Bilkent University

Department of Computer Engineering

Senior Design Project

T2314

Rhythmus

Detailed Design Report

21802067, Berk Baltacı, berk.baltaci@ug.bilkent.edu.tr

21802632, Cihan Can Kılıç, cihan.kilic@ug.bilkent.edu.tr

21600591, Azar Hasanaliyev,

azer.hasanaliyev@ug.bilkent.edu.tr

21802977, Emir Yavuz, emiryavuz@ug.bilkent.edu.tr

21903708, Edip Kerem Tayhan,ekeremt@gmail.com

Fazlı Can

Erhan Dolak,Tağmaç Topal

<13/03/2023>

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2

1. Introduction

As the world starts to recognize the horsepower behind A.I. and machine learning, a way to integrate it with music has been on the mind of many programmers and musicians. Musicians and music enthusiasts can use our tool Rhythmus to create melodies and variations. Rhythmus will be fed with large quantities of MIDI data of certain artists and thus offer a way to recreate variations of the input data while allowing the manipulation of certain attributes of the desired piece such as the key of the piece, the used scales, the tone of the piece, etc.

1.1 Purpose of the system

The project aims to help people who want to compose music for both professional and personal purposes. Many people get bored of listening to the same songs and wonder if there is a way to make songs just like their favorite ones. This is where we come into play. Our project makes it possible to determine patterns followed in a song and make a new song that contains similar patterns. Rhythmus can also inspire professional composers by giving them ideas about composing new music when they have a certain type of song in their minds.

As soon as the user signs up and logs in, he or she sees the home page of Rhythmus. There are two options in the home page for composing. The first one is to create a new project and the second one is to work on an existing project. When creating a project, the user needs to specify the parameters of the project such as musical instruments that are going to be used or beats per minute. All of these parameters can be changed later.

There can be several songs that the user wants to utilize in the composition of a new song. For this purpose, there is a music list in Rhythmus. The user can upload several songs for making another one. These songs are kept as long as the user desires. When the user logs out and logs in from another device, the list will still be available. The user chooses a song from the list and Rhythmus detects melodic patterns in the song along with the musical instruments played in it. Then, Rhythmus generates similar patterns and the user chooses one of them for composing his or her own song.

1.2 Design goals

1.2.1 Usability

All functionalities must be self explanatory so that everyone can intuitively figure out what is what. However, there must also be a help page where every instruction is given as clearly as possible. Our target user group includes every age. Therefore, the UI must be designed in a way that provides a convenient experience for elderly as well. For example, the texts in the app must be in a proper size. Also, the graphical components must be nice and appealing so that users do not get bored while using the app.

1.2.2 Reliability

When it comes to programming, reliability is one of the most important things to consider. Our program must be error free. It must be available 7/24 without any interruption. Nowadays, people are worried about the security of their data. So, security of users' data must be provided. Independent of the device or the operating system, the program must work properly. If the app crashes, the application must be resettable in no longer than 5 seconds.

1.2.3 Fast Response Time

Time is valuable for everyone including musicians and other people in our target user group. Therefore, the application must be working fast. Most operations such as navigating between pages, uploading or deleting songs to the music list, editing the song and setting the parameters of the project must be done almost instantly (no longer than 3 seconds). The operations that require processing data, such as generating similar music, will take maximum of 60 seconds.

1.2.4 Maintainability

Our project will be highly maintainable as we are using a layered network and a master slave model inside the logic layer. These design choices create an abstraction between classes so when we are updating, adding or removing a new functionality we don't have to work with the system all together. Instead we are going to implement the changes to the classes separately.

1.3 Definitions, acronyms, and abbreviations

MIDI: Musical Instrument Digital Interface.
A.I.: Artificial intelligence.
U.I.: User interface.
SHA-256: Secure Hash Algorithm 256 bit.
LSTM: Long short term memory.
UX: User experience.

1.4 Overview

The rest of this Detailed Design Report is organized as follows. Section 2 explains the proposed software architecture of our project. This section has subsections overview, subsystem decomposition, persistent data management and access control and security. In Section 3, the subsystem services are described. Then Section 4 discusses test cases with the expected results and outcomes. In Section 5, consideration of various factors, such as cultural factors, in engineering design is argued. Section 6 explains teamwork details of the project including each member's responsibilities.

2. Proposed software architecture

2.1 Overview

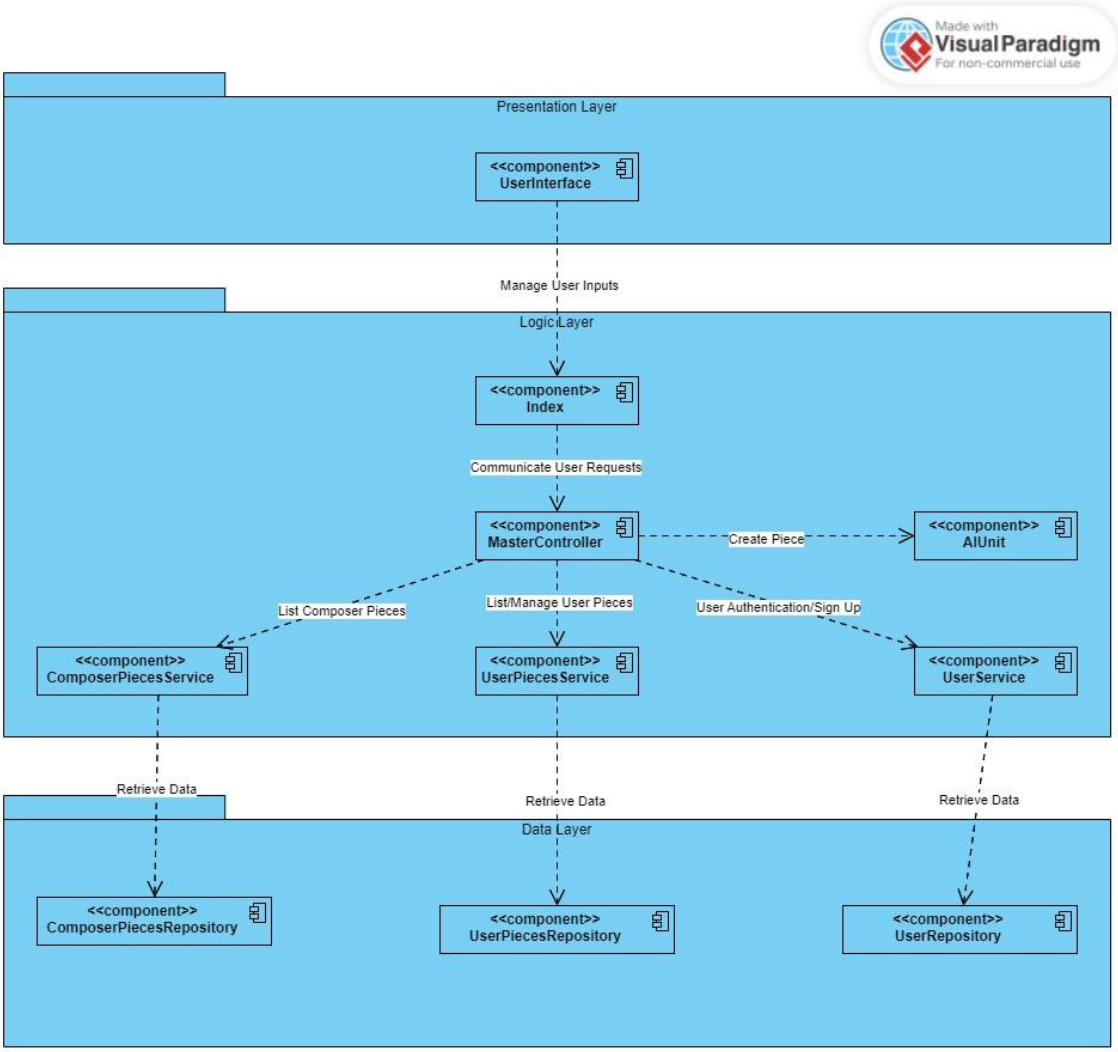
The main functionality of the system is to generate a musical piece with the help of a machine learning model. However we are aiming to keep the interface and music generation process as simple as possible. The reason for that is to make it possible for everyone to create their own A.I. generated musical piece without any knowledge on the musical theory or the machine learning processes.

Also we are aiming to implement an editing page where people can edit the pieces that the algorithm generated. This is a more advanced feature as it requires some knowledge on music theory. Also these pieces are going to be downloadable in MIDI format so that the users can download, store, and share their A.I. generated pieces.

We are using triple layer architecture for our software. These layers are presentation, logic and database. Inside the presentation layer, there are the subsystems that the users are interacting with. Then we have the logic layer which is responsible for all

the computations, data manipulations and the communication with both presentation and data layers. The logic layer has a master-slave pattern inside of it. Index is the master class and it communicates to UI and MasterController. MasterController divides the work ordered by Index and orders the relevant slave classes. Finally there is the data layer which is responsible for retrieving raw data from databases as requested from the logic layer. The presentation layer and data layer only communicate with the logic layer.

2.2 Subsystem decomposition



We used the triple layer architecture as it was the most compatible pattern for our project and our development strategy. The three layers are presentation, logic and data

layer. The presentation layer deals with user interactions and inputs. The logic layer is where all the data manipulations and modifications are done. The data layer is where raw data is communicated between the application and databases. Also we used a master slave pattern inside the logic layer because we wanted a better structure for the logic operations in order to make maintenance, updates and group work easier as it creates abstraction. The master slave implementation will be further explained in the following part of this section.

UserInterface: This is the subsystem which handles all user interactions with the application.

Index: This subsystem is the master of the logic layer. It communicates directly with the representation layer and MasterController class. This class works as a middleman between the logical operations and user interactions. It sends requests to the MasterController and receives data. Then this data is sent to the representation layer.

MasterController: MasterController works as a distribution center. It receives requests from Index and distributes them to the relative services. MasterController also works as a validator for the data. Before returning data to Index all data is checked here.

AIUnit: This subsystem receives instructions from MasterController and generates a musical piece with respect to them. After the models are trained, the node weights will be stored in a hdf5 file. Then these weights will be used to generate notes from a chosen music.

ComposerPiecesService: This subsystem receives requests from the MasterController and sends requests to the data layer. After the data is received, all the manipulations and modifications will be done here. So this can be referred to as a worker subsystem with all other services. Then finally the data is returned back to the MasterController for validations. It will be used for listing and choosing composer pieces while generating a musical piece.

UserPiecesService: This subsystem is also a worker subsystem and will be used to receive information about the user generated pieces.

UserService: This subsystem will be used for user authentication and sign-up process. This is again a worker subsystem.

ComposerPiecesRepository: Repositories are data layer subsystems so they will be receiving requests from the corresponding service then communicate to the corresponding database in order to return raw data to the services. ComposerPiecesRepository will be used to retrieve information about composer pieces.

UserPiecesRepository: This subsystem is a data layer subsystem as well and it will be used to receive data from or write data to the corresponding database.

UserRepository: This subsystem will be used to receive data from or write data to the user database where authentication information is kept.

2.3 Persistent data management

We are going to have three relational databases for information storage, a large cloud service for storing the notes of the songs. Only the relational database is crucial in terms of privacy so data security is not important for other databases as they don't contain sensitive or private data.

There are going to be two databases that store information about composer pieces and user generated pieces. These are non-private data so security isn't the first priority here and we are planning on using a firebase relational database.

The database that's going to store user information has to be secure in order to protect private user data. We are planning on using a relational firebase database too and store password information with a hash algorithm like SHA-256.

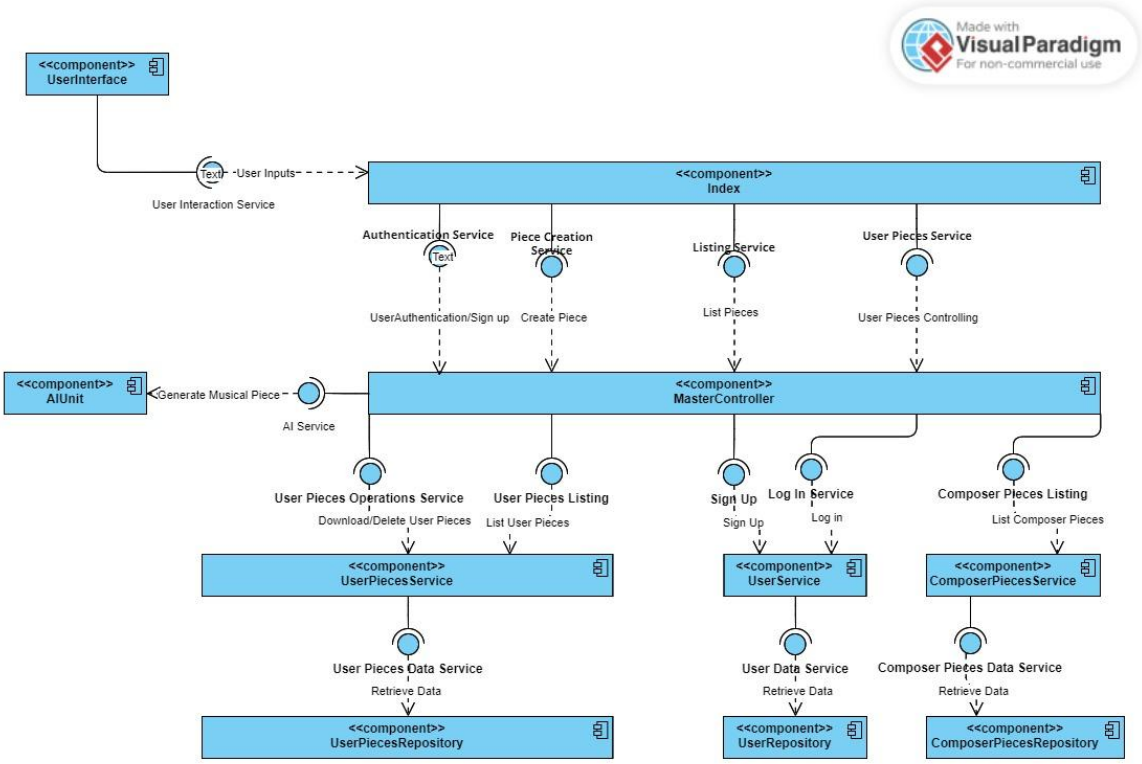
We haven't yet decided on how to store the songs. There are two options as follows; storing them in MIDI file format which would take up a large amount of storage space and storing only the notes. We are going to use a cloud service for large storage.

2.4 Access control and security

There are two types of users. They are signed-up users and visitors. Signed-up users will be able to create pieces; save, download, delete or edit pieces. However visitors won't have access to any of these functionalities and they can only listen to the demo pieces that we are going to create and upload to the entry page of the application.

Users have to sign-up and log-in to use the application. Their authentication information will be stored in secure databases to prevent the leak of sensitive information. Also a hash function will be used to encrypt passwords like SHA-256.

3. Subsystem services



User Interactions Service: This server handles all user interactions through the index subsystem.

Authentication Service: This server handles user authentications. Aside from login and signup, authentication service also plays a role in user detection while listing the correct user pieces for all users.

Piece Creation Process: In this service, Index subsystem is ordering piece creation with parameters taken from the user to the Master Controller Subsystem which will then divide the work to the related slave subsystems.

Listing Service: Listing Service handles all listing processes by directing them to the related subsystems.

User Pieces Service: This service handles the user pieces operations communications between the user and the logic layer.

AI Service: AI Service handles the main functionality of the application which is generating music through machine learning algorithms with previously constructed node weights. It also handles the inputs given by the Index subsystem which will be used in the music generation process.

User Pieces Operations Service: User Pieces Service handles all the processes on user pieces. These processes are downloading, saving, deleting and editing. All these functionalities will be handled by this process.

User Pieces Listing: This service handles the listing of user pieces.

Sign Up: This service handles saving and validating new user information.

Log In Service: This service validates the information entered by the user by comparing them to the data taken from the User Repository subsystem.

Composer Pieces Listing: This service handles the listing process of the composer pieces while the user is selecting a composer piece before creating a new piece.

User Pieces Data Service: This service gets raw data from User Pieces Database and sends them to User Pieces Service.

User Data Service: This service gets raw data from User Database and sends them to User Service.

Composer Pieces Data Service: This service gets raw data from Composer Pieces Database and sends them to Composer Pieces Service.

4. Test Cases

- Test ID: 1
- Test Type/Category: Functional
- Summary/Title/Objective: Test Case For The Log in
- Procedure of testing steps
    - Check the email is written.

    - Check the password is written.
    - Check the email exists in the database.
    - Check the written password is matched with the written email password
- Expected results/Outcome
    - Email is written or giving an error.
    - Password is written or giving an error.
    - Email exists in the database or gives an error.
    - Password is correct or giving an error.
- Priority/Severity: Critical

- Test ID: 2
- Test Type/Category: Functional
- Summary/Title/Objective: Test Case For The Sign Up
- Procedure of testing steps
    - Check the email is written.
    - Check the password is written.
    - Check the second password is matched with the first password
    - Check the email does not exist in the database.
- Expected results/Outcome
    - Email is written or giving error.
    - Password is written or giving error.
    - Password is matched or giving an error.
    - Email is not taken or giving an error.
- Priority/Severity: Critical

- Test ID: 3
- Test Type/Category: Functional
- Summary/Title/Objective: Test Case For The Sign Up Link
- Procedure of testing steps
    - Check the link direct to the sign-up page

- Expected results/Outcome
    - Link directs to the correct page or gives an error.
- Priority/Severity: Critical
- Test ID: 4
- Test Type/Category: Functional
- Summary/Title/Objective: Test Case For The Log-In Link
- Procedure of testing steps
    - Check the link direct to the log-in page
- Expected results/Outcome
    - Link directs to the correct page or gives an error.
- Priority/Severity: Critical

- Test ID: 5
- Test Type/Category: Functional
- Summary/Title/Objective: Listen to Demos
- Procedure of testing steps
    - Check the link direct to the demos.
    - Check that all demos are shown.
- Expected results/Outcome
    - The link does not go to the demos page, giving an error.
    - Demos are not showed gives an error.
- Priority/Severity: Critical

- Test ID: 6
- Test Type/Category: Installation
- Summary/Title/Objective: Test Case For Demo Pieces Created by Rythmus
- Procedure of testing steps
    - Check the demos are installed.
    - Check the demos are correct.
    - Check if the demos can listened.
- Expected results/Outcome

    - Demos are installed correctly or give an error.
    - Demos are correct or give names an error.
    - Demos are listened to or give an error.
- Priority/Severity: Critical

- Test ID: 7
- Test Type/Category: Installation
- Summary/Title/Objective: Test Case For My Pieces
- Procedure of testing steps
    - Check the pieces are shown.
    - Check the pieces have the composer and parameters correct.
- Expected results/Outcome
    - Pieces are installed correctly or give an error.
    - Pieces are correct or give an error.
- Priority/Severity: Critical

- Test ID: 8
- Test Type/Category: Installation
- Summary/Title/Objective: Test Case For My Pieces Download
- Procedure of testing steps
    - Check the pieces are installed.
    - Check the pieces are correct.
    - Check if the pieces can listen.
- Expected results/Outcome
    - Pieces are installed correctly or give an error.
    - Pieces are correct, giving names an error.
    - Pieces are not listened to, giving an error.
- Priority/Severity: Critical

- Test ID: 9

- Test Type/Category: Function
- Summary/Title/Objective: Test Case For Choosing Composer
- Procedure of testing steps
    - Check the composers are chosen.
- Expected results/Outcome
    - Composers are chosen or giving an error.
- Priority/Severity: Critical

- Test ID: 11
- Test Type/Category: Function
- Summary/Title/Objective: Test Case For Choosing Music
- Procedure of testing steps
    - Check the composers are the same for the music.
    - Check the list of pieces of music are correct
- Expected results/Outcome
    - Pieces of music are from the same composer or giving an error.
    - Pieces of the music list are correct or giving an error.
- Priority/Severity: Critical

- Test ID: 12
- Test Type/Category: Function
- Summary/Title/Objective: Test Case For Layers
- Procedure of testing steps
    - Check the layers are created correctly
- Expected results/Outcome
    - Layers are correct or giving an error.
- Priority/Severity: Critical

- Test ID: 13
- Test Type/Category: Function

- Summary/Title/Objective: Test Case For Saving Layer
- Procedure of testing steps
    - Check the save the new music piece correct
- Expected results/Outcome
    - Pieces of the music are correct or giving an error.
- Priority/Severity: Critical

- Test ID: 14
- Test Type/Category: Installation
- Summary/Title/Objective: Test Case For Download Music Layer
- Procedure of testing steps
    - Check the pieces of music are correctly downloaded
- Expected results/Outcome
    - The music is downloaded or giving an error.
- Priority/Severity: Critical

- Test ID: 15
- Test Type/Category: Security
- Summary/Title/Objective: Test Case For Password
- Procedure of testing steps
    - Check the password is not short
- Expected results/Outcome
    - The password is long enough or giving an error.
- Priority/Severity: Major

- Test ID: 16
- Test Type/Category: Usability
- Summary/Title/Objective: Test Case For Composers
- Procedure of testing steps
    - Check the composers are found easily

- Expected results/Outcome
    - The composers are found quickly.
- Priority/Severity: Minor

- Test ID: 17
- Test Type/Category: Usability
- Summary/Title/Objective: Test Case For Musics
- Procedure of testing steps
    - Check the list of music that is found quickly.
    - Check the list of music that is chosen easily
- Expected results/Outcome
    - The music lists are found quickly.
    - The music is chosen quickly
- Priority/Severity: Minor

- Test ID: 18
- Test Type/Category: Usability
- Summary/Title/Objective: Test Case For Layers
- Procedure of testing steps
    - Check the layers are chosen quickly
- Expected results/Outcome
    - The layers are found.
- Priority/Severity: Minor

- Test ID: 19
- Test Type/Category: Performance
- Summary/Title/Objective: Test Case Music pieces
- Procedure of testing steps
    - Check the music models are downloaded fast
- Expected results/Outcome

    - The music models are downloaded.
- Priority/Severity: Minor

- Test ID: 20
- Test Type/Category: Function
- Summary/Title/Objective: Test Case Delete My Pieces
- Procedure of testing steps
    - Check the music pieces are deleted correctly
- Expected results/Outcome
    - The music pieces are deleted.
- Priority/Severity: Major

- Test ID: 21
- Test Type/Category: Compatibility
- Summary/Title/Objective: Test Case For Compatible
- Procedure of testing steps
    - Check the application is working when it opens from the browser
- Expected results/Outcome
    - The program is worked fine in every web browser
- Priority/Severity: Major

- Test ID: 22
- Test Type/Category:Performance
- Summary/Title/Objective:Test Case Creating New Rhythms
- Procedure of testing steps
    - Check if the rhythms are created fast enough
- Expected results/Outcome
    - The new rhythms are created in an optimal time.

- Priority/Severity: Minor

- Test ID: 23
- Test Type/Category: Security
- Summary/Title/Objective: Test Case For Usernames
- Procedure of testing steps
    - Check if the username is already taken or not
- Expected results/Outcome
    - Error given if the username is taken to avoid duplicates
- Priority/Severity: Major

- Test ID: 24
- Test Type/Category: Non-Functional
- Summary/Title/Objective: Test Case for Heavy Load
- Procedure of testing steps
    - Check the performance levels after multiple users trying to login at the same time or try to use the functions in the app.
- Expected results/Outcome
    - The application does not crash under the heavy load.
- Priority/Severity: Critical

- Test ID: 25
- Test Type/Category: Usability
- Summary/Title/Objective: Test Case For Sign Up/Log In
- Procedure of testing steps
    - Check if the user can sign up easily.

- Check if the user can log in easily.

- Expected results/Outcome
  - The Sign Up and Log In process is done quickly.

- Priority/Severity: Minor

- Test ID: 26
- Test Type/Category: Compatibility
- Summary/Title/Objective: Test Case For Firebase
- Procedure of testing steps
  - Check if the database works optimally and stores the data properly.

- Expected results/Outcome
  - Firebase works optimally and stores data properly.

- Priority/Severity: Critical

- Test ID: 27
- Test Type/Category: Compatibility
- Summary/Title/Objective: Test Case For Keras
- Procedure of testing steps
  - Check if the library works optimally.

- Expected results/Outcome
  - Keras works optimally.

- Priority/Severity: Critical

- Test ID: 28
- Test Type/Category: Compatibility
- Summary/Title/Objective: Test Case For Cloud Services
- Procedure of testing steps
    - Check if the cloud services works optimally.
- Expected results/Outcome
    - Cloud services works optimally.
- Priority/Severity: Critical

- Test ID: 29
- Test Type/Category: Documentation
- Summary/Title/Objective: Test Case For Adaptation of Documentation
- Procedure of testing steps
    - Check if the application has its features mentioned in documentation.
- Expected results/Outcome
    - The documentation is integrated into the application.
- Priority/Severity: Major

- Test ID: 30
- Test Type/Category: Installation
- Summary/Title/Objective: Test Case For Uploading New Pieces
- Procedure of testing steps
    - Check if the pieces are uploaded.
- Expected results/Outcome
    - Pieces are uploaded to the system without any error.
- Priority/Severity: Critical

- Test ID: 31
- Test Type/Category: Security
- Summary/Title/Objective: Test Case For Security of User Data

- Procedure of testing steps
    - Check if the user data is accessed easily.
- Expected results/Outcome
    - The user data cannot be accessed by another user.
- Priority/Severity: Critical

- Test ID: 32
- Test Type/Category: Security
- Summary/Title/Objective: Test Case For Timeout of An User
- Procedure of testing steps
    - Check if the user automatically logs out after a certain time of inactivity or closing the browser.
- Expected results/Outcome
    - User logs out after a certain time of inactivity.
- Priority/Severity: Major

- Test ID: 33
- Test Type/Category: Performance
- Summary/Title/Objective: Test Case for Application Start
- Procedure of testing steps
    - Check if the application loads up fast.
- Expected results/Outcome
    - The application loads fast enough for users.
- Priority/Severity: Minor

- Test ID: 34
- Test Type/Category: Performance
- Summary/Title/Objective: Test Case for Smoothness of Log In
- Procedure of testing steps
    - Check if the login process time is optimally short.
- Expected results/Outcome
    - The login time is optimal for users.
- Priority/Severity: Minor

- Test ID: 35
- Test Type/Category:Performance
- Summary/Title/Objective: Test Case for Smoothness of Sign Up
- Procedure of testing steps
    - Check if the signup process time is optimally short.
- Expected results/Outcome
    - The signup time is optimal for users.
- Priority/Severity: Minor

- Test ID: 36
- Test Type/Category: Performance
- Summary/Title/Objective: Test Case for Smoothness of Uploading New Pieces
- Procedure of testing steps
    - Check if the uploading new pieces to the system is smooth for users.

- Expected results/Outcome
    - The uploading of new pieces to the system is smooth for users.
- Priority/Severity: Minor

- Test ID: 37
- Test Type/Category: Performance
- Summary/Title/Objective: Test Case for Smoothness of Uploading New Pieces
- Procedure of testing steps
    - Check if the uploading new pieces to the system is smooth for users.
- Expected results/Outcome
    - The uploading of new pieces to the system is smooth for users.
- Priority/Severity: Minor

- Test ID: 38
- Test Type/Category: Performance
- Summary/Title/Objective: Test Case for Smoothness of Listening Pieces
- Procedure of testing steps
    - Check if the process of listening to new pieces to the system is smooth for users.
- Expected results/Outcome
    - The process of listening to new pieces to the system is smooth for users.
- Priority/Severity: Minor

- Test ID: 39

- Test Type/Category: Performance
- Summary/Title/Objective: Test Case for Smoothness of the Application in different web platforms
- Procedure of testing steps
    - Check if the application is smooth enough for the users in different web platforms.
- Expected results/Outcome
    - The application is smooth enough for the users in different web platforms.
- Priority/Severity: Minor

- Test ID: 40
- Test Type/Category: Performance
- Summary/Title/Objective: Test Case for Smoothness of the Composers
- Procedure of testing steps
    - Check if the composer work is smooth and fast enough for the optimal user experience.
- Expected results/Outcome
    - The composer's work is smooth and fast enough for the optimal user experience.
- Priority/Severity: Minor

5. Consideration of Various Factors in Engineering Design

	Effect Level	Effect
Public Health	0	Our project doesn't have any effects on public health concerns.
Public Safety	8	We are going to be storing some private data about system users as email and passwords. However it is going to be handled safely by using secure databases and hashing algorithms.
Public Welfare	3	Our project doesn't seem like it can generate income for some people at first glance. However, as with most of the other A.I. projects we are foreseeing some of the users to find specific niches to use their A.I. generated music to generate some cash flow.
Global Factors	7	The music industry is very big and global. All around the world people are subscribing to music providers. This project might have a big impact on the music industry so it's a globally effective project.
Cultural Factors	3	Our project isn't affected by any cultures. However it might affect some musical cultures in the long term.
Social Factors	3	There isn't much social aspect in our project as users can't interact with each other. However this project might create some interactions in other social media platforms.

6. Teamwork Details

6.1 Contributing and functioning effectively on the team

As this is a complex project, we divided the work and assigned each member a part of the complete model. The distribution of work is as follows;
Cihan Can Kılıç: Machine learning models, generating music and parsing MIDI files.
Berk Baltacı: Machine learning models, generating music and parsing MIDI files.
Azar Hasanaliyev: Rest of the logic layer and UI/UX.
Emir Yavuz: Rest of the logic layer and UI/UX.
Edip Kerem Tayhan: Rest of the logic layer and UI/UX.

6.2 Helping creating a collaborative and inclusive environment

We divided the work equally to not overwhelm any of the members and everyone will have an equal impact in the end product. We are communicating through a Whatsapp group and online meetings via Zoom. We are trying to keep the meeting as short and effective as possible for time efficiency. Even though everyone has his own part, we help each other if necessary.

6.3 Taking lead role and sharing leadership on the team

We don't have a chosen leader. However, some members take the leadership role at different times and tasks. Also in the sub-groups which are formed by the division of work, some members take the leading role to make some of the decisions.

7. Glossary

LSTM: A machine learning model which has feedback connections.
SHA-256: A hashing algorithm that produces irreversible and unique hashes. It is used in cryptography for data privacy.

8. References

- *Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition*, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.