Wattson: An Innovative IELTS Speaking Assistant Android Application

Kaijie Lai
2034675
Kaijie.Lai20@student.xjtlu.edu.cn

Zhen Ma
2034590
Zhen.Ma20@student.xjtlu.edu.cn

Jinxing Li
20XXXXX
Jinxing.Li20@student.xjtlu.edu.cn

Yichen Liang
20XXXXX
Yichen.Liang20@student.xjtlu.edu.cn

Keming Shen
20XXXXX
Keming.Shen20@student.xjtlu.edu.cn

I. Introduction

This report presents development and evaluation of ”Wattson”, an innovative IELTS Speaking Assistant Android application designed to aid students in IELTS speaking practice. The report details the app design, including its user interface and functional features, followed by an evaluation of its effectiveness in aiding IELTS speaking preparation.

A. Motivation

For more than three decades, the International English Language Testing System (IELTS) has been a pivotal benchmark in assessing English language proficiency, trusted by over 12,000 institutions in approximately 140 nations. However, a significant challenge has been identified in the context of Chinese students' performance in the IELTS Speaking section. This challenge does not stem from a fundamental deficiency in English proficiency but rather from a critical shortfall in effective and practical oral language training.

As a student at Xi'an Jiaotong-Liverpool University, where many aspire to pursue postgraduate studies abroad, proficiency in the IELTS exam, particularly in the speaking part, is a common requirement. In my personal experience and observations, achieving a high score in the IELTS Speaking section necessitates repetitive practice and training, but students often struggle with rigid, rehearsed responses and lack the adaptability essential for genuine linguistic advancement.

Recognizing the inefficiency of traditional practice methods, which involve a cumbersome process of operating a recorder and managing speaking prompts, we were motivated to develop an innovative IELTS speaking practice application. This application aims to streamline the IELTS speaking practice by integrating a database of questions with an easy-to-use recording feature, allowing users to focus on enhancing their speaking skills without the hassle of juggling multiple devices or applications.

Our project is propelled by a dual commitment: firstly, to tackle specific obstacles encountered by Chinese learners in the Speaking section, and secondly, to make IELTS preparation more accessible and financially viable. Current software solutions in the market, such as voice assistant apps, lack targeted training features and do not address the need for structured and practical oral language training.

This initiative is not merely an endeavor to enhance test scores but a strategic effort to bridge a critical educational gap. By facilitating a more comprehensive and practical approach to mastering the spoken aspect of the English language, we aim to ensure that economic limitations do not obstruct individuals' journey towards English language mastery.

B. Contribution

Incorporating the emphasis on adherence to the Model-View-Controller (MVC) architecture into the description of the IELTS Speaking Practice Application, the revised content is as follows:

TABLE I
Comparative Analysis of IELTS Preparation Apps

APP	Question Library	Simulate Exam	Pronunciation Practice	Playback	Random Question	Note & Record Storage
Wattson	✓			✓	✓	✓
China Daily			✓			
Duolingo			✓			
Minor Station IELTS	✓	✓		✓		
IELTS Official Pre App	✓	✓		✓	✓	
Simon IELTS	✓	✓			✓	

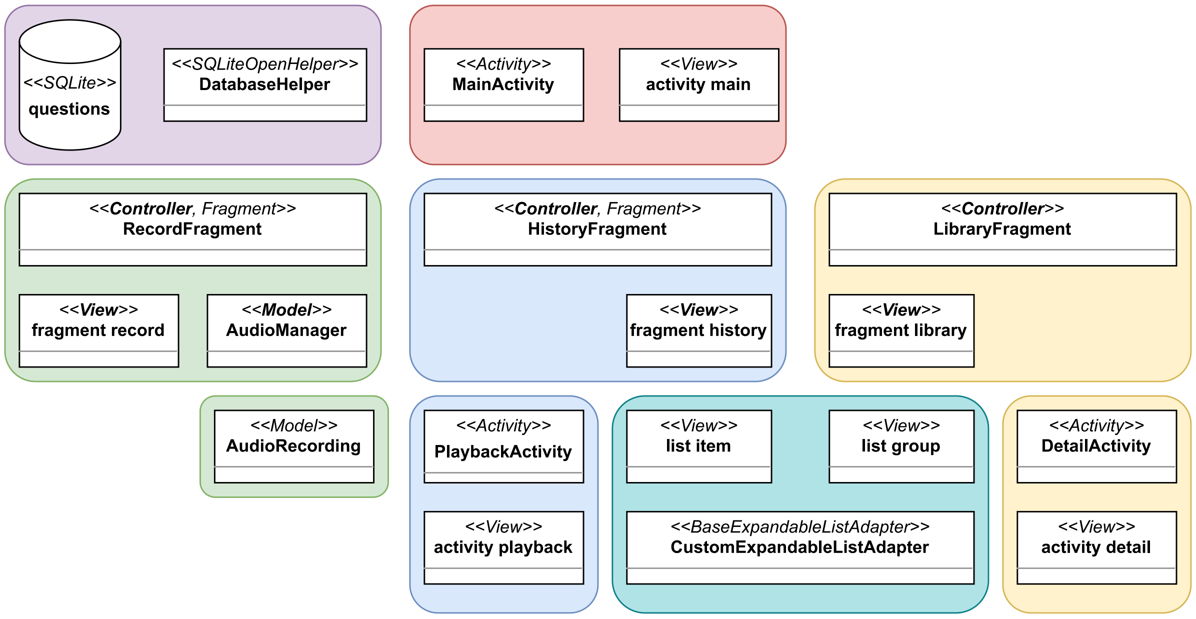


Fig. 1. MVC Architecture of Project

The Wattson offers a tailored platform for IELTS candidates, particularly focused on the speaking section of the test. Our application is designed following the strict principles of the Model-View-Controller (MVC) architecture, as depicted in Figure 1. This architectural approach enhances learning through realistic simulation and immediate feedback via recording features, providing a user-friendly experience that allows for personalized practice sessions. Its convenience, clarity, and structured approach to organizing study materials, aligned with the MVC model, significantly contribute to a user's ability to improve their speaking skills efficiently, making it a valuable tool in the IELTS preparation process.

A comparative analysis of existing market offerings was conducted to identify gaps in current applications and guide the development of our app. This analysis, summarized in a comparative table, revealed that popular applications like 'China Daily' and 'Duolingo' lack specialized features for IELTS Speaking Test preparation, such as a dedicated IELTS test bank and exam simulation. These applications predominantly focus on general speaking practice, omitting IELTS-specific content. Moreover, while other applications possess a broader range of functionalities, they exhibit notable usability issues.

Key issues identified in existing applications include disorganized question bank functionalities, which impede users' ability to efficiently categorize questions by parts of the speaking test. This disorganization complicates the process of locating specific questions for targeted practice. Additionally, the recording function in these apps does not permit users to save their recordings. Consequently, if a user exits the exam page, they lose access to previous recordings, hindering their ability to review and refine their speaking skills.

Our application, adhering to the MVC architecture, addresses these shortcomings by offering streamlined access to the question bank, enabling users to filter questions by test part swiftly. This feature significantly enhances both the efficiency and the user experience. Furthermore, our app's capability to save, edit, and retrieve historical recording records fills a critical gap in the market, addressing the limitations of existing applications that do not support recording preservation.

In summary, our IELTS Speaking Practice Application is distinctively positioned to cater to the specific needs of IELTS test takers. By offering tailored functionalities for IELTS Speaking Test preparation, within the MVC framework, our app provides a convenient and effective platform for users to practice and enhance their skills anytime and anywhere. These user-centric features, coupled with our adherence to the MVC architecture, are pivotal in attracting and retaining our user base.

II. App Design

In this section, we delve into the intricacies of two primary features of our application: the Record feature and the Bottom Navigation. These features are dissected from multiple perspectives to provide a comprehensive understanding of their design and functionality.

A. Record Feature

The Record feature is a crucial component of the app, and its design is considered from the following aspects:

- **User Interface (UI)**: The primary components of the View for the Record feature are defined in the `res/layout/fragment_record.xml`. The corresponding class diagram illustrates the object structure in Figure 5. Initially, the UI layout is as shown in the schematic Figure 2. Interaction

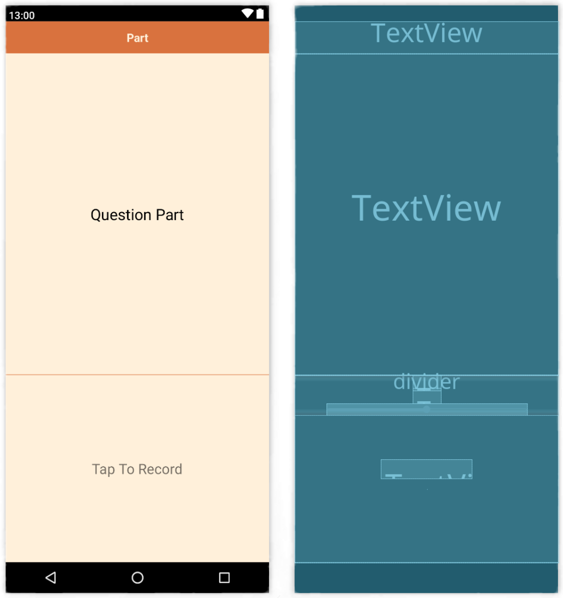


Fig. 2. Layout View of Record Fragment

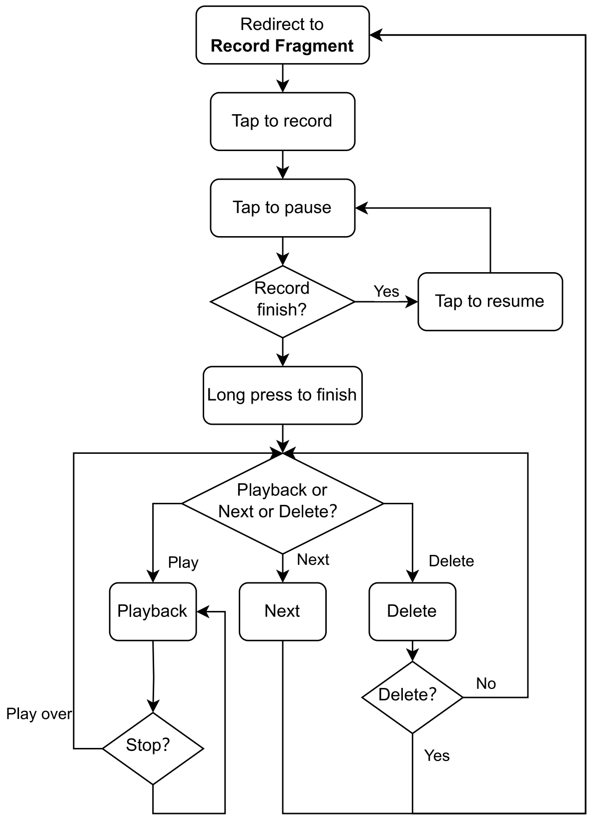


Fig. 3. Flowchart of Record Instruction

and visibility of different UI elements, such as buttons, are controlled in `RecordFragment` by modifying their visibility attributes (visible, invisible, or gone). The View utilizes components like `TextView`, `View`, `SeekBar`, and `ImageButton`, while the ViewGroup includes `FrameLayout`, `LinearLayout`, and `GridLayout`. The app's theme, set in `values/themes/themes.xml`, employs a `NoActionBar` style for a clean interface without a primary bar. Colors used throughout the application are defined in `values/colors.xml`, and all fundamental strings are set in `values/strings.xml`.

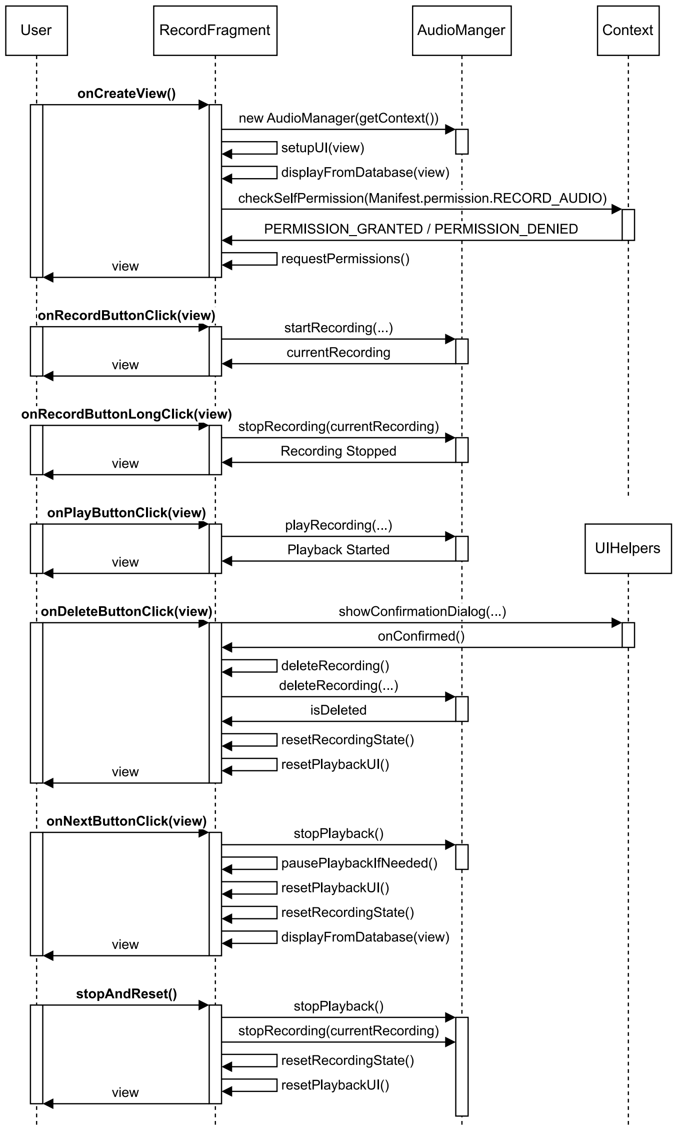
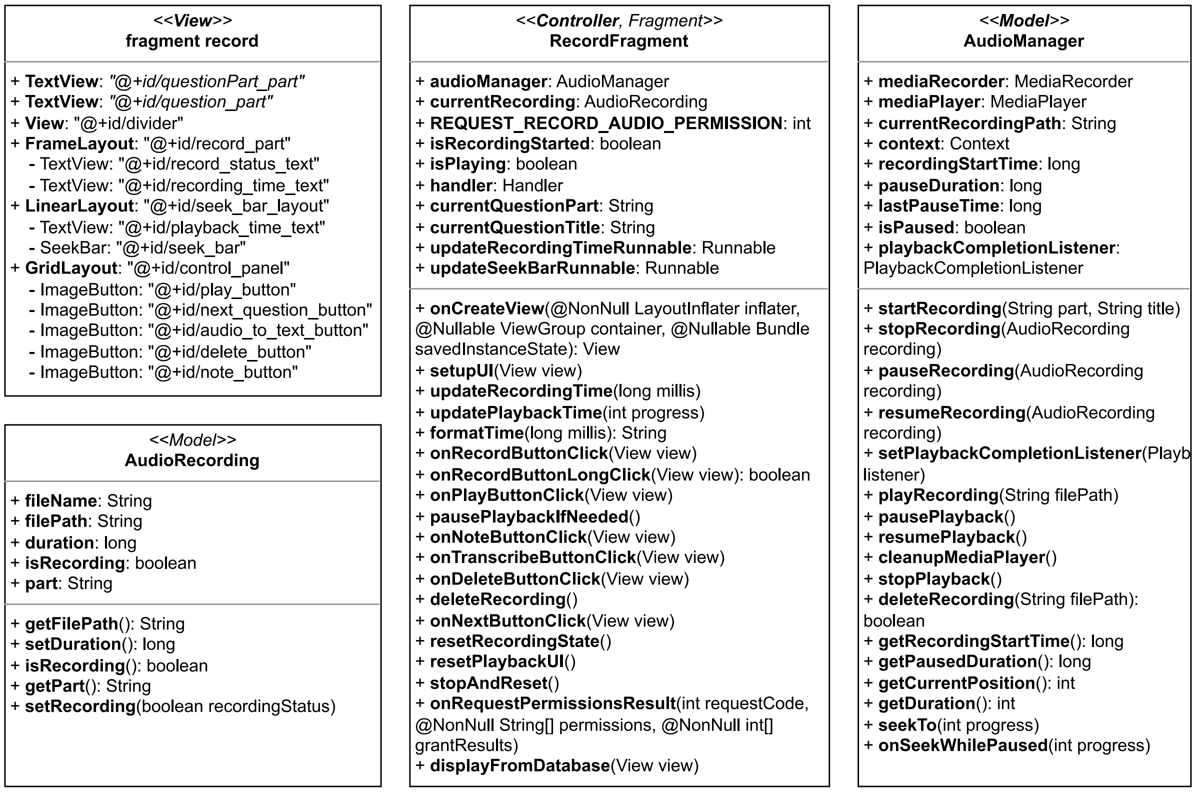


Fig. 4. Sequence Diagram of RecordFragment

- **User Experience (UX)**: Upon opening the app, users are automatically directed to the Record Part (as shown in the attached Figure 5). At the top of the page, users see a question from the database along with its corresponding Part. Following the prompt `Tap to record` begins



<<*View*>>
fragment record

+ **TextView**: *"@+id/questionPart_part"*
+ **TextView**: *"@+id/question_part"*
+ **View**: "@+id/divider"
+ **FrameLayout**: "@+id/record_part"
  - TextView: "@+id/record_status_text"
  - TextView: "@+id/recording_time_text"
+ **LinearLayout**: "@+id/seek_bar_layout"
  - TextView: "@+id/playback_time_text"
  - SeekBar: "@+id/seek_bar"
+ **GridLayout**: "@+id/control_panel"
  - ImageButton: "@+id/play_button"
  - ImageButton: "@+id/next_question_button"
  - ImageButton: "@+id/audio_to_text_button"
  - ImageButton: "@+id/delete_button"
  - ImageButton: "@+id/note_button"

<<*Model*>>
AudioRecording

+ **fileName**: String
+ **filePath**: String
+ **duration**: long
+ **isRecording**: boolean
+ **part**: String

+ **getFilePath**(): String
+ **setDuration**(): long
+ **isRecording**(): boolean
+ **getPart**(): String
+ **setRecording**(boolean recordingStatus)

<<***Controller***, *Fragment*>>
RecordFragment

+ **audioManager**: AudioManager
+ **currentRecording**: AudioRecording
+ **REQUEST_RECORD_AUDIO_PERMISSION**: int
+ **isRecordingStarted**: boolean
+ **isPlaying**: boolean
+ **handler**: Handler
+ **currentQuestionPart**: String
+ **currentQuestionTitle**: String
+ **updateRecordingTimeRunnable**: Runnable
+ **updateSeekBarRunnable**: Runnable

+ **onCreateView**(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState): View
+ **setupUI**(View view)
+ **updateRecordingTime**(long millis)
+ **updatePlaybackTime**(int progress)
+ **formatTime**(long millis): String
+ **onRecordButtonClick**(View view)
+ **onRecordButtonLongClick**(View view): boolean
+ **onPlayButtonClick**(View view)
+ **pausePlaybackIfNeeded**()
+ **onNoteButtonClick**(View view)
+ **onTranscribeButtonClick**(View view)
+ **onDeleteButtonClick**(View view)
+ **deleteRecording**()
+ **onNextButtonClick**(View view)
+ **resetRecordingState**()
+ **resetPlaybackUI**()
+ **stopAndReset**()
+ **onRequestPermissionsResult**(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults)
+ **displayFromDatabase**(View view)

<<*Model*>>
AudioManager

+ **mediaRecorder**: MediaRecorder
+ **mediaPlayer**: MediaPlayer
+ **currentRecordingPath**: String
+ **context**: Context
+ **recordingStartTime**: long
+ **pauseDuration**: long
+ **lastPauseTime**: long
+ **isPaused**: boolean
+ **playbackCompletionListener**: PlaybackCompletionListener

+ **startRecording**(String part, String title)
+ **stopRecording**(AudioRecording recording)
+ **pauseRecording**(AudioRecording recording)
+ **resumeRecording**(AudioRecording recording)
+ **setPlaybackCompletionListener**(Playb listener)
+ **playRecording**(String filePath)
+ **pausePlayback**()
+ **resumePlayback**()
+ **cleanupMediaPlayer**()
+ **stopPlayback**()
+ **deleteRecording**(String filePath): boolean
+ **getRecordingStartTime**(): long
+ **getPausedDuration**(): long
+ **getCurrentPosition**(): int
+ **getDuration**(): int
+ **seekTo**(int progress)
+ **onSeekWhilePaused**(int progress)

Fig. 5. Classes Diagram of Record Part

the recording process. A subsequent tap changes the prompt to `Tap to pause`, allowing users to pause the recording. Options then include `Tap to resume` or `Long press to finish`. After recording, users can see their recording with options to play it back by pressing the play button and navigate through the recording using the progress bar. The `Next` button allows users to proceed to the next question, saving the current recording. If unsatisfied, the `Delete` option is available to remove the recording, accompanied by a system prompt for confirmation and a toast message `Recording deleted successfully` upon deletion. This workflow constitutes the entire user experience of the Record functionality in the flowchart Figure 3.

- **Permissions and Data Management**: The class of `RecordFragment` manages permissions and data intricately. It checks for audio recording permissions using the `ContextCompat.checkSelfPermission` method and requests them if not already granted. This ensures that the app adheres to Android's security norms. Additionally, the class interacts with the `DatabaseHelper` like in Figure **??** to retrieve and display questions from the database. This involves fetching random questions and their corresponding details, demonstrating the app's ability to manage and utilize data effectively. The management of audio files, such as creation, playback, and deletion, is handled through the `AudioManager` class, showcasing a well-structured approach to data handling and storage.
- **Error Handling and User Feedback**: The code demonstrates careful consideration of error handling and user feedback. For instance, if the `Question TextView` or `Part TextView` is null, the app logs an error message, indicating a fail-safe approach to null pointer exceptions. During operations like deleting a recording, the app provides user feedback through `Toast` messages, such as ”Recording deleted successfully” or ”Failed to delete recording”, enhancing the user experience. The use of confirmation dialogs before deleting recordings is another example of thoughtful user interaction design. Moreover, the app logs various actions and errors, such as in `displayFromDatabase` and `onDeleteButtonClick` methods, which aids in troubleshooting and ensures a smooth user experience.
- **Functionality Implementation**: The implementation of the Record Fragment's functionality is illustrated in the sequence diagram in Figure 4 shown below. This diagram clearly depicts the flow of operations within the Record Fragment, providing a visual representation of the interactions and processes involved. The sequence diagram aids in understanding the sequential steps and the logical flow of the recording feature, from initiating a recording to handling playback and user interactions.

Event	**Stage**	***Duration (ms)***	***CPU Usage (%)***	***Memory Usage (MB)***
Touch Event Press	Start Record	*51.48*	*1*	*49.2*
Touch Event Press	Record Pause	*71.09*	*2*	*49.3*
Touch Event Press	Resume Record	*91.21*	*<1*	*49.2*
Touch Event Press	Record Finish	*882.02*	*1*	*49.1*
Touch Event Press	Playback	*143.09*	*<1*	*49.2*
Touch Event Press	Stop Playback	*147.17*	*<1*	*49.3*
Touch Event Press	Next	*142.03*	*2*	*49.7*
Touch Event Press	Delete	*103.32*	*7*	*50.1*

TABLE II
PERFORMANCE METRICS FOR RECORD PART PROGRESS

- **Performance Considerations**: The performance of the application during various stages of interaction in the Record Part is a critical aspect of the user experience. The following Table II presents a detailed analysis of the performance metrics associated with different events. These metrics include the duration of each event in milliseconds, CPU usage percentage, and memory usage in megabytes. The data provides valuable insights into the application's responsiveness and efficiency in resource utilization during key interactions such as starting and pausing recording, resuming recording, finishing recording, playback, and other functionalities. This analysis is instrumental in identifying potential areas for optimization to enhance the performance of the app.

B. History and Library Part

The History and Library features are integral components of the app, designed to enhance user interaction and data management. Their design is considered from the following aspects:

- **User Interface (UI)**: The UI in Figure 6 for both the History and Library parts is primarily defined in `res/layout/fragment_history.xml` and `res/layout/fragment_library.xml`, respectively. These fragments utilize an `ExpandableListView` to display the data in a hierarchical manner. The custom adapter `CustomExpandableListAdapter` is employed to manage the data presentation, offering a dynamic and interactive UI experience. The `list_group.xml` is used for displaying parts like Part1, Part2, etc., while `list_item.xml` is responsible for presenting individual items such as titles or specific recordings under each part. This modular approach in the UI design allows for efficient data organization and enhances the user's ability to navigate through the content. The UI layout is further enhanced with XML-defined styles and themes, ensuring a consistent and visually appealing interface throughout the app.
- **User Experience (UX)**: The History and Library features provide a seamless and intuitive user experience. In the History section, users can view and access their past recordings, grouped by parts, and interact with them through playback or deletion options. The Library section, on the other hand, presents a structured view of available titles, allowing users to explore and select items for further interaction. Both sections are designed with ease of navigation and interaction in mind, promoting an engaging and user-friendly experience.

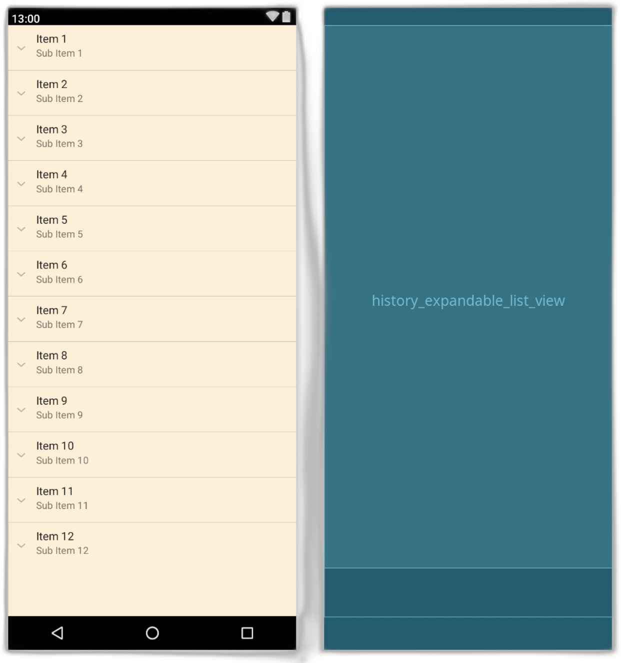


Fig. 6. Layout View of Record Fragment

- **Permissions and Data Management**: These features integrate smoothly with the app's data management system. In the History part, the `HistoryFragment` class manages the retrieval and display of recorded files, while the `LibraryFragment` class in the Library part interacts with the `DatabaseHelper` to fetch and display titles. This demonstrates the app's capability in handling data efficiently, ensuring a robust and responsive performance.

- **Error Handling and User Feedback**: Both History and Library parts include prudent error handling and user feedback mechanisms. For example, if there is an issue in fetching data or accessing files, the app provides informative feedback to the user through `Toast` messages or logs. This approach not only enhances the user experience but also aids in maintaining app stability.
- **Functionality Implementation**: The History and Library features' implementation can be understood through the key functions in `CustomExpandableListAdapter` and the data handling method in `HistoryFragment`.
  – `getGroupView` and `getChildView` methods are crucial for rendering the ExpandableListView. `getGroupView` is responsible for creating and populating the views for each 'group' item in the list, typically representing different parts like Part1, Part2, etc. Similarly, `getChildView` handles the creation and content of 'child' items under each group, which are the individual recordings or titles. These methods ensure that the data is presented in an organized and user-friendly manner.

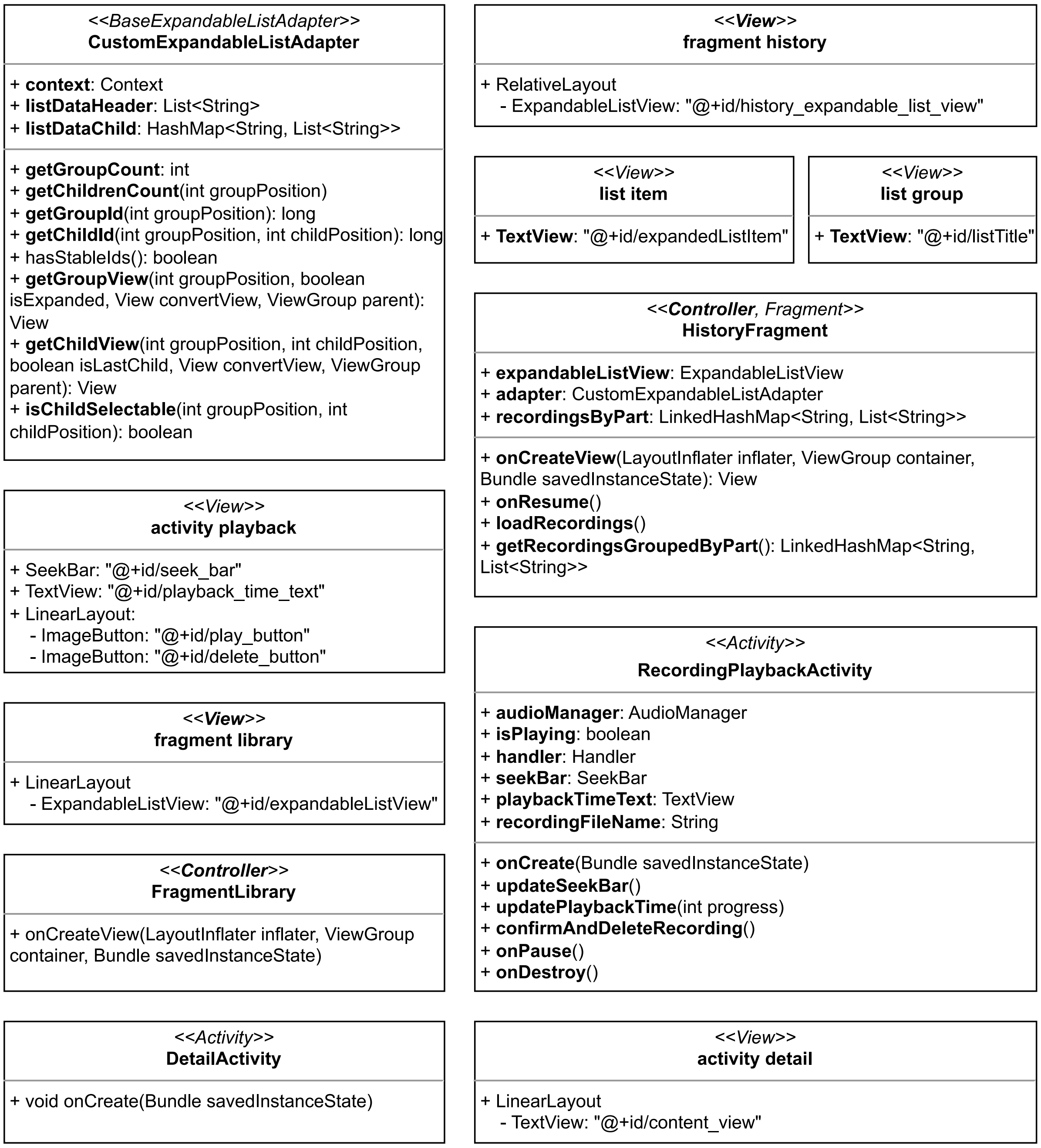

<<BaseExpandableListAdapter>>
CustomExpandableListAdapter
+ context: Context
+ listDataHeader: List<String>
+ listDataChild: HashMap<String, List<String>>
+ getGroupCount: int
+ getChildrenCount(int groupPosition)
+ getGroupId(int groupPosition): long
+ getChildId(int groupPosition, int childPosition): long
+ hasStableIds(): boolean
+ getGroupView(int groupPosition, boolean isExpanded, View convertView, ViewGroup parent): View
+ getChildView(int groupPosition, int childPosition, boolean isLastChild, View convertView, ViewGroup parent): View
+ isChildSelectable(int groupPosition, int childPosition): boolean
<<View>>
fragment history
+ RelativeLayout
- ExpandableListView: "@+id/history_expandable_list_view"
<<View>>
list item
+ TextView: "@+id/expandedListItem"
<<View>>
list group
+ TextView: "@+id/listTitle"
<<Controller, Fragment>>
HistoryFragment
+ expandableListView: ExpandableListView
+ adapter: CustomExpandableListAdapter
+ recordingsByPart: LinkedHashMap<String, List<String>>
+ onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState): View
+ onResume()
+ loadRecordings()
+ getRecordingsGroupedByPart(): LinkedHashMap<String, List<String>>
<<View>>
activity playback
+ SeekBar: "@+id/seek_bar"
+ TextView: "@+id/playback_time_text"
+ LinearLayout:
- ImageButton: "@+id/play_button"
- ImageButton: "@+id/delete_button"
<<View>>
fragment library
+ LinearLayout
- ExpandableListView: "@+id/expandableListView"
<<Controller>>
FragmentLibrary
+ onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
<<Activity>>
RecordingPlaybackActivity
+ audioManager: AudioManager
+ isPlaying: boolean
+ handler: Handler
+ seekBar: SeekBar
+ playbackTimeText: TextView
+ recordingFileName: String
+ onCreate(Bundle savedInstanceState)
+ updateSeekBar()
+ updatePlaybackTime(int progress)
+ confirmAndDeleteRecording()
+ onPause()
+ onDestroy()
<<Activity>>
DetailActivity
+ void onCreate(Bundle savedInstanceState)
<<View>>
activity detail
+ LinearLayout
- TextView: "@+id/content_view"

Fig. 7. Classes Diagram of Library and History Part

- `getRecordingsGroupedByPart` method in `HistoryFragment` plays a vital role in organizing the audio recordings. It groups recordings by their respective parts, facilitating easier access and a more structured view for the user.
- `LinkedHashMap<String, List<String>>` for storing these groupings is strategic. LinkedHashMap maintains the insertion order, which is crucial for presenting the data in a predictable and orderly manner, enhancing the user's navigational experience. This approach allows the app to display recordings or titles under the correct part headings, ensuring that the user interface remains intuitive and efficient. The combination of these methods and data structures forms the backbone of the functionality for the History and Library parts of the app, providing a smooth and coherent user experience.

- **Performance Considerations**: The performance analysis of the History and Library parts is crucial for ensuring a smooth user experience. Table III present the performance metrics for these features, including load times, memory usage, and responsiveness. This analysis is essential for identifying optimization opportunities, thereby enhancing the overall efficiency of the app.

C. Bottom Navigation Part

The Bottom Navigation is an essential feature of the app's user interface, providing users with a convenient way to navigate between the main sections of the app.

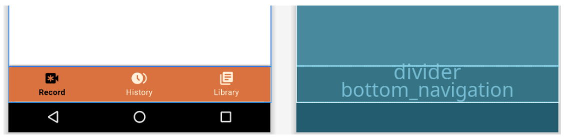


Fig. 8. Layout View of Specific Menu Design

- **User Interface (UI)**: The UI for the Bottom Navigation is defined by two XML files: `activity_main.xml` for the overall layout, including the BottomNavigationView as a container at the bottom of the app, and `menu/bottom_navigation_menu.xml` for the specific items within the navigation menu. In `activity_main.xml`, a FrameLayout identified as `fragment_container` serves as the placeholder for the various fragments switched by the Bottom Navigation. Below this, a BottomNavigationView widget is configured with parameters such as `itemIconTint` and `itemTextColor` to control the appearance of the navigation icons and text. This setup ensures a consistent and coherent visual representation across different fragments and contributes to the overall aesthetic and functional design of the navigation experience.
- **User Experience (UX)**: The Bottom Navigation is designed for ease of use, allowing users to seamlessly transition between the Record, History, and Library fragments by tapping on the corresponding icons in the navigation bar.

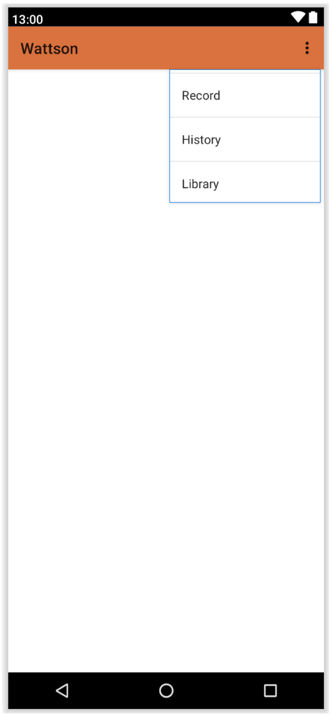


Fig. 9. Layout View of Bottom Navigation

- **Functionality Implementation**: The functionality of the Bottom Navigation is intricately designed within the `MainActivity` class of the app. Two pivotal methods, `navListener` and `switchFragment`, are central to its operation. The `navListener` is an instance of `navListener` and serves as the primary method for monitoring user interactions with the Bottom Navigation items. When an item is selected, this listener invokes the corresponding action, determined by the item's ID. Each ID is associated with one of the app's fragments; for instance, `R.id.navigation_record` corresponds to the `RecordFragment`. The `navListener` ensures that the correct fragment is selected based on the user's choice, facilitating a responsive and interactive navi-

Event	**Stage**	***Duration (ms)***	***CPU Usage (%)***	***Memory Usage (MB)***
Touch Event Press	Open Details	*46.18*	*1*	*28.2*
Touch Event Press	Open Records	*90.17*	*1*	*28.9*

TABLE III
PERFORMANCE METRICS FOR LIBRARY AND HISTORY PART

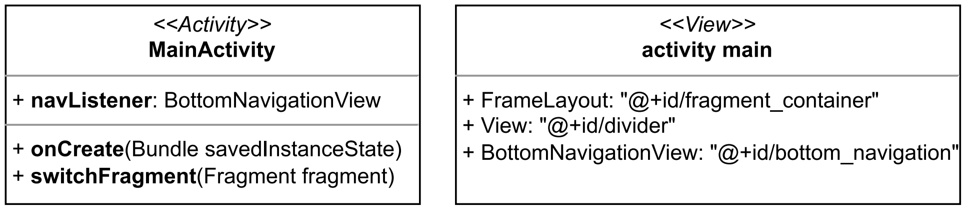

<<Activity>>
MainActivity
+ navListener: BottomNavigationView
+ onCreate(Bundle savedInstanceState)
+ switchFragment(Fragment fragment)
<<View>>
activity main
+ FrameLayout: "@+id/fragment_container"
+ View: "@+id/divider"
+ BottomNavigationView: "@+id/bottom_navigation"

Fig. 10. Classes Diagram of Bottom Navigation Part

gation experience. The `switchFragment` method is responsible for the dynamic and seamless replacement of fragments within the `fragment_container`, which is defined in the `activity_main.xml`. This method performs a fragment transaction, which is a series of steps to add, remove, or replace fragments in the activity's view hierarchy. Before switching, `switchFragment` checks if the current fragment is an instance of `RecordFragment` and, if so, stops any ongoing operations within it to reset the state. This is crucial to prevent any potential interference with the incoming fragment. After this check, the method executes the transaction to display the selected fragment. By managing these transitions, the method ensures that the user interface remains responsive and the transitions appear smooth to the user, thereby enhancing the overall user experience. In summary, these methods together form a robust navigation system that not only responds promptly to user input but also maintains the integrity and continuity of the user interface. This careful implementation underpins the app's navigation efficiency and contributes significantly to its usability.

- **Performance Considerations**: The performance of the Bottom Navigation is a critical aspect of the app's overall user experience. Fast and efficient fragment transitions are imperative to maintain a fluid interaction. The app is designed to minimize the load times and resource usage when switching between different sections, ensuring that the user can navigate the app with minimal latency.

D. Database Management Part

The Database Management component is vital for the app's functionality, handling the storage, retrieval, and processing of user and application data.

- **Permissions and Data Management**: The class of `DatabaseHelper` encapsulates the complexity of database creation, upgrade, and data management within the app. In Android, SQLite databases are inherently private to the application, which means that no special permissions are required for an app to read from or write to the databases it creates. This class leverages this architecture to handle data within the confines of the app's secure storage without necessitating explicit user permissions, thus simplifying the data management process.

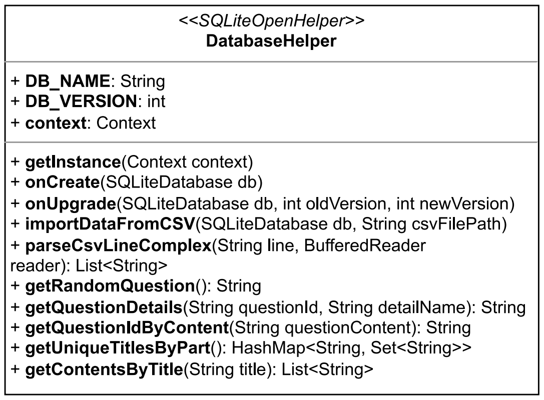

<<SQLiteOpenHelper>>
DatabaseHelper
+ DB_NAME: String
+ DB_VERSION: int
+ context: Context
+ getInstance(Context context)
+ onCreate(SQLiteDatabase db)
+ onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
+ importDataFromCSV(SQLiteDatabase db, String csvFilePath)
+ parseCsvLineComplex(String line, BufferedReader reader): List<String>
+ getRandomQuestion(): String
+ getQuestionDetails(String questionId, String detailName): String
+ getQuestionIdByContent(String questionContent): String
+ getUniqueTitlesByPart(): HashMap<String, Set<String>>
+ getContentsByTitle(String title): List<String>

Fig. 11. Class of DataHelper

When the `DatabaseHelper` is instantiated, it either creates a new database if it does not exist or opens an existing one for reading and writing purposes. The class implements the `SQLiteOpenHelper` interface, which provides a framework for managing database creation and version management. SQLite's role within the Android ecosystem is pivotal. It is a lightweight, file-based database management system that's integrated into Android, making it an ideal choice for mobile applications that require a robust data storage solution without the overhead of a full-fledged database server. SQLite supports standard SQL syntax and is highly optimized for performance on mobile devices, providing mechanisms such as transactions to maintain database integrity even when operations are interrupted. The `DatabaseHelper` class ensures that all database interactions are performed efficiently and correctly. It manages the flow of data to and from the database, encapsulating the complexities of raw SQL handling.

Event	**Stage**	***Duration (ms)***	***CPU Usage (%)***	***Memory Usage (MB)***
Touch Event Press	Switch Fragment	*100.18*	*<1*	*10.4*

TABLE IV
PERFORMANCE METRICS FOR BOTTOM NAVIGATION PART

- `onUpgrade` method facilitates the transition to new database versions, preserving data integrity and consistency across app updates.
- `onCreate` method is only called when the database is created for the first time, ensuring that the schema is set up correctly.
- `getWritableDatabase` and `getReadableDatabase`, it abstracts the underlying SQLite operations, providing a high-level interface for other components of the app to interact with the database. This approach guarantees data integrity and the security of the stored data, as SQLite enforces strict typing and transactional integrity, making the database resistant to corruption and ensuring the atomicity of operations.

Furthermore, the class employs best practices such as prepared statements and batch insertions to enhance performance and protect against SQL injection attacks. These measures are crucial for maintaining the security of the app's data and providing a seamless and responsive user experience.

- **Error Handling and User Feedback**: The class includes comprehensive error handling to cope with potential database access issues or data retrieval problems. It uses logging mechanisms, as seen with the `Log.e` calls, to notify developers of issues that occur during database operations. User feedback mechanisms are integrated into the app to inform users of any issues with data they are interacting with, providing a seamless experience.
- **Functionality Implementation**: `DatabaseHelper` class is endowed with several pivotal methods that facilitate various aspects of data manipulation and retrieval, crucial for the app's dynamic content management:
  - `getRandomQuestion()` method showcases the use of SQLite's `RANDOM()` function to fetch a random entry from the questions table. This method enhances user engagement by providing varied content in scenarios such as quiz apps or practice tests where unpredictability is desired.
  - `getQuestionDetails()` is tailored to retrieve specific details of a question, such as its type or content, based on a unique identifier. This function's precise data retrieval is essential for displaying question details or further processing within the app.
  - `getQuestionIdByContent()` inversely maps a question's content to its unique ID. This is particularly useful for functions like bookmarking or referencing questions where a stable identifier is necessary to recall specific records.
  - `getUniqueTitlesByPart()` compiles a list of unique titles for each part of the app's content, using a `HashMap` to ensure that each part is associated with a distinct set of titles. This method aids in categorizing content, making navigation and organization within the app more intuitive and efficient.

  The initial population of the database with data from a CSV file during the creation phase is handled methodically. The `importDataFromCSV()` function reads the CSV file line by line, parsing each line into individual columns, and inserts the data into the SQLite database using prepared statements. This process not only pre-populates the app with a rich dataset but also showcases the utility of batch processing in database operations for enhanced performance. These methods collectively underscore the sophisticated data handling capabilities of the `DatabaseHelper` class. By abstracting the complexities of raw SQL queries and database interactions, the class provides a streamlined and efficient mechanism for data querying, insertion, and updates. This functionality is paramount in maintaining the responsiveness and dynamism of the app, allowing it to serve up-to-date content to the user with minimal delay and ensuring a smooth user experience.
- **Performance Considerations**: The performance of database operations is a key concern for the app's overall responsiveness. `DatabaseHelper` is optimized to handle data-intensive operations with minimal performance overhead. The use of transactions during CSV import and prepared statements for database insertion help to reduce the time taken for bulk data operations. Furthermore, the class is designed as a singleton to prevent unnecessary database connections, which can significantly impact performance. The methods are also optimized to ensure that database queries are executed quickly and efficiently, providing fast access to data when needed by the app.

III. Evaluation

A. Challenges

During the development of our application, several significant challenges were encountered, ranging from engineering complexities to user interface design concerns. These challenges were critical to address as they significantly impacted the overall functionality and user experience of the application. The key challenges included:

1) A major engineering challenge was managing the lifecycle of `RecorderMedia` and `PlayMedia`. These components are essential in Android app development for handling audio recording and playback. The challenge lay in accurately implementing their opening and closing mechanisms. Inadequate management of these resources could lead to application crashes, undermining stability and reliability.
2) Designing the layout for `HistoryFragment` and `LibraryFragment` posed another significant challenge. We deliberated extensively over the choice of the ViewGroup to use, comparing `LinearLayout`, `FrameLayout`, and `GridView`. Our goal was to achieve an expandable view upon clicking, but these options did not fully meet our requirements.

3) The design of the navigation bottom posed a challenge as we initially considered using activity transitions to switch views. However, this approach was found to be resource-intensive and resulted in unnatural transitions, even when animations were removed using the `no_animation` method.

B. Solutions

To overcome these challenges, we implemented a series of solutions that enhanced both the functionality and user experience of our application:

1) To tackle the first challenge, we utilized `Log` and `Toast` messages during development for real-time situational assessment. We also established an `AudioManager` object to manage `RecorderMedia` and `PlayMedia` effectively. The use of dedicated `Recording` objects for sound management further ensured efficient resource utilization, resolving stability issues.
2) For the layout design challenge in `HistoryFragment` and `LibraryFragment`, we discovered that newer versions of Android API support `ExpandableListView`. This necessitated the use of a `CustomExpandableListAdapter` for efficient management. `ExpandableListView` allowed us to achieve the desired expandable effect upon user interaction, which was crucial for our app's design.
3) To address the navigation issue, we transitioned from using separate activities to implementing fragments. This was achieved through a `OnNavigationItemSelectedListener` and a `switchFragment(Fragment fragment)` method. This approach not only reduced resource consumption but also provided smoother and more natural transitions between different parts of the app.

C. HCI

In terms of Human-Computer Interaction, careful consideration was given to the app's usability. For the `Tap to record` feature, the lower third of the screen was chosen for interaction, as this area is most easily accessible with the thumb, making it convenient for users. This design allows users to start, pause, resume, and stop recording effortlessly, which is highly beneficial for concentrating on their oral practice without the need to constantly look at the phone. After recording, the interface efficiently uses three ImageButtons to execute different functions, making the app more intuitive and user-friendly.

D. Functionality and Integration

Regarding the question of whether the features of our app work as intended and their synergy with other features, the Record Part has been successfully implemented and integrates well with the overall functionality of the app. However, due to the solo nature of the development process and academic pressures, including language exams and assignments, two features could not be completed in time. These are the manual note-taking feature and the implementation of voice-to-text conversion using Android's native APIs.

E. Future Enhancements and Reworked Design

Given more time and resources, several enhancements and reworks are envisioned for the app. Keeping in mind the lecture on context awareness, the following improvements are proposed:

1) **Enhanced Context Awareness:** Incorporating more advanced context-aware features, such as adjusting the difficulty level of questions based on the user's proficiency and learning pace, would make the app more responsive to individual needs.
2) **Interactive Note-Taking Feature:** As previously mentioned, the manual note-taking feature was not completed. With additional time, this feature would be implemented to allow users to jot down notes or key points during their practice sessions, enhancing the learning experience.
3) **Voice-to-Text Functionality:** Implementing the voice-to-text feature using Android's APIs would allow users to see a transcription of their spoken responses, aiding in self-assessment and improvement.
4) **UI/UX Rework:** Based on user feedback and HCI principles, the user interface and experience could be further refined. This might include simplifying navigation, making the app more intuitive, and enhancing visual appeal to keep users engaged.

These proposed enhancements and redesigns are aimed at making the app not only more user-friendly but also more effective as a language learning tool, leveraging the potential of technology to provide a personalized and engaging learning experience.

IV. Conclusion

In conclusion, I successfully navigated through key issues in engineering and interface design, implementing effective solutions that have significantly improved the app's performance and usability. My focus on human-computer interaction principles has resulted in an intuitive and accessible tool for IELTS speaking practice. Wattson stands as a testament to the effectiveness of thoughtful design and user-centered development in creating applications that are not only functional but also user-friendly. I believe that this app will make a substantial difference in assisting users in their language learning journey.

References

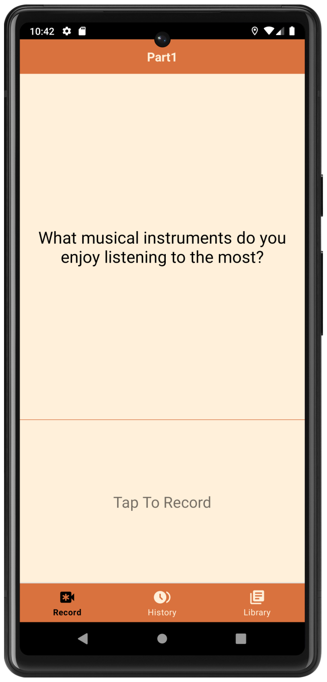
[1] Google Developers, "Documentation — Android Developers," available at https://developer.android.com/guide
[2] Android Developers, "View — Android Developers," available at https://developer.android.com/reference/classes
[3] Android Open Source Project, "Android OS Source Documentation — Android Open Source Project," available at https://source.android.com/docs
[4] Android Developers, "Services overview — Background work — Android Developers," available at https://developer.android.com/guide/components/services

Appendix

A. *Project Repository*

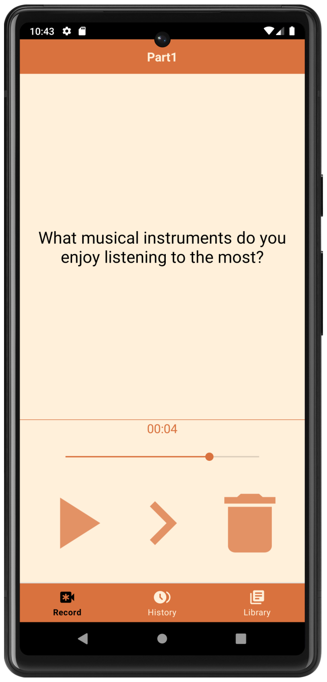
The source code and additional resources for this project are available in the GitHub repository at the following URL: https://github.com/jonlai211/Wattson

B. *Shortcuts*



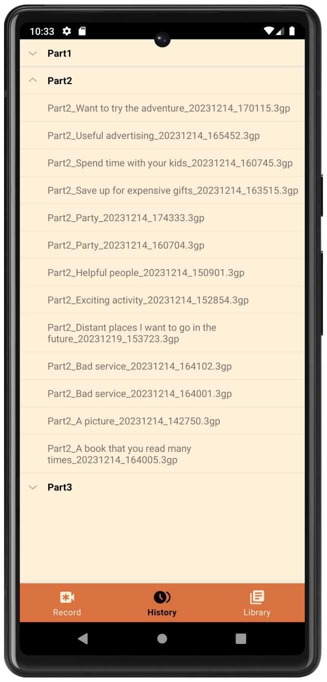
What musical instruments do you enjoy listening to the most?

Fig. 12. Record Shortcut



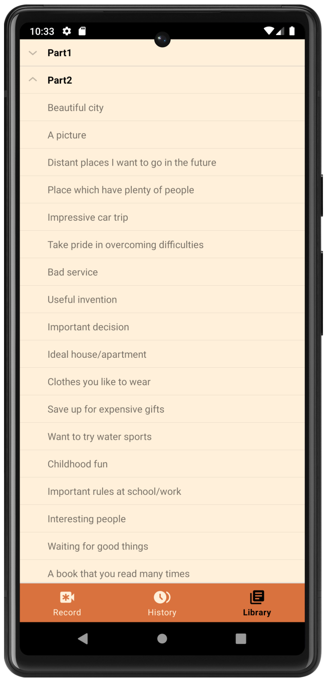
What musical instruments do you enjoy listening to the most?

Fig. 13. Control Shortcut



Part1
Part2
Part2_Want to try the adventure_20231214_170115.3gp
Part2_Useful advertising_20231214_165452.3gp
Part2_Spend time with your kids_20231214_160745.3gp
Part2_Save up for expensive gifts_20231214_163515.3gp
Part2_Party_20231214_174333.3gp
Part2_Party_20231214_160704.3gp
Part2_Helpful people_20231214_150901.3gp
Part2_Exciting activity_20231214_152854.3gp
Part2_Distant places I want to go in the future_20231219_153723.3gp
Part2_Bad service_20231214_164102.3gp
Part2_Bad service_20231214_164001.3gp
Part2_A picture_20231214_142750.3gp
Part2_A book that you read many times_20231214_164005.3gp
Part3

Fig. 14. History Shortcut



Part1
Part2
Beautiful city
A picture
Distant places I want to go in the future
Place which have plenty of people
Impressive car trip
Take pride in overcoming difficulties
Bad service
Useful invention
Important decision
Ideal house/apartment
Clothes you like to wear
Save up for expensive gifts
Want to try water sports
Childhood fun
Important rules at school/work
Interesting people
Waiting for good things
A book that you read many times

Fig. 15. Library Shortcut