Harmonica AMT

Student name: Liam Riche

Student number: B00789989

Course: Computing Systems PT

Date: 2022/2023

Supervisor: Glenn Hawe

Contents

[Abstract 3](#_Toc131092236)

[Acknowledgements 3](#_Toc131092237)

[Listing of Figures 3](#_Toc131092238)

[Listing of Tables 3](#_Toc131092239)

[Listing of Code Listings 3](#_Toc131092240)

[Introduction 3](#_Toc131092241)

[Background 3](#_Toc131092242)

[Project Aims 3](#_Toc131092243)

[Project Objectives 3](#_Toc131092244)

[Outline of Dissertation Structure 3](#_Toc131092245)

[Literature Review 3](#_Toc131092246)

[Existing Solutions 3](#_Toc131092247)

[Current System 3](#_Toc131092248)

[Desired Solution 3](#_Toc131092249)

[Potential Existing Solutions 3](#_Toc131092250)

[Weighing up the options 3](#_Toc131092251)

[Technical Background 3](#_Toc131092252)

[Choosing a Software Development Life-Cycle 3](#_Toc131092253)

[Considerations 3](#_Toc131092254)

[Software Development Life-Cycles 3](#_Toc131092255)

[Choosing a life-cycle model 3](#_Toc131092256)

[Choosing the tools 3](#_Toc131092257)

[Identifying the needs 3](#_Toc131092258)

[Choosing a web framework 3](#_Toc131092259)

[Choosing a database backend 3](#_Toc131092260)

[Development Environments and Other Tools 3](#_Toc131092261)

[Project Planning 3](#_Toc131092262)

[Resource Consideration 3](#_Toc131092263)

[Risk Assessment 3](#_Toc131092264)

[Data Management 4](#_Toc131092265)

[Knowledge and Skills Required 4](#_Toc131092266)

[Involving Stakeholders 4](#_Toc131092267)

[System Requirements 4](#_Toc131092268)

[Requirements Gathering 4](#_Toc131092269)

[Requirements 4](#_Toc131092270)

[Functional Requirements 4](#_Toc131092271)

[Non-Functional Requirements 4](#_Toc131092272)

[Design 4](#_Toc131092273)

[UX Design 4](#_Toc131092274)

[Design Principles 4](#_Toc131092275)

[UI Designs 4](#_Toc131092276)

[UI Implementations 4](#_Toc131092277)

[Data Modelling 4](#_Toc131092278)

[Database Design 4](#_Toc131092279)

[Data Models 4](#_Toc131092280)

[Implementation 4](#_Toc131092281)

[Architecture Overview 4](#_Toc131092282)

[Architecture 4](#_Toc131092283)

[Additional libraries used 4](#_Toc131092284)

[Code Overview 4](#_Toc131092285)

[Obstacles Encountered 4](#_Toc131092286)

[Testing & Evaluation 4](#_Toc131092287)

[Testing 4](#_Toc131092288)

[Unit Testing 4](#_Toc131092289)

[System Testing 4](#_Toc131092290)

[Manual Testing 4](#_Toc131092291)

[Compatibility Testing 4](#_Toc131092292)

[Evaluation 4](#_Toc131092293)

[Evaluation of technologies used 4](#_Toc131092294)

[Evaluation of system 4](#_Toc131092295)

[Results 4](#_Toc131092296)

[Fulfilment of project objectives 4](#_Toc131092297)

[Fulfilment of functional requirements 4](#_Toc131092298)

[Fulfilment of non-functional requirements 5](#_Toc131092299)

[Additional work completed 5](#_Toc131092300)

[Conclusion 5](#_Toc131092301)

[Reflecting on the project 5](#_Toc131092302)

[Project scope 5](#_Toc131092303)

[Products produced 5](#_Toc131092304)

[Managing the process 5](#_Toc131092305)

[Suggestions for future improvements 5](#_Toc131092306)

[References 5](#_Toc131092307)

[Appendices 5](#_Toc131092308)

[Appendix 1 – Initial Gantt Chart 5](#_Toc131092309)

[Appendix 2 – Unit tests listing 5](#_Toc131092310)

# Abstract

# Acknowledgements

# Listing of Figures

# Listing of Tables

# Listing of Code Listings

# Introduction

## Background

The harmonica is a wind instrument played by either inhaling or exhaling a breath on certain parts of the harmonicas front to produce different noises. Harmonicas can have many different tunings, with the most common being in the key of C with 20 possible notes at different pitches. The music notation for the harmonica is also an outlier to other sheet music as instead of the common stave format of 5 horizontal bars commonly seen with piano music, a harmonica music notation is much simpler, with notations often simply being the number of the note and an arrow indication wether the note is a draw or blow.

This project came into being from my interest in Automatic Music Transcription services (AMTs) that take an audio input for a specific instrument and convert it into a sheet music notation. When looking into these I found many AMTs for popular instruments like the guitar and piano, but very few for more niche instruments. I chose to base my project around the harmonica as it is niche enough that there is not an AMT already created for it, as well as the fact that the harmonica has enough unique aspects to it that a tailored solution would offer more to a user compared to a general use AMT

The intended user base for this would be people who play the harmonica and either want to:

1. Find the notes to a song they are listening to
2. Create sheet music for a song they have come up with to share with others

### Project Aims

My main aim for this project is to create an automatic music transcribing program that will take a user input from a user via uploading or recording a music file containing a harmonica piece and gather information about the recording, including:

* tempo (speed at which music is played),
* note length (how long a note is held / a silence lasts)
* pitch of note (which note is being played)
* key (the tuning of the instrument)
* blow direction (is the musician sucking or blowing into the harmonica)

the AMT will then piece these together with a renderer to generate sheet music for the sound file that was uploaded and offer the user a method to view or download the finished product

The software would be created using python with some dataset manipulation to extract important information from the sound file. This python program would be packaged into a website or application to offer a more user friendly method of interacting with the AMT rather than just having it run off the console.

## Project Objectives

At the beginning of this project I set myself a few key objectives to aim to complete during the course of the alotted time:

1. find and understand key requirements that the stakeholders of the project may have
2. generate a set of functional and non functional requirements for the finished program
3. identify and choose the correct software to fit the requirements of the project, as well as choose an effective software development cycle for the projects duration
4. perform a risk assessment of issues that could arise during the development of the project and come up with countermeasures to ensure mitigation of risks
5. research what software would be most effective for creating a final product that would fulfil the shareholders criteria
6. design and use a gantt chart to track and make effective use of my time when working on the project
7. create a product that allows a user to create and download a sheet music version of whatever tune was played into it
8. perform testing with potential users of the product and create a list of feedback and areas of improvement for the future
9. produce a readme for the project that would both teach the user how to use the product as well as how to set up a new instance of the product with ease

## Outline of Dissertation Structure

Chapter 2 – Literature Review & Technical Background

This section will include information detailing the basics of the project and the research into any prior existing products that fulfil similar roles in the market

Chapter 3 – Requirements Gathering & Analysis

This section will cover the gathering, ordering and verification of requirements for the project, as well as the time allotment for the different steps of the project and creating a list of user requirements

Chapter 4 – Project Planning & Preparation ---

This includes details regarding project management tools, version control, risk assessment, chosen software development lifecycle methodology, Gantt chart as well as an overview of the hardware and software utilised.

Chapter 5 – Design

This includes details regarding use case diagrams, screen design phases, colour palettes, accessibility and a discussion regarding the ER Diagram.

Chapter 6 – Implementation & Testing

This includes details regarding the implementation of user stories into features within the application. This highlights the code used to achieve each user story, using unit tests to confirm that user stories are fully met and providing screenshots to provide evidence of this.

Chapter 7 – Evaluation

# Literature Review

## Project foundations

In 2022 Spotify released an open source python module called basic pitch, which allows for a multitude of types of audio files to be passed through it, and using a neural network will output a midi file and an optional csv file with the predicted pitches that were played in said audio file as well as when they were played.

As well as this I had contact with multiple shareholders that played the harmonica and were interested in an application to encode songs they played on a harmonica into a sheet music version of the audio file.

## Existing Solutions

While searching for previously existing harmonica transcribers, I found many systems that offer plain audio to midi and sheet music transcribers (basic pitch is an example of such), however none of these would work correctly for a harmonica, as each note on a harmonica (bar the lowest and highest notes) are a mixture of different notes played at the same time with varying pitches, which would not return an easily recognisable notation format that the harmonica uses.

\*screenshots of audio to sheet notation programs (basic pitch, songify, ect.)\*

## Desired Solution

With the system I aim to create, the returned sheet music will be in the harmonicas sheet music format of a number corresponding to the harmonicas note played and an arrow or other icon / text to indicate the blow or draw direction (this could be done through an up or down arrow, or a negative operator symbol to indicate a draw note and lack of said symbol to indicate a blow note)

\*insert screenshots of harmonica notation and possible draw / blow notation

## Weighing up the options

Upon further researching into existing solutions for harmonica transcribers and general transcribers currently available, the best course of action would be to create an application that uses the existing basic pitch module to process an audio file passed into the applications frontend. The results from this module will then be passed through to the backend for processing that will further fine tune the results from this basic pitch into a json object of harmonica notes. This json object will then be passed back to the applications frontend and rendered in a format recognisable to the user. The rendered notes will then be downloadable for future reference of the user in their own time so that they do not have to use the application to view the generated notes.

# Technical Background

## Choosing a Software Development Life-Cycle

* Agile was chosen for use on the project for its benefits below:
  + Easy and swift change adaption in case of change or oversight in original plans
  + Transparency in current state of project compared to projected progress
  + Allows for continual testing of overall product more often than waterfall model
  + Better stakeholder engagement, updates on current project states more often

### Considerations

### Software Development Life-Cycles

### Choosing a life-cycle model

## Choosing the tools

### Identifying the needs

The proposed system for the harmonica transcriber would be a web app running either locally or hosted online, consisting of a frontend for the user to interact with (uploading files, displaying generated sheet music, downloading sheet music, etc) and a backend that will contain all functions needed to facilitate actions taken on the front end and to process any data.

The frameworks and technologies used will be chosen based on the requirements of the identified needs

### Choosing a web framework

As the program would need to perform data manipulation on received files / data, a programming language that can easily accomplish this must be chosen. Python was the main language that was used to fulfil this need, as with pandas and numpy modules imported to the python file, data manipulation is much easier than when using other languages like Java or .Net platform

For the front end of the website, a flask app was used for the web framework, as it is what Liam was most comfortable working with as he had prior experience creating and using a python flask webapp. This would allow development of the web application to be faster as Liam would not need to learn many new web frameworks, as well as flasks easy to get running design, taking a short amount of time to go from concept to working example.

### Choosing a database backend

The data storage aspect of the project is limited, as stated in the desired solution section It is mentioned that Liam does not want data / audio files to be permanently stored on the webapp, as it may cause issues with copyright if the music passed through and stored is not under fair use, therefore Liam planned to avoid the use of databases by having a simple temporary data folder in the website directory that holds the uploaded audio file, as well as the csv file of notes generated. Said temporary files would then be deleted at the end of the processing process, with the temporary file being empty 99.99% of the time the webapp is running.

### Development Environments and Other Tools

Since this project will use python, PyCharm community edition will be used as the IDE of choice. Liam has used this IDE before, allowing for him to easily set up a new working environment for the project and work more efficiently due to prior experience with the IDE. PyCharm offers syntax error highlighting for different versions of python, also allowing for integration of the project directly into the GitHub tab at the side of the PyCharm window, meaning faster versioning and backing up of the project. Liam has created a github repository to host all files of the project online, meaning he can work on the project from multiple devices, ensure all previous versions of the project can be found online in the case of the project being corrupted or inaccessible from one device, and bring in other developers for further iterations of the project after being used for the university dissertation project.

# Project Planning

## Resource Consideration

As this project is only going to deliver a simple web application, the resources needed are quite simple, boiling down to the following:

* A Computer to be used to develop / host the website during development and presentation. Liam owns a personal computer that he will use for this consideration
* Software to program the website. Liam used pycharm community edition, a free IDE for python development, as well as a free version of a pythonanywhere.com webhost to test online functionality of code when not running on a higher spec machine.
* A harmonica. Liam purchased a harmonica tuned to the key of C for this project to create test data to be used during the creation of the processing section of the webapp and to allow Liam to not have to rely on stakeholders or online hosted harmonica music clips for testing.

## Risk Assessment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Risk Event** | **Probability %** | **Impact, days** | **Score, days** | **Risk Mitigation plan** |
| 1 | Product does not work on other OS (mac) | 50 | 4 | 2 | Ensure testing of each new version of program on both windows and macOS when finished |
| 2 | Software progress loss during a stage | 20 | 7 | 1 | Keep consistently committing and pushing changes made on the project to github each time an update is made |
| 3 | User interface too complex, requiring reconfiguration | 10 | 5 | .5 | Check with stakeholders each time a new update is created |
| 4 | User data leaks requiring patch of system | 5 | 10 | 2 | Ensure all files are localised and removed internally after application is closed |
| 5 | Addition of extra features | 25 | 10 | 2.5 | Leave enough time between final handover and planned project finish date |
| 6 | Module component in program stop functioning / depreciates | 33% | 15 | 5 | Ensure all required modules are incorporated into the project and are not affected by external factors |

## Data Management

## Knowledge and Skills Required

As the development of the project hardware requirements are only a computer, a harmonica and a webserver hosting site for testing, Liam already had access to all hardware needed to complete the project and knows how to use them, meaning the only new knowledge and skills needed are related to programming.

Liam has 5+ years of html, css and javascript development for the frontend of the web application, therefore no new skills needed in that area, however when it comes to python and flask Liam has significantly less experience meaning he had to do some tutorials to refresh his skills on python, data manipulation and the flask framework. Liam would also use google and stack overflow if he encountered any issues whichever task he was currently trying to implement during development.

## Involving Stakeholders

# System Requirements

## Requirements Gathering

a software requirement is described as ‘a property that must be exhibited by something in order to solve some problem in the real world’ by The Software Engineering Body of Knowledge (SWEBOK) v3 (IEEE Computer Society, 2014). These requirements can further be split into functional and non-functional requirements.

Functional requirements refer to functions the end product would offer, or ‘features’. An example would be “developed system must accept an audio file on the frontend webpage and pass it through to the processing backend”.

Non functional requirements refer requirements like performance, security, efficiency and so on (requirements that are quantified). An example would be “data passed to the backend must be processed within 30 seconds of ‘process’ button being clicked”

## Requirements

### Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requirement** | **ID** |  |
| R1 | Application must be user friendly and easy to navigate |  |  |
| R2 | application must be able to record or upload sound clip |  |  |
| R3 | Application must return error message if file uploaded is not compatible |  |  |
| R4 | Application must return error message if file uploaded is longer than max length |  |  |
| R5 | Application must be self contained and have ability to function on its own |  |  |
| R6 | Application should return sheet notation of an uploaded clip in pdf format |  |  |
| R7 | Application should have accessible gui menu |  |  |

### Non-Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requirement** | **ID** |  |
| R1 | Application must be user friendly and easy to navigate |  |  |
| R2 | application must be able to record or upload sound clip |  |  |
| R3 | Application must return error message if file uploaded is not compatible |  |  |
| R4 | Application must return error message if file uploaded is longer than max length |  |  |
| R5 | Application must be self contained and have ability to function on its own |  |  |
| R6 | Application should return sheet notation of an uploaded clip in pdf format |  |  |
| R7 | Application should have accessible gui menu |  |  |

# Design

## UX Design

### Design Principles

### UI Designs

Shown below are some of the high-level UI designs created for the webapps pages, with the goal from most pages being minimalistic in style to not overwhelm a user, and to create a website that keeps up with the laws of UX (<https://lawsofux.com/>).

Shown below is the overall layout and user flow of the webapp, sections below that will contain more information on each page

Graphical user interface, diagram

Description automatically generated

#### Landing page

Graphical user interface, application

Description automatically generated

As the first page of the website, Liam kept the overall layout of the page simple, offering the name of the application (since changed from Harmonica Transcriber to Harmonizer), a quick description of what the webapp does, what is needed in the webapp and how it came about.

Underneath the description is a large button with the label text ‘Get started’ which will bring you to the upload page.

#### Upload page

Graphical user interface, text, application, chat or text message

Description automatically generatedLogo

Description automatically generated

On this page of the website the user will either record a new harmonica piece live or upload an existing audio file of a harmonica being played. On the left side of the screen would be settings for the web application, with potential settings listed in the figure above being written prior to exact setting requirements being generated. At the bottom of the screen are the buttons for uploading and recording the audio file to be passed through to processing backend. A new button of similar style (shown in fig.) to the upload and record buttons would be displayed after one of the displayed options has been chosen to bring user to processing page.

The description above the buttons would hold a quick description of what each of the settings do, guide to getting better outputs from recordings (ensure quiet room and no other sounds playing aside from harmonica) and description of next pages

#### Processing page

Graphical user interface, application, Teams

Description automatically generated

Processing page will simply be used to tell the user that the processing of the file is happening, as mentioned in (<https://www.nngroup.com/articles/response-times-3-important-limits/>), the limit of a user waiting for the webapp staying static is 10 seconds, therefore this page is designed to make it clear that the processing of the file is taking place, and that It may take longer than 10 seconds if the file is large.

Once file processing is complete on the backend, the page will then move onto the completion page

#### Completion page

Graphical user interface, text, application, chat or text message

Description automatically generated

The completion page will have a header stating that the file conversion has been completed successfully, with a description below it that is used to describe where the file has been saved to on the computer. Underneath the description is a button saying ‘convert another’, this button will bring you back to the starting page of the web app allowing you to start another transcription.

### UI Implementations

Listed below are the actual user interfaces created for the full journey of the webapp and to explain the differences and similarities between the planned UI designs and the final output:

\*insert frontend screenshots once finalised\*

## Data Modelling

As my project did not have a database built into it, I will instead talk about the structure of the CSV and dataframe data structures that get generated and what they will store and return. For the storing and processing of the data generated from the audio file I used pandas dataframes as they are easily manipulatable and allowed Liam to get information from rows, create new data from existing rose and modify the structure of the dataframe easily and quickly.

### Input CSV of notes data Structure

Shown below is the data structure off the CSV generated by basic pitch.

### Output JSON of processed notes

Shown below is the dataframe output of the processing file that Liam programmed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Dataframe  column names | start\_time\_s | end\_time\_s | avg\_pitch\_midi | note\_group | total\_length | direction |
| Example Data | 1.998200 | 2.137520 | 6 | 1 | 0.139320 | True |
| 2.381329 | 2.636749 | 5 | 2 | 0.255420 | False |
| Description | Predicted start time of the harmonica note played from the start of the uploaded file (in seconds) | Predicted end time of the harmonica note played from the start of the uploaded file (in seconds) | The predicted harmonica note that was played between start and end time | Used earlier in processing file to combine multiple midi notes played at the same time into one table row, each unique note played will incrementally increase note group by one (can be used for iteration) | The length of the predicted note played, generated by removing the start time of the note from the end time of the note (in seconds) | A true or false Boolean to indicate wether the note was blown or drawn.  True is blown, False is drawn. |
| Data types | Float | Float | Integer | Integer | Float | boolean |

# Implementation

## Architecture Overview

### Architecture

### Additional libraries used

## Code Overview

## Obstacles Encountered

# Testing & Evaluation

## Testing

### Unit Testing

### System Testing

### Manual Testing

### Compatibility Testing

## Evaluation

### Evaluation of technologies used

### Evaluation of system

# Results

## Fulfilment of project objectives

### Fulfilment of functional requirements

### Fulfilment of non-functional requirements

### Additional work completed

# Conclusion

## Reflecting on the project

### Project scope

### Products produced

### Managing the process

## Suggestions for future improvements

# References

# Appendices

## Appendix 1 – Initial Gantt Chart

A picture containing graphical user interface

Description automatically generated

## Appendix 2 – Unit tests listing

## https://blog.pythonanywhere.com/169/