An AI Piano Tutor
(Ma Zhiying’s Version)

Supervisor:
Dr. C. Wu
Department of Computer Science
The University of Hong Kong

Team Members:
Lin Yuyang (3035446899)
Ma Zhiying (3035334709)
Wang Ximing (3035448158)

2021.4.19
Abstract

As piano learners struggle to seek guidance for their daily practice, the need for an AI piano tutor becomes evident. Several mobile applications have already been launched to serve this purpose, but the guidance they provide is either limited to the simple correction of each note and or it separates the detail error detection from an overall performance evaluation. This project is therefore motivated to create a mobile application that could not only assess the correctness of each note but also generate a report on the overall performance at the same time. In order to achieve this goal, a client-server system structure is adopted. The two ends interact with each other through HTTP requests and responses. The music processing algorithm is implemented on the back end. To detect the notes and generate the reports, the algorithm utilizes the traditional way of digital signal processing, where the notes are identified by their frequencies.

Currently, a functional prototype is developed and could fulfil the basic use flow where the user follows the music sheet on the screen to practice, the system records the practice and submits it to the back end for analysis, and finally, a report is generated and displayed on the screen. However, there are still limitations to the current results. For instance, only one song is supported in this prototype; the front end is only tested on iOS platform; and the back end is not yet deployed onto a cloud service. Besides, the algorithm is only tested with single notes, which means recognition of superimposed notes, i.e., several notes played simultaneously, is not yet supported. Our next step will be to tackle these limitations by expanding the available song range, testing with Android, deploying the back end onto a cloud platform, and improving the algorithm.
Acknowledgements

This project plan could not be written without the help of my supervisor, Dr. C. Wu, and my teammates, Lin Yuyang and Wang Ximing.
# Table of Contents

ABSTRACT ........................................................................................................... I

ACKNOWLEDGEMENTS ............................................................................. II

LIST OF FIGURES ......................................................................................... IV

LIST OF TABLES ............................................................................................ V

LIST OF ABBREVIATIONS ........................................................................... V

1. INTRODUCTION ....................................................................................... 1
   1.1 BACKGROUND .................................................................................... 1
   1.2 EXPECTED DELIVERABLES ............................................................... 2
   1.3 CURRENT PROGRESS AND CONTRIBUTION .................................... 4
   1.4 OUTLINE OF THIS REPORT ............................................................... 4

2. REVIEW OF EXISTING APPS ................................................................. 5
   2.1 EXISTING SOLUTIONS ....................................................................... 5
   2.2 PROBLEMS WITH EXISTING SOLUTIONS ....................................... 10
   2.3 POTENTIAL IMPROVEMENTS ........................................................... 12

3. METHODOLOGY ....................................................................................... 13
   3.1 SYSTEM IMPLEMENTATION ............................................................... 13
      3.1.1 The Client-Server System Structure .............................................. 13
      3.1.2 Overview of the System Logic ...................................................... 15
      3.1.3 User Interface Design ................................................................. 21
      3.1.4 Implementation Details ............................................................... 27
   3.2 THE MUSIC PROCESSING ALGORITHM .............................................. 35
      3.2.1 The Underlying Theories ............................................................. 35
      3.2.2 Using Existing Solutions ............................................................. 36

4. PROJECT STATUS .................................................................................... 37
   4.1 OVERVIEW OF THE DEVELOPMENT CYCLE ..................................... 37
   4.2 RESEARCH PHASE ............................................................................. 37
   4.3 IMPLEMENTATION PHASE ................................................................. 38
   4.4 GENERAL TESTING AND ADJUSTMENT ......................................... 38
   4.5 PRELIMINARY RESULTS AND LIMITATIONS .................................... 40

5. CONCLUSION ............................................................................................ 43
   5.1 FUTURE WORKS ................................................................................ 43
List of Figures

Figure 1  Screenshot 1 of SimplyPiano  6
Figure 2  Screenshot 2 of SimplyPiano  7
Figure 3  Screenshot 1 of Xiaoyezi  8
Figure 4  Screenshot 2 of Xiaoyezi  8
Figure 5  Screenshot 3 of Xiaoyezi  9
Figure 6  Screenshot 4 of Xiaoyezi  10
Figure 7  An illustration of the client-server system structure  14
Figure 8  The flow chart  18
Figure 9  The sequence diagram of the main use flow  20
Figure 10  The home or main route  21
Figure 11  The detail route  22
Figure 12  The loading screen  23
Figure 13  Dialogue box for the general report  24
Figure 14  The report route  25
Figure 15  Dialogue box for BPM modification  26
Figure 16  The dropdown menu for BPM modification  26
Figure 17  Part of a music sheet for Ode to Joy retrieved online  27
Figure 18  Some samples of the graphics used for the scrolling music sheet  28-29
Figure 19  Some samples of the graphics used for the detailed report  30
Figure 20  Detail route with a snack bar  32
Figure 21  The structure of the front end  32
Figure 22  The structure of the back end  34
Figure 23  An illustration of waveforms of a sinusoid and of real-world sound [1]  36
Figure 24  The iOS simulator running on a computer  39
Figure 25  A UI problem detected in testing  39
Figure 26  The collection of major screens implemented in this app  42
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>The list of expected features in current phase</td>
<td>2-3</td>
</tr>
<tr>
<td>Table 2</td>
<td>The major use case of this project</td>
<td>16-17</td>
</tr>
<tr>
<td>Table 3</td>
<td>An Overview of the Project Schedule</td>
<td>37</td>
</tr>
<tr>
<td>Table 4</td>
<td>The list of features and limitations in current phase</td>
<td>40-41</td>
</tr>
</tbody>
</table>

List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APK</td>
<td>Android Application Package</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>BPM</td>
<td>Beats Per Minute</td>
</tr>
<tr>
<td>IPA</td>
<td>iOS App Store Package</td>
</tr>
<tr>
<td>pt.</td>
<td>Point (typographic unit)</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
</tbody>
</table>
1. Introduction

There is now a great demand for AI piano tutors running on smart phones. Therefore, this project aims to develop such an app that helps with piano learners’ daily practice. This section first provides an overview of the background. It then explicates the expected deliverables of this project, followed by an introduction of the current progress and contribution. Finally, it gives an outline of this whole report.

1.1 Background

Learning piano is popular among children and adults alike but practicing it could turn out to be quite problematic. Piano lessons usually take place only once a week, yet the students need practice much more often than that. Many of them turn to private tutors to fill in the gaps between lessons, while the others choose to practice on their own without any guidance. However, the two options both have deficiencies. On one hand, private tutorials could be expensive and inflexible regarding the schedule and traveling distance. On the other hand, practicing alone may compromise efficiency and accuracy due to the lack of supervision, correction and instruction.

For beginners in particular, the correctness of both the individual notes and the rhythm is essential to their daily practice. It would be very helpful if they could be informed of this basic information whenever they practice. For more fluent players, on the other hand, they might care more about the high-level features of their overall performance.

Therefore, we may assume that there is a potential market among these piano learners, where an AI piano tutor running on smart phones could prove to be helpful. If a mobile application could smartly detect basic errors in notes and rhythms for the beginners, and provide high-level overall evaluation for advanced players, it will alleviate the problem facing many piano learners who are now stuck between expensive piano tutors and inefficient self-practice. After all, AI piano tutor will be much cheaper than the private tutors and also much more flexible in terms of the schedule and venue. The user can literally start practicing any time at any places. Moreover, we may also expect the AI piano tutor to maintain coherent performance in its service. Unlike human-beings, computer will not be distracted or exhausted after long hours of continuous working.
Nowadays, it is increasingly common and popular for users from various age ranges to exploit digital education on platforms such as smart phones and laptops. With the coronavirus that has led to large-scale lockdowns and temporary suspension of face-to-face teachings, the needs for digital learning in various fields have roared greatly and have consequently drawn considerable attention. This high demand is also present regrading digital music learning. In our project, therefore, we would like to draw focus to the possibility of an AI piano tutor as a kind of innovative digital learning platform with state-of-the-art technologies. The goal is to help piano learners, from beginners to more fluent ones, with their daily practice.

1.2 Expected Deliverables
This project aims to develop an AI piano tutor app that guides piano learners’ practice. It should be able to provide an overall performance report that includes both the general evaluation and the detailed feedbacks, and in later phases, the possibility of customizing compositions.

In current phase, a functional prototype of the app should be developed. In particular, the resulting prototype should be able to realize some basic functions as follows.

<table>
<thead>
<tr>
<th>User Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI-1</td>
<td>Retrieve and display available songs</td>
</tr>
<tr>
<td>UI-2</td>
<td>Display the music sheet of the selected song</td>
</tr>
<tr>
<td>UI-3</td>
<td>Provide a real-time pointer to accentuate which note the users should play at a certain time</td>
</tr>
<tr>
<td>UI-4</td>
<td>Allow the users to change the BPM as they desire</td>
</tr>
<tr>
<td>UI-5</td>
<td>Allow the users to start/restart recording for their practice</td>
</tr>
<tr>
<td>UI-6</td>
<td>Allow the users to submit the recording for a performance report</td>
</tr>
<tr>
<td>UI-7</td>
<td>Provide the users with an overall report displayed in a user-friendly fashion</td>
</tr>
<tr>
<td>UI-8</td>
<td>Allow the users to check their individual errors in detail</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>AL-1</td>
<td>Able to recognize the sequence of notes in an audio input</td>
</tr>
</tbody>
</table>
| AL-2      | Able to identify errors, including:  
|           | a. wrong notes,  
|           | b. missed notes,  
|           | c. wrong rhythm,  
|           | d. redundant notes |
| AL-3      | Able to calculate scores from several different criteria:  
|           | a. overall performance,  
|           | b. notes correctness,  
|           | c. rhythm correctness,  
|           | d. speed |

**Table 1: The list of expected features in current phase**

As illustrated above, first, it should provide the user interface for users to select a song to practice, view the corresponding music sheet, record their practice, upload the recording, and view the report generated.

To generate that report, the app should also provide algorithms to analyze the recording, detect the sequence of notes in that recording, identify different types of errors including wrong notes, missed notes, wrong rhythm, redundant notes and so on, and finally generate scores based on the spotted errors.

For this functional prototype, at least one song should be supported, and the app should at least be fully functional on the iOS platform (which means, testing on Android is not compulsory). Since one song on one platform is already enough to showcase the functionality of the application, more songs and more platforms are not strictly required. For simplification, only the single note detection is considered in the initial phase. Recognition of superimposed notes is not strictly required.

Apart from the basic functions, in later phases, it should also include recognition of superimposed notes. It may also consider including more aspects in the overall report such as
some more advanced criteria like the expressiveness. When time permitting, the additional function of customizing compositions could also be researched on, but noteworthy, these three functions are not included in the initial scope of work at the current stage.

1.3 Current Progress and Contribution
This project is divided into three phases, namely the general research phase, implementation phase, and general testing phase. Each teammate contributed equally to the project. In the research and testing phase, all teammates collaborated together, and thus, no strict work distribution was present. In the implementation phase, however, the work was divided into two branches, the system implementation and the algorithm implementation. I was in charge of the system implementation, while my teammates were responsible for the algorithm implementation. More specifically, I designed and constructed the whole client-server system, designed and implemented the entire user interface, handled the interaction between the two ends, and developed part of the back end. (see Section 4.3 for more details).

In terms of our current progress, the following tasks have been finished:

- Existing music processing libraries have been examined, and relative ones have been selected.
- The client app and the server app have been constructed.
- On the back end, relevant APIs have been developed.
- On the front end, all features in Table 1, from UI-1 to UI-8, have been implemented.
- The two sides are able to send and receive requests and responses via APIs.
- For the algorithm, all features in Table 1, from AL-1 to AL-3, have been implemented.

(see Section 4.5 for more details).

1.4 Outline of This Report
Section 2 gives a general description of two selected apps that have functions similar to our project’s. It then discusses some deficiencies in those existing apps and proposes a couple of potential improvements that could be made in our project.

Section 3 elaborates on the project methodology from two aspects, the system and the algorithm. Since I am responsible for the system implementation, more details are given to
that part. In particular, it illustrates the overall client-server structure of the system, gives an overview of the system logic, explains the user interface design, and finally, specifies the system implementation details. It then introduces the music processing algorithm in general terms, which introduces its underlying theories and the existing libraries and code samples.

Section 4 describes the results of our project within the current development cycle. At the moment, the project has gone through a whole development cycle where a functional prototype has been developed. The prototype generally met the expected requirements, though some limitations are still present and need to be tackled in the next development cycle.

Section 5 concludes this report and proposes some works that could be done in the future to improve the results.

2. Review of Existing Apps

Nowadays, technical progress in signal processing has given rise to new possibilities for music information retrieval. Some mobile apps have already been launched in recent years based on these technologies to serve as AI piano tutors. These trials are excitingly creative, but they can still be improved on. This section therefore examines two selected apps, analyzes their deficiencies, and proposes some potential improvements that could be made in our project.

2.1 Existing Solutions

The advances in technology have now made it possible for computers to analyze and process audio signals by their acoustic waves [1]. The research in this field has already achieved an abundance of promising functions such as chord recognition, tempo and beat tracking, and various audio retrieval strategies [1]. There are also several open-source libraries dedicated to music information retrieval such as Essentia [2].

Based on these technical progresses, several AI piano tutoring apps have emerged for commercial purposes. Most of these apps, such as Xiaoyezi and SimplyPiano, provide an in-built database of songs or music compositions and allows for no customized compositions.
Figure 1 below shows how the user interface of SimplyPiano looks like when checking for user’s correctness.

![Image of SimplyPiano user interface](image)

**Figure 1**: A screenshot of the user interface of SimplyPiano, where the app checks whether the user is playing correctly

As illustrated in Figure 1, when the user is playing the piece, a music sheet is displayed on the screen, and there is a pointer moving along the sheet as the time passes to accentuate which note the user shall be playing at a particular time. The app only focuses on one note at a time. To be more specific, it looks for the supposed note in a fixed pre-defined duration, depending on the tempo and rhythm. If redundant notes are played, the app will not deem it wrong. This is probably a deliberate design because the apps are not always accurate in filtering out irrelevant audio signals. Besides, it might be difficult to reflect redundant notes on the UI. Since the feedback is real-time, the UI should also minimize distraction in its feedback so that the user could focus on the next note to play. As a result, in this app, as long as the desired note is detected in the pre-defined duration, the note will appear green on the music sheet, otherwise, it will appear red.

After the user has completed the composition, an overall report will be presented. Figure 2 below shows how the screen will look like in this scenario.
As can be seen above, the report only consists of three parts, the notes, the timing and assists. The first criterion, the notes, is a simple calculation of how many notes are deemed correct (that is, are shown in green), while the timing is also quite mechanical, as it is judged from how “punctually” the user plays on each note compared with the pre-defined timing. The last criterion, assists, simply means how often the user utilizes the assistance function.

Another application called Xiaoyezi applies a slightly different strategy in its AI tutoring functions. It divides its services into three different modes, namely Intro, Improve, and Evaluate. Figure 3 below shows the three modes for choice.
Figure 3: A screenshot of the user interface of Xiaoyezi, where the users are allowed to choose from three different practice modes

In the first mode, Intro, the focus is on playing each note correctly. On the screen, there will be a “bar” on the music sheet notifying which note to play. When the user plays the correct note, the system will color the note green, and the “bar” will jump to the next note on the sheet. Figure 4 below illustrates how this mode looks like.

Figure 4: A screenshot of the user interface of Xiaoyezi in Mode 1, Intro

Noteworthily, the “bar” only moves when the user plays the correct note. Otherwise, it will stay at the same place. This differs from SimplyPiano whose pointer is constantly moving regardless of the user input.
In the second mode, *Improve*, there is no “bar” to signal the notes. The composition is divided into several lines, and each time, the user is only allowed to play one line. After the system deems that a line is finished, it will return a feedback as illustrated in Figure 5.

![Figure 5: A screenshot of the user interface of Xiaoyezi in Mode 2, Improve, where feedback is given to reflect how the user performs](image)

As can be seen from the screenshot above, notes that are played correctly are colored in green, whereas those that are played wrong are colored in red. The app also indicates other errors such as missing notes, and notes being played too early or too late. With the extra information, the user could be aware of his/her performance both in terms of note correctness and the accuracy of beats, which is quite helpful for beginners to get familiar with the composition.

In the last mode, *Evaluate*, the user is asked to play the whole piece non-stop. Same as the previous mode, *Improve*, no “bar” is there to signify which note to play. After the app perceives that the end of the composition is reached, it returns an overall report as in Figure 6.
As illustrated above, in addition to an overall numeric score, the app also generates feedback in five different dimensions, including Pitch, Rhythm, Speed, Stability, and Completeness. However, in this mode, no other detailed information is given, and therefore, the user could not know where in particular he/she makes mistakes.

2.2 Problems with Existing Solutions

The existing solutions, as introduced above, still have some deficiencies that compromise their overall effectiveness.

In SimplyPiano, the progress bar together with the real-time feedback functions well to instruct the user of the timing of each note, and it also allows the user to check his/her own correctness timely. However, the constant feedback may also turn into distraction, especially for more advanced players who do not struggle that much with the accuracy of individual notes. Besides, the real-time strategy imposes high demand on the calculating power of the server. When there is too much network traffic that exceeds the capacity of the back-end server or when there are network delays, the real-time feedback will be largely compromised. Therefore, there is risk that the user may find themselves stuck in a certain note, waiting for the feedback, which interrupts their practice.

After the user finishes the piece, a simple report is displayed, but the user is not allowed to go back to the sheet to check the errors he/she made while playing. With this design, the user
does not have the opportunity to review and reflect on the errors. Instead, he/she only sees the specific errors in a real-time style, as he/she is playing. But as a matter of fact, some users would prefer a chance to check the errors in detail after he/she finishes. Moreover, the overall report generated at the end of the practice is simply an arithmetic addition of the real-time feedbacks. No general evaluation on the overall performance is provided other than the aforementioned mechanical calculation. Although this might be enough for elementary learners, as the complexity of the compositions increases, more powerful evaluations are necessary. The duration, strength and other aspects of each note, for instance, are not considered in this app. An advanced player, therefore, might not find this kind of report very helpful.

The other application, Xiaoyezi, provides no real-time feedback. It focuses on the correctness of each note in Mode 1, Intro, includes other errors such as beat errors in Mode 2, Improve, and generates an overall report in Mode 3, Evaluate. The three modes it provides correspond to the progression of learning focus from individual notes, each line, to the whole piece.

Mode 2 provides the most detailed feedback. However, it forces the user to stop at the end of each line. This implies that its underlying algorithm is only accurate within a small section. As the scope of the input gets bigger, most likely is this app uncapable of calculating the detailed feedback.

Therefore, in Mode 3, where the user is allowed to play the whole piece continuously, unlike in Mode 2, the final report generated is rather vague. Although it includes various dimensions that may be helpful both to beginners and more fluent players, it does not inform the user where in particular he/she makes mistakes. It is thus harder for the user to spot where exactly to improve on. Consequently, he/she may have to retreat to Mode 2 to achieve that. However, in that case, in order to hopefully locate where he/she previously plays wrongly, the user has to play the sheet line by line. This boring process of repeatedly going back and forth between the two modes may not be convenient or efficient enough. Thus, it would be preferrable if the app could provide the general report and the detailed individual errors simultaneously instead of separating them into two modes.

Besides, despite more dimensions in its general report, it is questionable whether the five dimensions that it reports on, respectively the Pitch, Rhythm, Completeness, Speed, and
Stability, are actually helpful. Pitch and Rhythm are in fact the same as the so-called Notes and Timing in SimplyPiano and are also mechanical calculation of the errors made in individual notes. Completeness is about whether every note is played, or some are omitted. This is a very basic criterion and may not be helpful for fluent players, either. Speed and Stability are likely reflected by the value of BPM of the user input. It might be better if more high-level information in the music is also taken into consideration, such as the emotions, expressiveness, and so on.

Another limitation to both apps is that they only allow users to play pre-defined compositions in their database and does not allow for customized pieces. This may also cause inconvenience for users who have their own musical preference.

2.3 Potential Improvements
Based on these observations, this project proposes to improve on existing solutions at two potential edges, namely, an overall performance report that includes both general evaluation and detailed feedback, and the possibility of customizing compositions.

As mentioned above, SimplyPiano limits its evaluation to the correctness of individual notes only. However, since music exists as a continuous and thus infinite stream of information, evaluating it only as separate notes may risk omitting lots of information in-between that could be equally vital to the quality of the piece. Therefore, if an overall evaluation can be calculated that includes more complex criteria, the user might get a much more helpful knowledge of their performance. Moreover, it is unlikely for pianists, especially the advanced ones, to stare at the screen all the time to check their real-time errors on each individual note. Instead, they will possibly prefer to play a whole piece before checking the overall result, which also implicates the insufficiency of the over-simplified real-time feedback adopted in apps like SimplyPiano.

Xiaoyezi, on the other hand, separates the detailed error feedbacks with the overall evaluation, and the user does not have access to both kinds of information at the same time. This may cause inefficiency and inconvenience in the user experience and is therefore undesirable.
This project, therefore, aims to develop an application that combines the advantages of these two apps by providing feedback on individual errors and an evaluation of the general performance at the same time.

In addition, the option of customizing compositions, if made possible, will increase the flexibility for users to choose their own preference of music at their own skill level. In particular, the users may upload compositions that are not in the built-in database by uploading photos of the music sheet and a playing sample of this piece. The app will then process the new composition in the same way as it treats existing ones. This project, in later phases, could also explore the possibility of this additional design.

3. Methodology
As a mobile application, the system architecture is essential to our project. Our system is designed to be composed of two parts, respectively the client side and the server side. In addition to the overall structure, the algorithm design also plays an important role, because music processing lies in the core of the functionality of this app. Therefore, the following sections introduce the methodology of our projects in these two aspects.

Noteworthily, since I am in charge of the system implementation, more emphasis is put on the system rather than the algorithm.

3.1 System Implementation
The system of the AI piano tutor application applies a distributed design, in the sense there are two ends running on two different machines. The front end, that is, the client-side app, runs locally on the users’ smart phones, whereas the back end, that is, the server-side app, runs on a remote server. The following sections illustrate the overall system structure, system logic, UI design, and implementation details.

3.1.1 The Client-Server System Structure
The AI piano tutor app of our project consists of a client-side app and a server-side app. Figure 3 below illustrates the overall structure of the system.
Figure 7: An illustration of the client-server system structure

As illustrated above, the client-side app implements the user interface, whereas the server side handles the algorithms for music processing. The client-side calls the APIs provided by the server-side so that the calculation could be carried out at the back end, and the results would be returned to the front end.

The client-side uses Flutter, a cross-platform toolkit that can be used to build native apps both for Android and iOS from a single codebase. Traditionally, developers use Android and Swift to implement two sets of codes for the two major mobile platforms, namely Android and iOS, but Flutter relieves developers from this redundant work and allows for one set of codes for both platforms. In addition, Dart, the programming language it uses, is proven to be fast on all the platforms it supports. Thus, in order to maximize efficiency, Flutter is chosen for our project.

The server-side, on the other hand, uses Django as its web framework. This framework uses Python as its programming language, and Python is largely preferred in our project because of its preeminent signal processing power. The existing audio processing packages such as pydub, librosa and PyAudio are all implemented for usage in Python only. Moreover, the majority of open-source libraries for music analysis, such as Essentia [2] or madmom [4], are either directly implemented in Python or available for usage in it. This project will rely completely on existing libraries and code samples to realize the music processing functions, and since the majority of the libraries are available in Python, it is reasonable to choose it as
our backend language. In addition to the language compatibility, Django is also rapid in development and pragmatic in its design, which all accounts for our commitment to it.

For this functional prototype, the requirement for the database is rather low, because it only has two major usages. More specifically, it is used to store the information about the available songs and the standard frequencies of each note. Therefore, the built-in SQLite database should be sufficient. SQLite is the default database format in Django, and it is already included in Python, which means that there is no need to install additional things to support it. This simplifies the process greatly and is therefore selected for storing the songs in this functional prototype. However, in later phases, when developing for production-level functionality, it should be migrated to more scalable database such as PostgreSQL, since more data and more complex data relations might be involved when additional functions are added.

For this functional prototype, instead of deploying the server to a cloud service, it runs at the localhost, namely http://127.0.0.1:8000/. This is sufficient for testing purposes. Online deployment is not included in this phase because it might cost extra money. Moreover, it will incur additional issues such as security measures, user authorization and authentication, and so on. However, in our current phase, the focus is mainly on the implementation of the user interface and the algorithm to fulfill the fundamental use flow, and other complications are therefore avoided.

In later phases, for production-level implementation, the server will be deployed to a cloud service platform. Azure is likely to be elected as the platform for our usage, especially during the developmental cycle, because the lowest price tier in Azure is actually free of charge. Although this price tier only provides limited storage space and calculation power, it should be enough for testing purposes. Therefore, with financial consideration, choosing Azure could be a rather practical decision.

3.1.2 Overview of the System Logic
In the current phase, for the functional prototype, only one major use case of the app is present. To give an overall picture of the functions of the system, this use case is illustrated in Table 2 below.
<table>
<thead>
<tr>
<th>Use Case:</th>
<th>Practice and Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Actor:</td>
<td>Piano Learner (Primary User to the APP)</td>
</tr>
<tr>
<td>Description:</td>
<td>Piano Learner uses this app to practice a selected piece and views the performance report.</td>
</tr>
<tr>
<td>Stakeholders and Interests:</td>
<td>Piano Learner:</td>
</tr>
<tr>
<td></td>
<td>• Wants visual aids to display the music sheet and to indicate which note to play.</td>
</tr>
<tr>
<td></td>
<td>• Wants a general report to evaluate his/her overall performance.</td>
</tr>
<tr>
<td></td>
<td>• Wants a detailed report to mark every individual errors.</td>
</tr>
<tr>
<td>Preconditions:</td>
<td>PRE-1: The sequence of notes in the selected piece is known to the system.</td>
</tr>
<tr>
<td></td>
<td>PRE-2: The graphics for the music sheet of the selected piece has already been created and saved in the system.</td>
</tr>
<tr>
<td></td>
<td>PRE-3: The user has given the application the permission to use the microphone.</td>
</tr>
<tr>
<td>Postconditions:</td>
<td>N/A</td>
</tr>
<tr>
<td>Normal Flow:</td>
<td>1. The system displays all songs available.</td>
</tr>
<tr>
<td></td>
<td>2. Piano Learner selects one of the listed songs.</td>
</tr>
<tr>
<td></td>
<td>3. The system displays the music sheet of the selected song on the screen.</td>
</tr>
<tr>
<td></td>
<td>4. Piano Learner chooses to start.</td>
</tr>
<tr>
<td></td>
<td>5. The system starts recording and uses visual aids to guide the player.</td>
</tr>
<tr>
<td></td>
<td>6. Piano Learner follows the visual aids to play the piece.</td>
</tr>
<tr>
<td></td>
<td>7. The system stops scrolling when the end of the piece is reached.</td>
</tr>
<tr>
<td></td>
<td>8. Piano Learner chooses to run a report for his/her performance.</td>
</tr>
<tr>
<td></td>
<td>9. The system analyzes the recording, generate a report, and displays the data in the report.</td>
</tr>
<tr>
<td></td>
<td>10. Piano Learner view the report.</td>
</tr>
<tr>
<td>Extensions:</td>
<td>1. Piano Learner chooses to change the BPM:</td>
</tr>
<tr>
<td></td>
<td>a. The system displays the BPM options.</td>
</tr>
</tbody>
</table>
b. Piano Learner selects one of the options and confirms the change.
c. The system changes the standard BPM.

2. Piano Learner chooses to restart anytime after step 4 in normal flow:
   a. The system stops recording and reverts the visual elements back to step 3.

<table>
<thead>
<tr>
<th>Assumptions:</th>
<th>1. The device (i.e., the smart phone on which the app is running) is stably connected to the Internet.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. The device’s microphone is functioning.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special Requirements:</th>
<th>1. Piano Learner should follow the visual aids to start playing.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Piano Learner should follow the music sheet displayed on the screen to play instead of playing a different version of the selected song or completely irrelevant songs.</td>
</tr>
<tr>
<td></td>
<td>3. For iOS devices, the iOS version should be 8.0 or larger.</td>
</tr>
</tbody>
</table>

**Table 2: The major use case of this project**

In addition to the textual information listed in the table above, the user flow of this application is also illustrated in a flow chart below (see Figure 8). This is basically a visual representation of the “Normal Flow” field and the “Extensions” field in Table 2.
In the flow chart above, the elements colored in blue illustrate the main user flow, where the user opens the app, selects a song, and starts recording his/her practice immediately. And then, after the user finishes playing the whole piece, he/she submits the recording and runs a report for it. Finally, the user checks the report and views the individual errors he/she made.

The elements in green are optional flows, that is, the extensions. The first additional function is the “Change BPM” function, which corresponds to UI-4 in Table 1. To be more specific, the user is allowed to change the BPM before he/she starts playing so that he/she may customize the standard speed at which he/she prefers to play. For instance, when the user is not familiar with the piece, he/she may set the BPM to be smaller so that he/she could play slower. As the user improves, he/she may then change the BPM back to a larger number.

Figure 8: The flow chart
Another optional function is the “Restart” function. In particular, the user is allowed to restart at any time after he/she has started recording. No report will be generated if the user chooses to restart. This gives the user the alternative to start over, if he/she realizes that he/she has been making too many mistakes and is therefore unwilling to see the report. Noteworthily, when the user chooses to restart, the previous recording will be discarded.

As mentioned before, the main user flow colored in blue in Figure 8 is also a representation of the normal flow of the major user case of this project (see Table 2). In this use flow, the user uses the app to practice a song and then, view the feedback of his/her performance. The way in which the client-side and server-side interact with each other, and subsequently serve the user in this use flow is illustrated in a sequence diagram (Figure 9) on the next page.

In Figure 9, the three objects colored in blue reside at the client side. They are three different “routes” implemented in Dart in the client-side app. In flutter, each page or screen is called a “route”, so these three routes actually represent three different screens on the front end (the layout and design details of these screens are introduced in the next section).

The other three objects in yellow belong to the server side and are implemented in Python within the Django framework. In particular, music.views.songs and music.views.upload are two view functions that handles web requests and returns the corresponding responses. Song, on the other hand, is a model that maps to a table in the database, that is, the built-in SQLite database. The information of all available songs has already been added to the database beforehand. The view function, music.views.songs, simply sends a query to get all objects in the Song table, and then send back the retrieved song list back to the front end in a json response. The other view function, music.views.upload, is the core functionality of the back end. It handles a multipart HTTP POST request from the front end and retrieves the recording file from the request (through the handle_recording function). It then processes the recording to generate the report (through the process_music function).

(next page) Figure 9: The sequence diagram of the main use flow
3.1.3 User Interface Design
As mentioned in the previous section, pages or screens in Flutter are called routes. To fulfill the features mentioned in Table 1 (UI-1 to UI-8), three routes are designed and implemented, namely the home route (see Figure 10 below), the detail route (Figure 11), and the report route (Figure 14). Meanwhile, another two dialogue boxes are also implemented. One is used to show the general report (Figure 13), and the other is for BPM modification (Figure 15). In order to achieve the best visual experience, same as SimplyPiano and Xiaoyezi, this app forces the screen orientation to be horizontal.

![Choose your music](image)

**Figure 10: The home or main route**

The home route (see Figure 10 above) is the main page, and is rendered when the user opens the app. It retrieves the list of songs available from the server and displays them accordingly. When the user taps on one of the songs, the app will navigate to the detail page of the corresponding song (see Figure 11 below).
This detail page has a design that is quite similar to that of SimplyPiano (see Figure 1), except that there is no real-time feedback. The automatic scrolling of the music sheet, together with the progress bar in the middle of the screen, is designed to regulate the user’s behavior to some extent. These two features inform the user of the timing to play the first note. The user should not play anything meaningful before this timing. This regulation is crucial, because the value of that timing will subsequently be used by the music processing algorithm at the server side as a way to reduce the pressure on the algorithm. Without this value, the algorithm must accurately filter out irrelevant noises at the beginning of the recording to identify the onset of the first note. With this timing value, however, there is a rough idea about the likely onset of the first note and thus eases the task.

Moreover, with the progress bar that moves constantly along the music sheet, the user is provided with more guidance on the rhythm. He/she will have a general idea about when to play each note, how long each note should last, and so on. This will also help the user learn and improve.

After the music sheet is scrolled to the end, the scrolling will be stopped automatically to indicate that the song has been finished. The user may then tap on the “Run Report” button at the bottom-right corner of the screen. After that, the app will send the data to the back end.
and wait for the report. As illustrated in Figure 12 below, while waiting, the screen will dim, and a loading icon is displayed.

![Image of a loading screen](image)

**Figure 12: The loading screen**

This design is to inform the user that his/her “Run-Report” request has been sent and the system is now processing it. It can effectively prevent the user from doing unpredictable actions during the waiting period, such as to click on the “Run Report” button repeatedly, which will result in duplicated HTTP requests, or to click on the “Restart” button, which will cause the system to drop the previous recording.

When the response is received at the front end, the system will render a dialogue box to display the general report of the user’s performance (see Figure 13 below).
The general report consists of an overall score and three sub-criteria, namely the *Note Accuracy*, the *Rhythm Accuracy* and the *Speed*. This design is similar to the overall report displayed in Mode 3, *Evaluate*, of *Xiaoyezi* (see Figure 6). To visualize the report properly, the overall score is transformed into a 5-star rating indicator placed in the top-center of the dialogue box. It is followed by three circular percentage indicators that illustrate the respective accuracy of the notes, rhythm, and speed. This user-friendly data-visualization approach is similar to the design of *SimplyPiano* (see Figure 2). With these visual components, this performance report is designed to make the underlying information easily and accurately comprehensible by the user.

If the user wants to view the details of his/her performance, he/she could click on the “Details” button on the bottom-right corner of the dialogue box. Then, the system will navigate to the report route (see Figure 14 below).

**Figure 13: Dialogue box for the general report**
As illustrated above, the report route renders the whole music sheet on the screen and colors all the errors in pink. The errors are further categorized into 5 groups, which are “wrong note”, “wrong rhythm”, “both the note and rhythm are wrong”, “note omitted”, and “extra notes played”. For now, each error category is represented by a single alphabet, which is marked on the music sheet under the note where the specific error is detected. This visual representation could be improved in later phases.

Noteworthily, the detailed report is separated from the general report and is displayed on another route. This is an intentional design, because in this way, it would be less likely for the user to get overwhelmed by receiving too much information at a time. And therefore, the user might perceive the data more accurately. What’s more, it provides the user with the freedom to choose whether he/she would like to view the detail information. If the user is busy or simply is uninterested, he/she may skip the detail report and start a new practice immediately. On the other hand, if the user prefers to know where exactly he/she has made mistakes, he/she is also welcomed to do so. Making the detail report an alternative view might make the user experience more enjoyable by catering for different user needs simultaneously.

In addition to the aforementioned screens that follow the main user flow, the additional function of changing the BPM is also included in the design. As mentioned before, this function allows the user to customize the BPM he/she would like to follow for his/her
practice. For instance, if the user is new to the song and wants to play at a slower speed, he/she may change the BPM to a lower value. In order to do that, the user may click on the “setting” icon on the top-right corner of the detail route (see Figure 11), and a dialogue box will then pop up as follows (see Figure 15).

![Figure 15: Dialogue box for BPM modification](image)

As can be seen in the Figure above, the dialogue box contains a dropdown button. When the user clicks on the dropdown button, a list of options is displayed in the dropdown menu (see Figure 16 below).

![Figure 16: The dropdown menu for BPM modification](image)
The user may click on an item in the dropdown menu, as shown in Figure 16, and then, the system will change the BPM to the selected value. If user click on the “OK” button, this change will be confirmed, otherwise, this change will be discarded, and the previous BPM value will still be in effect.

These are all the screens implemented at this stage. In general, a consistent theme color and theme font is applied to all of them. This aims to make the visual experience more pleasing. And it also helps to create a brand image.

3.1.4 Implementation Details
This section dives into the detailed specifications of the system implementation in several aspects, including the graphics, recording, front end, and back end.

3.1.4.1 Preparing the Graphics
In this application, the music sheet is displayed on the screen and is scrolled automatically in a way that corresponds to the rhythm of the composition piece. The progress bar has a fixed position in the center of the screen, whereas the “container” that holds the music sheet is scrolled horizontally to the left. When the center of a note superimposes the progress bar, it indicates that the note should be played.

For the music sheet, a fixed scrolling speed is maintained because unstable scrolling speed will make the viewer dizzy. Since the scrolling speed is constant, in order to make the progress bar accurate, the intervals between two notes must align with the expected duration of the note in question. Take Ode to Joy (see Figure 17 below) as an example.

![Figure 17: Part of a music sheet for Ode to Joy retrieved online](image)

Figure 17 is taken from a random music sheet for Ode to Joy retrieved online. The E4 note circled in green is a quarter note and should last for 1 beat, whereas the E4 note circled in
orange is a dotted note which lasts for one and a half beat. Assume the distance between the green E4 and the orange E4 is $x$ (see Figure 17), and the distance between the orange E4 and the next note, i.e., an eighth note of D4, is $y$. Then, in order to preserve the rhythm, the music sheet displayed in the application should have $y = \frac{3}{2}x$. Similarly, the distance between every two notes should be calculated in this way.

However, the existing music sheets on the Internet do not have accurate placing of every note. As you may see in Figure 17, the $x$ in that music sheet is even larger than $y$. Therefore, existing music sheets are not valid for this application. In order to make the relative placing accurate, a music sheet must be developed manually and note-by-note for this app.

To achieve this goal, I first researched on some existing Flutter packages that might help with music-sheet development. Although there was one package called sheet_music that claimed to have the capability of displaying sheet music, it lacked the necessary documentations. After examining the examples and source codes of this package, it was concluded that, instead of drawing a whole music sheet, it was only capable of drawing one single note at a time with this package.

Since no existing packages could help with the task of rendering the music sheet, it became necessary for me to develop the graphics on my own. With the help of the design software, Adobe Illustrator, each note (and note group) was drawn independently (see Figure 18 below). After all notes (and note groups) were drawn and added to the front end, they were then rendered in the correct on the screen to form the music sheet.
(2) A half note of D4

(3) An eighth note of C4 and a half note of C4 grouped together

**Figure 18:** Some samples of the graphics used for the scrolling music sheet

As in Figure 18, an individual graph is developed for each of the note (or note group) that appears in the music sheet. When developing the graphs, one beat is assigned with a standard width of 300 pt. This means, when a note is a quarter note, the distance from its center to the center of the next note should be 300 pt. When a note is an eighth note, this distance in-between should be 150 pt. Normally, the graph for a note has a fixed width of 300 pt. (e.g., Figure 18(1) and (2)). And the center of the note is 150 pt. away from the left edge of the graph.

In most cases, one note has one graph (see Figure 18(1) and (2)), so as to maximize reusability. However, for some special notes such as the pair of an eighth note of C4 and a half note of C4 grouped together (see Figure 18(3)), more than one notes must be drawn together because they are too close to each other and cannot be cut into halves easily. In Figure 18(3), for instance, the distance between the center (the center of the eclipse part without the tail) of the two notes must be 150 pt. As indicated by the blue line, the end of the first note is very close to the beginning of the second. Even if the two notes are cut into two graphs (referred to as the left half and the right half) along the blue line, they can hardly be reused in other cases because this cutting is not even and thus, imposes great limitation on the width and the position of the immediate note sitting next to them. As mentioned above, in a standard graph (e.g., Figure 18(1) and (2)), the distance between the center of the note and the left edge of the graph is 150 pt. Therefore, if a standard graph is put right next to the left-half of Figure 18(3), the distance between the center of the eighth note of C4 and the center of the note in the standard graph, will definitely be larger than the supposed 150 pt., which makes it inaccurate.
As a result, it is unlikely to develop a universal database of graphics for all music sheets. Instead, although some of the standard graphics may be reused, there still exist exceptional cases where graphics that are unique to one music sheet must be developed. This might limit the scalability of the available songs in this project to some extent, because for each additional song, the set of graphics must be examined and possibly re-developed manually.

For the detailed report route (see Figure 14), on the other hand, all the grouped notes are cut into individual graphs (see Figure 19 below). Each note has two different versions, one in pink color and the other in black. When rendering the report, the system will determine which version to use depending on whether this note is played correctly or not.

![Figure 19](some samples of the graphics used for the detailed report)

This cutting of note groups is necessary because only the wrong notes are colored in pink, and therefore, it will be most efficient to treat each note independently to cater for different error combinations. Since the distance between the notes in the report does not have to be accurate, cutting the groups into halves will not cause the aforementioned inaccuracy problem in this case.
3.1.4.2 Making the Recording

In order to make recordings on the front end, a Flutter package called `flutter_audio_recorder` is used. There are also other packages for audio recording in Flutter such as `flutter_sound`. However, `flutter_sound` requires the iOS deployment target to be higher than iOS 10.0, whereas `flutter_audio_recorder` has a requirement of only iOS 8.0. Therefore, `flutter_audio_recorder` is better at compatibility, as it includes more iOS versions. Moreover, the documentation of `flutter_audio_recorder` is clearer than that of `flutter_sound`, which made it easier to understand and use the package. Taking these into consideration, `flutter_audio_recorder` was selected over its counterpart.

This package supports 3 audio formats, namely .wav, .aac, and .m4a. Among them, .m4a was selected, because it is the default format for recordings made by the built-in recorder in iOS. Since all the testing audios were recorded by iOS and were thus in .m4a format during the algorithm development, it is rational to use the same format in the system as well.

To make the recording, the app will first ask for the permission to use the microphone. The recording function is only available when the user consents to give the permission. In general, there will be three states regarding the recording events in the life cycle of the app. They are Unset, Set, and Recording. Unset means that the user has not set the permission. Set means the user has set the permission and the system is ready to record. Recording means that the system is currently recording. To trace the change of these states, an enum object is created called `RecorderState`. The default value for the recorder state is `RecorderState.Unset`. As the detail route (see Figure 11) is initiated, the system checks for the permission. If the permission is granted, the system changes the state to `RecorderState.Set`. When the user clicks on the “Start” button on the detail route, the system checks the recorder state. If the state is `RecorderState.Set`, the system changes the state to `RecorderState.Recording`, and starts recording. Otherwise, the system will show a snack bar on the bottom of the screen as illustrated in Figure 20 below.
![Ode To Joy](image)

**Figure 20**: Detail route with a snack bar which is displayed when the user has not given permission to microphone usage

At RecorderState.Recording state, when the end of the music sheet is reached or when the user clicks on the “Restart” button (see Figure 11), the state will change back to RecorderState.Set and the system will stop recording.

### 3.1.4.3 Implementing the Front End

The front end is implemented in Flutter. Its overall structure is illustrated below in Figure 21.

```
piano_tutor/
    android/
    ios/
    fonts/
    graphics/
    lib/
        detail.dart
        main.dart
        report.dart
        pubspec.yaml
        ...
```

**Figure 21**: The structure of the front end (some directories and files are omitted)

The main codes are stored in “lib/” directory, where each dart file implements a route. The “fonts/” folder holds the custom font, and the “graphics/” folder holds all images used in this app. The “pubspec.yaml” file contains important information that are crucial to the Dart and
Flutter tooling. The dependencies, that is, all the packages imported in the project, are specified here. Custom fonts and assets such as images are also registered here. The “android/” and “ios/” folders contain platform-specific information and are used to create the IPA/APK files for the respective platform.

During the user interface implementation, several Flutter packages are used. First of all, in order to force the orientation of the app to be horizontal, a package named orientation is imported. Then, for the loading screen, another package named flutter_easyloading is used. The app also makes use of two other packages to draw the general report (see Figure 13), which are the flutter_rating_bar for the 5-start score indicator, and the percent_indicator for the three circular percent indicators.

In order to construct and send HTTP requests to the back end, another package called http is also utilized. In particular, it helps to send an HTTP GET request to get all the songs in “main.dart”. Then, in “detail.dart”, it helps to construct the HTTP multipart POST request with the recording file and several other data fields, and subsequently send this request to the back end. In both cases, the HTTP response is converted into a custom Dart object before the data in the response could be displayed. For the HTTP GET request, a class called Song is defined to hold each Song object, and another class called SongList is further created to handle a list of Song objects. Meanwhile, for the HTTP POST request in “detail.dart”, a class called Feedback is implemented to handle the json response.

### 3.1.4.3 Implementing the Back End

Noteworthily, I was in charge of the overall setup of the back end, and I also implemented part of the models and view functions, especially those directly involved in the client-server interaction. This section will introduce the back-end implementation details relevant to my tasks only (please refer to my teammates’ reports for other details).
Figure 22: The structure of the back end (some directories and files are omitted)

The “media/” directory stores assets such as the recordings uploaded from the front end and other audio samples for testing. The “piano_tutor_backend/” directory contains information about the whole project. For instance, the “piano_tutor_backend/settings.py” file holds the project configuration information such as the list of installed apps and databases. The “piano_tutor_backend/urls.py” provides the routing information of this project. This file further refers to “music/urls.py” for the next level of routing.

The “music/” directory is an app within this project. In Django, a project could have several apps, each being a Web application that has some particular functions. In this project, there is only one app called “music” that handles everything. The “urls.py” file under “music/” deals with the routing of URLs to their corresponding views implemented in “music/views.py” file. Apart from the basic “index” function, there are two view functions in the “views.py” file, “songs” and “upload”. They are two web APIs to which the front end could make requests. The first function, “songs”, takes an HTTP GET request and returns all of the Song objects stored in the database. The second function, “upload” takes an HTTP multipart POST request, and calls the handle_recording function in the “music/handle_recording.py” file,
which handles the FILES field of the request and writes the received file into the back end. After that, it calls the `process_music` function in the “music/algo” directory to process the music (see my teammates’ reports for more details).

The “music/models.py” defines the models for the database. Each model is implemented as a Python class in this file and is associated with a table in the database. In our database, there are three models, namely the `Song`, `Frequency`, and `Standard`. `Song` stores the list of available songs in the application. `Standard` stores the list of note sequence of a song. Each `Song` has a one-to-one mapping to `Standard`. `Frequency` stores the `(note, frequency)` pair that specifies the corresponding frequency value for each note. In the algorithm, when the front end sends an upload request to the back end, the back end first retrieves the name of the song from the request. It then finds the `Standard` object mapped to that song name. Finally, for each note in the `Standard` object, it looks for its corresponding frequency value in the `Frequency` table.

The “manage.py” file is a command-line utility that allows you to run commands such as “python manage.py runserver” to start running the backend application.

### 3.2 The Music Processing Algorithm

In order to generate a report based on the user’s performance, the app needs algorithms to process and analyze the audio inputs. In particular, we need to calculate the beat and the onset timing of each note, detect the note sequences, and give scores based on those results. The traditional and fundamental method to detect and identify notes is through the digital signal processing. Existing open-source libraries and code samples are to be imported into our backend application to help process the audio and generate the results.

In this section, the underlying theories in digital signal processing are introduced briefly. The selected music processing libraries and code samples are also mentioned. Please refer to my teammates’ reports for more details.

#### 3.2.1 The Underlying Theories

In digital signal processing, the waveform of an audio is examined in order to identify the notes. The pitch of a sound is closely related to the frequency of its sound wave. The higher the frequency, the higher the pitch. However, because human ears do not perceive slight
frequency changes, in practice, a range of frequencies are usually associated with one particular pitch. The figure below illustrates how the waveforms look like in different scenarios.

![Waveform of a sinusoid and of real-world sound](image)

**Figure 23**: An illustration of waveforms of a sinusoid and of real-world sound [1]

As illustrated in Figure 23a, a well-defined frequency is a sinusoid whose waveform is periodic. However, sounds in real-world scenarios do not possess a well-defined frequency, but rather, their waveforms are much more complex, as is illustrated in Figure 23b. Therefore, the sound wave of a single note played on a piano may have mixed and changing frequencies, which could be seen as a superposition of several sinusoids. Consequently, in order to determine the pitch of a sound, the usual practice is to find the fundamental frequency, which is the lowest partial present in the waveform, where a partial is any sinusoid that could be extracted from the complex waveform [1].

### 3.2.2 Using Existing Solutions

In our project, instead of implementing the algorithm on our own, several existing open-source libraries and code samples were examined and tested to realize the note identification. In general, two potential libraries and one code sample were spotted and experimented with. They were Essentia[2], madmom[4], and “Node Recognition In Python” [8]. During the algorithm development, the ones with the highest accuracy in the specific tasks during the process of note detection were incorporated into our system.

The experiments with the algorithms and the subsequent implementation were both carried out by my teammates, so please refer to their reports for more details.
4. Project Status

The current development cycle of our project is divided into several stages, from research, implementation to testing and refinement. The following sections first provide an overview of the development cycle and the progress in each phase, and then discuss the current results.

4.1 Overview of the Development Cycle

Our project mainly consisted of 4 phases, with a respective focus on research, implementation, and testing and refinement (see Table 3).

<table>
<thead>
<tr>
<th>Steps</th>
<th>Person(s) in Charge</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Plan</td>
<td>All</td>
<td>Oct 4, 2020</td>
</tr>
<tr>
<td>Phase 1: General Research</td>
<td>All</td>
<td>Oct 5 - Feb 28, 2020</td>
</tr>
<tr>
<td>Phase 2:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Implementation</td>
<td>Me</td>
<td>Mar 1 – Apr 10, 2021</td>
</tr>
<tr>
<td>Algorithm Implementation</td>
<td>Other Teammates</td>
<td></td>
</tr>
<tr>
<td>Phase 3:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>General Testing and Adjustment</td>
<td>All</td>
<td>Apr 10 – Apr 17, 2021</td>
</tr>
<tr>
<td>Finalized Test Implementation</td>
<td>All</td>
<td>Apr 18, 2021</td>
</tr>
<tr>
<td>Project Exhibition</td>
<td>All</td>
<td>May 4, 2021</td>
</tr>
</tbody>
</table>

Table 3: An Overview of the Project Schedule

The first phase, namely the research phase, aimed to gain practical insights into existing algorithms related to music processing. Then, with sufficient knowledge in related works, our project moved into the second phase, the implementation phase. In this phase, the work was distributed among the teammates, where I was in charge of the system implementation. Finally, after the implementation was generally finished, some rough testing was carried out, and slight changes were made accordingly.

4.2 Research Phase

In the research phase, we reviewed several papers, found a few open-source libraries, code samples and datasets, and gained an overall knowledge of the theories behind music processing. Based on our research, given the abundance of related works, we concluded that it was highly feasible to identify notes from recordings with much accuracy, especially since
there were well-established libraries like Essentia [2]. We subsequently experimented with Essentia and discovered some inefficiencies in it, especially that there is no direct algorithm in this library that recognizes notes from an audio input.

We also discovered that there was little existing works in determining the overall performance of a musical piece with high-level features. It is likely due to the great complexity of this task, for several factors such as crescendo and pace all play a part in the overall performance, hence it might be too complicated to evaluate them all at a time. We first tried to work around this problem by comparing the similarity between the user input and a built-in exemplar. In Essentia, there is one algorithm that is used to detect cover songs and returns a similarity value. However, this algorithm only takes into consideration the similarity between the note sequences in the two audio inputs, and none of the other features takes effect, which is not as helpful as we thought. As a result, for our current prototype, we decided to adopt the more practical option. The report was designed to be quite similar to that of Xiaoyezi. The system calculates the percentage of wrong notes and wrong rhythm detected, evaluates the speed correctness by comparing the standard BPM and that of the user input, and finally generates an overall score by taking an average or a weighted sum of these three criteria.

4.3 Implementation Phase
In the implementation phase, the work was distributed among the three teammates, where I was in charge of the system implementation. This included almost everything other than the music processing algorithm. In particular, I independently implemented the entire front end, the general structure of the back end, some of the models and database at the back end and the sending and receiving of files and data in-between two ends. Meanwhile, my teammates were responsible for further experiments with different libraries, implementing the music processing algorithm, and generating the report on top of that. The implementation details are discussed in Section 3.1.4, and the results will be illustrated below in Section 4.5.

4.4 General Testing and Adjustment
Since the back end is not yet deployed to a cloud service, the only way to test the whole app is by running the server and the front-end app on the same computer. For the front-end app, an iOS simulator is used (see Figure 24 below).
As is illustrated above, the iOS simulator resembles the screen of an actual iPhone and is a useful tool for prototyping. However, it cannot be distributed to a lot of users easily for the testing purpose. Therefore, the testing of the application at this stage is limited to the teammates only.

During the testing, some UI problems were detected and fixed. For instance, in some cases, even if the data in the HTTP response is rounded to the third decimal place (e.g., 0.952) the circular percentage indicator shows something as in Figure 25 below. This problem was then resolved by rounding the response data again before being displayed.
There also lacked proper UI feedback when an error was encountered at the back end, either the connection failed or the report could not be generated because, for example, the user uploaded an empty recording. This feature was subsequently added in the testing phase.

Besides, the automatic scrolling of the music sheet is not smooth enough. This will compromise the user experience greatly because the user is supposed to stare at the music sheet and follow the pointer on it during practice. When the scrolling is not smooth, the user may find it difficult or uncomfortable to read the notes on the sheet. It is still not clear whether this is a problem with the iOS simulator or the way the scrolling is implemented. In later phases, this problem will be examined.

At the back end, adjustments were also made. Initially, the recording file uploaded by the front end had a fixed name and was saved at a fixed location, but this might cause concurrency problem when several users tried to upload their recordings at the same time. Therefore, the name was later changed to be the timestamp of the receiving time, which helped resolve the concurrency problem.

Please refer to my teammates’ reports to see other adjustments made to the algorithm.

### 4.5 Preliminary Results and Limitations

As mentioned above, the objective of the current development cycle is to create a functional prototype of the AI Piano Tutor app. In light of this objective, the result is generally satisfying. The following table summarizes the results at the current stage.

<table>
<thead>
<tr>
<th>User Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI-1</td>
<td>Retrieve and display available songs</td>
</tr>
<tr>
<td>UI-2</td>
<td>Display the music sheet of the selected song</td>
</tr>
<tr>
<td>UI-3</td>
<td>Provide a real-time pointer to accentuate which note the users should play at a certain time</td>
</tr>
<tr>
<td>UI-4</td>
<td>Allow the users to change the BPM as they desire</td>
</tr>
<tr>
<td><strong>UI</strong></td>
<td>Allow the users to start/restart recording for their practice</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td><strong>UI</strong></td>
<td>Allow the users to submit the recording for a performance report</td>
</tr>
<tr>
<td><strong>UI</strong></td>
<td>Provide the users with an overall report displayed in a user-friendly fashion</td>
</tr>
<tr>
<td><strong>UI</strong></td>
<td>Allow the users to check their individual errors in detail</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Algorithm</strong></th>
<th>Able to recognize the sequence of notes in an audio input</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AL-1</strong></td>
<td>Able to identify errors, including:</td>
</tr>
<tr>
<td></td>
<td>e. wrong notes,</td>
</tr>
<tr>
<td></td>
<td>f. missed notes,</td>
</tr>
<tr>
<td></td>
<td>g. wrong rhythm,</td>
</tr>
<tr>
<td></td>
<td>h. redundant notes</td>
</tr>
<tr>
<td><strong>AL-2</strong></td>
<td>Able to calculate scores from several different criteria:</td>
</tr>
<tr>
<td></td>
<td>e. overall performance,</td>
</tr>
<tr>
<td></td>
<td>f. notes correctness,</td>
</tr>
<tr>
<td></td>
<td>g. rhythm correctness,</td>
</tr>
<tr>
<td></td>
<td>h. speed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Limitations</strong></th>
<th>Only one song, <em>Ode to Joy</em>, is available</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L-1</strong></td>
<td>Only iOS platform is tested</td>
</tr>
<tr>
<td><strong>L-2</strong></td>
<td>The server-side is not yet deployed to a cloud platform</td>
</tr>
<tr>
<td><strong>L-3</strong></td>
<td>The algorithm is only accurate to process one single note at a time, and is not yet fully functional with several notes played simultaneously (i.e., superimposed notes)</td>
</tr>
</tbody>
</table>

*Table 4: The list of features and limitations in current phase*
All requirements on the user interface (see UI-1 to UI-8 in Table 1) were basically fulfilled and the results are illustrated in Figure 26 below.

![User Interface Screenshots](image)

**Figure 26: The collection of major screens implemented in this app**

The expected features of the algorithm (see AL-1 to AL-3 in Table 1) were also generally achieved (see my teammates’ reports for more details).

However, there are several limitations as illustrated in L-1 to L-4 in Table 4. First, the prototype supports only one song, i.e., *Ode to Joy*. Although, as mentioned in Section 3.1.4.1, it is not entirely easy to prepare the graphics for different songs, it is still doable, except that each additional song will require customized graphics to be developed manually and exclusively for it.

In addition, only iOS platform has been tested at this point (L-2 in Table 4). If Android is to be included in later phases, extra configuration and testing will be needed. But since Flutter is inherently compatible both to Android and iOS, this may not be a problem.

Meanwhile, the back end is not yet deployed to a cloud service such as Azure or AWS. Currently the back end is tested by running it on the localhost, and the Python interpreter it uses is the one installed on the local computer. In order to let it run correctly on a cloud service, a Python interpreter will need to be configured online and the relevant Python packages will also need to be installed there. Moreover, online deployment inevitably incurs...
additional issues such as the security measures, which is not within the current scope. This will be tackled in later phases and is expected to be somehow time-consuming.

Last but not least, there are still limitations to the algorithm. At this point, no superimposed notes are tested, and it is still not clear whether the same approach (detecting notes by their frequencies) could apply to superimposed notes. Please refer to my teammates’ reports for more discussion on the algorithm.

5. Conclusion

This project is initiated with a hope of alleviating the difficulties facing piano learners when practicing piano. It aims to create a mobile application that is easily accessible and fully functional in identifying errors and evaluating the performance of the users. Currently, detailed research in related works has been carried out, a functional prototype has been implemented, and some general testing has also been executed. There are still several limitations with the current implementation, and further testing among potential users is also necessary to discover more underlying problems.

5.1 Future Works

As mentioned in Section 4.5, there still exist several limitations to the current prototype. In the next development cycle, more songs should be supported. The front end should be tested on Android platform. The back end should be deployed onto a cloud service and should be running successfully online. The algorithm should be able to detect the correctness of superimposed notes. In later phases, more advanced features in the general report could also be considered, such as to evaluate the expressiveness of the performance.

Meanwhile, people from our target user group, i.e., the piano learners, should be constantly involved in the following development cycle so as to make sure that our user interface design is pleasant, and that our functions meet the users’ actual needs. This user-centric practice is crucial, as it will help keep our product development on the right track.
6. References


