A-Level Coursework

Contents

[Analysis 2](#_Toc1397740)

[Inroduction 2](#_Toc1397741)

[Identification of End Users 2](#_Toc1397742)

[User Requirements 2](#_Toc1397743)

[Acceptable Limitations 3](#_Toc1397744)

# Analysis

## Inroduction

Learning a musical instrument takes a lot time. This time is generally divided up into two parts – Time spent with a teacher, and time to practice alone. A good balance of these is key to learning quickly and effectively. At our school, when learning a musical instrument, the student gets one half hour lesson a week one-on-one or two-on-one time with a music teacher who will then set pieces of music to practice for the next lesson. With this system, generally what happens is the student is intruded to a new piece of music in the music lesson which will then be practiced alone over the week until the next lesson, where the teacher will see how well they are coming along, and either introduce a new piece of music to the student if they had been able to play it well, or, spend the lesson listening to the student playing and giving feedback on how to improve. Unfortunately, due to the short lessons and the long wait between them, the student may be sent away to practice a piece of music over the week that they have not been able to understand in the lesson. This means that during the week, the student is faced with mastering a piece of music that they may never played all the way through and so do not how should sound or what they may be doing wrong and so by the next lesson, the student has not been able to make much progress learning the piece of music.

## Identification of End Users

The end users of the solution will be the students who are learning an instrument as well as the music teachers who are teaching the student.

After having a discussion with one of the music teachers, I have learnt that often when a student is practicing a piece of music alone, they may be learning the music incorrectly by repeatedly practicing something one way because it’s the way they think it should sound even if it is incorrect. Often students may just need a small bit of guidance to tell them that they are playing it incorrectly to start improving.

## Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | ID | Definition | Reason | Type |
| Displaying Music | R1.1 | Notes the student should play must be shown on a musical staff | This is done so that the students can get used to learning to read sheet music | Must |
|  | R1.2 | Notes should be displayed as they would be on a standard musical score | This is done so that the students can get used to learning to read sheet music | Must |
|  | R1.3 | User’s should be able to choose whether the fingering for each note should be displayed underneath each note | This is done to help students learn/associate what notes require what hand positions | Must |
|  | R1.4 | The musical stave must show the time signature | This is required to let the student know how the piece should be played | Must |
| Exercises | R2.1 | Exercises should be stored in external files | This is done to allow for new exercises to be added to the program to accommodate many different students at many different ability levels | Must |
|  | R2.2 | Students should be able to import or read exercises given to them or downloaded from other sources | This is done to allow students to practice many different musical pieces at varying difficulty levels |  |
|  | R2.3 | Students must be offered a selection of pre-defined exercises to choose | This is done to allow students to decide what pieces that they wish to practice at home | Must |
|  | R2.4 | The student must have the option to ‘Play / Rewind / Cancel / Replay’ the exercises | So students can re-practice or replay parts they miss | Must |
| Feedback | R3.1 | Students must be displayed a piece of music to play along to that they will be tested on | This will be the music they will play and then be analysed | Must |
|  | R3.2 | Recorded music must be digitally analysed to assess the accuracy of the played music against the exercise | This will allow the program to give feedback to the student | Must |
|  | R3.3 | Feedback should provide students with an assessment of which notes were of the correct pitch | This will highlight sections of the piece that the student needs to work on | Must |
|  | R3.4 | Feedback should provide students with an assessment which notes were of the correct time | This will highlight sections of the piece that the student needs to work on | Must |
| Instrument Database | R4.1 | Specific instrument data for each supported instrument should be recorded in a database or be readable from a file | This will allow the program to be expaned to include many different instruments | Should |

## Acceptable Limitations

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | ID | Definition | Acceptable limitation |
| Playing instructions | R1.1 | Notes the student should play must be shown on a musical stave | Musical representation may not contain all features of actual music. Eg, no “Tied” notes or Dynamics. This is deemed acceptable because this is aimed at students who are just starting to learn so advanced features wouldn’t need to be included. |
|  | R1.2 | Notes should be displayed as they would be on a standard musical score | Some aspects may not be rendered 100% correctly. Eg, A bar may have 3 and half beats in a bar not 4. This is acceptable because it does not render the music unplayable or unreadable. |
| Exercises | R2.1 | Notes must be read from pre-defined MIDI files not pre-programmed | Not every format of MIDI File will be compatible with the system. |
|  | R2.2 | Students should be able to import or read exercises given to them or downloaded from other sources. (So students can play different pieces) | It may not be possible for users to write their own MIDI files with their own MIDI creation programs due to the differing MIDI formats available. |
|  | R2.3 | Students must be offered a selection of pre-defined exercises to choose (So students can practice an appropriate piece) | A few pieces would be sufficient to illustrate the program |
| Feedback | R3.2 | Recorded music must be digitally analysed to assess the accuracy of the played music against the exercise | The analysis does not have to happen live, and instead may be given at the end once the user has finished playing |
|  | R3.3 | Feedback should provide students with an assessment of how many of the notes were of the correct pitch | The program may not be able to tell the user how to improve directly but would be able to highlight areas of difficulty to the user and show what should’ve been played vs what was actually played. |
|  | R3.4 | Feedback should provide students with an assessment of how many of the notes were of the correct time | The program may not be able to tell the user how to improve directly but would be able to highlight areas of difficulty to the user and show what should’ve been played vs what was actually played. |
| Instrument Database | R4.1 | Specific instrument data for each supported instrument should be recorded in a database or be readable from a file (So students can learn different instruments) | For instrument data to be recorded as part of configuration of the system rather than via a user-friendly process allowing the user to configure the characteristics of an instrument |

# Documented Design

The proposed solution will consist of 2 main features:

* The practice section - The student is shown the sheet music for the piece along with a several learning aids to allow them to practice learning and getting to grips with the piece of music
* The testing section – The student plays along to a piece of music and the program will listen to the audio and show a comparison of what they played and what they should’ve played as per the exercise file

To store the exercise data, it was decided to use a MIDI file since this would allow the files to be created and sent to others easily and by sticking to a known standard, compatibility with future versions could be ensured.

## Table of Features

|  |  |  |  |
| --- | --- | --- | --- |
| Solution | ID | Definition | Requirements fulfilled |
| A database of songs and instruments | S1 | The program should use a small database to store the information about the MIDI exercises that the student will be able to select. This will include: a link to the file on the disk; the difficulty of the exercise and the instrument used for the piece along with the instrument data itself. This will allow the student to select which song that they would like to practice for their instrument at the difficulty that they want to play. The database will need to store all the information that will be instrument specific, Eg: the range of notes it can play; the key the instrument is in; if the instrument transposes the music and the fingering for each of the notes. | @@ Dad – Please can you match up the requirement IDs with this column here please? |
| A method to load in a MIDI file to a format that can be used | S2 | The program will need to open a file that is referenced in the database and load it into the program memory in a format that will be easy and assessable for all the other parts of the program to use |  |
| A class to store the information about the currently loaded MIDI file | S3.1 | The program will need a class that will store all the notes that were stored in the MIDI file along with any other information such as the time signature, tempo and so on. |  |
| The same class to generate images to display what notes should be played | S3.2 | The class will contain methods that would allow musical scored to be generated that could be displayed to the user |  |
| The same class to generate the image of what notes the user played against what they should’ve played | S3.3 | The class will contain methods that would generate an image showing the notes played by the user and the notes and the notes that should’ve been played. These images can then be shown to the user side by side to show the user where they can improve and where they went wrong |  |
| A method to process a recording of what the user was playing to generate a list of when the user was or wasn’t playing | S4.1 | This method will generate a list that will store when the user was actually playing a note or not based on weather it reached a certain threshold. This will then be used in S3.3 to generate the image |  |
| A method to process a recording and produce a list of what pitches that the user was playing | S4.2 | This method will generate a list of pitches that the user was playing at a certain point. This will then be used by S3.3 to generate the image that will be compared to what they user should’ve played |  |
| A method to move a cursor on the screen to show what point the user is playing | S5 | This method will display a cursor over-top the currently displayed music to show at what point the user should be playing. |  |
|  |  |  |  |
|  |  |  |  |

## Feature Deveopment

### S1 - database of songs and instruments

The database will store all the possible information that the program will need. This includes the location of all the exercises that can be played and all the information about the properties of the notes that each instrument can play.

2 tables will be used, one to store all the information about the exercises, like the name, the file path, the difficulty and so on, and the other will store all the information about the notes, this will store the instrument that plays it, at what position it should be drawn on the staff

These tables show the names of each of the fields in each of the tables and show what will be stored in each table

Table 1 - Exercises:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | DataType | Description | Explanation |
| ExerciseID | small | (Primary Key) Stores a unique ID to identify each exercise by. | Int is used since this will be dealing with natural numbers only and it’s unlikely that more than 32,767 will be needed |
| FilePath | varChar(256) | Stores the file path to the exercsise MIDI file | Var char is used because file paths can be different lengths and it would be a waste to dedticate unused storage space to it. Most file paths are shorter than 256 charaters so it would not be neccersary to have anything bigger than that. |
| FileName | varChar(50) | Stores the Name of the exercise that will be displayed to the user | Var char is used for the same reasons as above. The max length is 50 because it is unlickely that any song names would be longer than that |
| Difficulty | tiny | (Allows null) Stores the difficulty of the piece which allows the user to know what level the piece is at | The difficulty is levelled at the music grades and so will never go beyond 8, so a tiny int was used to save space  This can be null because it is not vital to the functioning of the program |
| Instrument | varChar(15) | Stores the instrument that the exercise was written for. Different instruments can play different notes so it is necessary to distinguish between exercises so that the user knows what instrument it is designed for | Var char is used for the same reason as above varchars.  Most common instrument names are shorter than 15 chars long, so will not need to store anything more than that. |
| FileExsists | bit | (Allow null) Stores whether the file was found at the specified file path. If it is, it is written as a “1” else it is a “0” | This only needs to store true or false so a bit is used to save space.  This can be null because when a new file is added, the program will have to check it is there before it can say it is there or not and so will allow null |

DDL:

CREATE TABLE [dbo].[Exercises] (

[ExerciseID] INT NOT NULL,

[FilePath] VARCHAR (256) NOT NULL,

[FileName] VARCHAR (50) NOT NULL,

[Difficulty] SMALLINT NULL,

[Instrument] VARCHAR (15) NOT NULL,

[FileExists] BIT NULL,

PRIMARY KEY CLUSTERED ([ExerciseID] ASC)

);

Table 2 – Notes:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | DataType | Description | Explanation |
| Note | tinyInt | (Primary Key) Stores the MIDI note number | This stores the MIDI note number of the particular note |
| Instrument | varChar(15) | (Primary Key) Stores the instrument that the note is for | Varchar is used so as to not take up too much space  Most common instrument names are shorter than 15 chars long, so will not need to store anything more than that. |
| FingeringDrawing | Image | Stores the image of the fingering diagram that will be show along with the note if so desired | It was decided to store the image directly in the database rather than storing a file path to it due to the small size of the images and to avoid having to import each image from an outside source to the program when required. |
| BinNum | smallnt | Stores the “Bin” that the note can be found in after the FFT. (baised on the pitch) | A small int is used because the FFT buffer is 4096 bins long so it would never need to be bigger than that |
| StaveLocation | Tinyint | Stores the position the note should be displayed on the stave with 0 being middle C | Since there are 15 useable position on the trebble clefs staff, it would never need to be bigger than that. So A tiny in is used to save space. |

DDL:

CREATE TABLE [dbo].[Notes] (

[Note] TINYINT NOT NULL,

[Instrument] VARCHAR (15) NOT NULL,

[FingeringDrawing] IMAGE NULL,

[BinNum] SMALLINT NOT NULL,

[StaveLocation] TINYINT NOT NULL,

PRIMARY KEY CLUSTERED ([Note] ASC, [Instrument] ASC)

);

### S2 – Method to load MIDI file in

//Explanation of midi file format

#### Midi File overview

MIDI files are split into “chunks”. Each chunk consists of 3 things:

|  |  |  |
| --- | --- | --- |
| Chunk type | Length of chunk | Chunk data |
| 4 bytes long | 4 bytes long | *Length* bytes long |

There are 2 types of chunks, head chunks (with a chunk type “MThd” in ASCII) and track chunks (“MTrk” in ASCII”).

A MIDI file always consists of 1 head chunk followed by 1 or more Track Chunks dependant on the format of the file.

0A MIDI file can be 1 of 3 formats. Each MIDI file consists of 1 “Head chunk” which stores 3 peies of information, the format, the number of other chunks in the file, and the “devision” of the piece (The number of ticks per beat). The three formats are as followed:

1. In this format, there are only 2 chunks. The Head chunk, followed by a single track chunk. This track chunk stores all of the MIDI events to be read sequencially.
2. In this format, there is a head chunk and then 2 track

//Format that will need to be converted to

//How to get to that format with all the statements and so on with psudo code and so on

## Structure design