Bilkent University

Department of Computer Engineering

Senior Design Project

T2314

Rhythmus

Design Project Final 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

<19/05/2023>

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. This report aims to detail our project and the ways we will aim to achieve our goal.
Rhythmus will be fed 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.

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.

2. Requirements Details

2.1 Functional Requirements

Begin with, the users must be able to use Rhythmus Windows, or macOS devices through local web..

2.1.1 Navigation Through User Interface

- The user interface must be fully navigable by left click, scroll up and scroll down actions.
- After the login page, the home page must occur.
- From the home page, there must be buttons for opening the start from scratch page and the open existing project page.
- When creating a new project, there must be the parameters page.
- After parameters are set, the website must open the layer page, where the new song will be composed

  Begin with, the users must be able to use Rhythmus Windows, or macOS devices.

5

2.1.2 Music List

- The user must be able to create a list of music that he or she uploaded and keep it as long as he or she wants.
- There must be an option for choosing a music from the list to be used for making another music.

2.1.3 Parameters

- In the parameters section, there must be options for setting the following properties of the project:
    - beats per minute
    - scale
    - key
    - time signature

2.1.4 Pattern Detection

- The melodic patterns in the music chosen by the user must be detected.

- The midi played in the chosen music must be detected separately.

2.1.5 Generating Similar Music

- Rhythmus must generate several similar versions of melodic patterns in the chosen song.
- There must be an option for choosing the instrument of the pattern generated.
- It must be possible to cut certain parts of the generated music to be used.

2.2 Non-functional Requirements

2.2.1 User Interface

- There must be a nice UI design that does not make the user get bored.

- All graphical components must be fully visible for everyone including the elderly.

- All functionalities must be self explanatory.

● The help page must have instructions as clear as possible.

2.2.2 Documentation

● Documentation for explaining functionalities to the users must be available.

● There must be source code documentation for future developers.

2.2.3 Performance Characteristics

● The program must perform following operations in no more than 3 seconds. ○ navigate between pages
○ upload a song to the list
○ delete a song from the list
○ edit the song(e.g. cut some part of it)
○ set parameters of the project

● The program must perform following operations in no more than 60 seconds. ○ detect melodic patterns in the music
○ detect musical instrument played in the music
○ generate similar patterns

2.2.4 Error Handling and Extreme Conditions

● The crash rate of the program must be less than %0.5.
● In case of a crash, the data of the user must not be lost.
● In case of a crash, the user must be informed via a notification.

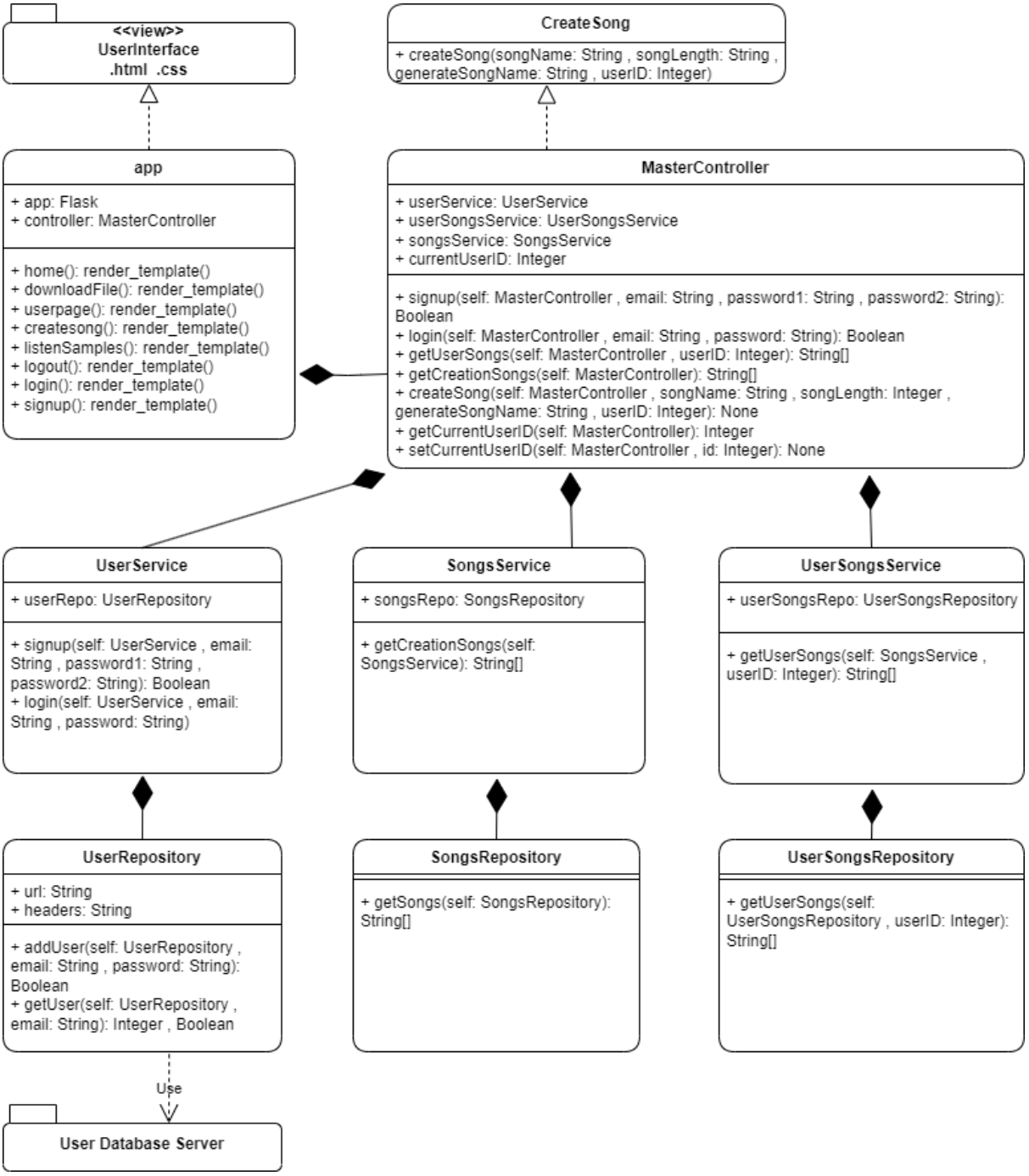
2.2.5 Quality Issues

● The program must be available 7/24.
● Security of the users' data must be provided.
● The program must be error-free.
● In case of a failure, the application must be restartable in less than 5 seconds. ● Independent of the device, the program must work properly.

2.3 Pseudo Requirements

- Git/Github must be used for version control.
- Planning of the project must be done.
- Discussions with the supervisor must be done regularly.

3. Final Architecture and Design Details



- **App**

  App class is the entry point of the application. All of the user commands that came through the interface will be sent to and handled by this class. App will use an instance of the MasterController class and Flask library to forward the requests of the user to the lower layers of the system and to pass the information coming from the lower layers to the user.

- **MasterController**

  This class will only be used by the App class. MasterController will be working as the handler of all services of the system. It has an instance of UserSongsService, UserService, SongsService and CreateSong. When MasterController receives a request from the App it is going to pass this request to the relevant service and forward the output to the App after doing some operations on the data if needed.

- **UserService**

  UserService will receive login or signup requests from the MasterController and forward it to the UserRepository through an instance of it. While logging in credential validations are going to be done here with the raw data received from the UserRepository. So the services will be the center of data operations, manipulations and validations.

- **SongsService**

  SongsService will be the operation center between MasterController and SongsRepository. It is going to use an instance of the SongsRepository.

- **UserSongsService**

  UserSongsService will be the operation point between MasterController and UserSongsRepository. As pointed out above, services are the operation centers.

- **UserRepository**

  This class will communicate with a server that stores user information in JSON format with the specifications received from the UserService and send the raw data to the UserService.

- **SongsRepository**
  This class handles data requests coming from SongsService. Songs are stored in a static local file. Therefore there isn't a server or database connection here.
- **UserSongsRepository**
  This class handles requests from UserSongsService in the same way as SongsRepository. Again all songs are stored locally in the static file of the app. The

ownership of the songs by users are determined through a naming system as "[songname]_[userID].mid".

- **CreateSong**

  This class actually stores a single function and MasterController imports this function. This function uploads the weights of a previously trained LSTM network and runs this network with the specified inputs to create a unique MIDI file. This file is created directly at the same place with where we store the user songs.

- **User Database Server**

  This system will be further explained below. In summary, it runs a server on the specified host and stores user information. This information contains email, password, and user. In this database server, we use MySQL to hold the information and in the server, it checks whether there is a user like that if the user exists it gives errors. There are three important parts used to get the user to find the created songs and to check login and sign-up information.

4. Development/Implementation Details

First we started with the AI module. After some research we found out that the best network to use for this specific task was LSTM. We used Tensorflow's Keras library in order to build the LSTM model. Then we gathered MIDI files to use in the training and prediction processes. These MIDI files were parsed with the Music21 library in python. Also we used Music21 while generating MIDI files as well. After these steps we trained the first model. However for a long time we couldn't get the system to work as we desired. We encountered an underfitting problem which led to all predicted notes to be the same note. In order to solve this problem we started enlarging the model and the training data. We experimented with different layer sizes and kinds of layers. Then finally we got the model to work. The training of the model took around 8 days (this time is subject to change in different hardware systems). In the final model we used three LSTM, two BatchNormalization, two Dropout, and two Dense layers. The forming of the training data was as follows at the beginning; we placed first 100 notes in the input data and 101th note to the output data then we placed notes 102 to 201 in the input data and 202th note in the output data. However this technique drastically reduced the number of samples and wasn't successful at generating different

outputs as a result of small changes like 1-2 notes. That is the reason why our first models were generating the same note over and over. In the final product we gathered data as follows; we placed notes 1 to 100 into the input data and 101th note into the output data. Then we placed notes 2 to 101 into the input data and 102th note into the output data. This way the model became capable of capturing small changes and acting accordingly. However there were still problems in an intermediate model we built which took about 20 hours to train, the output was sometimes good but other times the model under fitted again. So we enlarged the model and the training data and built the final model which is used in the end product.

After the model was created and worked as we desired all the time, we started working on the website and the server. We wrote the model in python so we decided to continue with python for the website as well. Flask library was used in order to create, manage and run the endpoints of the website. We created simple html files for each endpoint and prioritized the functional requirements of the system. The endpoints are /home, /download/<path:filename>, /userpage, /createsong, /logout, /login, /signup and /listensamples. We used the send_from_directory() function of Flask for the download operation. For playing MIDI files we used the cdn's midi-player class for html. Then we integrated the prediction code we wrote earlier to the system. Also we saved the weights of the trained model in a .hdf5 file, and we also saved the parsed notes that were used for model training and put both of them into the static file of our app.

After we concluded the functional parts we started working on the UX. We modified the html files and added some css code in order to make it prettier. One of our main goals was to create a website that would be incredibly easy to use so that people who don't know anything about musical theory or machine learning could enter and use the system to generate their unique AI song.

The user database has basic information to hold the user information and their songs. We choose the Python Django framework for the user database, and we have a MySQL for the database server. The main three methods are login, sign up and get users to find their unique id and match them with their created songs. The see user method is the get method, and the others are post methods.

5. Test Cases and Results

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Fail does not check

- 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
- Date Tested and Test Result
    - Fail does not have composers.

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Fail does not have a delete piece.

- 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

- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Fail does not check

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- Test ID: 26
- Test Type/Category: Compatibility
- Summary/Title/Objective: Test Case For Firebase

- Procedure of testing steps

  o Check if the database works optimally and stores the data properly.

- Expected results/Outcome

  o Firebase works optimally and stores data properly.

- Priority/Severity: Critical
- Date Tested and Test Result
  - Pass

- Test ID: 27
- Test Type/Category: Compatibility
- Summary/Title/Objective: Test Case For Keras
- Procedure of testing steps

  o Check if the library works optimally.

- Expected results/Outcome

  o Keras works optimally.

- Priority/Severity: Critical
- Date Tested and Test Result
  - Pass

- 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

- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
  - Pass

- 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
- Date Tested and Test Result
  - Fail, it has no such functions in it at the moment.

- 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
- Date Tested and Test Result
  - Pass

- 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
- Date Tested and Test Result
  - Pass

- 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
- Date Tested and Test Result
  - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

- 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
- Date Tested and Test Result
    - Pass

6. Maintenance Plan and Details

We are using many libraries that we have to keep updated. Therefore we should always follow the updates on these libraries and change the source code accordingly in order to keep the program up to date. Also we should follow and implement python updates as well. We wrote all of the code in python except for html and css files so python updates will play a big role in the maintenance. Also aside from libraries we should always follow new technological improvements in the AI field in order to implement better and faster technologies into our app.

We are storing our static files like MIDI files locally. We are planning on storing them in the cloud. One of the most important updates we are going to do in the future are adding more instruments. Right now the system only generates piano songs. Actually the only thing we should do to add more instruments is to gather MIDI songs for that instrument and train another model with those MIDI songs. The structure of the model is going to be the same, the only thing changing is going to be the training songs. However due to the limited amount of time and the fact that the training of the piano model took more than a week we couldn't implement other instruments but it's a priority for the future. Another thing we should

implement is the note offset. Right now all of the notes are lasting the same amount of time and there is a standard amount of time between each note. This makes the songs lack rhythm. Again we can use the exact same model and the exact same dataset but again due to lack of time we couldn't implement it as well.

7. Other Project Elements

7.1. 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.

7.2. Ethics and Professional Responsibilities

The ethical responsibility of the rhythmus project is based on the copyright issue. As described before, the project uses different kinds of music, with midi files. During this process, it can violate the copyright of the music. Therefore, we must ensure that this program does not infringe the copyright.

Professional responsibility is the main issue for our group because we are the five people who need to learn to know each other when the project begins and all of us should do their work before the deadline. In addition to that, the inside group deadline should be before the actual deadline because if the responsibilities still need to be done, there has to be time to compensate for that.

7.3. Teamwork Details

7.3.1. Contributing and functioning effectively on the team

Cihan Can Kılıç:

- Contributed to reports.
- Contributed to AI training and AI music generation codes.
- UX (html, css)
- UI (Flask module (App))
- Logic classes (MasterController, Service classes, CreateSong)

Berk Baltacı

- Contributed to reports.
- Contributed to AI training and AI music generation codes.
- Server side application.
- Repository classes.
- Created basic web page for the documents and info for the project

Emir Yavuz

- Contributed to reports

Azer Hasanaliyev

- Contributed to reports.

Edip Kerem Tayhan

- Contributed to reports.

7.3.2. Helping creating a collaborative and inclusive environment

Unfortunately we couldn't function well as a team. First semester we formed sub-groups in our group and divided the work. However this strategy didn't work well and we couldn't create a collaborative and inclusive environment.

7.3.3. Taking lead role and sharing leadership on the team

Cihan and Berk took the leading role in most parts. Apart from that in the first semester the leadership role was more dynamic. Different members of the group took the leadership role at different times. However in the second semester due to our inefficient group dynamics we couldn't create a collaborative environment. Therefore the project became dependent on individual efforts rather than a team effort which made the leadership role fade away.

7.3.4. Meeting objectives

We met most of our objectives that we set during meetings. We implemented almost all of the functional requirements. The ones we couldn't meet are the midi editing page, complete song with different instruments, and musical parameters while creating a song. Editing page was going to be like a musical composition app where users can change tones, notes according to the timeline. Unfortunately we couldn't find time to implement this feature. Another shortfall was the multi-instrumental song creation. As stated in the report the training time of the working model was more than a week so we couldn't make time for this feature. However the process is the same for each instrument so we are going to add different instruments in the future. The last requirement is the musical parameters like keys, tones etc. In order to implement this functionality we have to learn musical theory and unfortunately we couldn't find any time for this as well.

7.4 New Knowledge Acquired and Applied

At the beginning of the year we knew almost nothing about machine learning and AI. Therefore we had to do a wide research about the topic and analyze different machine learning problems. After this research we started experimenting with different models. Both

theoretical and experimental processes taught us a lot about machine learning and AI technology which will become very handy in our future working lives.
Another subject we weren't very familiar with was web development. Some of us had small experiences in the past but none of us ever built a complete website from scratch. Therefore after we were done with AI we started doing research and experimentations about website development. Web app development is also a popular subject in software development. Therefore we believe that the things we learned about web applications throughout this project are going to be very useful as well.
Also as we wrote all the code in python so our python skills are improved drastically as well. For most of us this project was the first complete full-stack project we have done using python.

8. Conclusion and Future Work

As AI is the most popular technology right now around the world we believe that this project taught us a lot that will be of use in the future. We learned so many things about machine learning algorithms throughout the course of this project. Also we believe that we developed our problem solving skills in a significant way. We encountered countless problems both in terms of teamwork and technical problems. After all, being an engineer is mostly about being able to solve all kinds of problems. Despite all of these issues we are proud of and happy about the end product we delivered.
In the future we are planning on adding more instruments and note offsets in order to create more sophisticated pieces. Also we are planning on giving the users more choice while creating their unique piece. Some of these choices might be selecting which instruments to be a part of their piece and giving them more options while choosing a song to base on their unique piece.

9. 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.

10. 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.