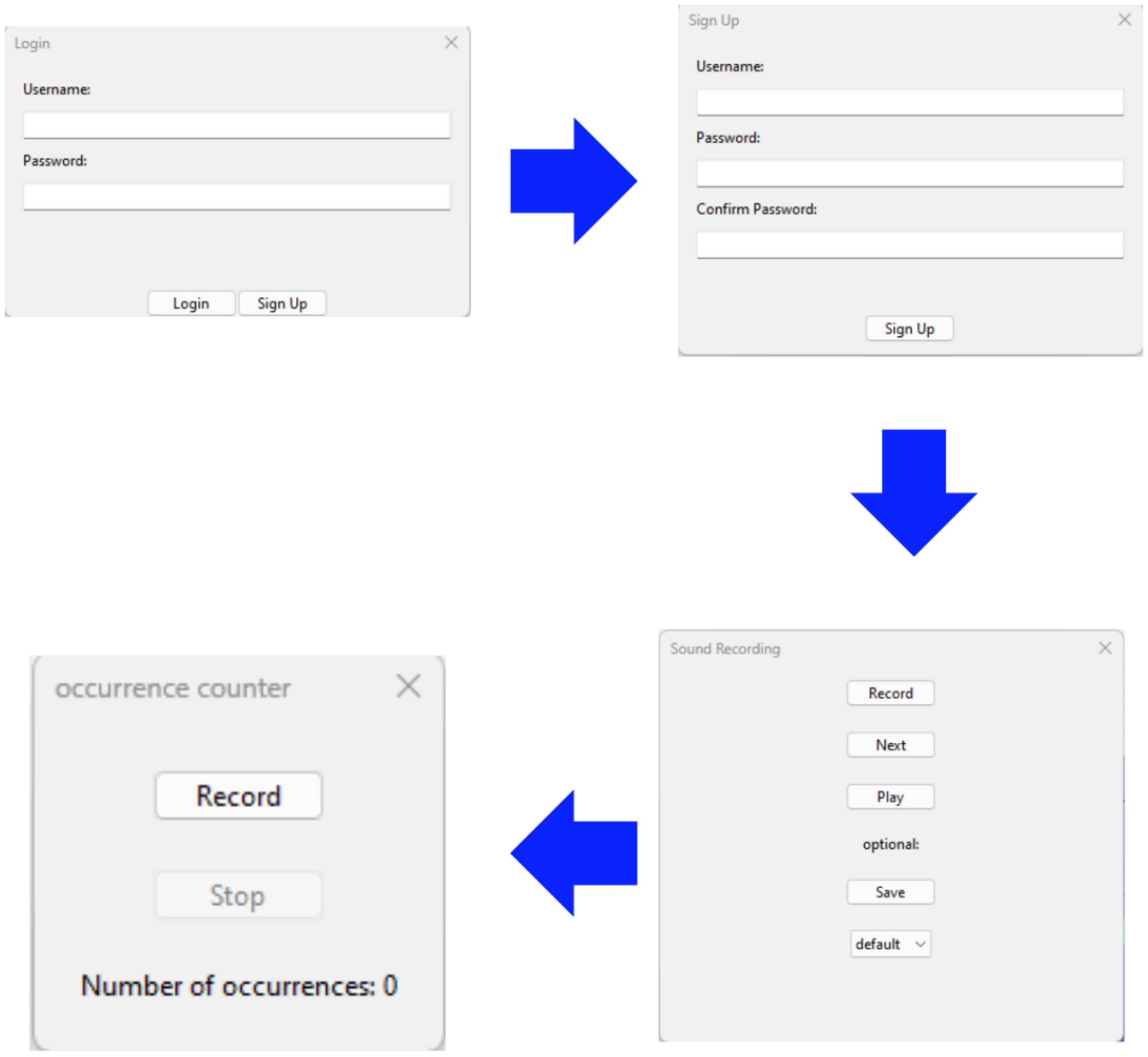
  Sound Recording:
  - **Description:** Users can record a short sound or select from saved ones, save new recordings, and listen to the new recorded sound.
  - **Function:** Capture new sounds, display saved recording names, save new sounds, and play the new sound recorded.

  Longer Recording Screen and Sound Counter:
  - **Description:** Enables users to record the longer audio stream and display the current sound occurrences.
  - **Function:** Captures the longer recording that will be sent to the server for analysis as well as receiving the server results.

Phone


Harmonic
User Name
Password
Password
CREATE
Harmonic
User Name
Password
LOG IN
SIGN UP
Harmonic
Record a short sound
Item 1
SAVE
Harmonic
Record the main sound
Number of occurrences: 0

PC:


Login
Username:
Password:
Login
Sign Up
Sign Up
Username:
Password:
Confirm Password:
Sign Up
Sound Recording
Record
Next
Play
optional:
Save
default
occurrence counter
Record
Stop
Number of occurrences: 0

- ## Files structure:

DB structure:

DB file name: Login.db

Table: Users
Fields:
  - username (TEXT PRIMARY KEY)
  - salt (BLOB)
  - password (TEXT)

Example Values:
  - username: rick_astley
  - salt: (binary data)
  - password: [Hashed Password]

Table: SavedSounds
Fields:
  - sound_name (TEXT)
  - file_name (TEXT)
  - userid (INT)
  - FOREIGN KEY (userid) REFERENCES Users(username)

Example Values:
  - sound_name : Bop
  - file_name : rick_bop_long.wav
  - userid : 1
  - SoundName: "Sample Sound"

The sound files will be saved in the server-specific folder to ensure easy access and good file management.

• Overview of weaknesses and threats:

Application Layer:

SQL Injection-
To mitigate SQL injection, the application will use prepared statements, ensuring that user inputs are treated as data rather than executable code. This prevents malicious SQL code from being injected into queries.

Login Process, Authentication, and Verification-
To reinforce secure authentication practices, the application will implement robust cryptographic techniques such as AES encryption for storing and transmitting sensitive data, like passwords. Furthermore, Diffie-Hellman key exchange will be employed to establish secure communication channels, enhancing the integrity and confidentiality of authentication processes. These measures ensure that user credentials are safeguarded against unauthorized access or manipulation.

MITM (Man-in-the-Middle) Attacks-
To mitigate against MITM attacks, the application will employ encryption, leveraging AES for data confidentiality and integrity. Additionally, Diffie-Hellman key exchange will be utilized to securely establish shared secret keys between the client and server, thwarting potential eavesdropping or tampering attempts. This approach ensures that communication remains confidential and resistant to unauthorized interception.

DDoS/DOS-
To mitigate against DDOS attacks, the application will use two different meggers. The first one will be to use socket.listen(x) in the connection of the users. This method will return an error to all the users/bots that are trying to connect when all x places are occupied. This method will reduce major waste of time in dealing with bots because they will get an extremely fast error. The second method will be used is, to kick anyone who doesn't log in within five minutes. Five minutes is more than sufficient time to connect even for users who struggle with the application. This method is required to make sure that bots will not occupy space that could be used by users.

Traffic Layer:

TCP Protocol and 3-Way Handshake-

Addressing threats related to the TCP protocol. Using secure and well-established TCP settings ensures the integrity of the communication protocol, guarding against potential vulnerabilities. In my application, the 3-way handshake happens automatically in the connection between the server and the client.

Encryption-

For secure data transmission, the application will implement strong encryption, AES, and DH.

System Vorenabilitys-

The system does not offer promises that the sound was transferred perfectly. If there were an MITM that would capture only the sound files he would be able to sabotage the counting. The application does not offer this protection because it will elongate the time that is required for file transmissions.

Project implementation

Details of the implementation of the system
Part A:

- **Overview of all the modules of the system:**
  Server:
  Imported Modules:
  - **numpy-** A fundamental package for scientific computing with support for large, multi-dimensional arrays and matrices.
  - **librosa-** A package for music and audio analysis. It provides functions for feature extraction, such as loading audio files and analyzing audio signals.
  - **sys.byteorder-** A module for determining the byte order of the system.
  - **array.array-** A module for creating and manipulating array-like objects.
  - **struct.pack-** A module for packing data into binary strings.
  - **hashlib-** A module for hashing the password using the salt.
  - **loguru-** Records messages about my program's execution for debugging and analysis.
  - **sqlite3-** Enables interacting with SQLite databases to store and retrieve data.
  - **uuid-** Generates universally unique identifiers (UUIDs) to uniquely identify things.
  - **Wave-** Method used to edit and create a wave file.
  - **Crypto.Cipher.AES-** A method that is used for the AES.
  - **Crypto.Util.Padding.pad/unpad-** Method that is used for setting a fighting message size in the encrypted communication.
  - **base64.b64encode/b64decode-** Used to transfer from readable to binary format.
  - **datetime.datetime/timedelta-** used to kick the client if he doesn’t logIn in 5 minutes.
  - **asyncio-** Used for running the sound streaming while analyzing the received parts.
  - **atexit-** Used to register a shutdown function called on_exit that is designed to perform clean-up actions when the program exits.
  - **os -** Used to manage files and get information such as size.
  - **random-** Make sure file names will not collide.
  - **socket-** Makes the communication with the client.

- **concurrent.futures.ThreadPoolExecutor-** Make managing multiple client communication possible.
- **sys-** Provides access to system-specific parameters and functions.
- **re-** Used to compile regular expressions.
- **scipy-** Used for calculation like FFT.
- **scipy.signal -** Used for signal processing, such as Fourier transforms.
- **matplotlib.pyplot -** Used to visualize the sound waves.

Developed Modules-

- **DBHelper-**
  Role: Manage DB. Run method, retrieve values, and more.
  Class variables:
  - Table names and table columns are used to create and point.

  Methods:
  - ***create_table***
    - ***Method input:*** none
    - ***Method performance:*** Create the two tables that the server uses.
  - ***insert_data_to_users_table***
    - ***Method input:***
      username - The client's username
      password - The client's password
    - ***Method performance:*** Create the salt for hashing the password. Additionally, insert the user credential and salt into the “users” table. Return success or failure. (boolean)
  - ***insert_data_to_sounds_table***
    - ***Method input:***
      username - The client's username
      sound_name - The name of the sound the user chose
      file_name - The name of the file that contains the sound
    - ***Method performance:*** insert the sound name and file name and match it to the foreign key using the username. Return success or failure. (boolean)
  - ***check_username***
    - ***Method input:*** username- the name the client wishes to use.
    - ***Method performance:*** return whether or not the username is in use. (boolean)
  -

- ■ ***check_username_password***
    - ***Method input:***
      username - The client's username
      password - The client's password
    - ***Method performance:*** Checks if the username and password match an existing user and return the result. (boolean)
- ■ ***check_user_sounds***
    - ***Method input:***
      username - The client's username
    - ***Method performance:*** Return all the sound names that are connected to the username. (list)
- ■ ***get_file_name_from_sound***
    - ***Method input:***
      sound_name- The sound name
    - ***Method performance:*** Return the file name associated with a sound name. (string)
- ■ ***close***
    - ***Method input:*** none
    - ***Method performance:*** Closes the database connection.

○ **send_receive_encrypted-**
Role: Send and receive encrypted messages.
Class variables:

- ■ CLIENTS_AES_INFO- A dictionary containing the user's AES information, client_addr:(aes_key, iv).

Methods:

- ■ ***new_client_key***
    - ***Method input:***
      client_addr - The address of the client
      aes_key - The key for the AES decryption/encryption
      iv - fixed-size input used alongside the encryption key to ensure unique ciphertexts for the same plaintext input
    - ***Method performance:*** Insert the user AES info into the dictionary.
- ■ ***send_encrypted***
    - ***Method input:***
      sock - Client socket.
      client_addr - Client address.
      to_send - The content that is wished to be sent.
      loop - A variable that is needed for the multithreading
    - ***Method performance:*** Send the content to the client safely.
- ■

- ***recv_decrypted***
    - ***Method input:***
      sock - Client socket.
      client_addr - Client address.
      loop - A variable that is needed for the multithreading
    - ***Method performance:*** Receive a message from the client securely and return it. (bytes)

- **tcp_by_size-**
  <u>**Role:**</u> Send and receive messages.
  <u>**Class variables:**</u>
    - SIZE_HEADER_FORMAT - The header that contains the size of the message to ensure the full arrival of the packet.
    - size_header_size - the size of the header itself. The header has a size therefore you need to count it in the overall size.

  <u>**Methods:**</u>
    - ***send_with_size***
        - ***Method input:***
          sock - Client socket.
          bdata - The content that is wished to be sent.
          loop - A variable that is needed for the multithreading
        - ***Method performance:*** Send the content to the client.
    - ***recv_by_size***
        - ***Method input:***
          sock - Client socket.
          loop - A variable that is needed for the multithreading
        - ***Method performance:*** Receive a message from the client and return it. (bytes)

-

- **Process-**
  <u>**Role:**</u> Count the number of sound occurrences of a sound in a recording.
  <u>**Class variables:**</u>
  - dtype - How is the data stored
  - n_fft - The size needed to calculate the FFT
  - win_length - The Amout of samples to be adited to gether.
  - hop_length - set the spacing betwin the samples
  - hz_count - The max hertz
  - sample_rate - Recording sample rate
  - sample_crop_start - skip the first 4 because they seem to get damaged.
  - sample_crop_end - At what sample from the end to stop searching.

  <u>**Methods:**</u>
  - ***counter***
    - ***Method input:***
      source_path - Path to recording file.
      matching_sample - Path to the short sound file.
    - ***Method performance:*** Returns the amount of occurrences of the sound in the recording.
  - ***analyze_sound***
    - ***Method input***
      sample_path- Path to the short sound file.
    - ***Method performance:*** Creat a image of the wave of the sound and analyze the sound.

  - ***stft_raw***
    - ***Method input:***
      series - The data that we want to edit.
      sample_rate - The sound sample rate.
      win_length - The Amout of samples to be adited to gether.
      hop_length - set the spacing betwin the samples
      hz_count - The max hertz
      dtype - How is the data stored
    - ***Method performance:*** Anelyze the sound using FFT.

- **main-**
  Role: Manage and run the server.
  Class variables:
  - executor - Thread for shutting down
  - file_short_record - List that hols the segment of the short sound file.
  - file_long_record - List that holds the segment of the record file.
  - FRAMES_PER_SECOND - The frame rate of the sound
  - p_str - Random number to make the encryption stronger.
  - G\P\A - Vairebuls for the AES encryption.
  - srv_DPH_key - The server private key.

  Methods:
  - ***prepare_hybrid_encryption***
    - ***Method input:***
      sock - client socket
      client_addr - client address
      loop - used to make the method async
    - ***Method performance:*** Prepar the encryption betwin the client and server.
  - ***login_user***
    - ***Method input:***
      username - user username
      password- user password
      DB - The class that manage the server DataBase
      login_timeout_task - kick the client if he doesn’t connect to his user in 5 minutes.
    - ***Method performance:*** log in user into DB and return success or failure type.
  - ***sign_up_user***
    - ***Method input:***
      username - user username
      password- user password
      DB - The class that manage the server DataBase
    - ***Method performance:*** sign up user into DB.
  - ***save_short_record***
    - ***Method input:***
      username - user username
      state - mark if the packet is the last one or not.
      content - The sound content. (byte array)
    - ***Method performance:*** save sound to single use.

- ***make_short_record***
  - ***Method input:***
    username - user username
    sound_name - User sound choice out of the saved ones
    DB - The class that manage the server DataBase
  - ***Method performance:*** convert the saved files into a short record format.
- ***count_occurrences***
  - ***Method input:***
    username - user username
    content - The sound content. (byte array)
  - ***Method performance:*** count the number of occurrences of the short sound in the long recording.
- ***count_occurrences_py***
  - ***Method input:***
    username - user username
    state - mark if the packet is the last one or not.
    content - The sound content. (byte array)
  - ***Method performance:*** count the number of occurrences of the short sound in the long recording.

- ***save_record***
  - ***Method input:***
    sound_name - User sound choice out of the saved ones
    username - user username
    state - mark if the packet is the last one or not.
    content - The sound content. (byte array)
    DB - The class that manage the server DataBase
  - ***Method performance:*** save user sound record for future use.
- ***return_sound_names***
  - ***Method input:***
    username - user username
    DB - The class that manage the server DataBase
  - ***Method performance:*** Return user's sound names.
- ***handle_request***
  - ***Method input:***
    request_code - Hold the request code and all the textual information.
    data - Sound file in byte form.
    DB - The class that manage the server DataBaselogin_timeout_task - kick the client if he doesn’t connect to his user in 5 minutes.
  - ***Method performance:*** Handle client request.

- ***on_new_client***
  - ***Method input:***
    client_socket - client socket
    addr - client address
  - ***Method performance:*** Handles communication with a single client.
- ***login_timeout***
  - ***Method input:***
    client_socket - client socket
  - ***Method performance:*** Disconnects client if he doesn’t login in 5 minutes.
- ***main***
  - ***Method input:***
    None
  - ***Method performance:*** Run the server.
- ***on_exit***
  - ***Method input:***
    None
  - ***Method performance:*** Close all the threads when the server is shuting himself down.

Client Phone:
Imported Modules:

- **android.content.Intent-** Alou transmission between screens.
- **android.os.Bundle-** Used to store and retrieve the state of the activity.
- **android.os.Handler-** Make the connection between the client and the server.
- **android.os.Looper-** Along with the Handler class, it handles communication between a background thread and the main UI thread.
- **android.util.Log-** Used for logging and help debugging.
- **android.widget.Button-** Enables the user to interact with the app through buttons.
- **android.widget.EditText-** Enables the user to interact with the app using text boxes.
- **android.widget.Toast-** Used to interact with the user using text messages.
- **androidx.annotation.NonNull-** Used to mark that some method can't return a null.
- **androidx.annotation.RequiresApi-** Used to require that the API will be at a certain level.
- **androidx.appcompat.app.AppCompatActivity-** It allows activities to work on older Android versions.
- **androidx.core.app.ActivityCompat-** Used to handle runtime permissions for recording audio and reading media files.
- **android.Manifest-** Used to handle runtime permissions for recording audio and reading media files.
- **android.content.pm.PackageManager-** Used to check and set runtime permissions for recording audio and reading media files.
- **import android.media.MediaPlayer-** Used to play the sound recording.
- **android.media.MediaRecorder-** Used to record the short sound.
- **android.os.Build-** Used to ensure that a certain part of the code only executes on devices running Android Tiramisu (API level 33) or higher.
- **android.widget.ArrayAdapter-** Used to adapt the ArrayList into a spinner.
- **java.util.ArrayList-** Used to create the spinner in the app.
- **android.widget.Spinner-** Enables the user to interact with the app through a spinner.
- **android.widget.ImageButton-** Used to create a button with a specific look.
- **android.app.AlertDialog-** Used to create a pop-up dialog that allows the user to enter a name for a saved recording.
- **java.io.BufferedInputStream-** Used to improve the efficiency of reading the recorded audio data from the storage when sending it to the server.

- **java.io.File-** Used to create and manage the sound file.
- **java.io.FileInputStream-** Used to read a stream of bytes from a recorded audio file.
- **java.io.IOException-** Used to catch errors that are related to file management.
- **java.nio.file.Files-** Used for working with files and directories more conveniently.
- **java.util.Collections-** Used to add all the sound names into the spinner.
- **java.util.List-** Used to store recording names that will be in the spinner
- **java.util.Objects-** Used to compare elements in the spinner list to a value.
- **java.util.Scanner-** Used to make sure the sound name is legal.
- **androidx.core.content.ContextCompat-** Used to check if the user has permission.
- **android.media.AudioRecord-** Used to record oudio.

Developed Modules-

- **RecordingRunnable-**

  Role: Record and send to the server the recording. It was created to be run on a different thread.

  Class variables:

  - SAMPLE_RATE - Defines the number of samples of audio collected per second.
  - CHANNEL_CONFIG - specifies the channel configuration for the audio recording. (stereo/mono)
  - AUDIO_FORMAT - The sound format the sound will be saved (pcm-8)
  - BUFFER_SIZE - Set the size of the temporary storage container for the sound data captured by the microphone before it is sent to the server.
  - TAG - The tag of the log info at this part.
  - recording - whether or not to keep recording.
  - microphone - handler to interact with the device's microphone
  - occurrences - The visual text of the number of occurrences.

  Methods:

  - ***recordAndSend***
    - ***Method input:*** none
    - ***Method performance:*** Record and send the segments to the server.
  - ***stop***
    - ***Method input:*** none
    - ***Method performance:*** Stop the recording.
  - ***prepareRecord***
    - ***Method input:*** none
    - ***Method performance:*** Prepare the microphone for the recording.
  - ***sendToServer***
    - ***Method input:*** none
    - ***Method performance:*** Send the recording segment to the server.

- **SocketHandler-**
  Role: Handel the conection betwin the client and the server.
  Class variables:
  - socket - Server socket.
  - Hreceiver - Holds the handle connection between the client and server.

  Methods:
  - ***getSocket***
    - ***Method input:***
      none
    - ***Method performance:*** Returns the socket.
  - ***setSocket***
    - ***Method input:***
      socket - server socket.
    - ***Method performance:*** Set the server socket.
  - ***getHreceiver***
    - ***Method input:***
      none
    - ***Method performance:*** Returns the handler.
  - ***setHreceiver***
    - ***Method input:***
      Hreceiver - the handle connection between the client and server.
    - ***Method performance:*** Set the connection handler.

- **SendRecv-**
  <u>Role:</u> Record and send to the server the recording. It was created to be run on a different thread.
  <u>Class variables:</u>
  - SendRecv - Holds the handle connection between the client and server.
  - Aes_key - The key for the AES decryption/encryption
  - ivParams - fixed-size input used alongside the encryption key to ensure unique ciphertexts for the same plaintext input
  - encripted - Whether or not the encryption process happened
  - random - Used to create strong encryption.

  <u>Methods:</u>
  - ***setAes_key***
    - ***Method input:***
      Aes_key - The new key for the encryption
    - ***Method performance:*** Save the new encryption key.
  - ***setIv***
    - ***Method input:***
      iv - The new iv for the encryption
    - ***Method performance:*** Save the new encryption iv.
  - ***setAes_key***
    - ***Method input:***
      Aes_key - The new key for the encryption
    - ***Method performance:*** Save the new encryption key.
  - ***send***
    - ***Method input:***
      mHandler - The handler for the communication.
      ip - The ip of the client.
      ba - The information we want to send.
    - ***Method performance:*** Send to the server the information.
  - ***receiveData***
    - ***Method input:***
      none
    - ***Method performance:*** receive data from the server.
  - ***padToMultipleOf16***
    - ***Method input:***
      input- The byte array that needs to be padded
    - ***Method performance:*** Padded byte array into base16 for the encryption process.

- *setEncryption*
  - ***Method input:***
    mHandler - The comulnication handler with the server
  - ***Method performance:*** Set the communication process.
- *send_encrypted*
  - ***Method input:***
    ba - The byte array that is wished to be sent
  - ***Method performance:*** Send to the server the information secretly.

- **TcpSendRecv-**
  <u>**Role:**</u> Send and receive messages from the server.
  <u>**Class variables:**</u>
  - ip - client ip.
  - LEN_SIZE - Used for reading the messages.
  - bdata - The message that will be sent\recived.

  <u>**Methods:**</u>
  - *TcpSendRecv*
    - ***Method input:***
      mHandler - The communication handler.
      ip - client ip.
      bdata - The message that will be sent\recived.
    - ***Method performance:*** Set the basic values for the communication.
  - *run*
    - ***Method input:***
      None
    - ***Method performance:*** Connect with the server.
  - *TcpSendRecv*
    - ***Method input:***
      mHandler - The communication handler.
      ip - client ip.
      bdata - The message that will be sent\recived.
    - ***Method performance:*** Set the basic values for the communication.

- **MainActivity-**
  Role: Handle all the action you can do in the login screen.
  Class variables:
  - user - user username.
  - mHandler - communication handler.
  - ip - device ip.
  - TAG - used for debugging to locate where the message is from.

  Methods:
  - ***getmHandler***
    - ***Method input:*** none
    - ***Method performance:*** Returns the handler.
  - ***getIp***
    - ***Method input:*** none
    - ***Method performance:*** Return the device ip.
  - ***getUsername***
    - ***Method input:*** none
    - ***Method performance:*** Return user username.
  - ***btnLogin.setOnClickListener***
    - ***Method input:*** none
    - ***Method performance:*** Send to the server the user credentials for confirmation.
  - ***btnSignup.setOnClickListener***
    - ***Method input:*** none
    - ***Method performance:*** Move to the sign up screen.

- **CreateAccount-**
  Role: Managing the sign up screen.
  Class variables:
  - TAG - The tag of the log info at this part.
  - MINIMUM_PASSWORD_LENGTH - The minimum length a password is allowed to be.

  Methods:
  - ***btnBack.setOnClickListener***
    - ***Method input:*** none
    - ***Method performance:*** Let the user go back to the login screen.
  - ***btnCreate.setOnClickListener***
    - ***Method input:*** none
    - ***Method performance:*** Send to the server the user credentials for registration.

- **ShortRecord-**
  <u>Role:</u> Managing the sound record screen.
  <u>Class variables:</u>
  - TAG - The tag of the log info at this part.

  <u>Methods:</u>
  - ***btnRecord.setOnClickListener***
    - ***Method input:*** none
    - ***Method performance:*** Record the short sound.
  - ***btnPlay.setOnClickListener***
    - ***Method input:*** none
    - ***Method performance:*** Play the recorded sound.
  - ***btnNext.setOnClickListener***
    - ***Method input:*** none
    - ***Method performance:*** Move to the next screen and send the recorded sound or the sound selection.
  - ***switchOnRecorder***
    - ***Method input:*** none
    - ***Method performance:*** Switch the record button on.
  - ***switchOffRecorder***
    - ***Method input:*** none
    - ***Method performance:*** Switch the record button off.
  - ***switchOnPlayer***
    - ***Method input:*** none
    - ***Method performance:*** Switch the play button on.
  - ***switchOffPlayer***
    - ***Method input:*** none
    - ***Method performance:*** Switch the play button off.
  - ***sendToServerSound***
    - ***Method input:***
      code - Message code. (explained in the communication protocol)
      Current - How much from the recording was already sent.
      fileLength - The total message length.
      contents - The message.
    - ***Method performance:*** Send to the server the sound.
  - ***sendToServerSelection***
    - ***Method input:***
      soundName - The user sound selection.
    - ***Method performance:*** Send to the server the chosen sound out of the saved ones.

- ***getSounds***
  - ***Method input:*** none
  - ***Method performance:*** Get all the names of the user's saved sounds.
- ***saveSound***
  - ***Method input:***
    audioSaveFile - The sound file that is wished to be saved.
  - ***Method performance:*** Send to the server the sound and the name of the sound that the user want to save.
- ***checkPermissions***
  - ***Method input:*** none
  - ***Method performance:*** Check if the user has sufficient permissions.

- **RecordingRunnable-**
  <u>Role:</u> Managing the sound record screen.
  <u>Class variables:</u>
  - TAG - The tag of the log info at this part.

  <u>Methods:</u>
  - ***btnRecord.setOnClickListener***
    - ***Method input:*** none
    - ***Method performance:*** Creat a method that record the user.
  - ***switchOn***
    - ***Method input:*** none
    - ***Method performance:*** Switch the record button on.
  - ***switchOff***
    - ***Method input:*** none
    - ***Method performance:*** Switch the record button off.
  - ***checkPermissions***
    - ***Method input:*** none
    - ***Method performance:*** Check if the user has sufficient permissions.

Client PC:
Imported Modules:
- **os -** Used to manage files and get information such as size.
- **random-** Used to make sure the encryption is strong.
- **socket-** Makes the communication with the server.
- **threading-** Used to record and send while making the application buttons usable.
- **wave-** Used to listen to recorded sound.
- **hashlib.sha256-** Used in the encryption process.
- **math.ceil-** Used in the encryption process.
- **loguru-** Records messages about my program's execution for debugging and analysis.
- **wx-** Used for client GUI
- **Crypto.Cipher.AES-** A method that is used for the AES.
- **pyaudio-** Used for recording.
- **re-** Used to compile regular expressions.
- **base64.b64encode/b64decode-** Used to transfer from readable to binary format.
- **time-** Used to make sure the recording segments are not to long
- **struct.pack-** A module for packing data into binary strings.
- **sys.byteorder-** A module for determining the byte order of the system.
- **array.array-** A module for creating and manipulating array-like objects.

Developed Modules-
- **tcp_by_size-**
  <u>**Role:**</u> Send and receive messages.
  <u>**Class variables:**</u>
  - SIZE_HEADER_FORMAT - The header that contains the size of the message to ensure the full arrival of the packet.
  - size_header_size - the size of the header itself. The header has a size therefore you need to count it in the overall size.

  <u>**Methods:**</u>
  - ***send_encrypted***
    - ***Method input:***
      sock - Client socket.
      bdata - The content that is wished to be sent.
      loop - A variable that is needed for the multithreading
    - ***Method performance:*** Send the content to the client.
  - ***recv_decrypted***
    - ***Method input:***
      sock - Client socket.
      loop - A variable that is needed for the multithreading
    - ***Method performance:*** Receive a message from the client and return it. (bytes)

- **send_receive_encrypted-**
  <u>**Role:**</u> Send and receive encrypted messages.
  <u>**Class variables:**</u>
  - is_encrypted - Make sure the client will not do the encryption process from the start.
  - Iv_parms - fixed-size input used alongside the encryption key to ensure unique ciphertexts for the same plaintext input
  - aes_key - The key for the AES decryption/encryption

  <u>**Methods:**</u>
  - ***set_encryption***
    - ***Method input:***
      Sock - The address of the server
    - ***Method performance:*** Set the encryption between the client and the server.
  - ***send_encrypted***
    - ***Method input:***
      sock - Client socket.
      ba- The content that is wished to be sent.
    - ***Method performance:*** Send the content to the server safely.
- **tcp_by_size-**
  <u>**Role:**</u> Send and receive messages.
  <u>**Class variables:**</u>
  - SIZE_HEADER_FORMAT - The header that contains the size of the message to ensure the full arrival of the packet.
  - size_header_size - the size of the header itself. The header has a size therefore you need to count it in the overall size.

  <u>**Methods:**</u>
  - ***send_with_size***
    - ***Method input:***
      sock - Server socket.
      bdata - The content that is wished to be sent.
    - ***Method performance:*** Send the content to the server.
  - ***recv_by_size***
    - ***Method input:***
      sock - Server socket.
    - ***Method performance:*** Receive a message from the server and return it. (bytes)

- **sound_manager-**
  <u>**Role:**</u> Record and edit sounds.
  <u>**Class variables:**</u>
  - FORMAT - The sample formats.
  - RATE - The sample rate.
  - THRESHOLD - What is the maximum sound decibels I consider silence. Used to mute background noise.
  - CHUNK_SIZE - The amount of samples in every record buffer.
  - rec - Used to stop the long record.

  <u>**Methods:**</u>
  - ***stop_record***
    - ***Method input:***
      None
    - ***Method performance:*** Set the rec variable to False.
  - ***record***
    - ***Method input:***
      None
    - ***Method performance:*** Used to record the short sound.
  - ***is_silent***
    - ***Method input:***
      snd_data - The data that is wished to be checked.
    - ***Method performance:*** Returns whether or not the sound is below the 'silent' threshold
  - ***normalize***
    - ***Method input:***
      snd_data - The data that is wished to be normalized.
    - ***Method performance:*** Average the volume out.
  - ***trim***
    - ***Method input:***
      snd_data - The data that is wished to be trimmed.
    - ***Method performance:*** Trim the quit spots at the start and end.
  - ***add_silence***
    - ***Method input:***
      snd_data - The recording.
      seconds - how many seconds to add at the ends.
    - ***Method performance:*** Add silence to the start and end of 'snd_data' of length 'seconds' (float).
  - ***record_sound***
    - ***Method input:***
      path - Path to the file that the sound will be saved at.
    - ***Method performance:*** Records from the microphone and outputs the resulting data to 'path'.

- ***record_with_val_stop***
  - ***Method input:***
    None
  - ***Method performance:*** Record audio from the microphone and stop when a specific time has passed or a value has changed.
- ***record_to_file***
  - ***Method input:***
    path - Path to the file that the recording will be saved at.
  - ***Method performance:*** Records from the microphone and outputs the resulting data to 'path'.

- **Main-**
  <u>**Role:**</u> Manage the PC client.
  <u>**Class variables:**</u>
  - HOST - Server ip.
  - PORT - Server conection port.
  - server_socket - The server socket conection.
  - username - user username.
  - target_sound_file - file path to the short sound file.
  - recording_file - file path to the recording file.
  - rec - variable that set to stop the recording.
  - occurrences - The number of the total occurrences.
  - MINIMUM_PASSWORD_LENGTH - The minimum password length I allow the user to set.

  <u>**Methods:**</u>
  - ***stop_record***
    - ***Method input:***
      None
    - ***Method performance:*** Set rec to false to stop the recording.
  - ***send_sound***
    - ***Method input:***
      file_path - The file path to the path that the user want to send.
      code - The message code.
    - ***Method performance:*** Send the file the user chose to the server.

- ***save_sound***
  - ***Method input:***
    file_path - The file path to the path that the user want to send.
    sound_name - The sound name the user chose to save a sound as.
  - ***Method performance:*** Send a sound and his chosen name to the server to be saved.
- ***send_recording***
  - ***Method input:***
    file_path - The file path to the path that the user want to send.
    code - The message code.
    text - the text displayed to showcase the number of occurrences.
  - ***Method performance:*** Run unsync and record and send the recording.
- ***send_selection***
  - ***Method input:***
    sound_name - The sound the user chose
  - ***Method performance:*** Send to the server wich sound he chose out of the ones he saved.
- ***signUp***
  - ***Method input:***
    username - User username
    password - user password
  - ***Method performance:*** Send to the server the user credentials.
- ***convert_with_length_prefix***
  - ***Method input:***
    text - The messag the user want to save
  - ***Method performance:*** Add a byte to the start of the text to indicate his length.
- ***main***
  - ***Method input:***
    None
  - ***Method performance:*** Start the client.

- **LoginDialog-**
  <u>**Role:**</u> Manage the login screen.
  <u>**Class variables:**</u>
  - None

  <u>**Methods:**</u>
  - ***on_login***
    - ***Method input:***
      Nome
    - ***Method performance:*** Send the user credentials for verification.
  - ***on_signup***
    - ***Method input:***
      Nome
    - ***Method performance:*** Move to the sign up screen.

- **SignupDialog-**
  <u>**Role:**</u> Manage the sign up screen.
  <u>**Class variables:**</u>
  - None

  <u>**Methods:**</u>
  - ***on_signup***
    - ***Method input:***
      None
    - ***Method performance:*** Send the user credentials for registration.

- **RecordSound-**
  <u>**Role:**</u> Manage the sound recording screen.
  <u>**Class variables:**</u>
  - None

  <u>**Methods:**</u>
  - ***initialize_spinner***
    - ***Method input:***
      None
    - ***Method performance:*** Initialize the spinner with the user's saved sound names.
  - ***get_sounds***
    - ***Method input:***
      None
    - ***Method performance:*** Get the user's sound names.

- ***on_save***
  - ***Method input:***
    None
  - ***Method performance:*** Open a screen that will let the user to save his recording for suter use.
- ***on_play***
  - ***Method input:***
    None
  - ***Method performance:*** Play the recording the user recorded.
- ***on_record***
  - ***Method input:***
    None
  - ***Method performance:*** Record the user’s sound.
- ***on_next***
  - ***Method input:***
    None
  - ***Method performance:*** Move to the next screen for the counting.

- **Counter-**
  <u>Role:</u> Send and receive messages.
  <u>Class variables:</u>
  - SIZE_HEADER_FORMAT - The header that contains the size of the message to ensure the full arrival of the packet.
  - size_header_size - the size of the header itself. The header has a size therefore you need to count it in the overall size.

  <u>Methods:</u>
  - ***on_record***
    - ***Method input:***
      None
    - ***Method performance:*** Runs a thread that record and send.
  - ***on_stop***
    - ***Method input:***
      None
    - ***Method performance:*** Stop the recording by seting the ‘rec’ to False.

Part B:

- ## Explanation of the ability:

  Basic Explanation:
  This method detects the number of occurrences of a target sound within a recording based on their similarity. The algorithm calculates the similarity score between audio segments and the target sound to identify matches that exceed a specified threshold.

  Key Concept:
  1. **Loading Audio Files**: The code uses the librosa.load function to load the target sound and the recording from the provided file paths.
  2. **Similarity Calculation:** It computes the similarity score between audio segments using the dot product and normalization of the audio arrays.
  3. **Threshold Comparison:** The algorithm compares the similarity score with a predefined threshold to determine if a match is found.

- **Code:**

```
for sample_path in files:
   if os.path.isfile(sample_path):

       sample_series, sample_sr = librosa.load(sample_path, sr=None)
# load the sound from file

       sample_frames, fft_window, n_columns = stft_raw(sample_series,
sample_sr, win_length, hop_length, hz_count,
                                                        dtype)

       # Pre-allocate the STFT matrix
       sample_data = np.empty((int(1 + n_fft // 2),
sample_frames.shape[1]), dtype=dtype, order='F')

       for bl_s in range(0, sample_data.shape[1], n_columns):  #
process the data
           bl_t = min(bl_s + n_columns, sample_data.shape[1])
           sample_data[:, bl_s:bl_t] = scipy.fft.fft(fft_window *
sample_frames[:, bl_s:bl_t], axis=0)[
                                       :sample_data.shape[0]]

       sample_data = abs(sample_data)

       sample_height = sample_data.shape[0]
       sample_length = sample_data.shape[1]

       x = 0
       sample_start = 0
       while x < sample_length:
           total = 0
           for y in range(0, sample_height):
               total += sample_data[y][x]
           if total >= 1:
               sample_start = x
               break
           x += 1
       sample_start += sample_crop_start  # The first few frames seem
to get modified, perhaps due to compression?
       sample_end = (sample_length - sample_crop_end)

       samples.append([
           sample_start,
           sample_end,
           os.path.basename(sample_path),
           sample_data
       ])

       logger.info('  {} ({}/{})'.format(sample_path, sample_start,
sample_end))

# ---------------------------------------------------
# Processing

logger.info('Processing')

source_frames, fft_window, n_columns = stft_raw(source_series,
sample_rate, win_length, hop_length, hz_count, dtype)

if config['source_frame_end'] == None:
   config['source_frame_end'] = source_frames.shape[1]

logger.info('From {} to {}'.format(config['source_frame_start'],
config['source_frame_end']))
logger.info('From {} to
{}'.format(((float(config['source_frame_start']) * hop_length) /
sample_rate),
                                  ((float(config['source_frame_end'])
* hop_length) / sample_rate)))

matching = {}
match_count = 0
match_last_time = None
match_skipping = 0
matches = []

results_end = {}
results_dupe = {}
for sample_id, sample_info in enumerate(samples):
   results_end[sample_id] = {}
   results_dupe[sample_id] = {}
   for k in range(0, (sample_info[1] + 1)):
       results_end[sample_id][k] = 0
       results_dupe[sample_id][k] = 0

for block_start in range(config['source_frame_start'],
config['source_frame_end'], n_columns):  # Time in 31 blocks

   block_end = min(block_start + n_columns,
config['source_frame_end'])

   set_data = abs((scipy.fft.fft(fft_window * source_frames[:,
block_start:block_end], axis=0)).astype(dtype))

   logger.info('{} to {} - {}'.format(block_start, block_end,
str(datetime.timedelta(
       seconds=((float(block_start) * hop_length) / sample_rate)))))

   x = 0
   x_max = (block_end - block_start)
   while x < x_max:

       if match_skipping > 0:
           if x == 0:
               logger.info('Skipping {}'.format(match_skipping))
           match_skipping -= 1
           x += 1
           continue

       matching_complete = []
       for matching_id in list(matching):  # Continue to check matches
(i.e. have already started)

           sample_id = matching[matching_id][0]
           sample_x = (matching[matching_id][1] + 1)

           if sample_id in matching_complete:
               continue

           hz_score = abs(set_data[0:hz_count, x] -
samples[sample_id][3][0:hz_count, sample_x])
           hz_score = sum(hz_score) / float(len(hz_score))  #
calculate similarity

           if hz_score < config['matching_min_score']:  # Is it above
or below the minimum to be count

               if sample_x >= samples[sample_id][1]:

                   match_start_time = ((float(x + block_start -
samples[sample_id][1]) * hop_length) / sample_rate)

                   logger.info(
                       'Match {}/{}: Complete at {} @
{}'.format(matching_id, sample_id, sample_x,

match_start_time))

                   results_end[sample_id][sample_x] += 1

                   if (config['matching_skip']) or (match_last_time ==
None) or (
                           (match_start_time - match_last_time) >
config['matching_ignore']):
                       match_last_ignored = False
                   else:
                       match_last_ignored = True

                   matches.append([sample_id, match_start_time,
match_last_ignored]) #Add the match to the list
                   match_last_time = match_start_time

                   if config['matching_skip']:
                       match_skipping = ((config['matching_skip'] *
sample_rate) / hop_length)
                       logger.info('Skipping
{}'.format(match_skipping))
                       matching = {}
```

```
                        break  # No more 'matching' entires
                    else:
                        del matching[matching_id]
                        matching_complete.append(sample_id)

                else:

                    logger.info(
                        'Match {}/{}: Update to {} ({} <
{})'.format(matching_id, sample_id, sample_x, hz_score,

config['matching_min_score']))
                    matching[matching_id][1] = sample_x

            elif matching[matching_id][2] < sample_warn_allowance and
sample_x > 10:

                logger.info('Match {}/{}: Warned at {} of {} ({} >
{})'.format(matching_id, sample_id, sample_x,

samples[sample_id][1], hz_score,

config['matching_min_score']))
                matching[matching_id][2] += 1

            else:

                logger.info('Match {}/{}: Failed at {} of {} ({} >
{})'.format(matching_id, sample_id, sample_x,

samples[sample_id][1], hz_score,

config['matching_min_score']))
                results_end[sample_id][sample_x] += 1
                del matching[matching_id]
```

Part C:

- ## Explanation of the tests:
  - ### Sound Recording Test:

    Purpose of the Test: To assess the application's ability to accurately record sound segments of varying lengths under different environmental conditions.
    What Was Done: I recorded myself snapping my fingers in different environments.
    Results of the Test: Recording in a noisy environment decreases the dominance of the sound.

  - ### Sound Occurrence Counting Test:

    Purpose of the Test: To check if the system accurately counts the occurrences of a reference sound within longer segments.
    What Was Done: Recorded various sounds in various recordings.
    Results of the Test: The application indeed manages to recognize the sound in the recording most of the time in the PC. The client in the phone was not abel to count acouretly the acourences.
    Problems Discovered and Solutions: The method the counts didn’t work properly for a very long time. I used many resources to try to improve it till I got to the method I have now. Additionally, I found out that the sound and the recording need to be recorded and saved with the same values like the sample rate.

    I also discovered that the phone's recording volume was too low for accurate counting. To address this, I used a library to normalize the captured sounds. While this improved the clarity of the longer recording part, it unfortunately distorted the short sound, making it unrecognizable in the longer recording. Consequently, I decided not to include the counting feature in the app for the final test.

  - ### Similarity Threshold Test:

    Purpose of the Test: To check which similarity threshold works the best with the app.
    What Was Done: run the counting method with different similarity thresholds until I found the most feating.
    Results of the Test: 15% is the golden point. Any more and it will count sounds that are not similar and any less will not count sounds that I think are similar.

- ## User Interface Test:

  Purpose of the Test: To evaluate the user-friendliness, intuitiveness, and effectiveness of the app's interface.
  What Was Done: I gave both the app and the PC application to people I know struggle with electronic usage (my grandparents and my little sister).
  Results of the Test: The two applications are user-friendly. I additionally learned that putting as less buttons as possible helps.
  Problems Discovered and Solutions: users consistently went back and forth in the app. To fix it I removed all the back buttons so they could only move forward. If a user wishes to go back they can use their phone's back button.

- ## Data Transmission Test:

  Purpose of the Test: To validate the secure transmission of messages.
  What Was Done: I used scapy to sniff all the packets that are going through my computer (server & client) and tried to get the message that was sent from the user to the server.
  Results of the Test: I couldn't get the original message ("Hello world), which means that the communication is encrypted well.

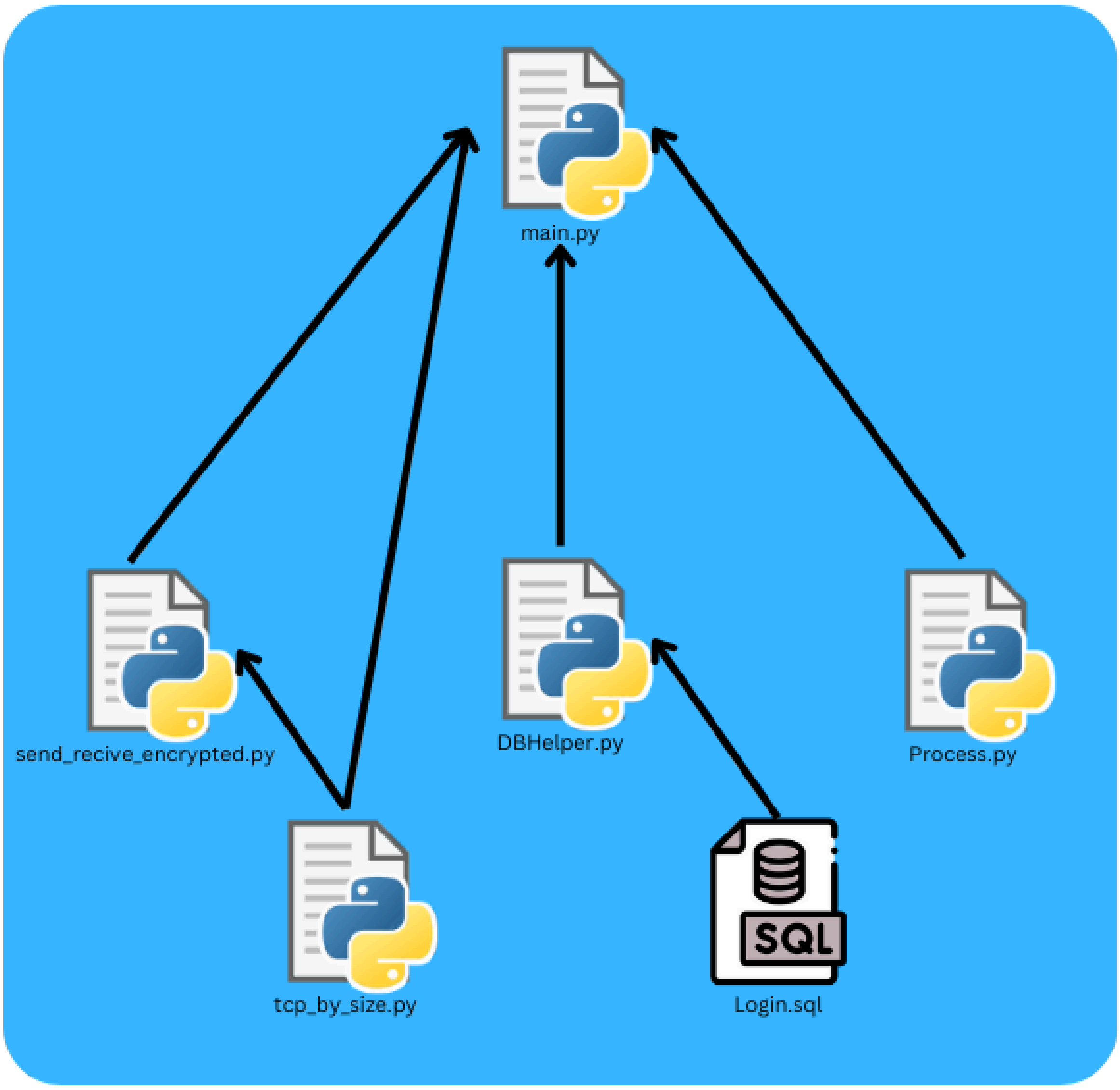
- ## Error Handling Test:

  Purpose of the Test: To assess how the system responds to and manages error scenarios, including incorrect file formats and non-functional internet connections.
  What Was Done: I created a Python client and tried to send the server messages he couldn't handle.
  Results of the Test: Most of the messages indeed returned an error but the server was still on and managed to talk with the proper client perfectly fine.
  Problems Discovered and Solutions: Some errors are in a while loop so I added a break point.

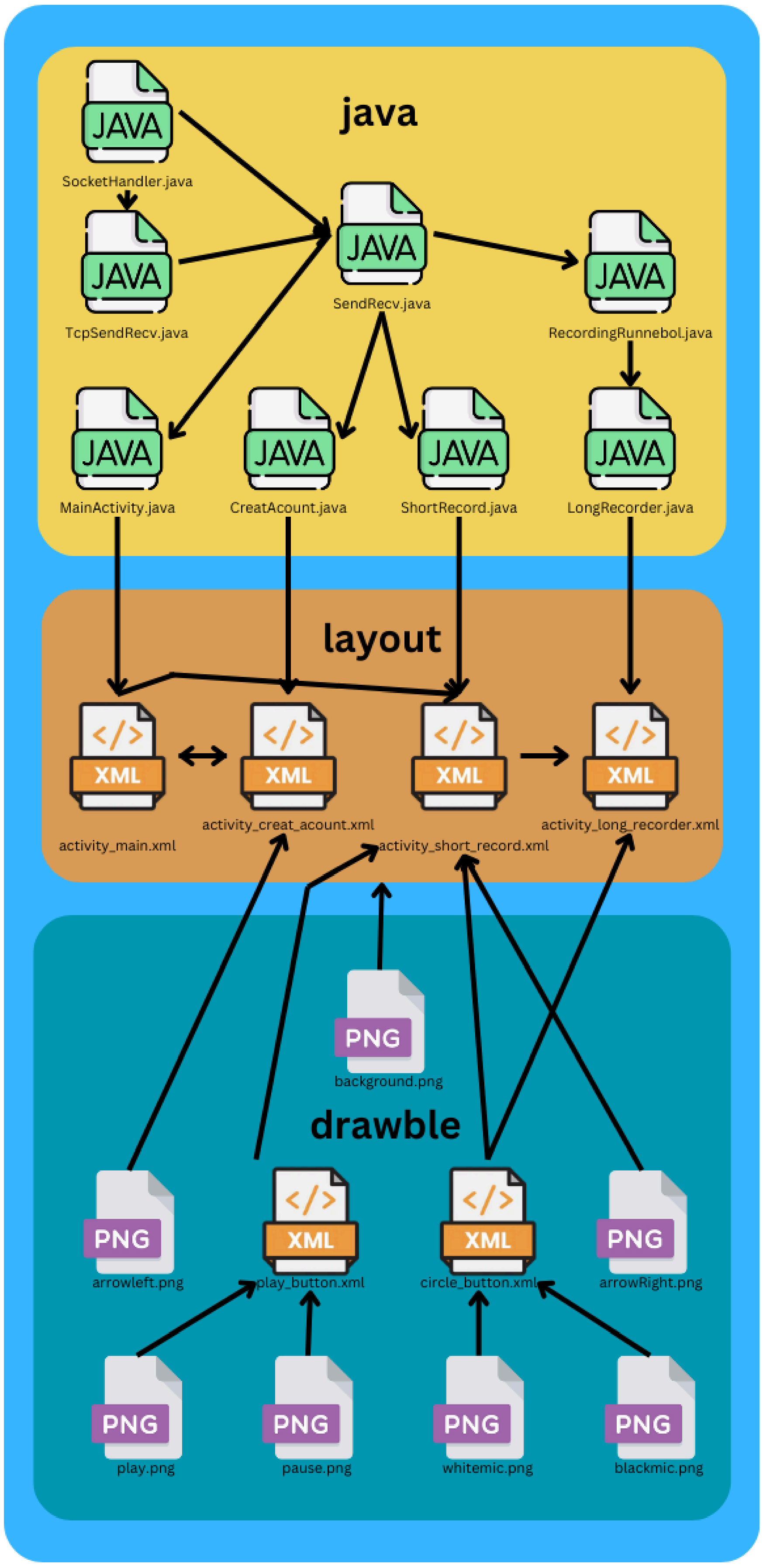
User Manual

- **File Tree:**

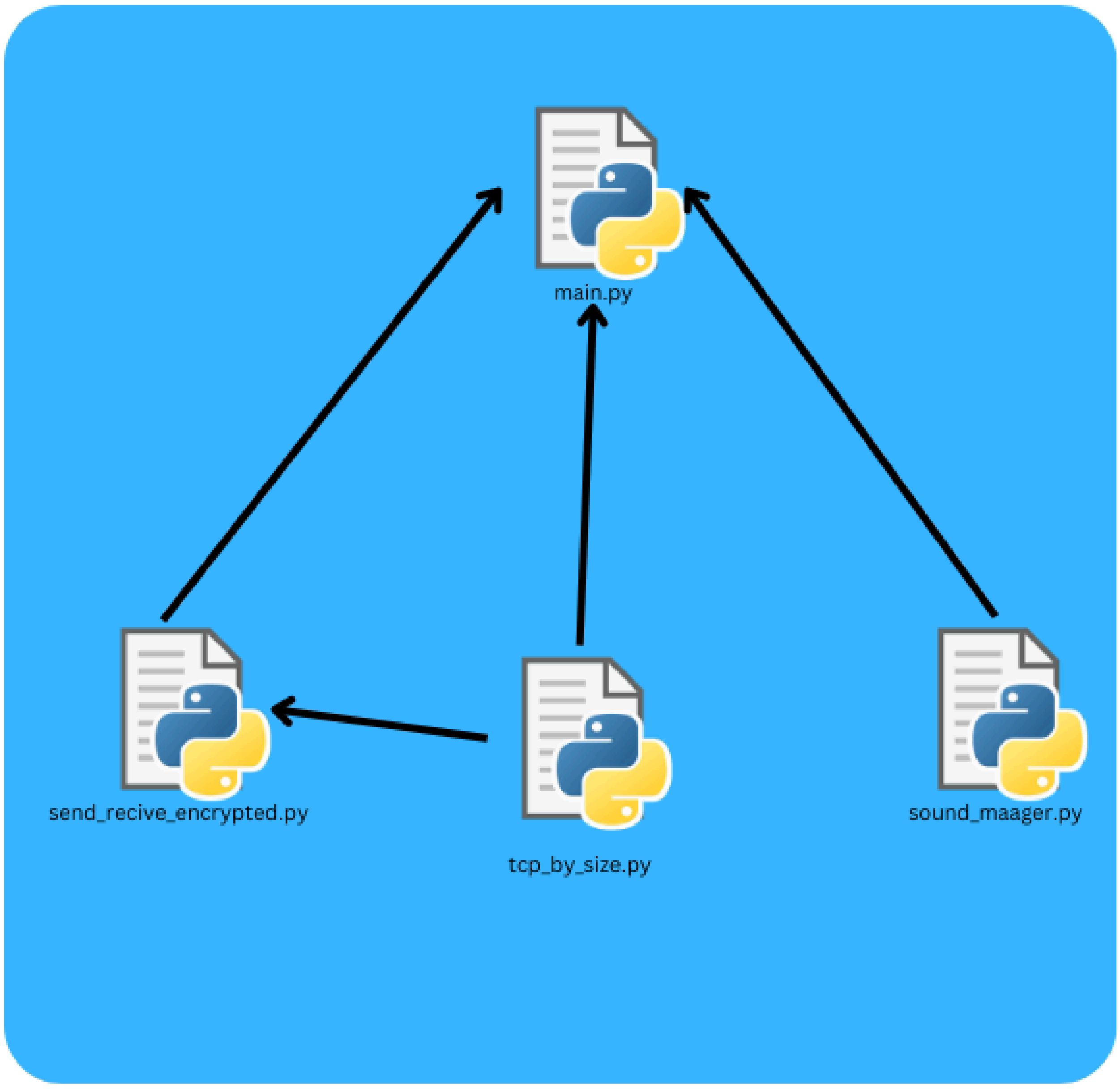
Server:



Client Phone:



Client PC:


main.py
send_recive_encrypted.py
tcp_by_size.py
sound_maager.py

• System download:

The required environment-

Python, SQLite, Android Studio

The tools required-

A computer to run the server. If you chose to use the android app you will need an Android device, or a device that can run an Android emulator, a microphone. If you chose to use the PC application you will need a computer to run the client.

File locations-

As you can see in the file tree

Network-

The entire system must be run on the same private network (NAT)

Minimum architecture required-

Run on Windows 10 or newer, it is advisable to use a server with strong processing power, to perform the sending of messages and the calculation in an optimal and fast way.

• System Users:

The system administrator - to manage the system you must run the server in Python.

End user - Using the client’s Android app is very similar to using its computer counterpart. All of the differences are caused by system GUI design.

Basic explanation of the end user- to use the system you need to download it on your device whether it’s a phone or a computer. After that you will need to create a user using the registration button or log in as an existing user. On the first screen in the phone, you must record a sound by pressing the big button in the middle onc. If you use the computer you will need to press the “record” button. To stop the recording the user must press again on the same button (On the computer it nows to stop alone). To move to the next screen the user needs to press the arrow button on the right bottom corner in his phone or the “next” button on the PC. On the second screen, the user starts the search by pressing the button on the screen. The user can stop counting all the user needs to do is press the same button again on the phone or the “stop” button on PC.

Sign-up screen-
The user needs to input their preferred username and a password that is at least 6 characters long. They will then press the registration button. They will then get a message of whether they registered successfully or not. Additionally, the user may get a message that says that the username is in use; if so, they will have to choose a different name.

Log-in screen-
The users need to input their username and password. After that, they will need to press the “Login” button. The users will move to the next screen or they will be told that the username and password do not match. If the credentials are incorrect, the user will have to re-enter their username and password or create a new user by pressing the “sign up” button.

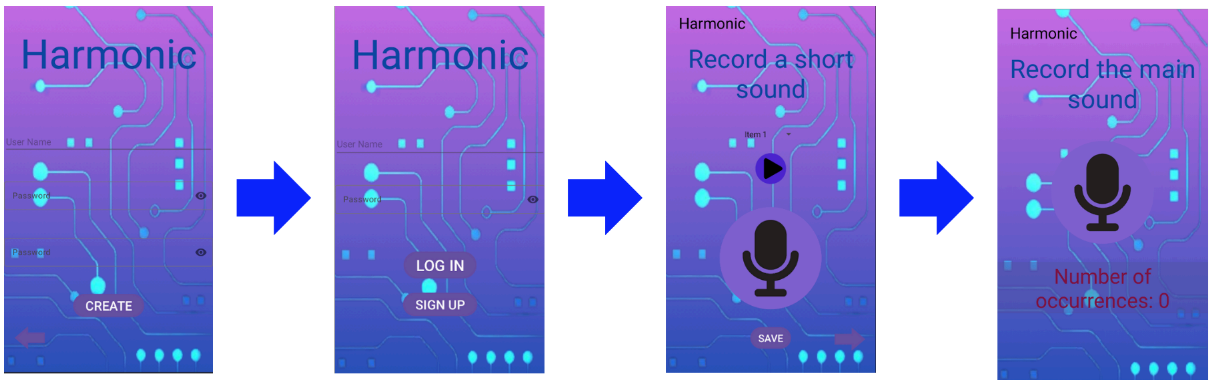
Short sound recording screen-
The users here will have to record the sound they wish to count. To start recording, they will need to press the big button in the middle if they use the Android app. If the Users use the PC application they will need to press the “record” button. When they think they've captured the sound, they need to press the same button again if they use the Android app, The PC application stop alone. The user may press the play button to hear their recording. Additionally, they can press the save button that is on the bottom to save the current sound. If the save button is pressed, the users will be faced with a pop-up screen that will ask them to enter the sound name. The sound name needs to be new and not one that is in use by them. In case the name is in use, the users will get a message about it and will have to enter a new name. If the users wish to use a sound that they previously saved, they can press the spinner to choose their sound. After the users record or choose a sound from the spinner, they will be able to move to the next screen by pressing the arrow \ “next” button that is located at the bottom.

If the user recorded a sound and then chose a sound in the spinner, they can re-choose the recording by setting the spinner to "default".

Long record screen-
In this screen, the users will have to press the button to start the recording. If the users wish to stop recording, they just need to press that button again in the android app or the “stop” button in PC application. While the app records the user will see on the bottom the overall number of occurrences of the sound in the recording.

(press means tap and not press and hold)

All the screens:
Android app



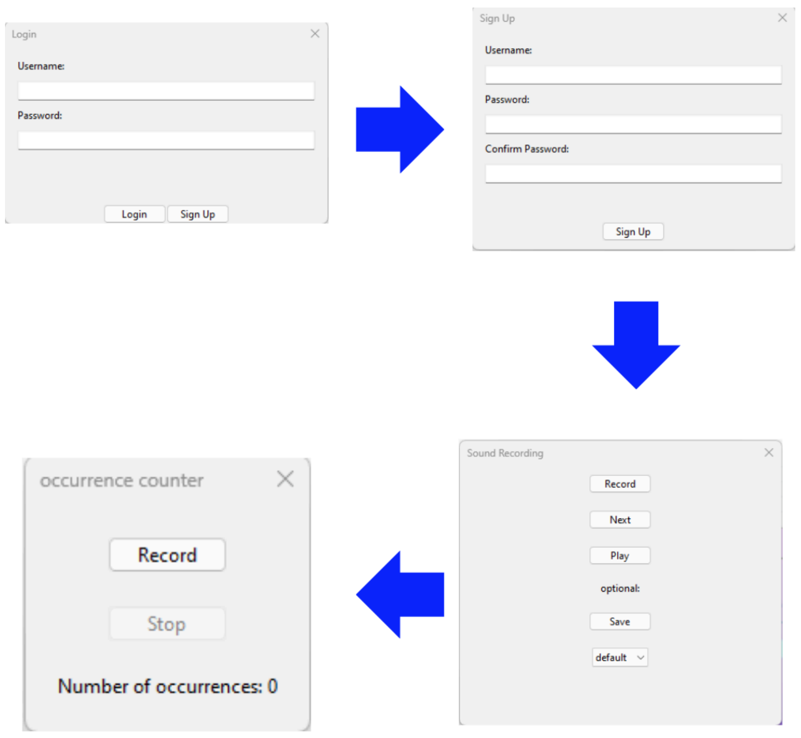
Sign-up screen - In this screen, there is one normal text box that will contain the username and two text boxes that hide the content one for password and the other for password repit. The users can see the content in the last two text boxes by pressing the eye icon. There are also two buttons, one that tries to create the user and one that moves the users back to the login screen.

Log-in screen - In this screen, There are two text boxes. One text box is a normal one that will contain the username and one discreet text box that will contain the password. There are also 2 buttons, One button sends the user input for verification, and the other one moves to the Sign-up screen.

Short sound recording screen- This screen has a big button in the middle to start/stop the recording. Above the big button, there is also a small button that plays the recording. Above that button, there is also a spinner that holds all the sound the user saved. On the bottom, there is a button that is used to save the sounds.

Long record screen - In the middle of the screen there is a big button that start/stop the recording. On the bottom, there is text that showcases the total number of occurrences.

PC application



Sign-up screen - In this screen, there is one normal text box that will contain the username and two text boxes that hide the content one for password and the other for password repit. There is also a button that tries to create the user. The user can go buck to the login screen by pressing the X button of the window.

Log-in screen - In this screen, There are two text boxes. One text box is a normal one that will contain the username and one discreet text box that will contain the password. There are also 2 buttons, One button sends the user input for verification, and the other one moves to the Sign-up screen.

Short sound recording screen- This screencontain many buttons so we will go over them from top to bottom. The “Record” button start the recording proces. There is no “stop” button because it nows to stop alone. Underneath there is the “Next” button that let the user to move to the next screen. The next button is the “Play” button that let the user hear what he recorded. Under that there is a text that says “optional:” because all the elements Underneath are extra. There are the “Save” button that let the user to save their sounds. At the button youcan find the spinner that let the user to chose one of his saved sounds.

Long record screen - In the middle of the screen there is a big button that start/stop the recording. On the bottom, there is text that showcases the total number of occurrences.

Reflection

My project journey was different from most. I had to start my project with coding and not with preparation using this file like the others. I had to do so to prove that my concept was possible. To prove so I had to write the code that counts the occurrences. This start was super difficult for me for several reasons. First, I didn’t get extra time on the first submission. That meant that I had to work extra every day. The second difficulty is that creating the code was extremely difficult. To create the code I had to learn a lot from code examples, articles, and YouTube videos. I had a lot of prototypes of the code before I was happy with what I made. Thirdly above all that, I had to prepare for tests for the army.

After the difficult start I got into a pace, each day I made sure to work on the project even if it meant just reading what I wrote. Although I did that I still had sometimes to work until the late hours of the night to finish things before the deadline. Although I did my best to make sure it would not happen, sometimes it is inevitable for example in a month with a lot of tests.

While working on the project I had a lot of failures. I didn’t manage to communicate between the client and server, I didn’t manage to record the sound and so much more. Luckily I had solutions to my problems. For most of my problems, I managed to overcome them through the Google searching skill that I picked up during the project. When I didn’t manage to overcome the obstacles I had friends and family that helped me. I and my threads worked together countless times to help each other in the creation of the project. My family didn’t help less. My father is a senior developer so he was happy to help me. As I mentioned before both my mother and grandmother have a master's degree in math, and both of them helped me understand some of the math concepts.

From this project, I learned that if I have spare time and I know that there is something that I need to do, I need to use that time to finish that something earlier. If I had to start my project again I would have done exactly this. Besides time management, the only thing that I would change in my work is that I should have made the counting method even better before I started the other things. If I would continue working on this project I will try to use as less libraries as I can to make my project more custom.

Of course, I would be honored to thank my friends and family who helped me along the way.

Bibliography

- Senin, M., & Goldberg, I. (2009). "A Comparative Study of Dynamic Time Warping, Cross-Correlation, and Waveform Similarity for Audio Signal Matching."
- Brigham, E. O., & Morrow, R. E. (1969). "Fast Fourier Transform and Its Applications."
- Bishop, C. M. (2006). "Pattern Recognition and Machine Learning."
- Smith, J. (2007). "Fourier Transforms in Audio Analysis."
- Salamon-Jespersen, S., & McGowan, R. W. (1994). "Dynamic Time Warping for Speech Recognition."
- Yurika Permanasari (2019). "Speech recognition using Dynamic Time Warping"
- Pavel Senin (2008). "Dynamic time warping algorithm review"
- Eamonn Keogh & Chotirat Ann Ratanamahatana (2004) "Exact indexing of dynamic time warping"
- Kongming Wang & Theo Gasser (1997). "ALIGNMENT OF CURVES BY DYNAMIC TIME WARPING"