MusicGuru: Multi-task and Multimodal Representation Learning for Music Applications

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Software Engineering

By

Bhuvana Gopala Krishna Basapur
Arpitha Gurumurthy
Mayuri Lalwani
Somya Mishra

December 2021

APPROVED

Professor Vijay Eranti, Project Advisor

Professor Dan Harkey, Director, MS Software Engineering

[Department Chair's Name], Department Chair

ABSTRACT

MusicGuru: Multi-task and Multi-modal Representation Learning for Music Applications
By
Bhuvana Gopala Krishna Basapur, Arpitha Gurumurthy, Mayuri Lalwani, Somya Mishra

The way in which we perceive the world is by using our sense of sight, hearing and smell, which is multimodal in nature. Research shows that multimodal learning helps in better understanding and retention of information in humans. Multimodality in AI refers to training models by combining multiple modes of input such as visual, audio and textual representations. There are correlations between them which further help to uncover useful patterns in the data that may not be obvious when using a unimodal approach. Similarly, a multi-task learning approach in AI trains a model to perform different tasks. This helps in generalizing the model as it learns the commonalities between these tasks and helps in achieving better performance for all the tasks. Introduction of multimodality and multi-task applications in AI will train models to be more 'human-like' and help in moving one step closer to 'self-improving AI'.

Since multimodal learning involves engaging multiple senses in the processing of information, it is considered more effective than single-modality learning. The concept of music is also multimodal. Using only audio to express music would be very restrictive. As discussed in the literature search section of this workbook, it is seen that most of the previous works in the field of music understanding focused on using only audio tracks or symbolic music as input. Text based approaches have also been employed but research using multimodal input is still limited. Music related tasks would hugely benefit from multimodal learning approaches as it can learn the fine grain details present in the language of music. Musicians and composers have mostly lacked a similar tool for exploring and mixing musical ideas, but the objective of this application is to change this. MusicGuru, is an application that would help creating palettes for blending and exploring musical notes.

The goal of this project is multimodal music understanding using intelligent DL techniques. To take it one step further, we aim at building a model that can jointly handle multiple music related downstream tasks to solve issues concerning extensive data and computational requirements. The concept of multimodality and multitasking in the field of music help in better understanding of music structure in terms of its melody, rhythm, beat and chord. This in turn results in better performance for tasks like melody completion, changing the pitch while keeping the rhythm constant, changing the rhythm while keeping the pitch constant, changing the melody of the song with the existing chord progression and changing the chords of the song with the existing melody progression as the model has a provision of visualizing, hearing and finally analysing the same input.

Acknowledgments

The authors are deeply indebted to Professor Vijay Eranti for his invaluable comments and assistance in the preparation of this study and Professor Dan Harkey for his invaluable insights leading to the writings of this project report.

Table of Contents

List of Figures

List of Tables

Chapter 1. Project Overview

1.1 Introduction

In this day and age, music is perceived in different modes such as audio, video, and text. Hence, music related applications would benefit from multimodal inputs. However, most of the previous works in music understanding depended only on audio sources and features. Music understanding is required to perform various tasks such as genre classification, style classification, and melody completion among others. Traditional Machine Learning (ML) approaches used handcrafted audio features such as Mel Frequency Cepstral Coefficients (MFCCs) while Deep Learning (DL) approaches used visual representations of these audio signals as input to CNNs (Convolution Neural Network) for performing these tasks [1]. Text based approaches were also employed in music understanding where customer reviews and song lyrics served as input [1]. Research in music applications with input as images is still limited. Multimodal approaches for these music applications have also been explored but its research using DL techniques is still in infancy [1]. This powerful combination of multimodal input with DL is what this paper will focus on.

The project involves building an application called ‘MusicGuru’ that aims to serve downstream applications such as melody completion, integrating new melody to an existing chord progression or or vice versa, changing the pitch while holding the rhythm constant or vice versa. All the downstream tasks offered by the MusicGuru application allows music composers to explore different melodies, chord progressions, pitches, and rhythms. This would in turn help them find the tune that resonates with them the most.

1.1.1 Introduction to MusicGuru

The MusicGuru application uses a multitasking model for performing various music related tasks. It uses a transformer based architecture for achieving the same. The model is deployed on the flask server which comes up on port 5000. The UI server comes up on port 8080. The predictions are then stored on the S3 bucket with an idea to make retrieval faster and store space. MusicGuru also has an authentication service using AuthO in order to respect user privacy.

1.1.2 Introduction to Symbolic Music

Symbolic music representation is the language of music. It involves encoding the components of music such as notes, pitches and rhythms in a human/machine readable format. Musical Instrument Digital Interface (MIDI) is one such format.

MIDI encodes information for each note such as the note onset, note offset and intensity/velocity. A MIDI file contains a list of MIDI messages and other related metadata.

The below screenshot shows a sample MIDI file (Levels - Avicii) used in our application:

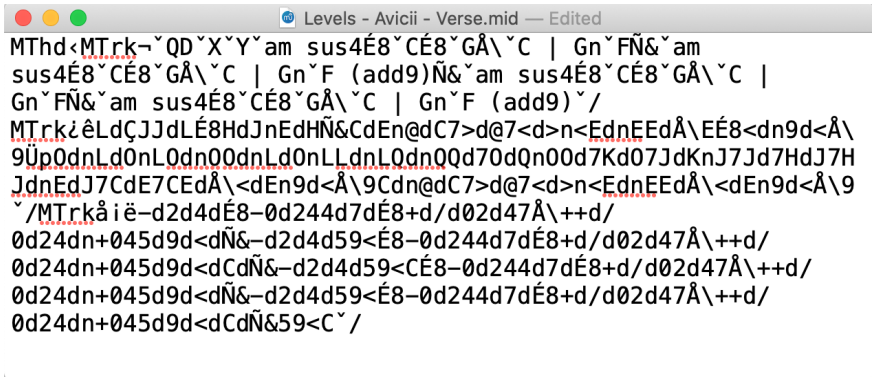


Figure 1. Sample MIDI file

We use MuseScore3 to convert this into sheet music. MuseScore is an open source music notation software that accepts MIDI files as input and output. Below is a screenshot of the same MIDI file (Levels - Avicii) as viewed on MuseScore:

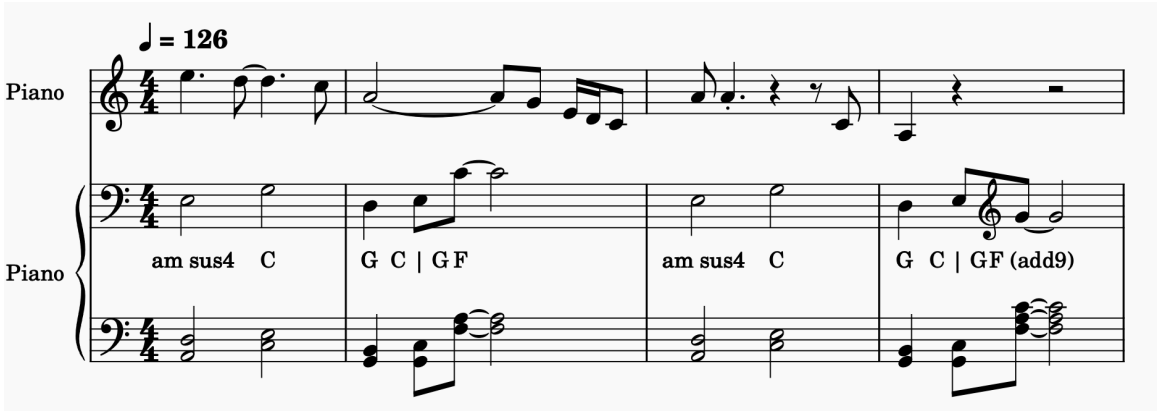


Figure 2. Sample MIDI file as viewed on MuseScore3

1.1.3 Importance of Multitask Learning

Multitask learning is a type of supervised learning that involves fitting a model on one dataset and thereby addresses multiple related problems. This enables parameter sharing and saves the burden of not requiring huge amounts of data.

1.1.4 Music downstream tasks implemented in MusicGuru

The MusicGuru application performs the below downstream tasks:

- Melody Autocompletion
- Changing the pitch while keeping the rhythm constant.
- Changing the rhythm while keeping the pitch constant.
- Changing the melody of the song with the existing chord progression.
- Changing the chords of the song with the existing melody progression.

1.1.5 Introduction to Transformers

Transformer is a class of models in deep learning that uses the concept of attention which means every word in the input of words is analyzed to determine which words get the most weightage. It is very widely used in the fields of Natural Language Processing (NLP) and Computer Vision (CV). They have revolutionized the NLP space.

1.1.6 Transformer variations

There are several types of Transformer variations. Although they have the same types of attention layers, they are specialized for different tasks. In project we are using 3 types of Transformers:

- Sequence to Sequence Translation (Seq2Seq): A class of neural networks that transforms a given input sequence into a different sequence. They particularly specialize in translation tasks in which a series of words in a language is translated into a series of words in a different language. It consists of an encoder and a decoder. The encoder accepts the input and converts it into a higher dimensional vector which is fed into the decoder to generate the output.
- Bert: BERT stands for Bidirectional Encoder Representations from Transformers. It is pre-trained on a large corpus of unlabelled text including the entire Wikipedia and Book Corpus. It specializes in predicting masked or missing tokens. Bi-directionality helps in providing complete context.
- TransformerXL: A class of neural networks that specializes in text generation. It consists of a forward directional decoder.

1.2 Proposed Areas of Study and Academic Contribution

1.2.1 Importance of symbolic music understanding

Symbolic music understanding is crucial in performing music related tasks as discussed above. It is necessary to accurately represent the nature of symbolic music and be able to embed key music information into a lower dimension so that complex music information can be understood as a computational process. Although symbolic music shares similarity with natural language, Natural Language Processing (NLP) techniques cannot be used directly, because symbolic music has a complex structure consisting of harmony, beat, chord, pitch, rhythm, dynamics and instrument information [2]. Naïve encoding schemes result in very long representations of music songs causing information leakage.

1.2.2 Evolution of different approaches to understand music

Chordripple [3] and Chord2vec [4] are chord-based approaches which aim to learn just a set of simultaneous notes i.e. chord representations. These works only contribute to harmony and cannot generate universal embeddings for symbolic music. Melody2vec [5] model keeps track of the sequences of notes (known as motifs). It extracts motifs from melody tracks using a rule-based approach. It also trains on most frequent notes to overcome the long-tail issues caused by huge vocabulary leading to incomplete learning. Piano roll-based methods are inefficient for long notes because the encoding results in a 2D matrix, one for pitch and other for time step [2], [6] and cannot model melody and harmony.

PiRhDy is a model proposed by Hongru Liang et al., that produces embeddings to leverage pitch, rhythm and dynamics information simultaneously and also encodes both melodic and harmonic contexts comprehensively, in order to optimize knowledge from music into useful embeddings [7]. While PiRhDy is trained on the Lakh MIDI Dataset (LMD), MusicBERT is trained on the larger Million MIDI Dataset (MMD) making it more efficient and pushing State of the Art results by a good margin [2].

MusicBERT is a pre-trained model which uses efficient encoding and masking strategies for understanding symbolic music [2]. Zeng et al., 2021 proposed OctupleMIDI encoding which encodes each note as a tuple of 8 elements (musical note, time signature, tempo, bar, position, instrument, pitch, duration, and velocity). This reduces the length of music sequences and eases their processing. The OctupleMIDI encoding is then converted into a single vector using a linear layer. They also proposed a bar level masking strategy that helps in avoiding information leakage as in the original BERT [2]. MusicBERT is finetuned on four downstream applications namely melody completion, accompaniment suggestion, genre classification and style classification. It is

shown to outperform all the previous State-of-the Art models for each of these applications [2].

1.2.3 Approaches towards handling multimodal inputs

When it comes to multimodal inputs, UniT, a Unified Transformer model is another such pre-trained model that simultaneously learns multiple tasks across different domains, ranging from object detection to natural language processing related tasks and multimodal reasoning [8]. It also performs tasks such as Visual Question Answering (VQA) and Visual Entailment.

The authors at Facebook AI Research describe it as a unified transformer based encoder-decoder architecture wherein the complete model is jointly trained to multitask and output predictions and losses for each task individually. The separate transformer encoders encode each input modality in the form of hidden states/ feature vectors. The predictions are made on each of the tasks with a shared decoder over the encoded input representations which are projected on task-specific output heads.

Each task shares the same model parameters instead of fine-tuning specific task models. Some of the previous research works have only focused mostly on a single task, have involved task-specific fine-tuning for each of the models instead of sharing parameters across tasks. The UniT model takes both- images and text as input. For images based inputs, a Convolutional Neural Network is used for feature extraction and for text/ language based inputs, a 12 layer BERT is used to encode input words into corresponding hidden states [8].

Once input modalities are encoded into hidden states, the decoder is applied based on the modality of the input for the task- uni-modal for a single task and multimodal for multiple tasks. This model is beneficial because of its simplicity and flexibility. Due to its simple and understandable architecture, it can be easily extended to more inputs, modalities and tasks of any domain, for instance music for the current project. As compared to previous research work on multi-task learning using transformers,

Similar to UniT is 12-in-1, which is a also pretrained model to learn visio linguistic representations that can be fine tuned jointly to specific tasks. Fine tuning from a single multi task model achieves better results than the state of the art. Compared to independently trained single task models, jointly trained tasks lead to parameter reduction from 2 billion to 270 million [9].

The four categories of tasks include visual question answering, caption based image retrieval, grounding referring expressions, and multi modal verification. Many times there is similar association between visual and language tasks and hence require similar understanding.Training multiple tasks jointly helps the research community avoid overfitting to specific datasets and metrics [9]. The datasets used for multiple tasks can be different in size. To deal with this issue, authors of this paper have introduced dynamic stop and go scheduler, task dependent input tokens, and simple parameter heuristics. By using this pre-trained multi task model to fine tune

for single tasks can achieve an improvement of 2.98 points on an average as compared to baseline single task models.

1.3 Current State of the Art

The following review of literature highlights the leading edge technologies available in the field of multitasking and multimodality for music related applications.

1.3.1 PiRhDy

PiRhDy is a State-of-the Art model which integrates pitch, rhythm and dynamics information for symbolic music seamlessly. This model adopts a hierarchical approach and has two main steps: (1) token or note event modeling where pitch, rhythm and dynamics are represented separately and then integrated into a single token embedding and (2) context modeling where melodic and harmonic knowledge is used to train the token embedding [7].

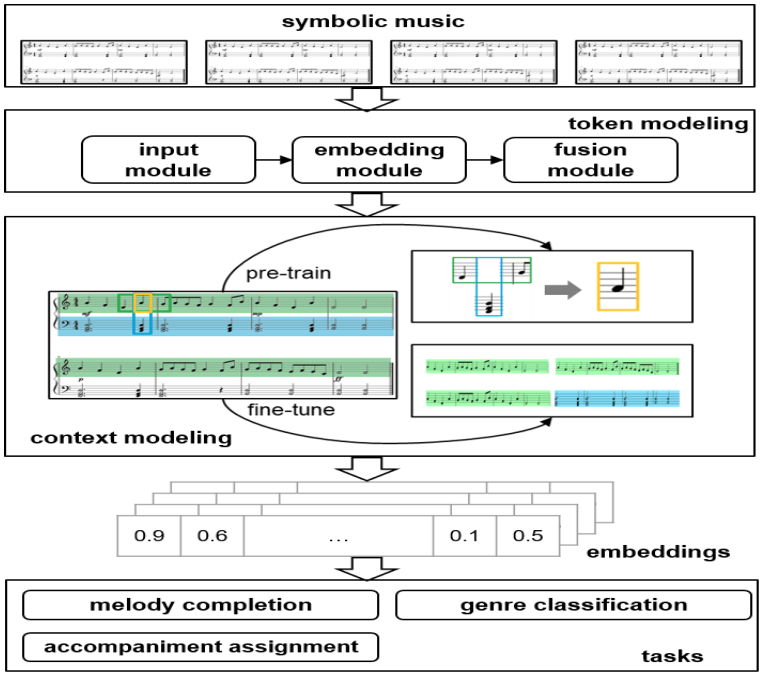


Figure 3. PiRhDy Architecture

PiRhDy model embedding is validated in three downstream applications - genre classification, style classification and melody completion and results show better performance when compared to previous models.

Model	Top-MAGD		MASD	
	AUC-ROC	F1	AUC-ROC	F1
SIA	0.753	0.637	0.761	0.455
P2	0.816	0.649	0.815	0.431
melody2vec_F	0.885	0.649	0.777	0.299
melody2vec_B	0.883	0.647	0.776	0.293
tonnetz	0.871	0.627	0.794	0.253
pianoroll	0.876	0.64	0.774	0.365
PiRhDy_GH	0.891	0.663	0.782	0.448
PiRhDy_GM	**0.895**	**0.668**	**0.832**	**0.471**

Figure 4. PiRhDy results of genre classification

1.3.2 MusicBert

MusicBert is a State-of-the Art pre-trained model embedded with a universal OctupleMIDI encoding and an effective bar level masking strategy trained on a large-scale symbolic music corpus of over 1 million music songs [2]. The goal of this SOTA is to learn the representations of symbolic music.

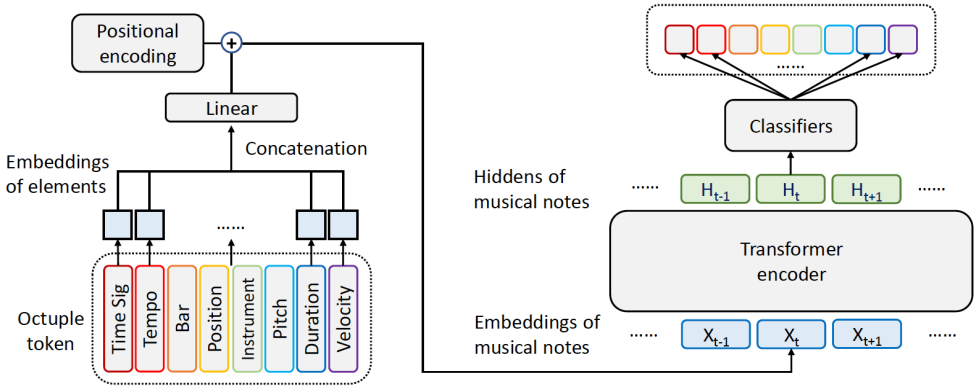


Figure 5. MusicBERT architecture

Since the transformers are incapable of handling sequences with lengths longer than 1000, OctupleMIDI encoding approaches help in significantly reducing the length of these tokens. The bar level masking strategy masks all tokens having the same type (e.g., time signature, bar, instrument, or pitch) preventing information loss and helps in better understanding of symbolic music representations [2]. On fine tuning MusicBERT on the selected downstream applications,

it achieves SOTA performance when compared to the previous baseline models including PiRhDy.

Model	Melody Completion					Accompaniment Suggestion					Classification	
	MAP	HITS @1	HITS @5	HITS @10	HITS @25	MAP	HITS @1	HITS @5	HITS @20	HITS @25	Genre F1	Style F1
melody2vec$_F$	0.646	0.578	0.717	0.774	0.867	-	-	-	-	-	0.649	0.299
melody2vec$_B$	0.641	0.571	0.712	0.772	0.866	-	-	-	-	-	0.647	0.293
tonnetz	0.683	0.545	0.865	0.946	0.993	0.423	0.101	0.407	0.628	0.897	0.627	0.253
pianoroll	0.762	0.645	0.916	0.967	0.995	0.567	0.166	0.541	0.720	0.921	0.640	0.365
PiRhDy$_{GH}$	0.858	0.775	0.966	0.988	0.999	0.651	0.211	0.625	0.812	0.965	0.663	0.448
PiRhDy$_{GM}$	0.971	0.950	0.995	0.998	0.999	0.567	0.184	0.540	0.718	0.919	0.668	0.471
MusicBERT$_{small}$	0.982	0.971	0.996	0.999	1.000	0.930	0.329	0.843	0.993	0.997	0.761	0.626
MusicBERT$_{base}$	**0.985**	**0.975**	**0.997**	**0.999**	**1.000**	**0.946**	**0.333**	**0.857**	**0.996**	**0.998**	**0.784**	**0.645**

Figure 6. MusicBert outperforms the previous SOTA for each of the downstream tasks

MusicBERT also demonstrates the importance of pre-training in achieving good performance.

Model	Melody	Accom.	Genre	Style
No pre-train	92.4	76.9	0.662	0.395
MusicBERT	**96.7**	**87.9**	**0.730**	**0.534**

Figure 7. Comparing results with and without pre-training

Chapter 2. Project Architecture

2.1 Introduction

This section focuses on two key aspects of the MusicGuru application architecture:

1. System architecture
2. Model architecture

The System architecture describes the functional components of the system that begins with an authentication layer, feeding into a UI layer for downstream tasks. This is followed by a backend layer for the APIs and a database layer to persist the music related data. The ML module is the data layer that serves predictions.

The Model architecture describes the functioning of the overall model and the individual language models used to accomplish multitasking in the application. .

2.1.1 Understanding the specialty and applications of transformer variations

Every variation of a Transformer architecture has an area of specialization. Below is the explanation for the transformer variations and their specializations:

2.1.1.1 TransformerXL

Since the TransformerXL neural network consists of a decoder architecture and it specializes in text generation, it is best suited for generating music for the MusicGuru application. The Melody Autocomplete downstream task of the MusicGuru uses TransformerXL.

2.1.1.2 BERT

Since BERT consists of an encoder architecture and it specializes in predicting masked/missing tokens, it is best suited for remixing songs. The below tasks are implemented for the MusicGuru application using the BERT architecture:

- Changing the pitch while keeping the rhythm constant
- Changing the rhythm while keeping the pitch constant

2.1.1.3 Seq2Seq

Since Seq2Seq consists of an encoder decoder architecture, it can be considered as a combination

of BERT and TransformerXL. These models are specialized for translation tasks and work well for the below downstream tasks of MusicGuru applications:

- changing the melody of the song with the existing chord progression.
- changing the chords of the song with the existing melody progression.

2.1.2 Building a multitasking model by combining the transformer variations

Sequence to Sequence Transformer is the backbone for the Multitasking model. Since it consists of an encoder and a decoder, the encoder can be reused to train the required Bert model and the decoder can be reused to train the TransformerXL model.

2.2 System Architecture

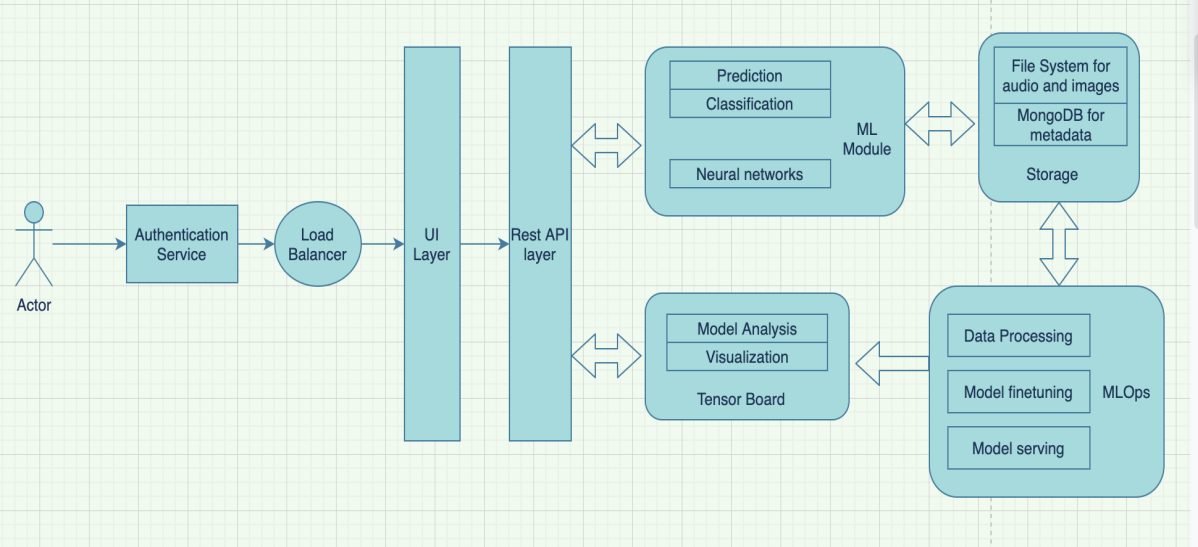


Figure 8. MusicGuru high level Architecture diagram

Below are the different elements in a System architecture:

Authentication service: To safeguard the privacy of the users, the application provides authentication and authorization services. The users can save their work before publishing them. This includes performing tasks like melody completion, changing the pitch while keeping the rhythm constant, changing the rhythm while keeping the pitch constant, changing the melody of the song with the existing chord progression and changing the chords of the song with the existing melody progression.

Load balancer: It routes incoming requests to different servers based on traffic, thereby improving performance. Preferred load balancer in this use case is nginx.

UI layer: It acts as a mediator between the users and the application. There will be different UIs designed for each of the downstream applications. The UI layer will be designed using react JS.

Rest APIs: It offers an interface for the UI to interact with data and ML models to offer predictions.

ML module: It consists of a neural network that is finetuned on the selected downstream applications. This is responsible for serving predictions based on user input.

Storage: The application would use file system and MongoDB/SQL respectively to store images, audio and text metadata.

MLOps and TensorBoard: MLOps would handle end to end processing from data preprocessing to model fine tuning. The logs would then be recorded and visualized on tensorboard.

2.3 Model Architecture

Below are the detailed architecture models for TransformerXL and transformer variations and their specializations:

2.3.1 Transformer

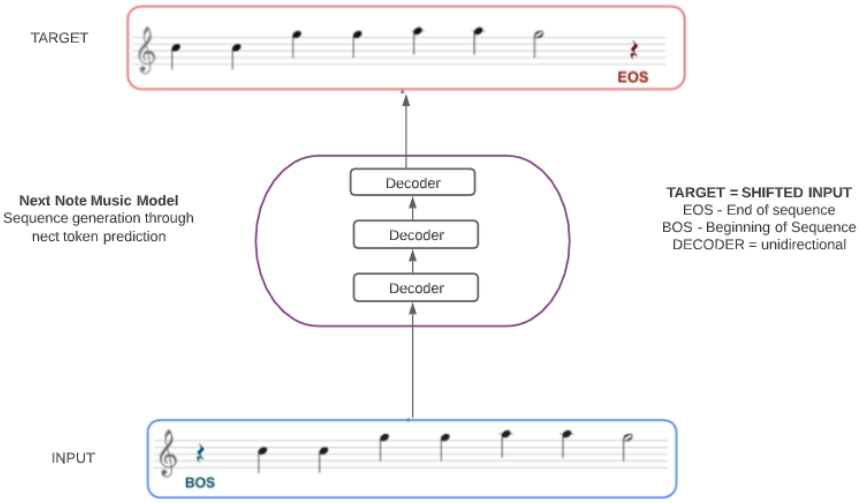


Figure 9. Architecture of TransformerXL

2.3.2 Seq2Seq

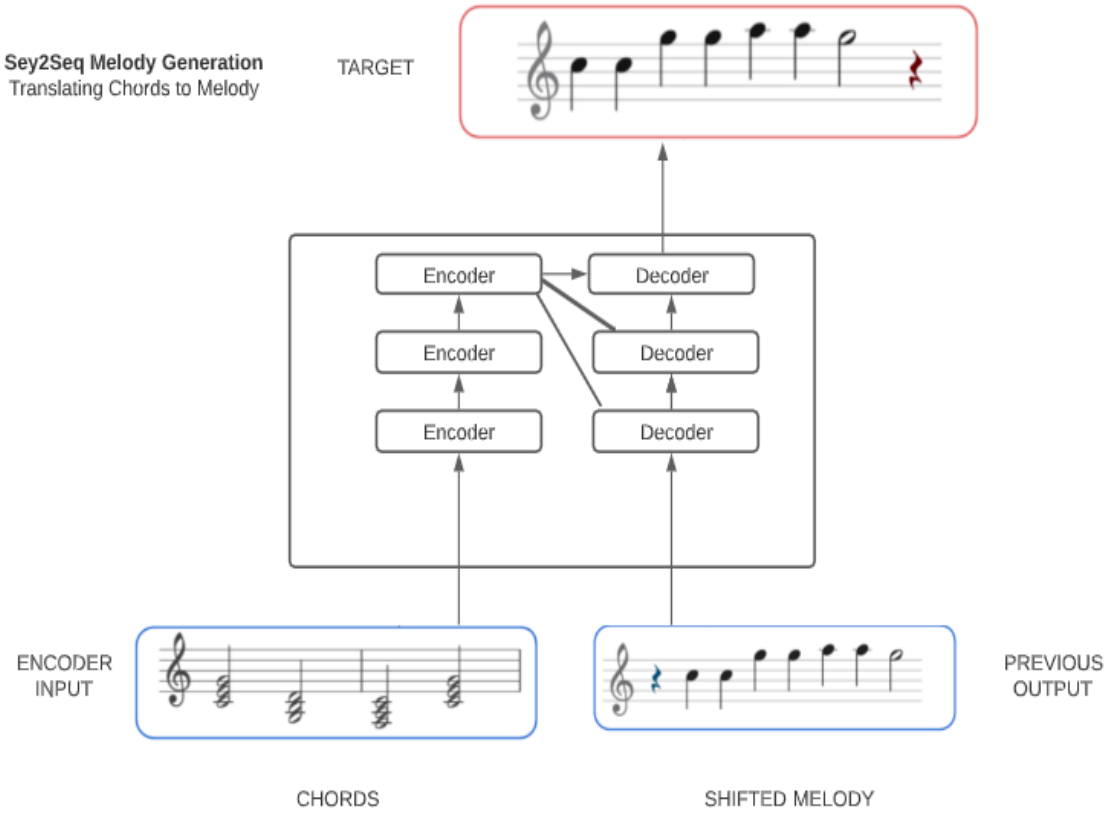


Figure 10. Architecture of Seq2Seq

2.3.3 BERT

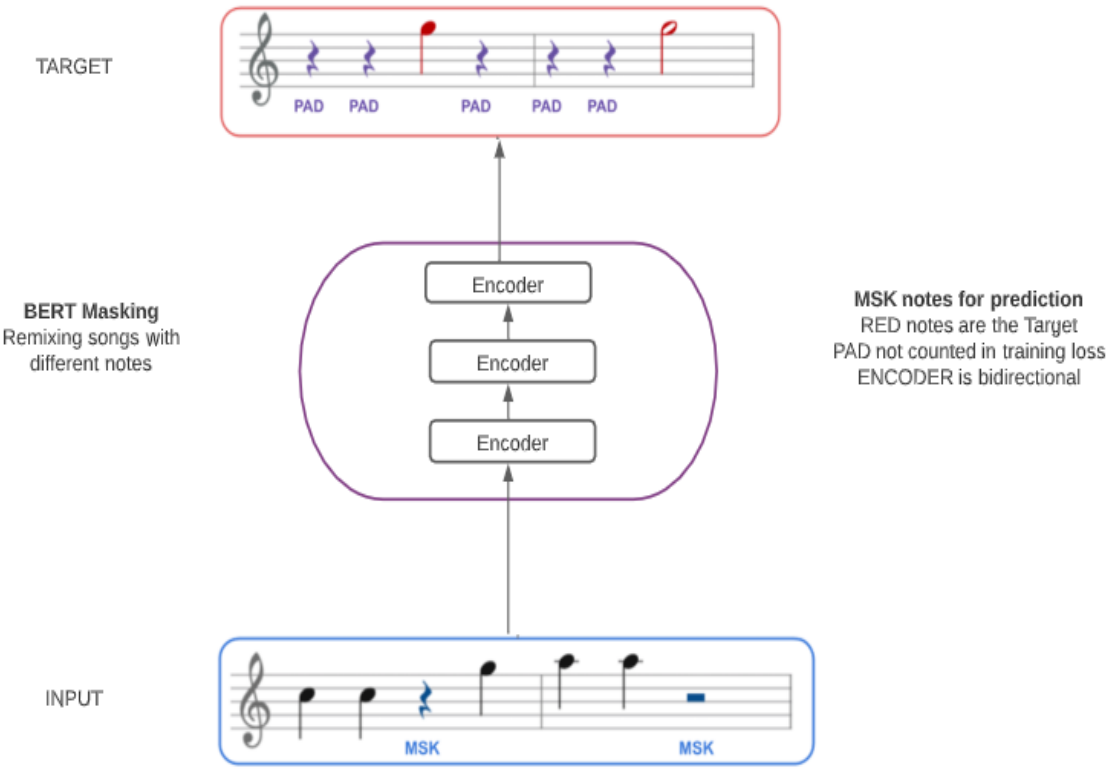


Figure 11. Architecture of BERT

2.3.5 Architecture of the multitasking model

The Multitasking model follows the architecture of the Sequence to Sequence Transformer (shown above in Figure 10) that consists of both an encoder and a decoder. This combination is used to perform tasks such as changing the melody of the song while keeping the chords constant and vice versa. The encoder in this architecture is reused to train BERT. This model handles tasks related to remixing the songs such as changing the pitch and rhythm. The decoder is also reused to train TransformerXL which performs Melody Completion.

Chapter 3. Technology Descriptions

3.1 User Interface

We have built the MusicGuru project User Interface using Vue JS. It is an open source javascript framework with HTML like syntax making it very beginner friendly. We maintain dependencies in ‘package.json’ and use npm to manage these dependencies. We use Node JS as our front end server which comes up on port 8080.

We used bulma and vuetify libraries for styling. Below are the important dependencies that we have installed for the project.

```
..
"dependencies": {
  "@tonejs/midi": "^2.0.27",
  "axios": "^0.22.0",
  "bulma": "^0.9.3",
  "core-js": "^3.6.5",
  "file-saver": "^2.0.5",
  "fuse.js": "^6.4.6",
  "loadash": "^1.0.0",
  "opensheetmusicdisplay": "^1.2.0",
  "pug": "^3.0.2",
  "pug-plain-loader": "^1.1.0",
  "tone": "^14.7.77",
  "vue": "^2.6.11",
  "vue-analytics": "^5.22.1",
  "vue-lodash": "^2.1.2",
  "vue-router": "^3.2.0",
  "vue-search-select": "^2.9.5",
  "vuetify": "^2.5.9",
  "vuex": "^3.6.2"
},
```

Figure 12. Dependencies for UI

3.2 Application Server

We used conda to install the required dependencies for the backend implementation. The below command installs them:

- conda env update -f environment.yml
- conda activate musicautobot

```
# To Run: conda env create -f environment.yml
name: musicautobot
channels:
- fastai
- pytorch
- defaults
dependencies:
- pytorch
- fastai==1.0.61
- jupyter
- ipyparallel
- pip
- python>=3.6
- pip:
  - music21
  - pebble
```

Figure 13. Dependencies for Application Server

The backend web server is a python Flask server that comes up on port 5000 and serves REST APIs that the Vue JS front end consumes.

3.3 Data-Tier Technologies

We have set up an S3 bucket that stores and serves predictions from the Multitasking model in the form of MIDI files. For every prediction, a Json file and a MIDI file gets written to the bucket which is then served on the frontend.

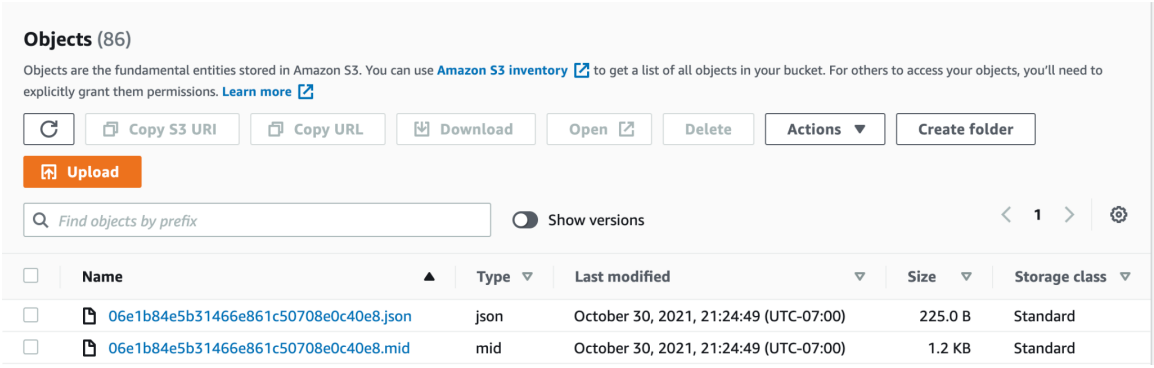


Figure 14. Sample S3 Bucket contents

Chapter 4. Project Design

4.1 UML Diagrams

4.1.1 Class Diagram

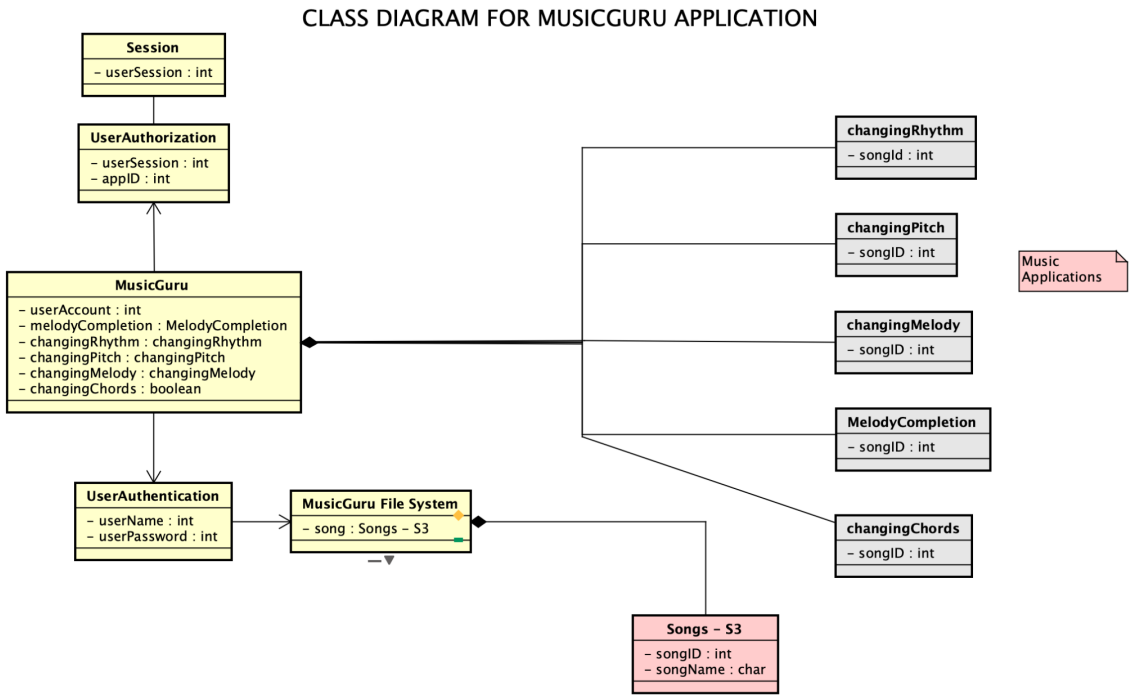


Figure 15: Class Diagram

4.1.2 Deployment Diagram

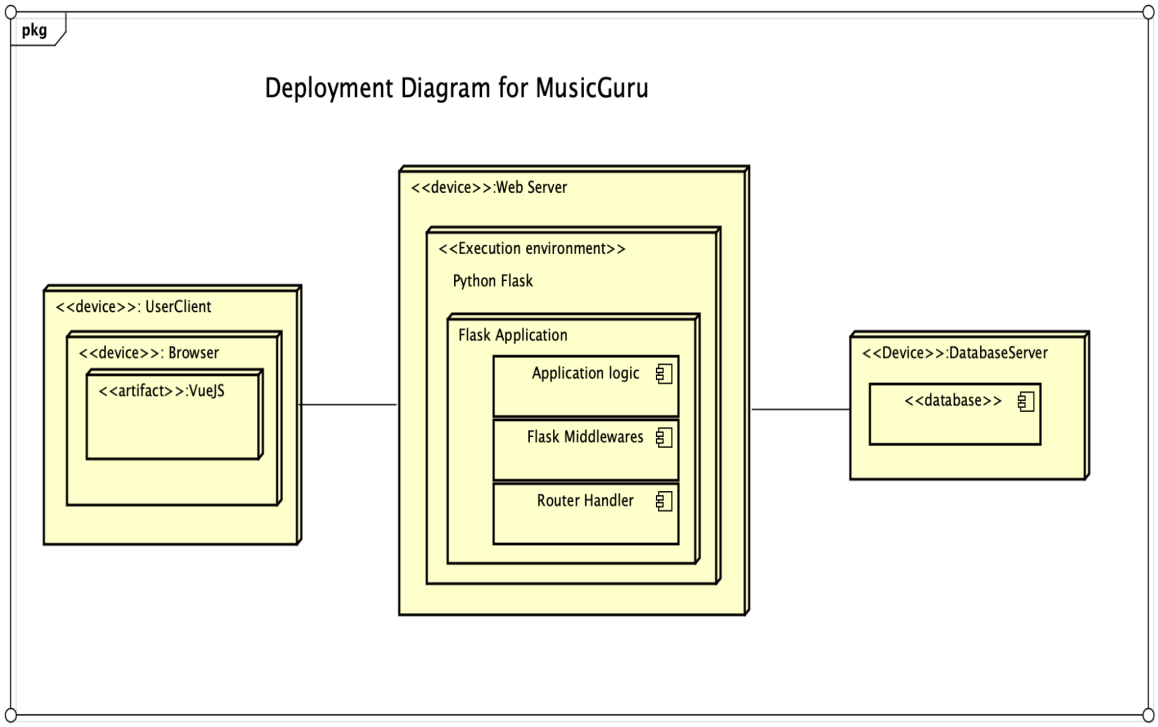


Figure 16: Deployment diagram

4.1.3 Use Case Diagrams

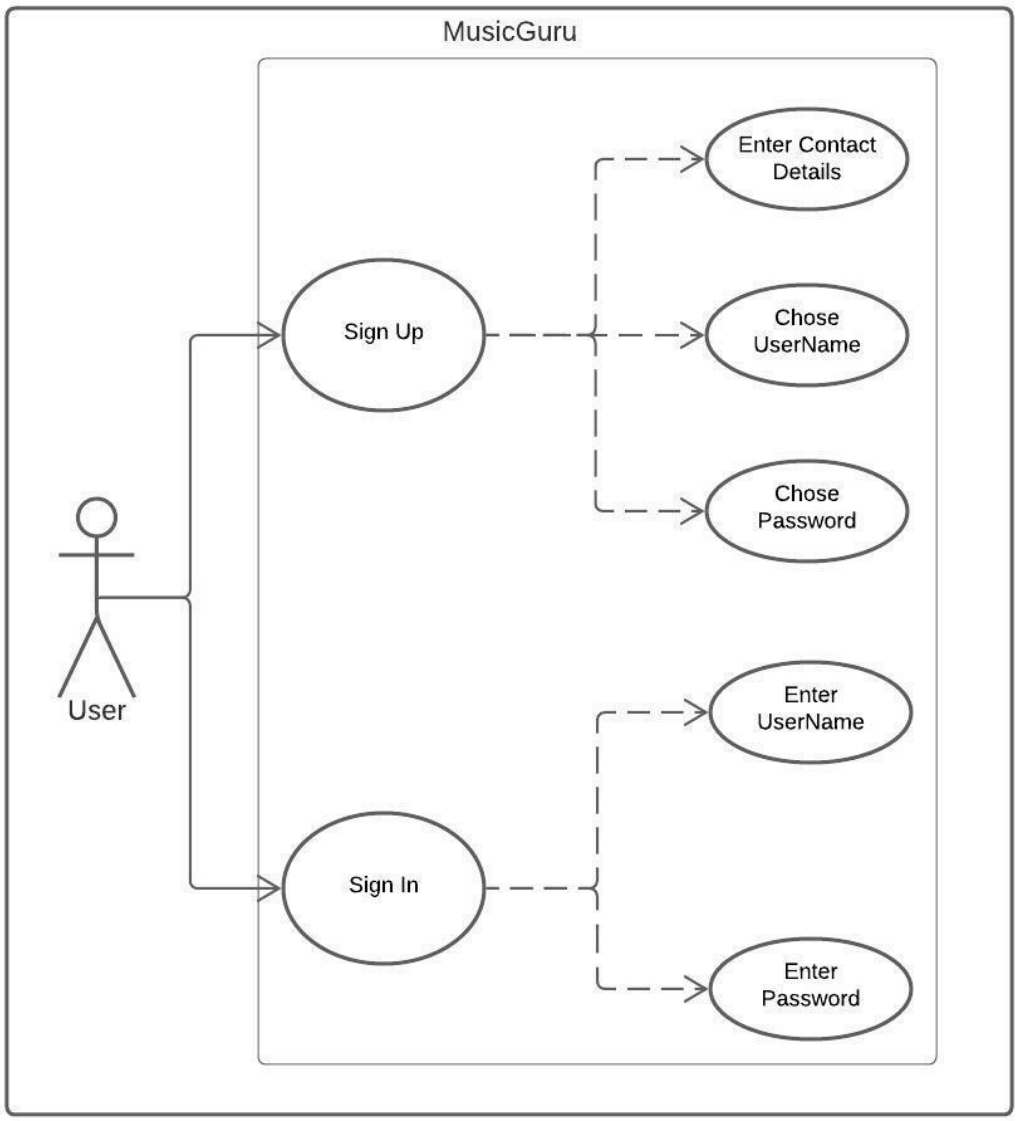


Figure 17: Use case diagram for sign up/ login

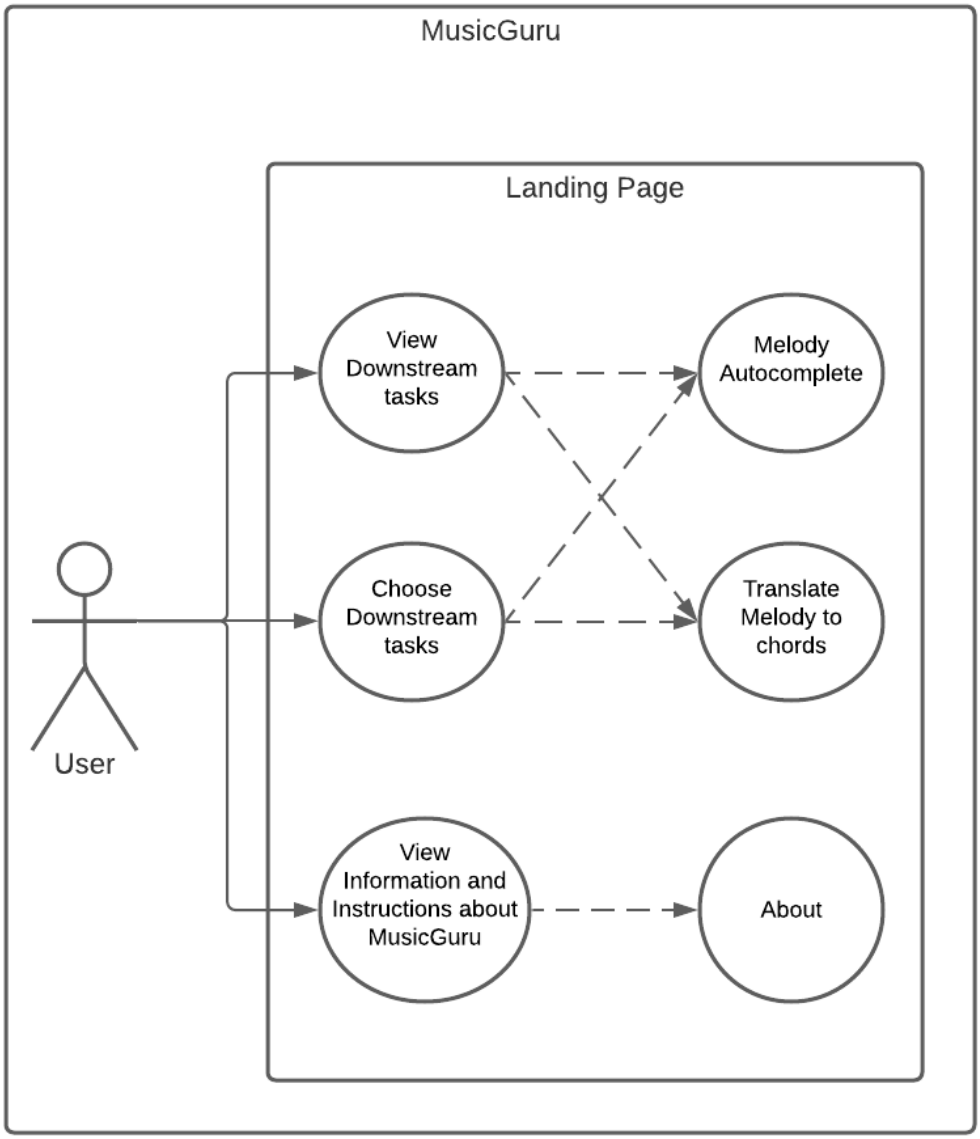


Figure 18: Usecase Diagram for Landing Page of MusicGuru

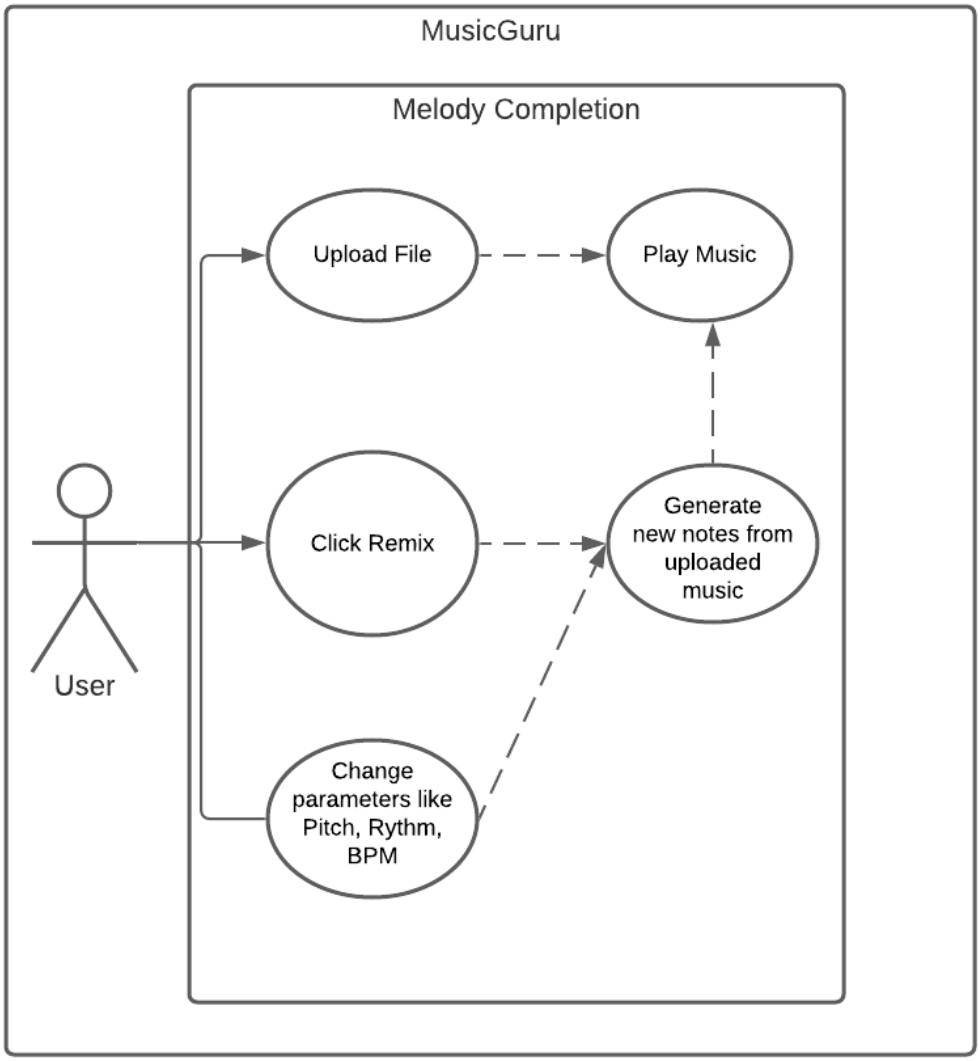


Figure 19: Usecase Diagram for Melody Completion

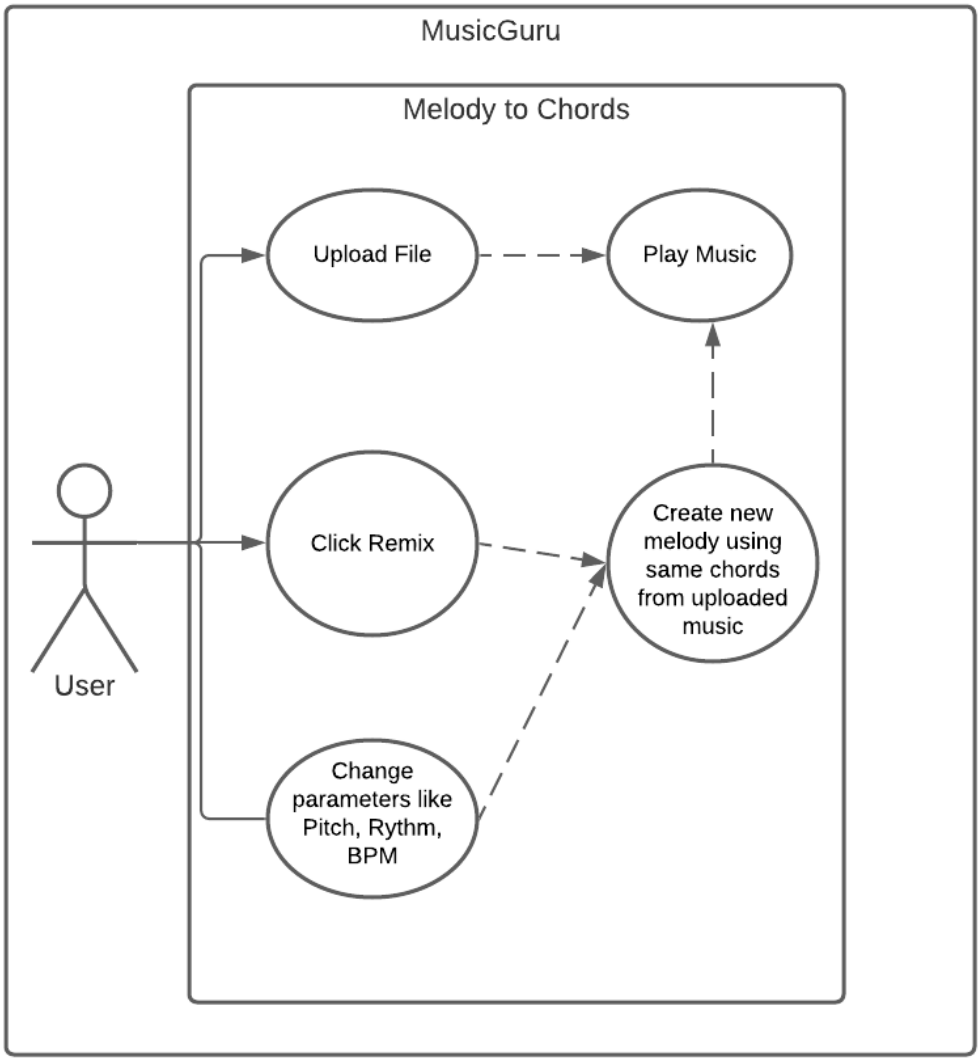


Figure 20: Usecase Diagram for Melody to chords

4.1.4 State Diagram

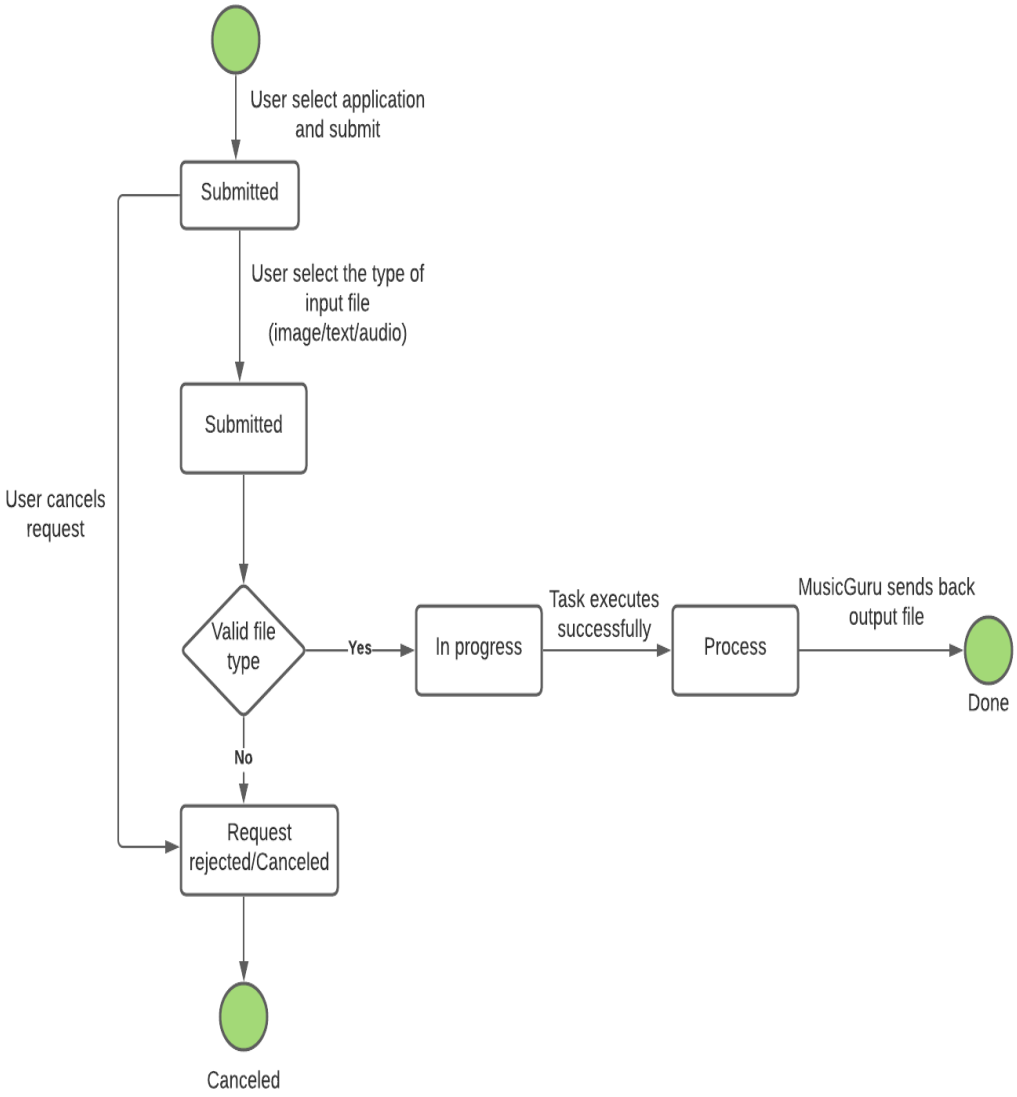


Figure 21: State Diagram

4.1.5 Database Entity Diagram

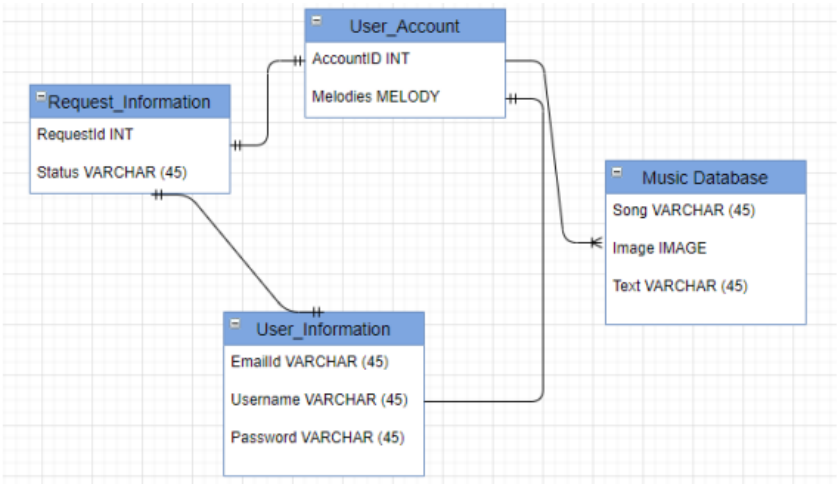


Figure 22: Database Entity diagram

4.2 User Interface

4.2.1 Wireframes

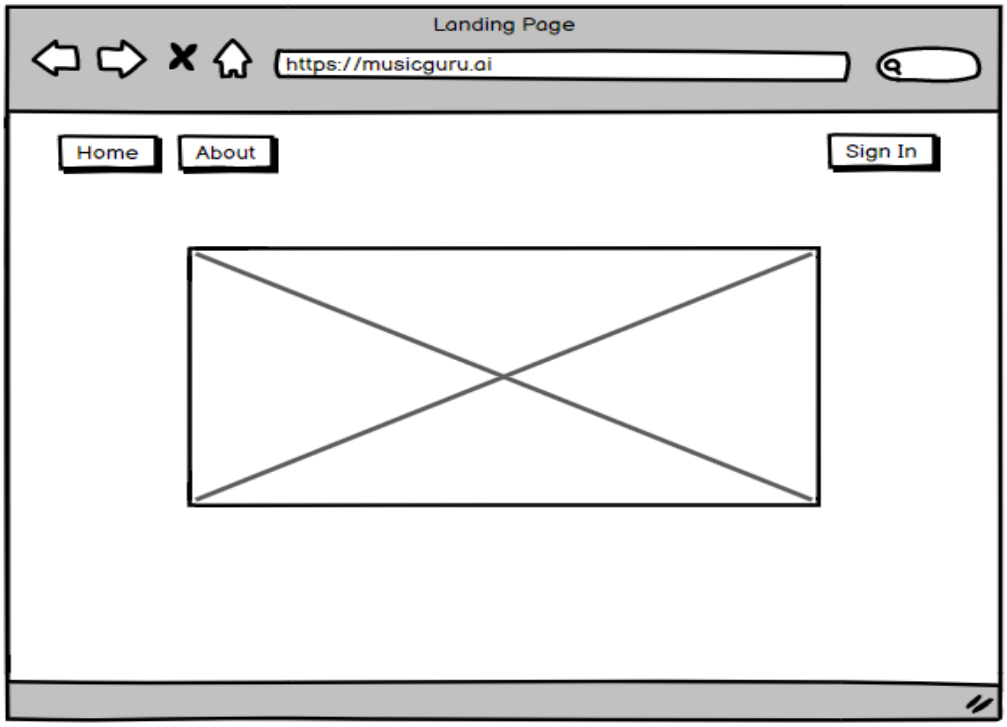


Figure 23: Wireframe diagram for Landing page

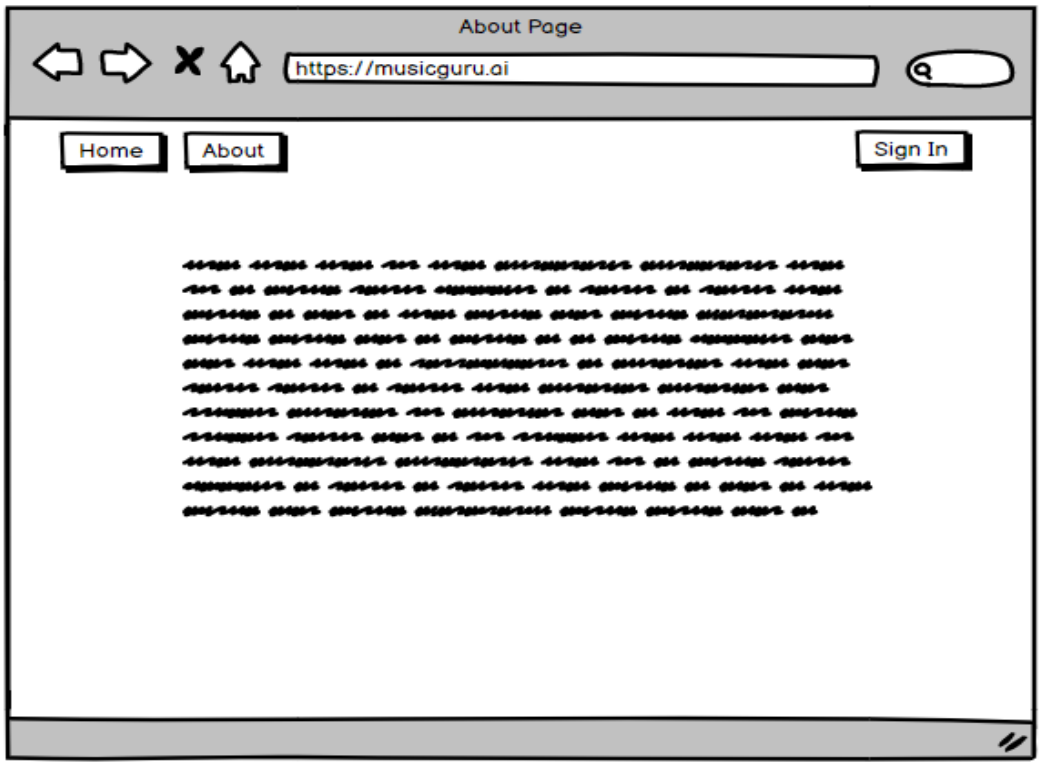


Figure 24: Wireframe diagram for About page

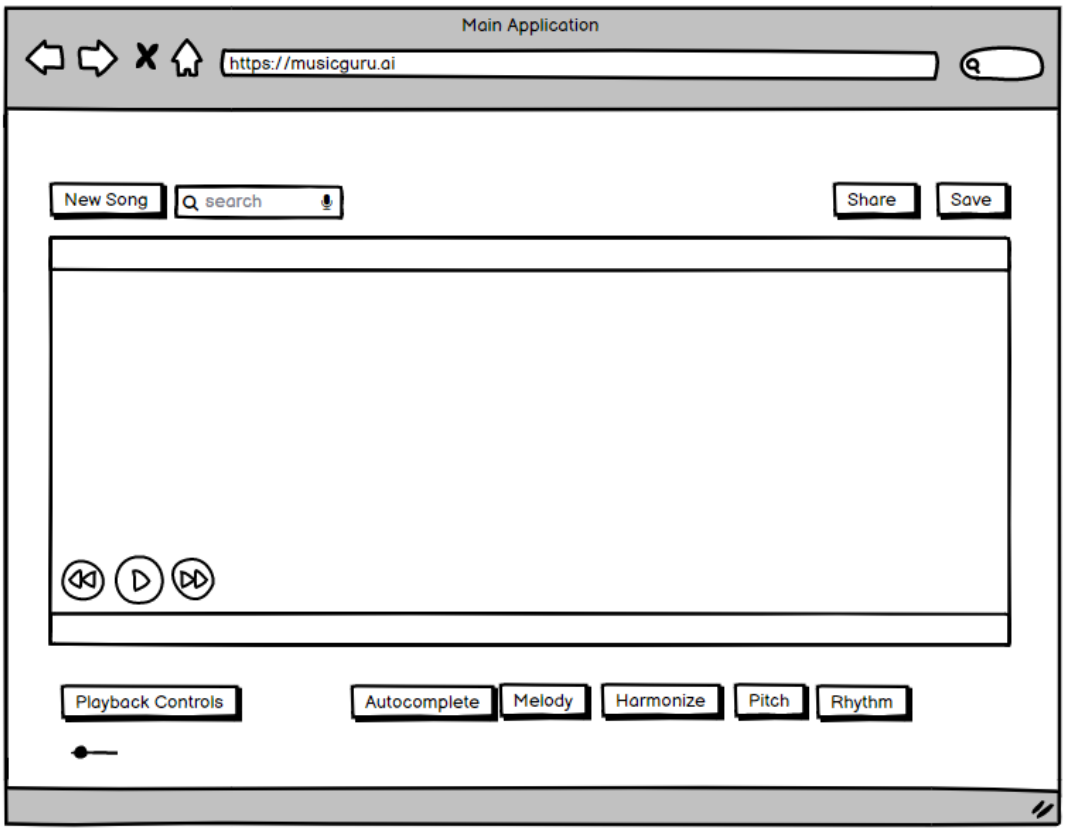


Figure 25: Wireframe diagram for Downstream tasks

4.3 Application Server

We use a web server to host rest APIs for our application and a Python Flask which serves as our application server provides a secure and easy platform for consuming these APIs. It provides annotations using which routes can be designed based on the action (predict, convert, save) required.

Below line shows annotations for the predict action:
@app.route('/predict/midi', methods=['POST'])

4.4 Data-Tier Design

We are using a pretrained model for our Multitasking application. The model serves and saves its predictions for all the downstream tasks to an S3 bucket that we have configured on AWS. As far as user authentication is concerned, we are using AuthO which alleviates the need for a database.

4.4.1 S3 bucket configurations

We created an s3 bucket to store the predictions from the multitasking model. The below screenshot shows the overview of the same:

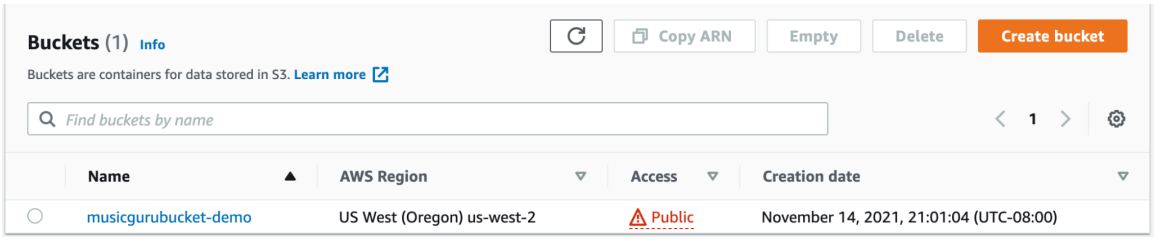


Figure 26: S3 bucket on AWS

Chapter 5. Project Implementation

5.1 User Interface Implementation

We are using Vue JS to implement the front end of our MusicGuru application. We see that the UI comes up on port 8080 (deployment on GCP in progress) on running the command - npm run serve as seen below:

```
DONE  Compiled successfully in 6599ms


  App running at:
  - Local:   http://localhost:8080/
  - Network: http://192.168.0.13:8080/

  Note that the development build is not optimized.
  To create a production build, run yarn build.
```

Figure 27: NodeJS running on port 8080

5.1.1 Home page for User Login

The below screenshot shows the Homepage of our MusicGuru application where the user can sign in to perform the offered downstream tasks:

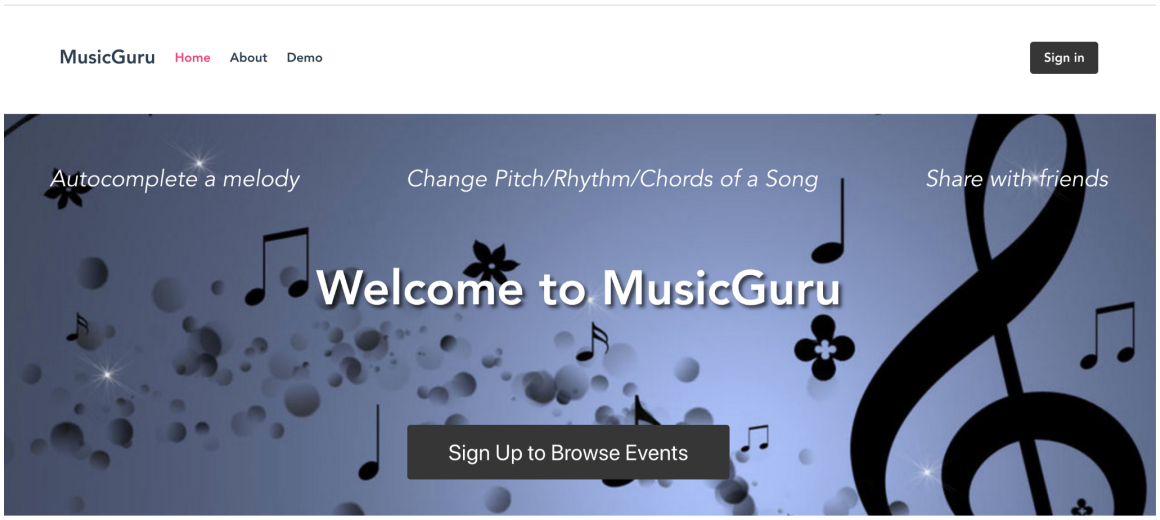


Figure 28: MusicGuru Home page

5.1.2 Authentication using Auth0

In order to implement AuthO into our application, we created a single page web application under https://auth0.com/ as shown in the below figure:

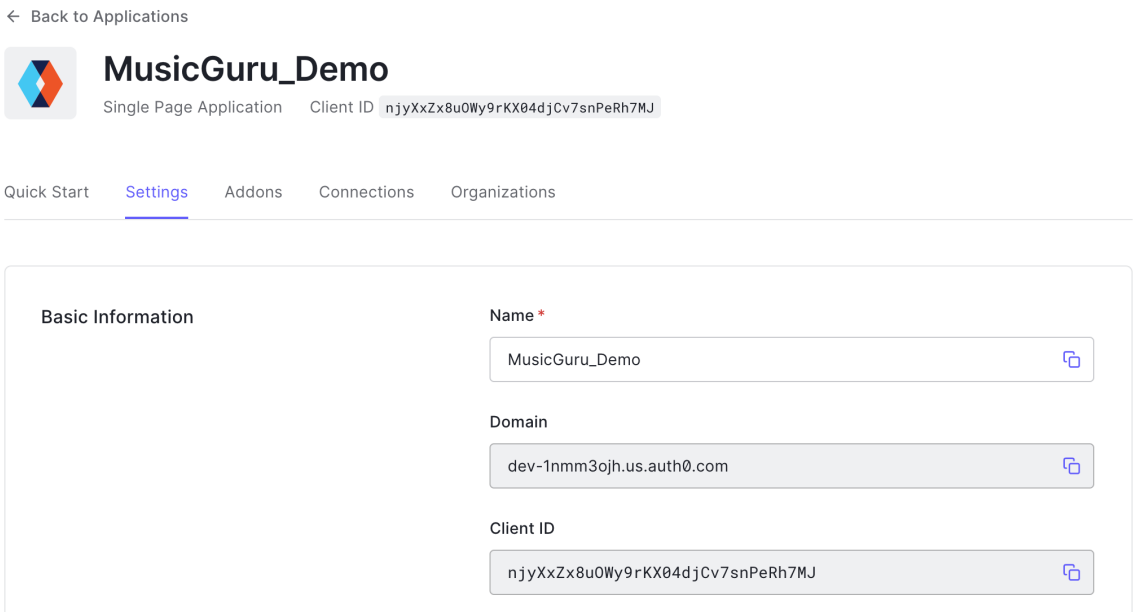


Figure 30: AuthO Single Page Application

Below are the settings to fill in some information that Auth0 needs to configure authentication for our MusicGuru application:

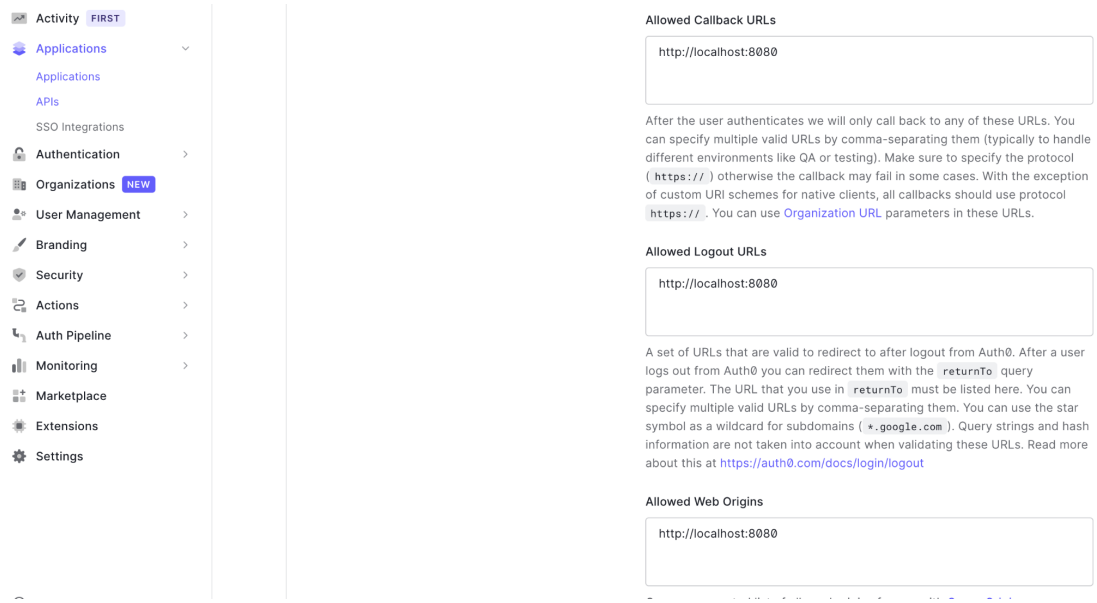


Figure 31: AuthO Single Page Application configurations

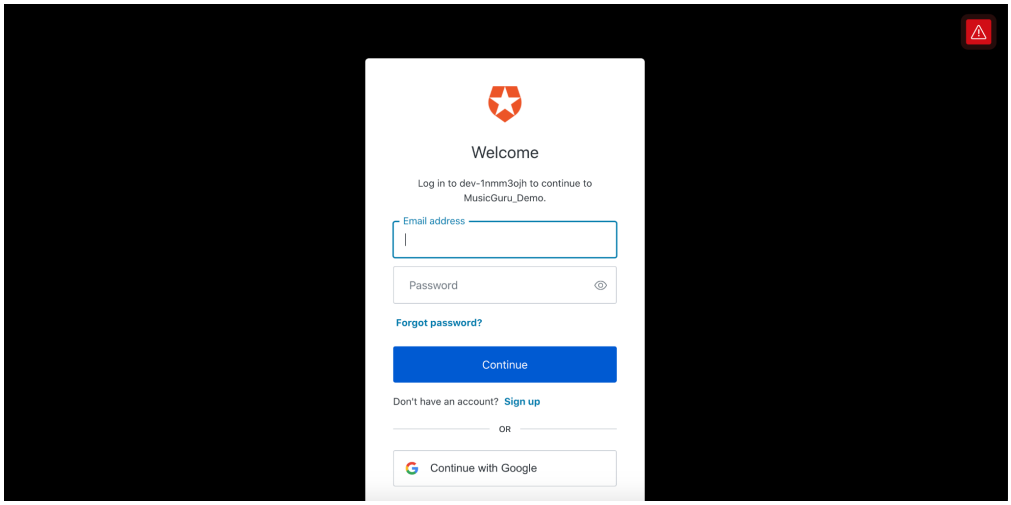


Figure 32: AuthO login

5.1.3 About Page

Figure 33: MusicGuru About page

5.1.4 Downstream tasks

The application uses cards on the home page which when clicked redirects the user to the corresponding downstream task they wish to perform.

Below screenshot shows the downstream application's cards displayed to the user:

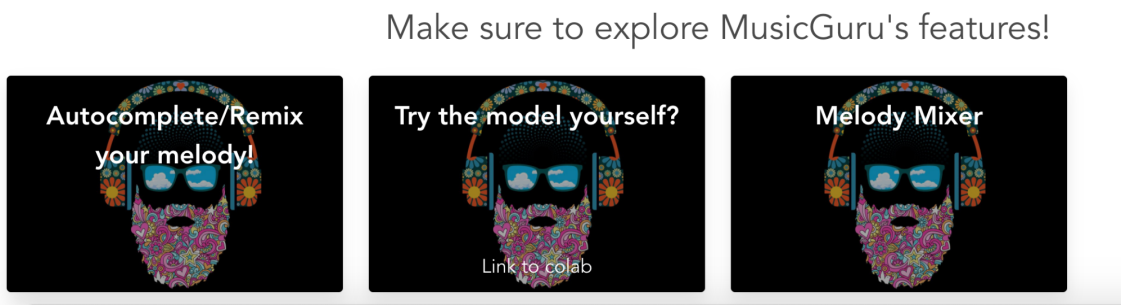


Figure 34: Downstream application cards

The below screenshot shows how the user can choose any song that they want to experiment/remix with from the dropdown or import from their local file.

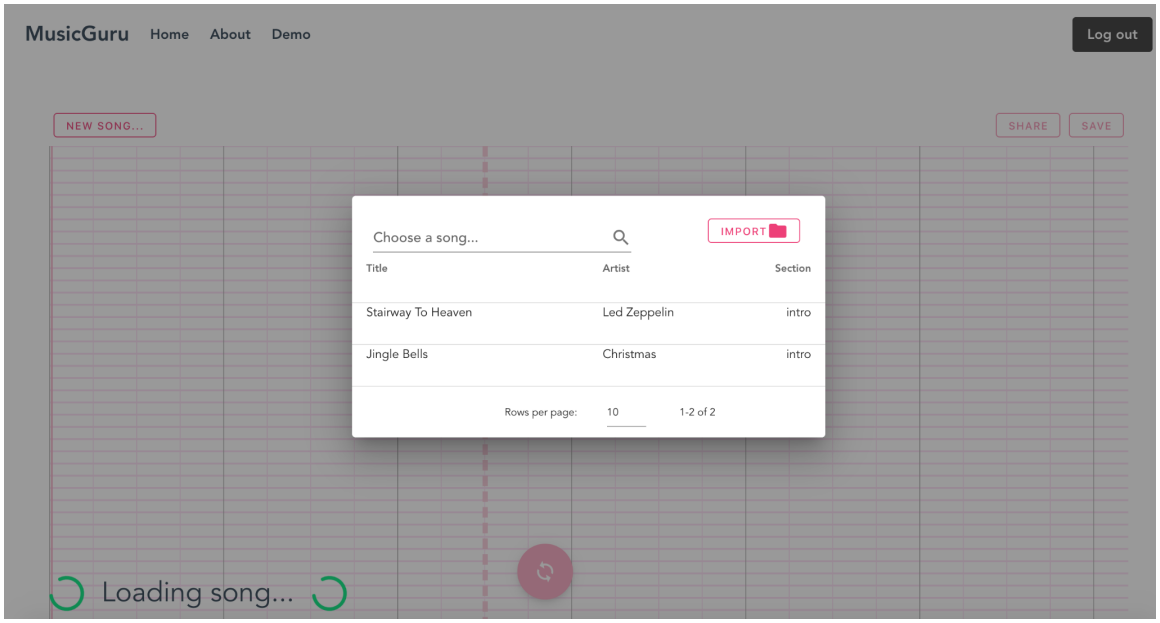


Figure 35: UI for importing a song for a user

After importing the required song, and clicking on the “make music” button for a particular task (Eg: Melody completion), the below screen shows up. This indicates that the model is processing the uploaded input MIDI file and getting ready for the prediction.

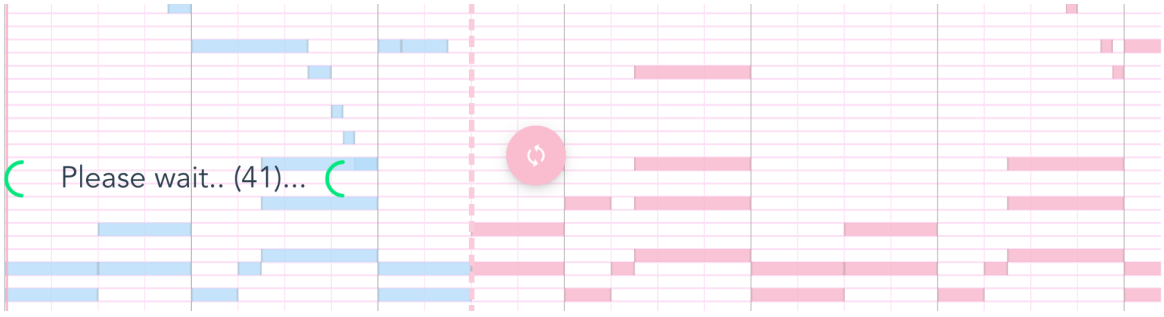


Figure 36: UI for model processing input MIDI file for prediction

Once the prediction is ready, the user can click on the “Play” button to listen to the music generated by the model.

Figure 37: UI Model prediction - Part 1

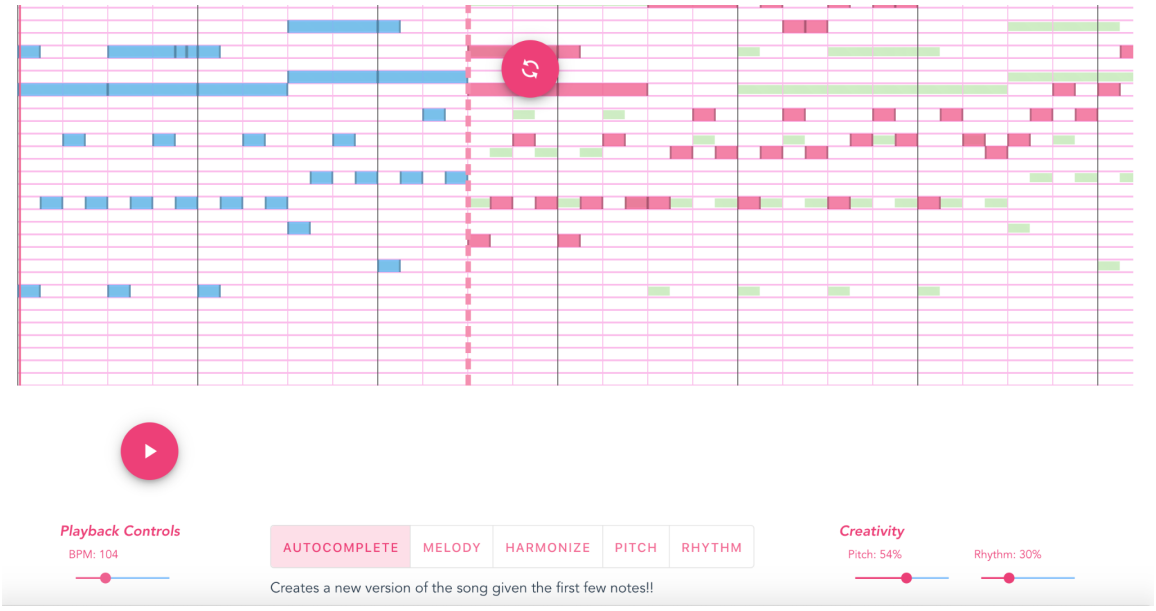


Figure 38: UI Model prediction - Part 2

5.1.5 Twitter Integration

The MusicGuru application is also integrated with Twitter and a logged in user can tweet about a newly generated melody or remixed song, as shown below:

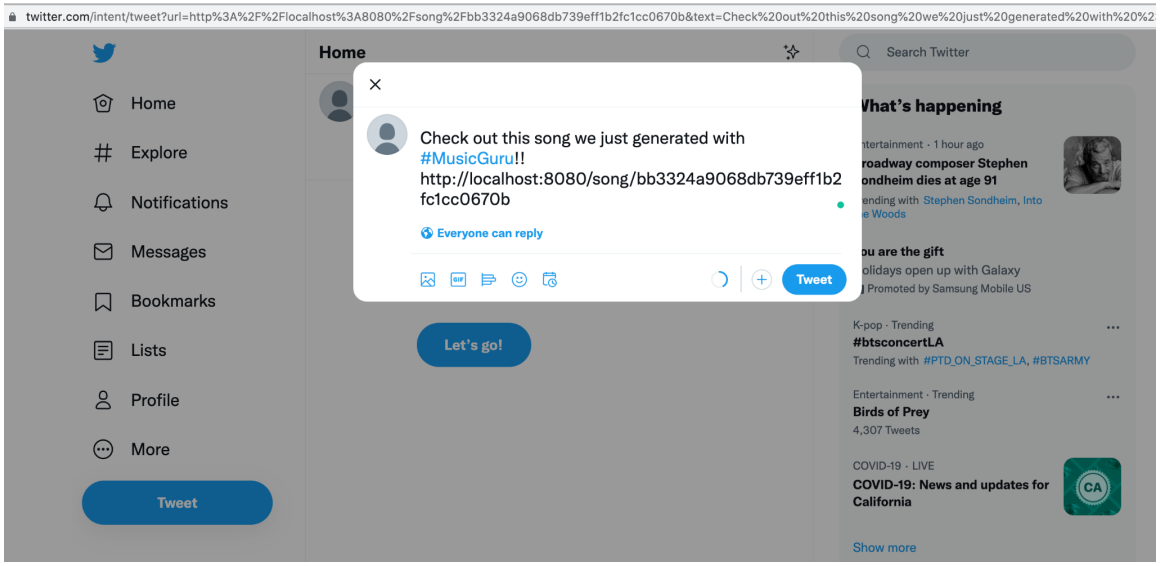


Figure 39: Twitter integration with MusicGuru

5.1.6 Demo Page

This page contains links to the recorded videos explaining what MusicGuru is and its capabilities. Any user visiting the MusicGuru website can view this page without having to log in. Below are screenshots of the demo page.

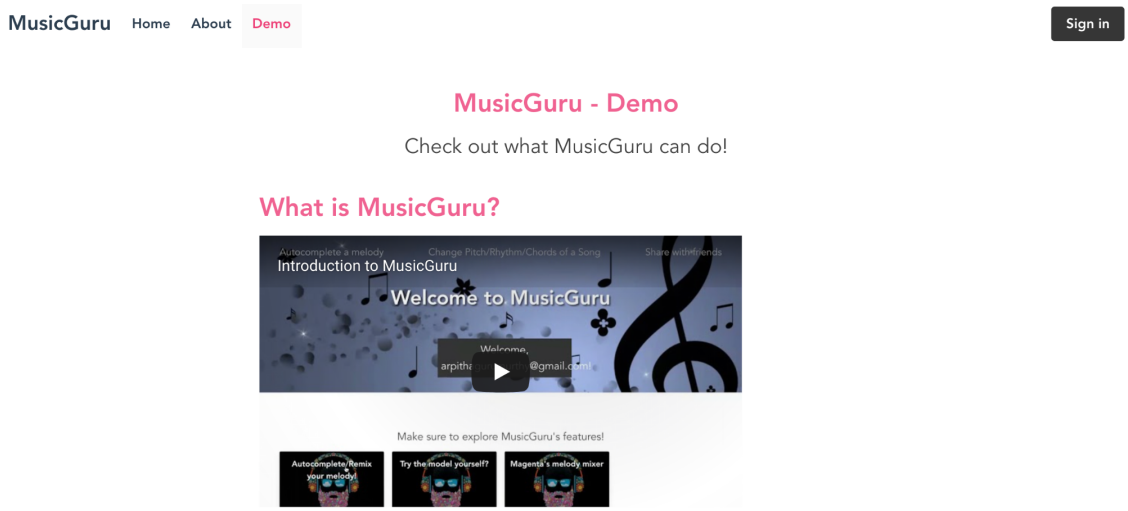


Figure 40: MusicGuru demo page - Part 1

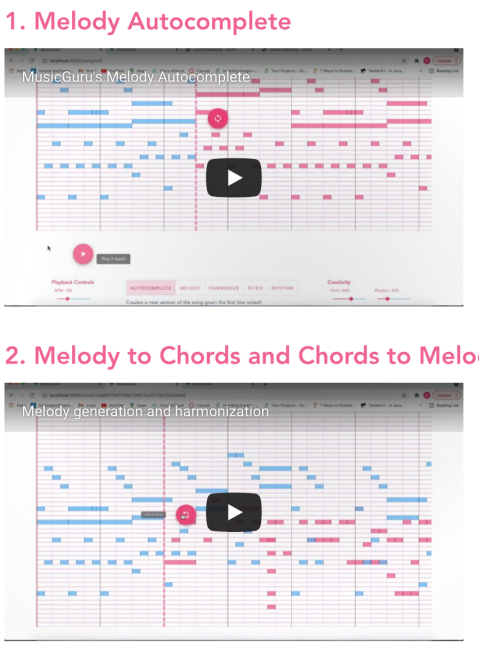


Figure 41: MusicGuru demo page - Part 2

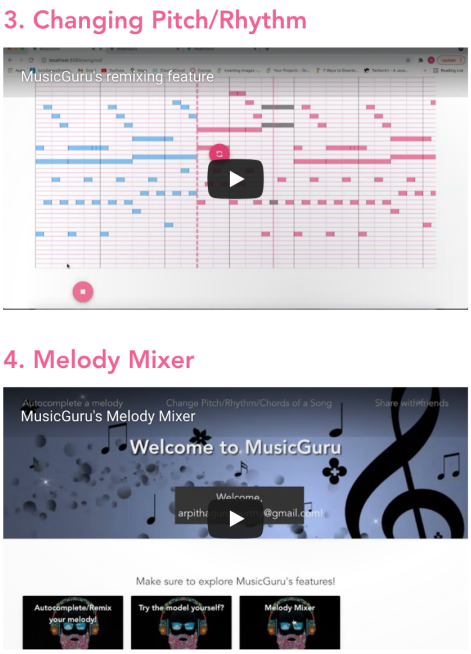


Figure 42: MusicGuru demo page - Part 3

5.2 Backend Implementation

5.2.1 Model Generation

We are using a pre-trained model which consists of an encoder and a decoder. The encoder and decoder together serves as the Sequence to Sequence architecture that performs Melody Generation and Chords Generation. The encoder serves as a BERT that performs all remixing applications and decoder serves as a TransformerXL which performs melody completion. The model is trained on MIDI files (Eg: Lakh MIDI dataset) and then fine tuned to perform each of these tasks.

Model architecture is summarised as:

- Encoder - Bi-directional attention layers
- Decoder - Uni-directional double-stacked attention layers
- Head - Dense layer for token decoding

5.2.2 Model Implementation

The model is implemented based on the below rules:

1. If the input is masked ('msk'), train the encoder.
2. If the input is shifted ('lm'), train the decoder.
3. If the input contains both translation input ('enc') and previous tokens ('dec'), use both encoder and decoder.

We are using Python Flask to implement the back end of our MusicGuru application. We see that the flask server comes up on port 5000 on running the command - python run.py:
Once the model is trained and deployed and, on receiving a request for prediction, the below screen shows up to indicate that the model is processing the input:

At the backend:

```
/Users/arpitha/opt/anaconda3/envs/musicautobot/lib/python3.9/site-packages/torch/_tensor.py:575: UserWarning: floor_divide is deprecated, and w
ill be removed in a future version of pytorch. It currently rounds toward 0 (like the 'trunc' function NOT 'floor'). This results in incorrect
rounding for negative values.
To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for actual floor division, use torch.div(a, b, rounding_mode='floo
r'). (Triggered internally at  /Users/distiller/project/conda/conda-bld/pytorch_1623459065530/work/aten/src/ATen/native/BinaryOps.cpp:467.)
  return torch.floor_divide(self, other)
 |████████████████████████████████████████----------| 65.00% [260/400 00:30<00:16]
```

Figure 43: Model predicting at the backend

The below screenshot indicates that the prediction is written to the S3 bucket from where it is served on UI:

Figure 44: Model prediction at the frontend

5.2.3 Flask server

We are using Python Flask to implement the back end of our MusicGuru application. We see that the flask server comes up on port 5000 on running the command - python run.py:

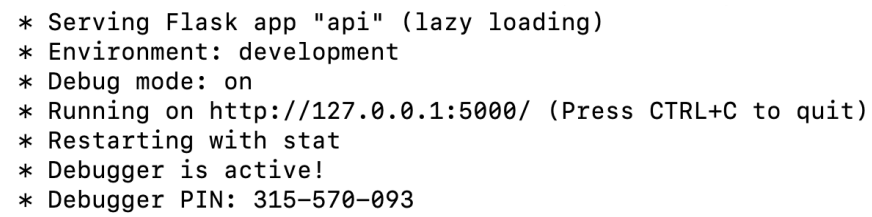


Figure 45: Flask Server running

We can see from the below screenshot that the model predictions are saved and served from the S3 bucket:

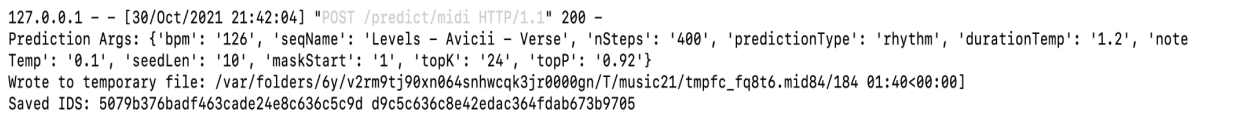


Figure 46: Predict API writing to the S3 bucket

5.3 Data Tier Implementation

The below screenshot shows the overview of the S3 bucket that saves and stores predictions from our multitasking model.

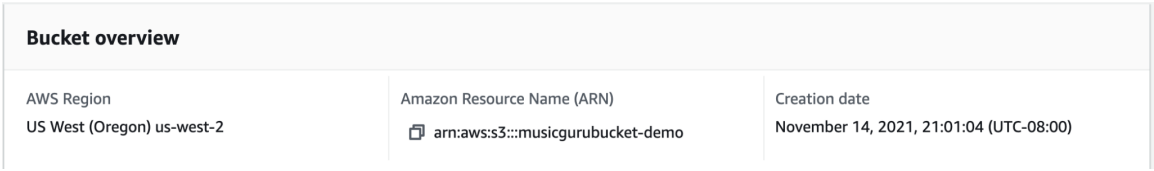


Figure 47: S3 bucket overview at AWS

We have added a user to enable access to the s3 bucket with the following permissions:

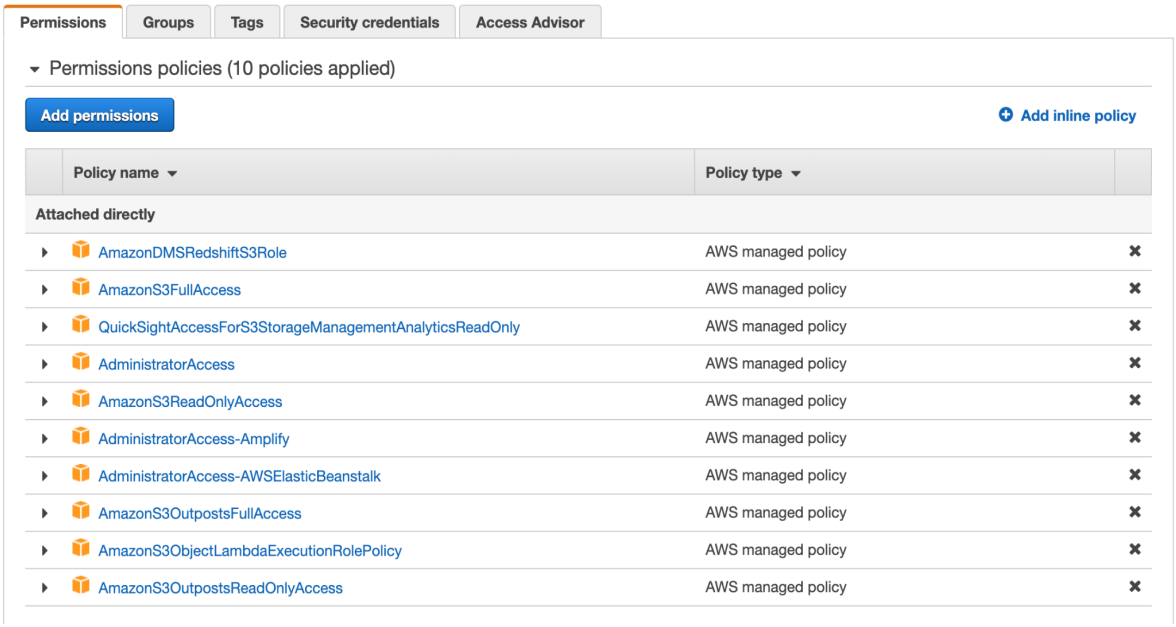


Figure 48: User permissions set for S3 bucket

We also added some policies in the s3 bucket permissions to allow for reads and writes:

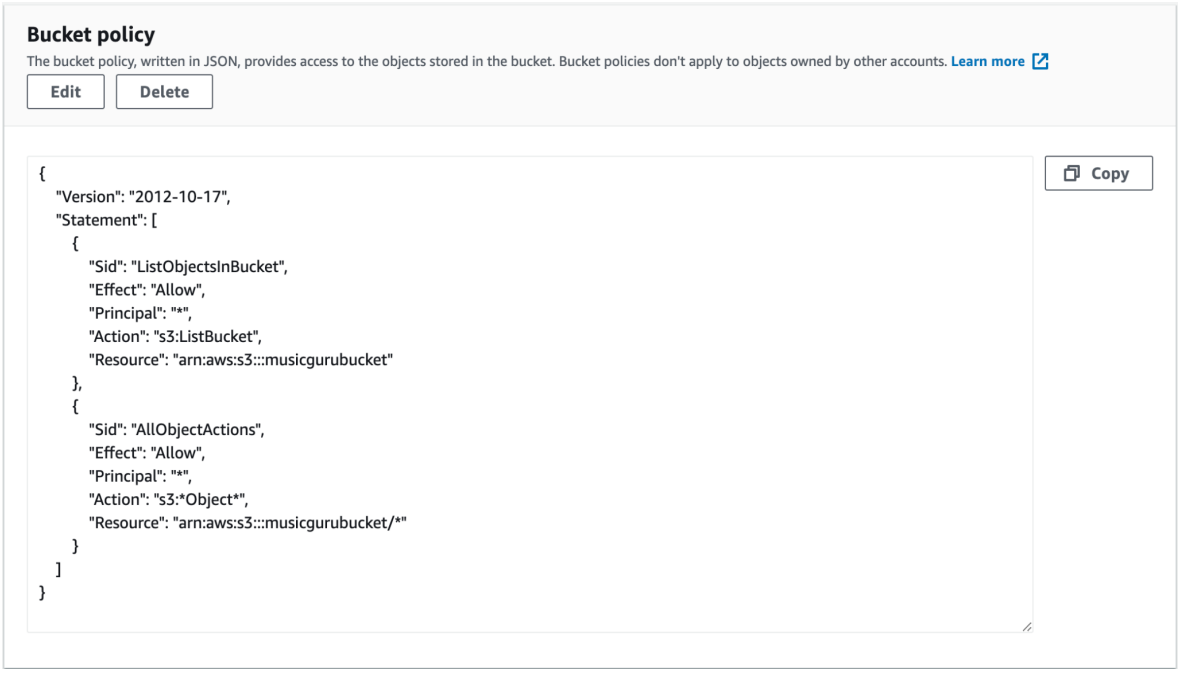


Figure 49: Policy for S3 bucket

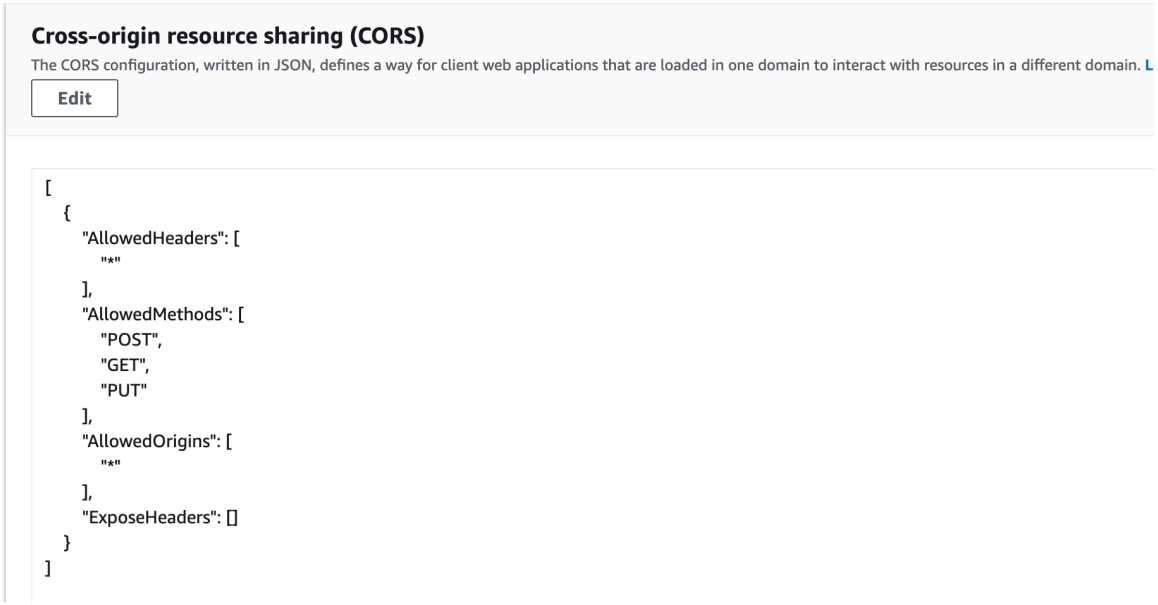


Figure 50: CORS configuration for S3 bucket

All the predictions from the model are stored in the generated/ folder as shown below:

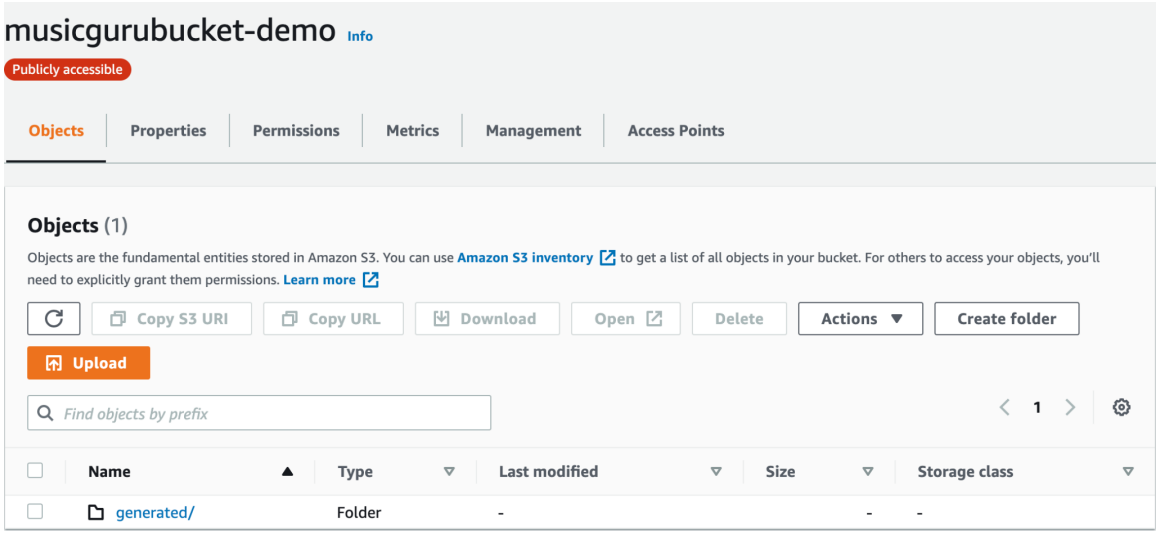


Figure 51: Model predictions saved in S3 bucket - part 1

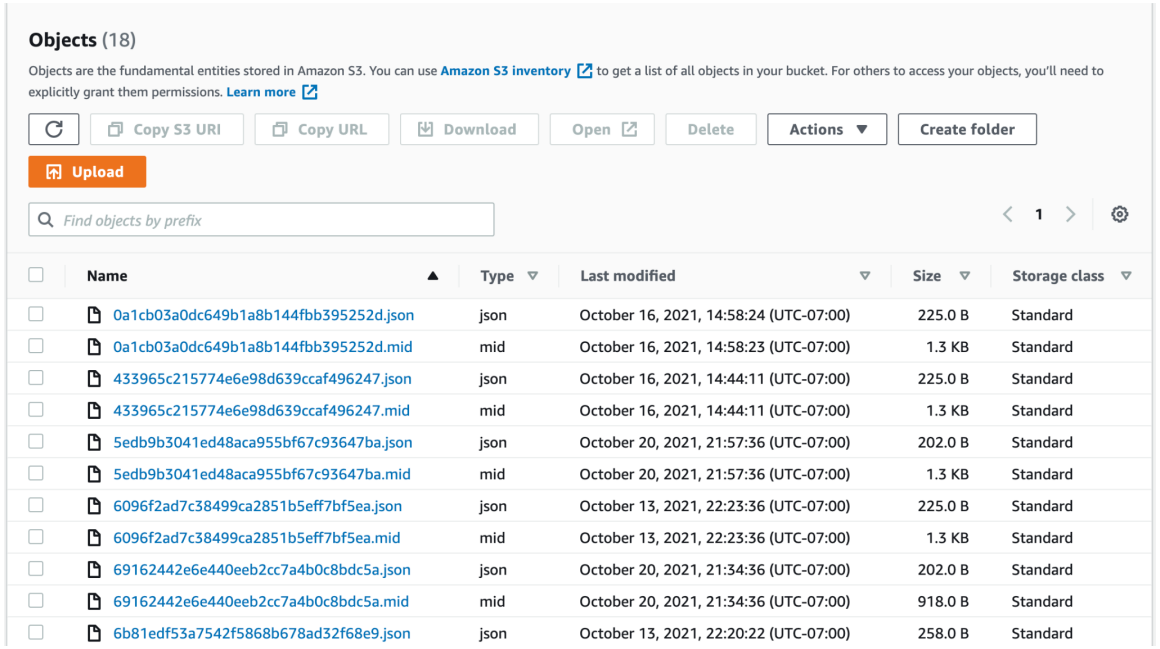


Figure 52: Model predictions saved in S3 bucket - part 2

Chapter 6. Testing and Verification

6.1 Authentication testing

AuthO has been successfully implemented into the application. We can see from Figure - that by default the application is telling the user to login to use all its features. The user is only allowed to view the home, about and demo pages without AuthO login.

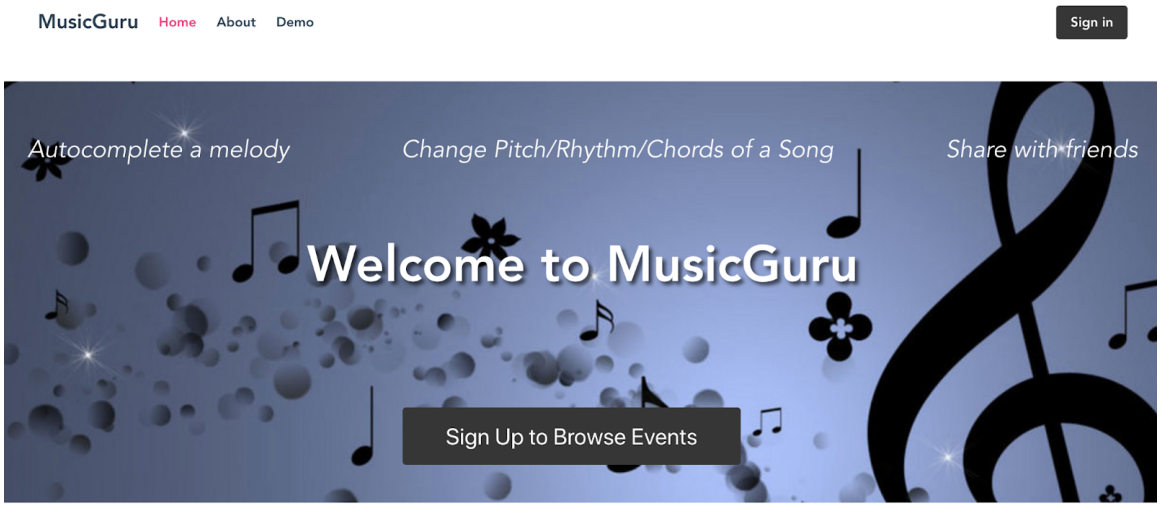


Figure 53: Before AuthO login

On successful authentication, AuthO lets the user login and use all the features of the application. The user on logging in can see the welcome message and also a log out button.

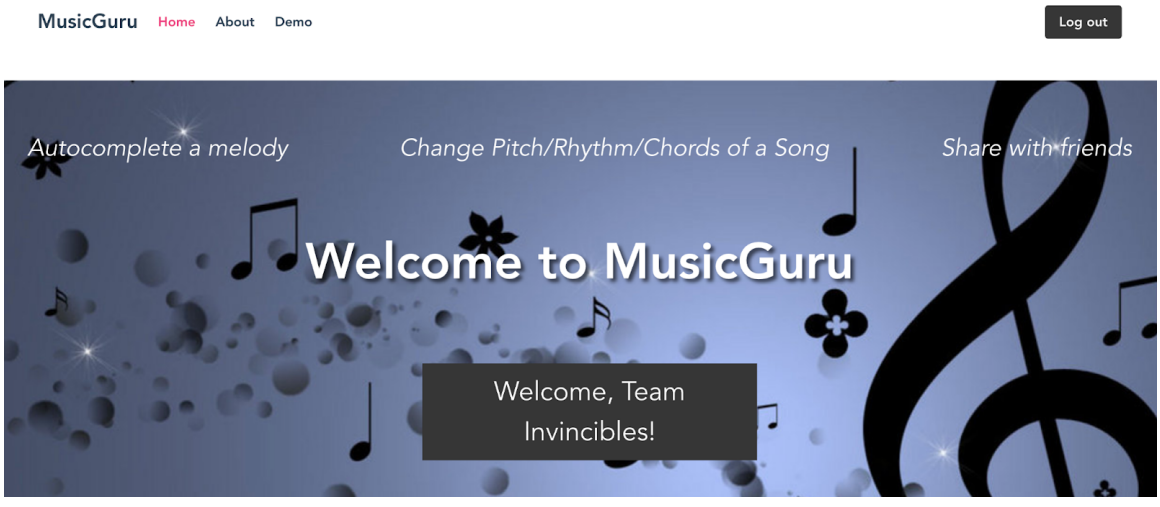


Figure 54: After AuthO login

6.2 Integration testing

Sl no	Component	Test case	Steps	Expected Output	Actual Output
1	Register	User registration	1. Create a request object containing required user information. 2. Invoke the register API and pass in the above created request object	Registration successful if all required fields are present in the request	Successful
2	Authentication	User login with correct password	1. Create a request object containing username and encrypted password. 2. Invoke the login API and pass the above created request object (Uses AuthO)	Login successful	Successful
3		User login with incorrect password	1. Create a request object containing username and encrypted password. 2. Invoke the login API and pass the above created request object (Uses AuthO)	“Invalid password”	Successful
4		Invalid user login	1. Create a request object containing a non existing username and password. 2. Invoke the login API and pass the above created request object (Uses AuthO)	Login in must throw error stating “Account does not exist”	Successful
5		User log out	1. Create a request object containing just the username 2. Invoke the logout API and pass the above created request object (Uses AuthO)	Log out successful	Successful

6	Melody Auto Completion	User performs melody completion	1. Invoke register API to create a user. 2. Invoke the login API to authenticate the user 3. Users upload the song of their choice. 3. In the melody completion, the input is shifted and the appropriate portion (TransformerXL) of the multitasking model is activated and predictions are stored and served from S3. The song is trimmed and then auto completed by the model.	The predicted output by the model is fetched from the S3 and served to the user.	Successful. Prediction is served by the decoder component of the model.
7	Changing the pitch while keeping the rhythm constant	User tries to change the pitch of the song	1. Invoke register API to create a user. 2. Invoke the login API to authenticate the user 3. Users upload the song of their choice. 3. In this use case, the input contains masked tokens in place of pitch, the encoder portion (Bert) of the multitasking model is activated and predictions are stored and served from S3. New notes are predicted for the input song by the model.	The predicted output by the model is fetched from the S3 and served to the user.	Successful. Prediction is served by the encoder component of the model.

8	Changing the rhythm while keeping the pitch constant.	User tries to change the rhythm of the song	1. Invoke register API to create a user. 2. Invoke the login API to authenticate the user 3. Users upload the song of their choice. 3. In this use case, the input contains masked tokens in place of rhythm, the encoder portion (Bert) of the multitasking model is activated and predictions are stored and served from S3. New rhythms are predicted for the input song by the model.	The predicted output by the model is fetched from the S3 and served to the user.	Successful. Prediction is served by the encoder component of the model.
9	Changing the melody of the song with the existing chord progression.	User tries to generate new melodies for the input song	1. Invoke register API to create a user. 2. Invoke the login API to authenticate the user 3. Users upload the song of their choice. 3. In this use case, the input contains both the previous decoder token and translation input, the encoder and decoder portion (Seq2Seq) of the multitasking model is activated and predictions are stored and served from S3. New melodies are predicted for the input song by the model.	The predicted output by the model is fetched from the S3 and served to the user.	Successful. Prediction is served by the encoder and decoder component of the model.

10	Changing the chords of the song with the existing melody progression.	User tries to generate new chords for the input song	1. Invoke register API to create a user. 2. Invoke the login API to authenticate the user 3. Users upload the song of their choice. 3. In this use case, the input contains both the previous decoder token and translation input, the encoder and decoder portion (Seq2Seq) of the multitasking model is activated and predictions are stored and served from S3. New chords are predicted for the input song by the model.	The predicted output by the model is fetched from the S3 and served to the user.	Successful. Prediction is served by the encoder and decoder component of the model.

Table 1: Test plan for Integration Testing

6.3 End to end testing

Sl no	Component	Test case	Steps	Expected Output	Actual Output

1	Manual testing Melody Auto Completion	User performs melody completion	1. User visits MusicGuru website and login via AuthO authentication 2. User then clicks on the event card - “Autocomplete/Remix your melody” and imports the required song. 3. User chooses the Autocomplete task from the downstream tasks. 4. User then clicks on the make music button. 5. After waiting for an adequate amount of time, the user clicks on play. 6. The user can share or save his work now.	The model’s prediction for melody autocomplete is complete and the user can now save/share.	Successful
2	Manual testing changing the pitch/rhythm of song	User tries to change the pitch/rhythm of the song	1. User visits MusicGuru website and login via AuthO authentication 2. User then clicks on the event card - “Autocomplete/Remix your melody” and imports the required song. 3. User chooses the Pitch/**Rhythm** task from the downstream tasks. 4. User then clicks on the make music button. 5. After waiting for an adequate amount of time, the user clicks on play. 6. The user can share or save his work now.	The model’s prediction for Pitch/Rhythm is complete and the user can now save/share.	Successful

3	Manual testing translating melody to chords and vice versa.	User tries to change the melody/chords of the song.	1. User visits MusicGuru website and login via AuthO authentication 2. User then clicks on the event card - "Autocomplete/Remix your melody" and imports the required song. 3. User chooses **Melody/Harmonize** tasks from the downstream tasks. 4. User then clicks on the make music button. 5. After waiting for an adequate amount of time, the user clicks on play. 6. The user can share or save his work now.	The model's prediction for Melody/Chords is complete and the user can now save/share.	Successful

Table 2: Test plan for end to end testing

6.4 Downstream tasks for various types of MIDI files

11	Different MIDI files	Experimenting with MIDI files of different genres - **Indian Classical**	1. Upload the song with Indian classical genre and invoke the downstream tasks	The model should predict as expected for all tasks.	Successful
12		Experimenting with MIDI files of different genres - **Rock**	1. Upload the song with rock genre and invoke the downstream tasks	The model should predict as expected for all tasks.	Successful

Table 3: Test plan for using different types of MIDI files

Chapter 7. Performance and Benchmarks

7.1 Testing achieved latency and scalability

The time taken by the Python flask server on receiving the request to predict the next token on GCP was significantly lesser than the time taken by it locally.

The use of S3 bucket for saving and serving predictions by the multitasking model helps in scalability as the predictions can be accessed by any number of users. The work of the user is safe on the S3 bucket and hence MusicGuru is fault tolerant.

7.2 Availability testing

Since the application is deployed on GCP, it is available to the users at any point in time. The VM instance chosen is c2-standard-4, with Intel Cascade Lake CPU and it ensures minimal downtime from the limited testing done for academic purposes.

Chapter 8. Deployment

8.1 Deployment on GCP

The application was initially deployed locally to verify that it is running end to end. This was then migrated to a cloud environment (GCP) for public access. In order to achieve this, first an S3 bucket was set up to store and serve predictions. A python Flask Web Server was deployed to host all the REST APIs. For the front end, a VueJS application is also hosted that connects to the backend.

Below is the screenshot of the VM Instance created on GCP that will host the MusicGuru application:

Figure 55: VM instance hosting MusicGuru application

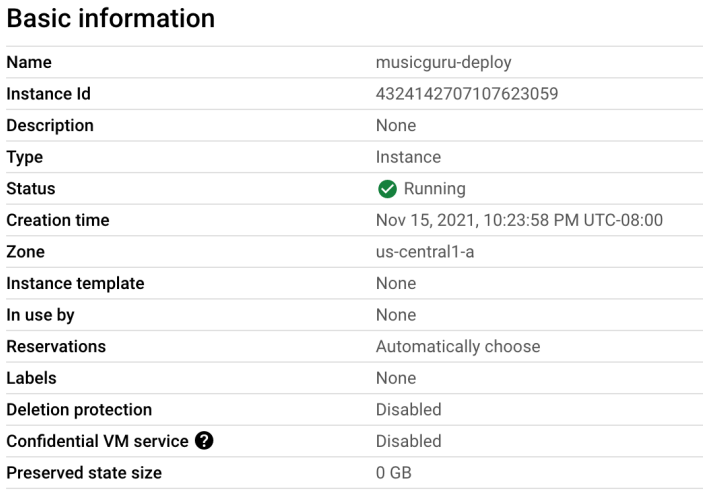


Figure 56: VM instance basic settings and information

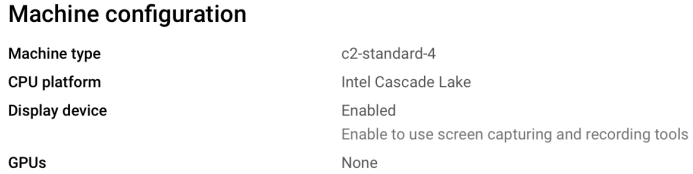


Figure 57: VM instance configurations

We then ssh into the GCP instance and clone the MusicGuru Git repository. The below screenshot shows the same:

```
arpithagurumurthy_ml@musicguru-deploy:~$ ls
arpithagurumurthy_ml@musicguru-deploy:~$ git clone https://github.com/bhuvanabasapur/TeamInvinsibles.git
Cloning into 'TeamInvinsibles'...
Username for 'https://github.com': arpithagurumurthy
Password for 'https://arpithagurumurthy@github.com':
remote: Enumerating objects: 495, done.
remote: Counting objects: 100% (495/495), done.
remote: Compressing objects: 100% (409/409), done.
remote: Total 495 (delta 121), reused 436 (delta 71), pack-reused 0
Receiving objects: 100% (495/495), 14.26 MiB | 39.05 MiB/s, done.
Resolving deltas: 100% (121/121), done.
arpithagurumurthy_ml@musicguru-deploy:~$
```

Figure 58: Cloning MusicGuru Git repository on GCP

We also create firewall rules to enable access to ports 5000, 8080 and 8081 for the flask backend, Vue JS frontend and Magenta respectively.

☐	allow-5000	Ingress	Apply to all	IP ranges: 0.(	tcp:5000	Allow	1000	default	Off
☐	allow-8080	Ingress	Apply to all	IP ranges: 0.(	tcp:8080	Allow	1000	default	Off
☐	allow-8081	Ingress	Apply to all	IP ranges: 0.(	tcp:8081	Allow	1000	default	Off

Figure 59: Firewall rules on GCP

We then install all the required dependencies by issuing the command “conda env update -f environment.yml”.

```
Using Anaconda API: https://api.anaconda.org
Fetching package metadata .............
Solving package specifications: .
_libgcc_mutex- 100% |################################################################################################################################################| Time: 0:00:00   5.51 MB/s
blas-1.0-mkl.t 100% |################################################################################################################################################| Time: 0:00:00  10.67 MB/s
ca-certificate 100% |################################################################################################################################################| Time: 0:00:00   6.21 MB/s
intel-openmp-2 100% |################################################################################################################################################| Time: 0:00:00  41.86 MB/s
ld_impl_linux- 100% |################################################################################################################################################| Time: 0:00:00 107.55 MB/s
libgfortran-ng 100% |################################################################################################################################################| Time: 0:00:00  33.46 MB/s
libgfortran4-7 100% |################################################################################################################################################| Time: 0:00:00 114.37 MB/s
libstdcxx-ng-9 100% |################################################################################################################################################| Time: 0:00:00 108.67 MB/s
pytorch-mutex- 100% |################################################################################################################################################| Time: 0:00:00   5.77 MB/s
tzdata-2021e-h 100% |################################################################################################################################################| Time: 0:00:00  69.80 MB/s
libgcc-ng-9.1. 100% |################################################################################################################################################| Time: 0:00:00  58.28 MB/s
mkl-2021.4.0-h 100% |################################################################################################################################################| Time: 0:00:04  54.89 MB/s
brotli-1.0.9-h 100% |################################################################################################################################################| Time: 0:00:00 102.21 MB/s
bzip2-1.0.8-h7 100% |################################################################################################################################################| Time: 0:00:00  76.05 MB/s
cudatoolkit-10 100% |################################################################################################################################################| Time: 0:00:10  54.98 MB/s
expat-2.4.1-h2 100% |################################################################################################################################################| Time: 0:00:00  83.52 MB/s
giflib-5.2.1-h 100% |################################################################################################################################################| Time: 0:00:00  57.78 MB/s
gmp-6.2.1-h253 100% |################################################################################################################################################| Time: 0:00:00  61.88 MB/s
icu-58.2-he671 100% |################################################################################################################################################| Time: 0:00:00  57.12 MB/s
jpeg-9d-h7f872 100% |################################################################################################################################################| Time: 0:00:00  98.45 MB/s
lame-3.100-h7b 100% |################################################################################################################################################| Time: 0:00:00 106.50 MB/s
libffi-3.3-he6 100% |################################################################################################################################################| Time: 0:00:00  63.22 MB/s
libiconv-1.15- 100% |################################################################################################################################################| Time: 0:00:00 106.54 MB/s
libsodium-1.0. 100% |################################################################################################################################################| Time: 0:00:00 104.94 MB/s
libtasn1-4.16. 100% |################################################################################################################################################| Time: 0:00:00  62.53 MB/s
libunistring-0 100% |################################################################################################################################################| Time: 0:00:00 103.56 MB/s
libuuid-1.0.3- 100% |################################################################################################################################################| Time: 0:00:00  31.72 MB/s
libuv-1.40.0-h 100% |################################################################################################################################################| Time: 0:00:00 111.84 MB/s
libwebp-base-1 100% |################################################################################################################################################| Time: 0:00:00  93.03 MB/s
libxcb-1.14-h7 100% |################################################################################################################################################| Time: 0:00:00  92.03 MB/s
lz4-c-1.9.3-h2 100% |################################################################################################################################################| Time: 0:00:00  83.20 MB/s
ncurses-6.3-he 100% |################################################################################################################################################| Time: 0:00:00 103.64 MB/s
openssl-1.1.1l 100% |################################################################################################################################################| Time: 0:00:00 108.55 MB/s
pcre-8.45-h295 100% |################################################################################################################################################| Time: 0:00:00  91.63 MB/s
xz-5.2.5-h7b64 100% |################################################################################################################################################| Time: 0:00:00  99.71 MB/s
yaml-0.2.5-h7b 100% |################################################################################################################################################| Time: 0:00:00  63.78 MB/s
zlib-1.2.11-h7 100% |################################################################################################################################################| Time: 0:00:00  72.72 MB/s
glib-2.69.1-h5 100% |################################################################################################################################################| Time: 0:00:00 107.20 MB/s
```

Figure 60: Installing dependencies for MusicGuru on GCP - Part 1

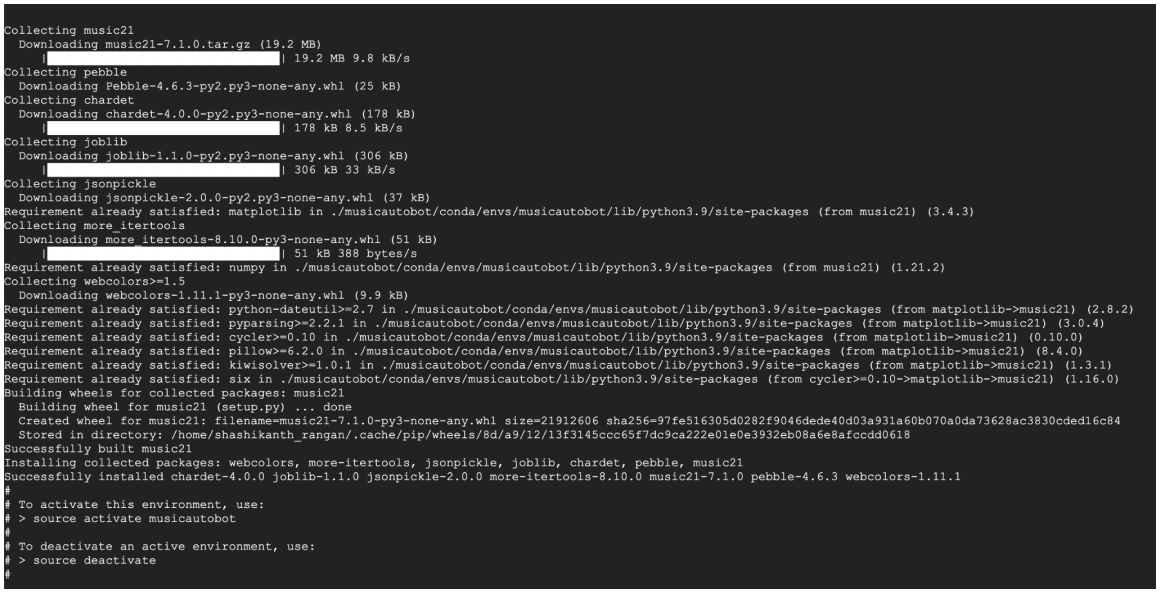


Figure 61: Installing dependencies for MusicGuru on GCP - Part 2

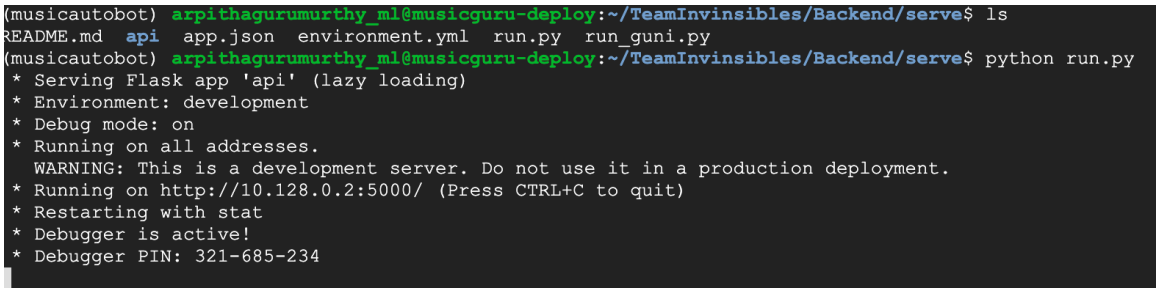


Figure 62: Python flask server coming up on port 5000 on GCP

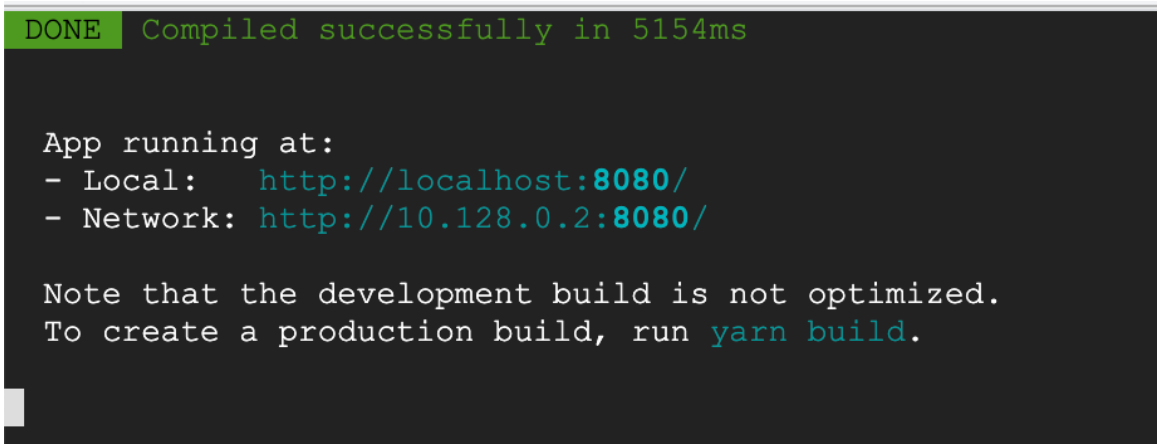


Figure 63: Vue JS frontend coming up on port 8080 on GCP

The below screenshot shows that the application comes up on VM instance's external IP on port 8080 at 35.72.49.148

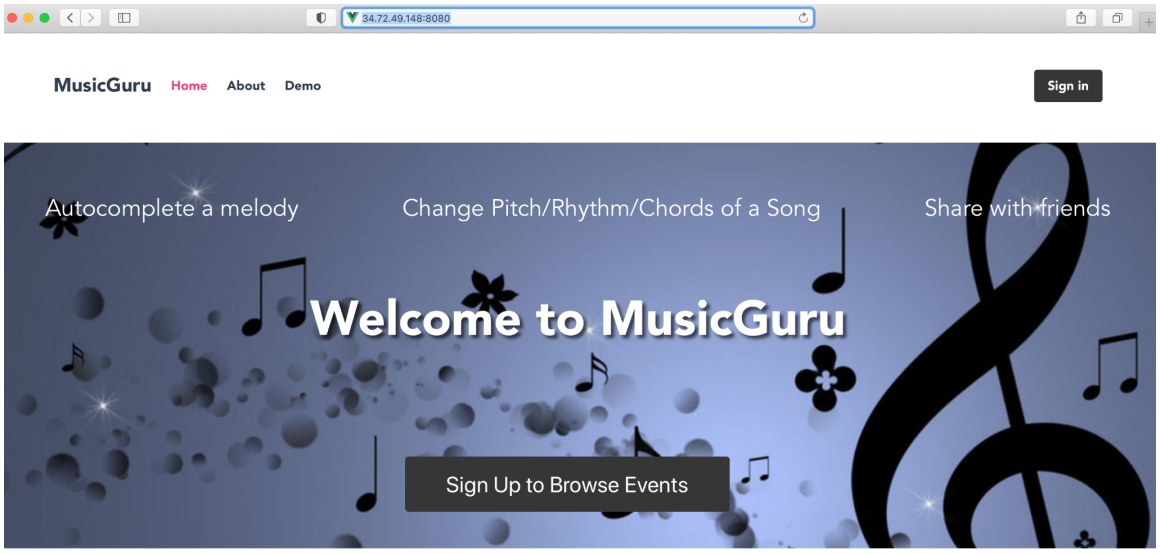


Figure 64: MusicGuru application on GCP's external IP

Chapter 9. Summary, Conclusions, and Recommendations

9.1 Summary

Advancements in the field of Natural Language Processing have proved to produce great results in generating text. One of the reasons for its success is the Transformer architecture. Passing a few words through a Transformer model can help generate whole paragraphs of text. A Transformer architecture is chosen for the project since it trains faster and has a larger long term memory when compared to some of the previous language models.

The project aims at leveraging these powerful language models and applying them to the music domain and implementing several music related downstream tasks.

The base model uses a TransformerXL to take a sequence of music notes (midi files) as input and based on these, predict the next note and generate melody. On top of the Transformer architecture, the model is trained on multiple music related tasks like Autocomplete, Melody, Harmonization, tuning the Pitch and Rhythm of an input track.

The BERT model is used for token masking/ missing tokens and is best suited for remixing songs, changing the pitch while keeping the rhythm constant and changing the rhythm while keeping the pitch constant.

A Seq2Seq model can be considered as a combination of the Bert and TransformerXL models and most importantly, a backbone for the multitasking model, wherein the encoder can be reused to train the required Bert model and the decoder can be reused to train the TransformerXL model.

9.2 Conclusion

The goal of this project was multimodal music understanding using intelligent DL techniques. To take it one step further, we built a model that can jointly handle multiple music related downstream tasks to solve issues concerning extensive data and computational requirements. The concept of multimodality and multitasking in the field of music help in better understanding of music structure in terms of its melody, rhythm, beat and chord. This in turn results in better performance for tasks like changing pitch/ rhythm, chords and melody completion as the model has a provision of visualizing, hearing and finally analysing the same input.

9.3 Recommendations for Further Research

One of the potential extensions to the current research work could be to combine the two resultant models for multitasking and multimodality and create an ensemble model that possesses both the functionalities and can perform multiple downstream tasks on different input types- audio, video, text and midi files at the same time.

Another possible area of future scope can be using OctupleMIDI encoding for the purpose of multitasking, instead of vanilla MIDI files for input to the models.

There is also a potential of extending the downstream tasks and including tasks like genre classification and style classification and using image input files for these classification tasks. Music genres and styles can be used to organize music files in a better way by making use of the commonalities they share. The databases of music streaming services are huge containing millions of music files. Genre and style classification can greatly help such services in organizing their songs better by clustering songs with similar genres/style together. Sorting songs by their genres also helps in making their retrieval and search faster.

Glossary

BERT (Bidirectional Encoder Representations from Transformers): A model architecture for text representation. A trained BERT model can act as part of a larger model for text classification or other ML tasks.

Convolutional Neural Network: A neural network in which at least one layer is a convolutional layer.

Decoder: In general, any ML system that converts from a processed, dense, or internal representation to a more raw, sparse, or external representation. Decoders are often a component of a larger model, where they are frequently paired with an encoder.

Deep model: A type of neural network containing multiple hidden layers.

Encoder: In general, any ML system that converts from a raw, sparse, or external representation into a more processed, denser, or more internal representation. Encoders are often a component of a larger model, where they are frequently paired with a decoder.

Ensemble: A merger of the predictions of multiple models.

Epoch: A full training pass over the entire dataset such that each example has been seen once. Thus, an epoch represents N/batch size training iterations, where N is the total number of examples.

Fine Tuning: Perform a secondary optimization to adjust the parameters of an already trained model to fit a new problem. Fine tuning often refers to refitting the weights of a trained unsupervised model to a supervised model.

GPT (Generative Pre-trained Transformer): A family of Transformer-based large language models developed by OpenAI.

Machine Learning: A program or system that builds (trains) a predictive model from input data. The system uses the learned model to make useful predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model. Machine learning also refers to the field of study concerned with these programs or systems.

Neural Network: A model that, taking inspiration from the brain, is composed of layers (at least one of which is hidden) consisting of simple connected units or neurons followed by nonlinearities.

Pre-trained model: Models or model components (such as embeddings) that have already been trained.

Sequence model: A model whose inputs have a sequential dependence.

Sequence-to-sequence task: A task that converts an input sequence of tokens to an output sequence of tokens.

Token: In a language model, the atomic unit that the model is training on and making predictions on.

Training: The process of determining the ideal parameters comprising a model.

Transfer Learning: Transferring information from one machine learning task to another. Transfer learning might involve transferring knowledge from the solution of a simpler task to a more complex one, or involve transferring knowledge from a task where there is more data to one where there is less data.

Transformer: A neural network architecture that relies on self-attention mechanisms to transform a sequence of input embeddings into a sequence of output embeddings without relying on convolutions or recurrent neural networks.

References

[1] Oramas, Sergio, Oriol Nieto, Francesco Barbieri, and Xavier Serra, “Multi-label music genre classification from audio, text, and images using deep features”, 2017, arXiv preprint arXiv:1707.04916.
*Note: This paper was presented at the 18th International Society for Music Information Retrieval Conference and published at ‘https://ccrma.stanford.edu/’

- The paper introduces a new dataset called MuMu with over 31 thousand albums classified among 250 genres. For each album, they have aggregated multimodal data which include cover images, corresponding amazon text reviews, and audio tracks.
- They experiment with different combinations of inputs and demonstrate that the model performs best when all trained on all modalities.
- While most of the previous works focus on classifying the music elements into broad genres, this paper takes into account the fine grain details of music and classifies them into multilabel genres.

[2] Zeng, M., Tan, X., Wang, R., Ju, Z., Qin, T. and Liu, T.Y., “MusicBERT: Symbolic Music Understanding with Large-Scale Pre-Training”, 2021, arXiv preprint arXiv:2106.05630.
*Note: The article is accepted by ACL 2021 Findings

- The paper explains the importance of understanding the melody, rhythm and music structure to perform tasks like genre classification, style classification, melody completion or accompaniment suggestion.
- It shows how pre-training the model using efficient encoding and masking strategies helps in improving its performance by a large scale.
- The authors propose OctupleMIDI encoding which encodes all 8 elements of a musical note (time signature, tempo, bar and position, instrument, pitch, duration, and velocity).
- They also propose a bar-level masking strategy.
- They develop a pre-trained model called MusicBERT that is incorporated with the above discussed OctupleMIDI encoding and bar level masking strategies for music understanding.

[3] Cheng-Zhi Anna Huang, David Duvenaud, and Krzysztof Z Gajos. 2016. “Chordripple: Recommending chords to help novice composers go beyond the ordinary.” In Proceedings of the 21st International Conference on Intelligent User Interfaces. 241–250.

- The paper proposes a creativity support tool called ‘chordripple’ that helps composers by making chord recommendations.
- They leverage the word2vec model from NLP to achieve the same.

[4] Sephora Madjiheurem, Lizhen Qu, and ChristianWalder. 2016. “Chord2vec: Learning musical chord embeddings". In Proceedings of the constructive machine learning workshop at 30th conference on neural information processing systems.

- The above paper focuses mainly on chord representations to generate embeddings of symbolic music.
- It makes chord recommendations that aim to be both diverse and appropriate to the current context.

[5] Tatsunori Hirai and Shun Sawada. 2019. “Melody2Vec: Distributed Representations of Melodic Phrases based on Melody Segmentation”. Journal of InformationProcessing 27 (2019), 278–286.

- This paper is based on a motif-based approach to generate embeddings for symbolic music.
- It combines pitch, rhythm and melody aspects of symbolic music to generate embeddings.

[6] Shulei Ji, Jing Luo, and Xinyu Yang, “A comprehensive survey on deep music generation: Multilevel representations, algorithms, evaluations, and future directions”, 2020, arXiv preprint arXiv:2011.06801.
*Note: The article is published at ACM

- This is a survey paper that helps in understanding the current research and development in the field of music generation using deep learning.
- It talks about the three stages involved in music generation namely score generation, performance generation, and audio generation.
- It also summarizes the list of techniques available for music generation and their shortcomings.

[7] Hongru Liang, Wenqiang Lei, Paul Yaozhu Chan, Zhenglu Yang, Maosong Sun, and Tat-Seng Chua. 2020. “Pirhdy: Learning pitch-, rhythm-, and dynamics-aware embeddings for symbolic musi”c. In Proceedings of the 28th ACM International Confer- ence on Multimedia, pages 574–582.

- This is a State-of-the Art model to generate embeddings for symbolic music.
- This model combines pitch, rhythm and dynamics components of music to generate embeddings.

[8] Hu, R., & Singh, A. (2021). “UniT: Multimodal Multitask Learning with a Unified Transformer”. arXiv preprint arXiv:2102.10772.
*Note: This is a research paper by Facebook AI Research (FAIR) conducted in March, 2021.

- This paper proposes UniT, a Unified Transformer model that simultaneously learns multiple tasks across different domains.
- The proposed model is jointly trained end-to-end, handles 7 tasks on 8 datasets and multimodality.
- It works on tasks such as object detection, visual question answering (VQA), visual entailment, & natural language understanding related tasks.
- It is a single unified encoder-decoder model built using a transformer framework.

- This paper focuses on having the same model parameters for all tasks instead of working on fine-tuning individual models for each of the tasks.

[9] Lu, Jiasen, Goswami, Vedanuj, Rohrbach, Marcus, Parikh, Devi, and Lee, Stefan. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2020. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Web.

- This paper focuses on jointly training four tasks based on a single model.
- It describes how fine tuning for downstreaming tasks from a single multi task model achieves better performance than the state of the art.

[10] https://towardsdatascience.com/a-multitask-music-model-with-bert-transformer-xl-and-seq2seq-3d80bd2ea08e

[11] https://github.com/bearpelican/musicautobot

[12] https://musicinformationretrieval.com/symbolic_representations.html#:~:text=Symbolic%20music%20representations%20comprise%20anyalphabet%20of%20letters%20or%20symbols.

[13] https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)

[14] https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

[15] https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/

[16] https://auth0.com/blog/beginner-vuejs-tutorial-with-user-login/

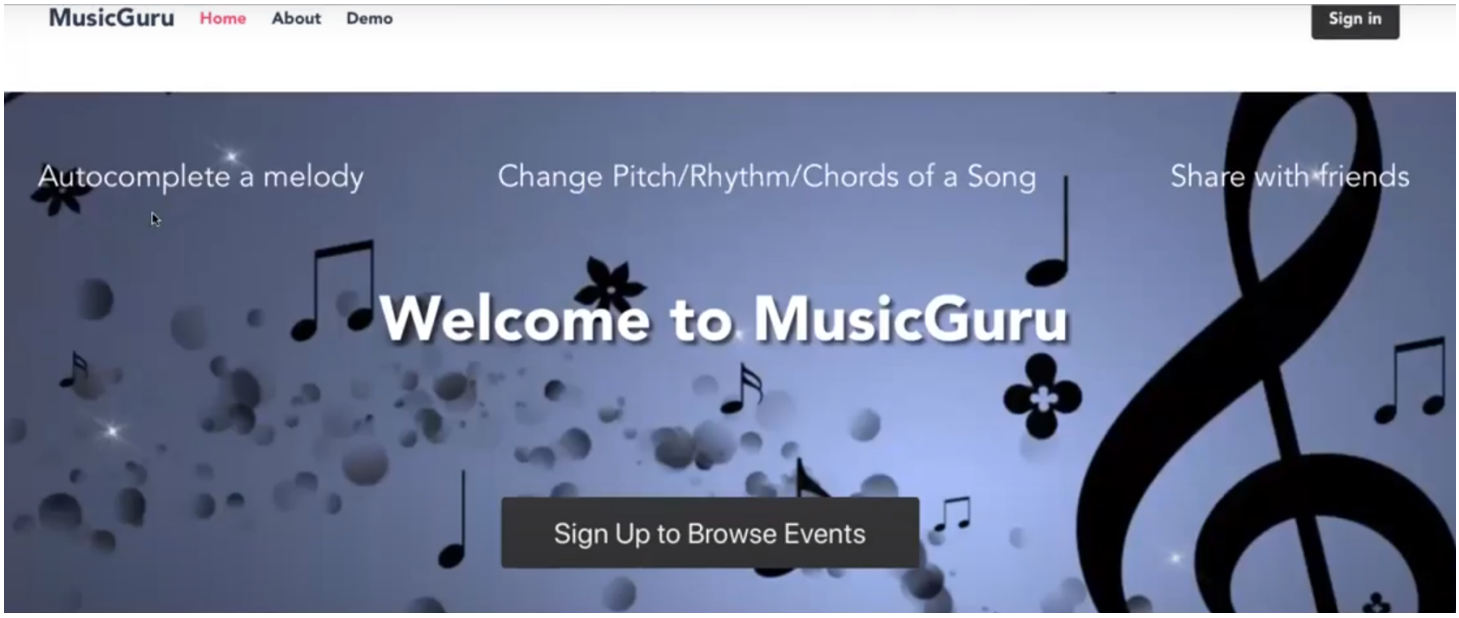
[17] Glossary reference: https://developers.google.com/machine-learning/glossary#s

Appendices

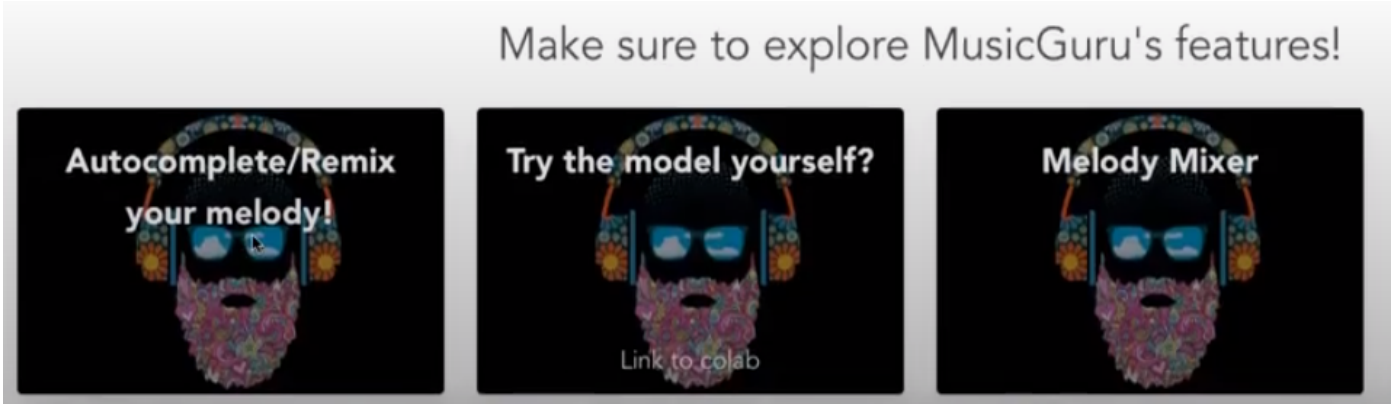
APPENDIX A: Overview of useful music related terms

term	notation	definition
note event	*n*	the playing of a note at a certain time span (duration).
pitch	*P*	a set of human-defined numbers describing the frequency degree of music sound, e.g., A4.
chroma	*c*	a.k.a, pitch class, the octave-invariant value of a pitch. For example, both C3 and C4 refer to chroma C (*do*), and the pitch interval from C3 to C4 indicates an octave (notated as *o*).
chord	–	a set of two or more simultaneous notes.
duration	–	the temporal length of a musical note, e.g., 1/4.
onset	–	the beginning of a musical note.
note state	*s*	the current state of a note event, involving *on*, *hold*, and *off* indicating the beginning, playing, and ending of a note, respectively.
inter-onset-interval	*i*	abbreviated to IOI, the duration between the onsets of two consecutive notes.
rhythm	*R*	the temporal pattern of notes, related to note duration, note onset, and IOI.
dynamics	*D*	the variation in loudness between notes or phrases.
velocity	*v*	the dynamics markings (e.g., "*mf*" and "*p*").
motif	–	a.k.a, motivate, a group of note events forming the smallest identifiable musical idea, normally one-bar long.
phrase	*ph.*	a group of motifs forming a complete musical sense, being normally four-bar long.
period	*per.*	a pair of consecutive phrases.
melody	–	a sequence of single notes from the melody track[1], which plays the most impressive sounds among the entire song.
harmony	–	the simultaneous or overlapping notes on accompaniment tracks to produce an overall effect along with the melody.

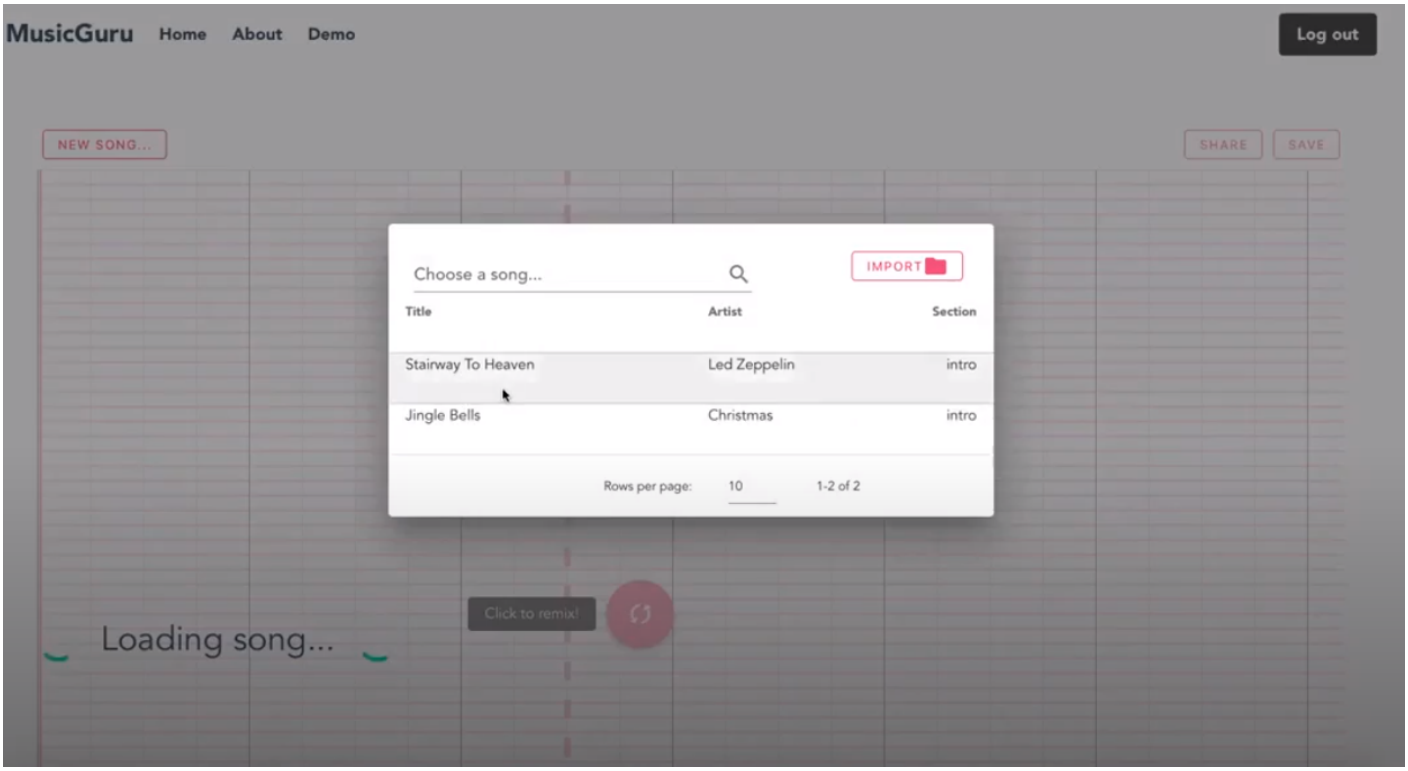
APPENDIX B: MusicGuru Landing Page Post GCP Deployment



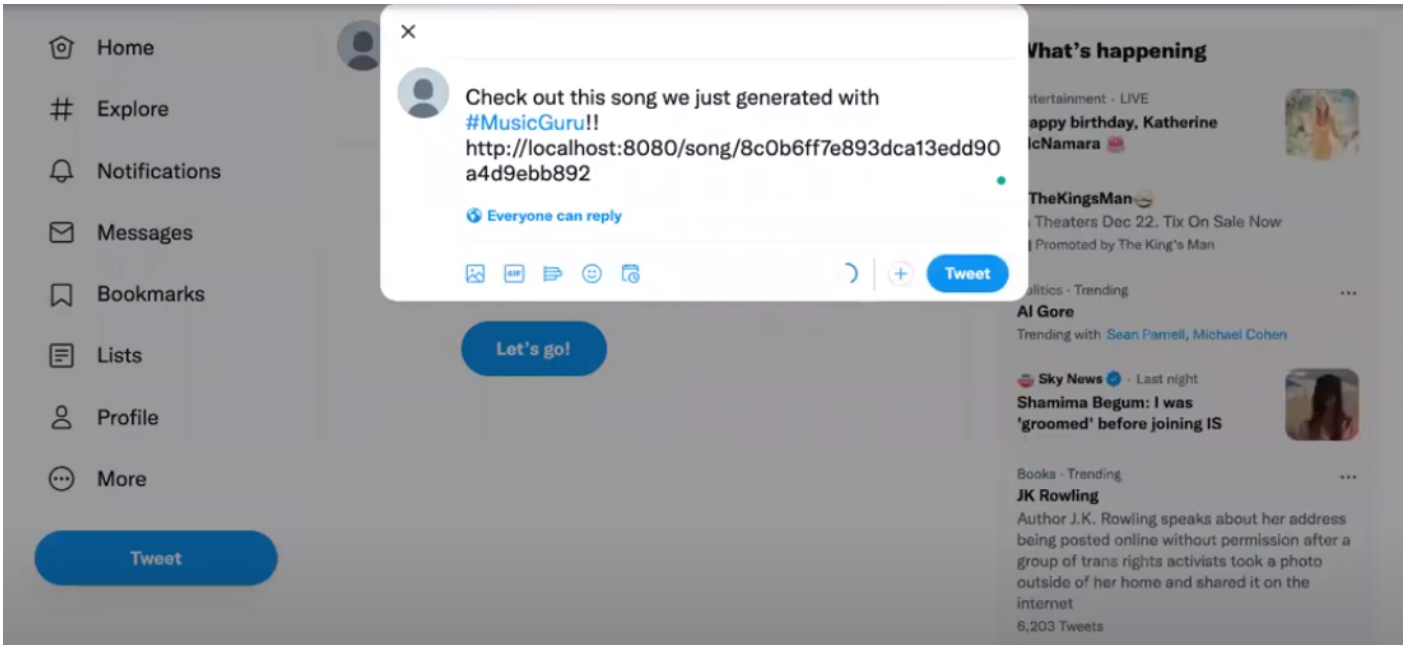
APPENDIX C: MusicGuru Landing Page, Downstream Tasks



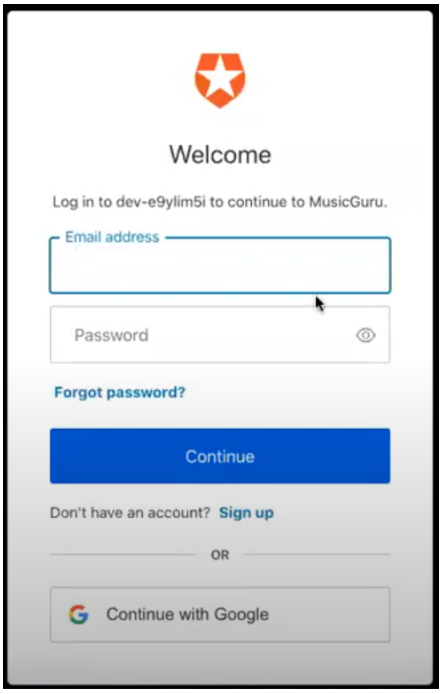
APPENDIX D: Importing a song for Downstream Tasks' Application



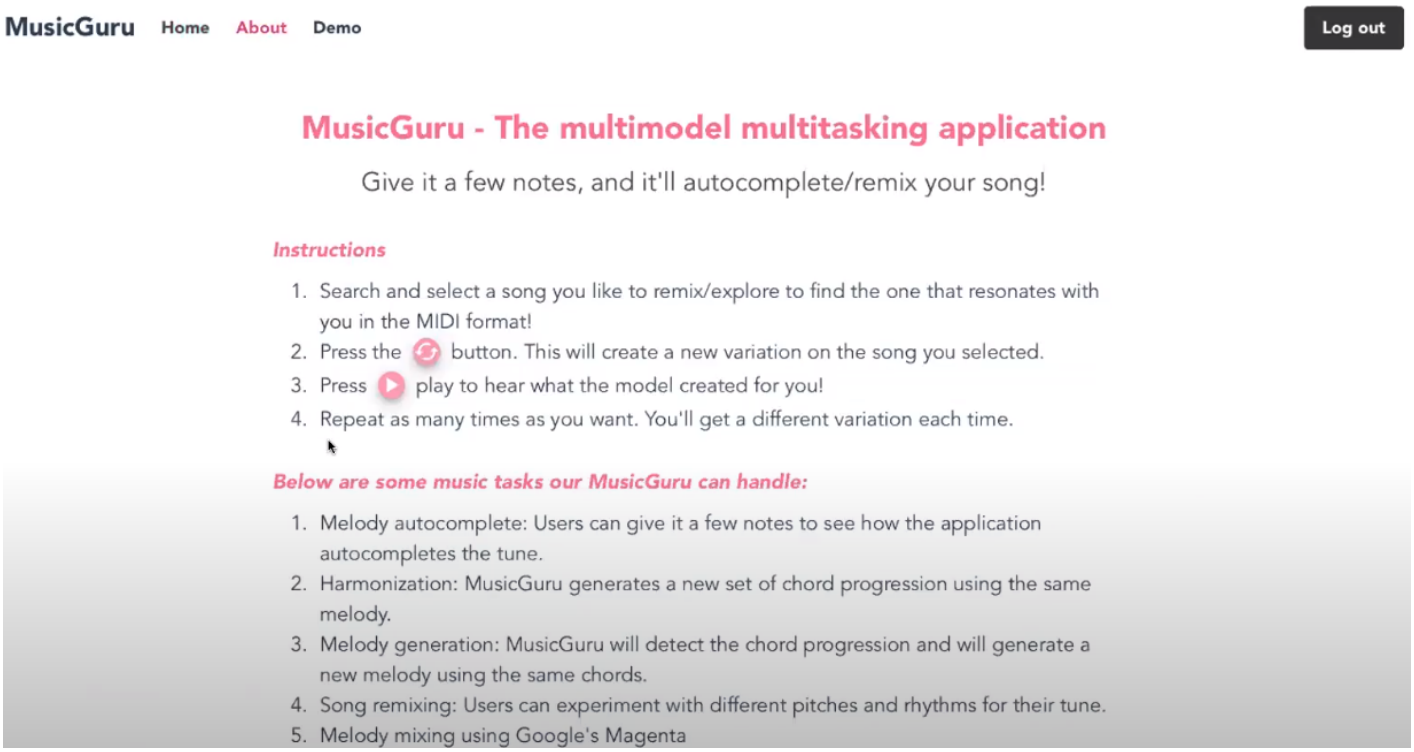
APPENDIX E: Sharing on Twitter about a newly created song on MusicGuru

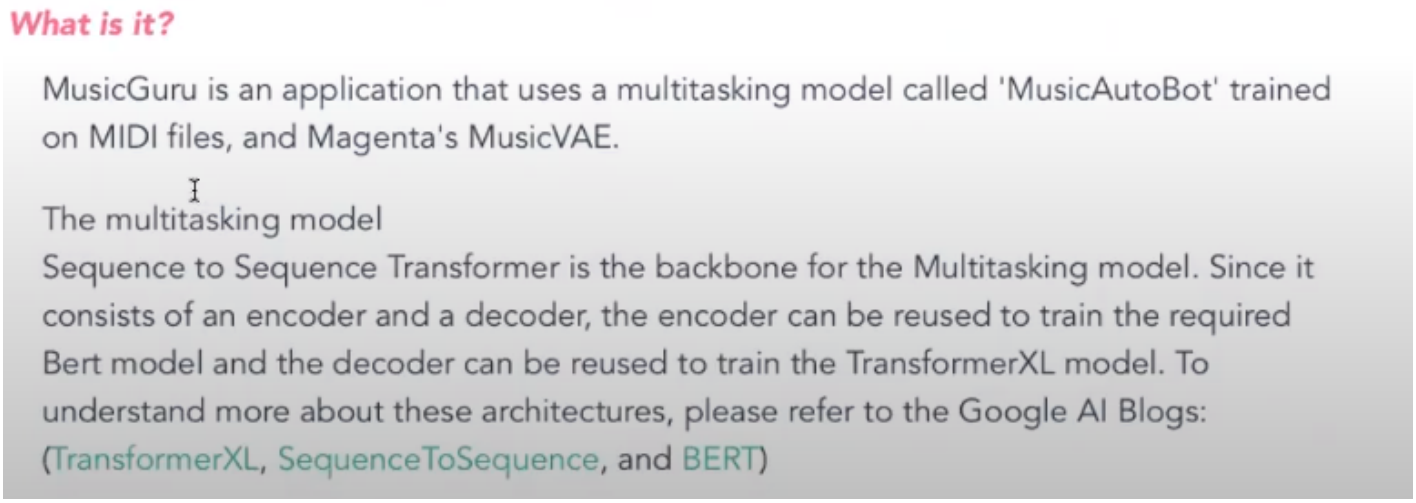


APPENDIX F: AuthO Authentication for MusicGuru

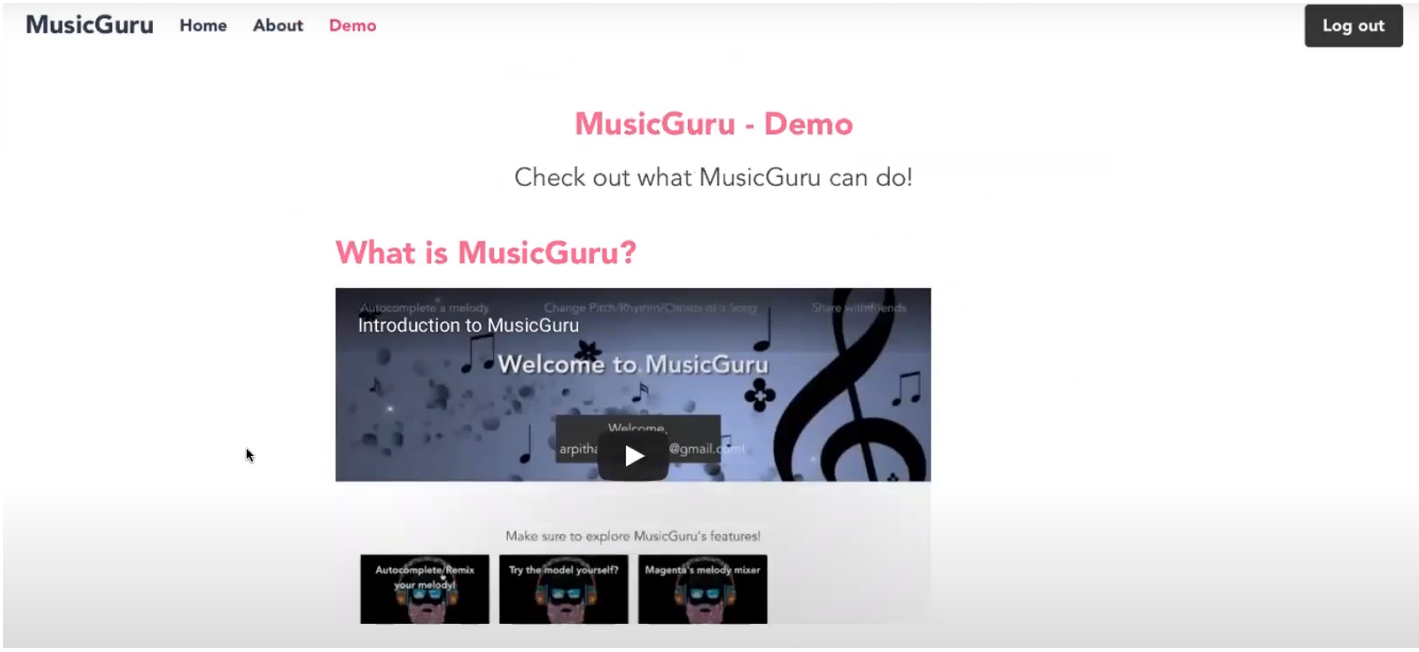


APPENDIX G: MusicGuru About Page

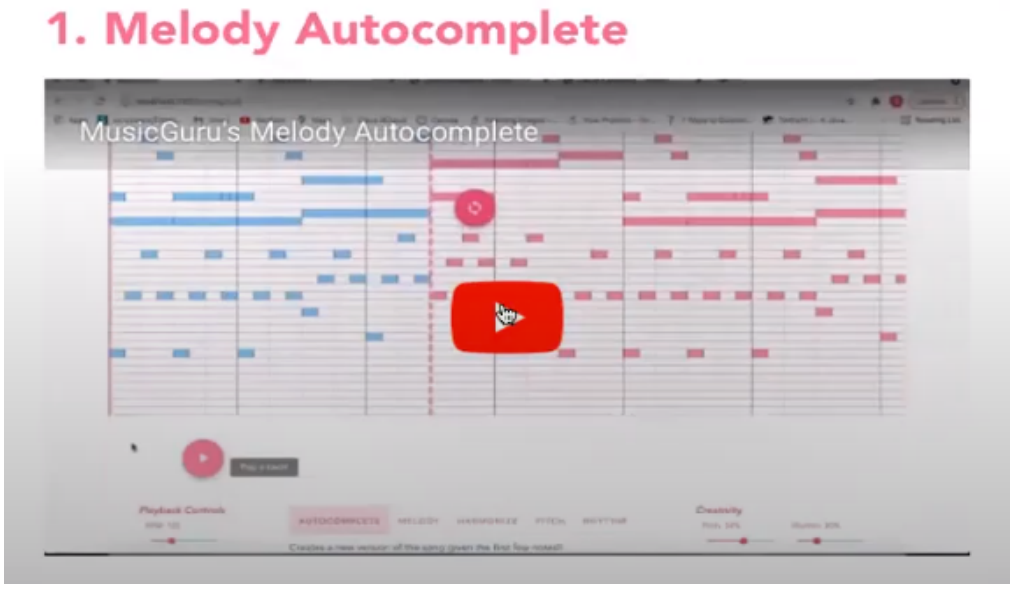




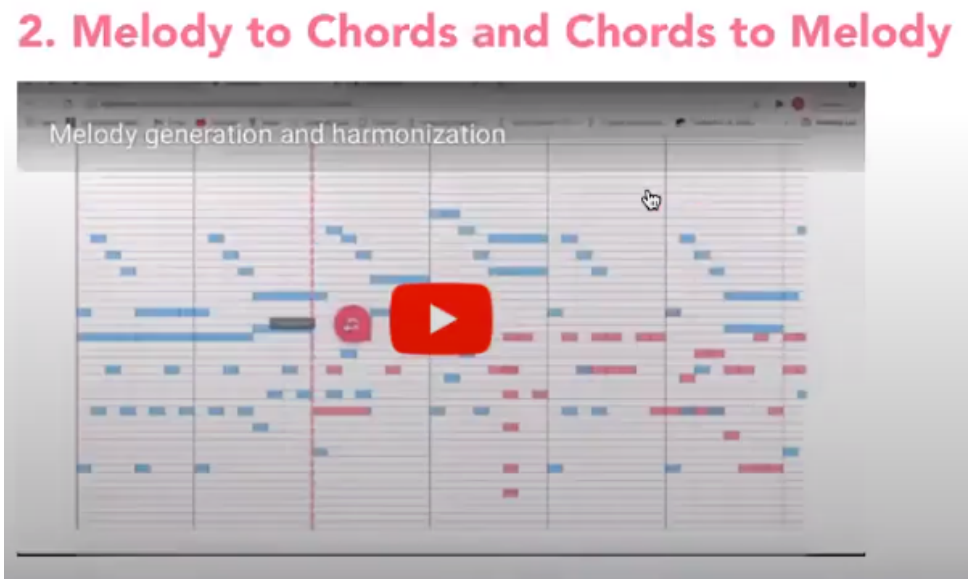
APPENDIX H: MusicGuru Introduction Video



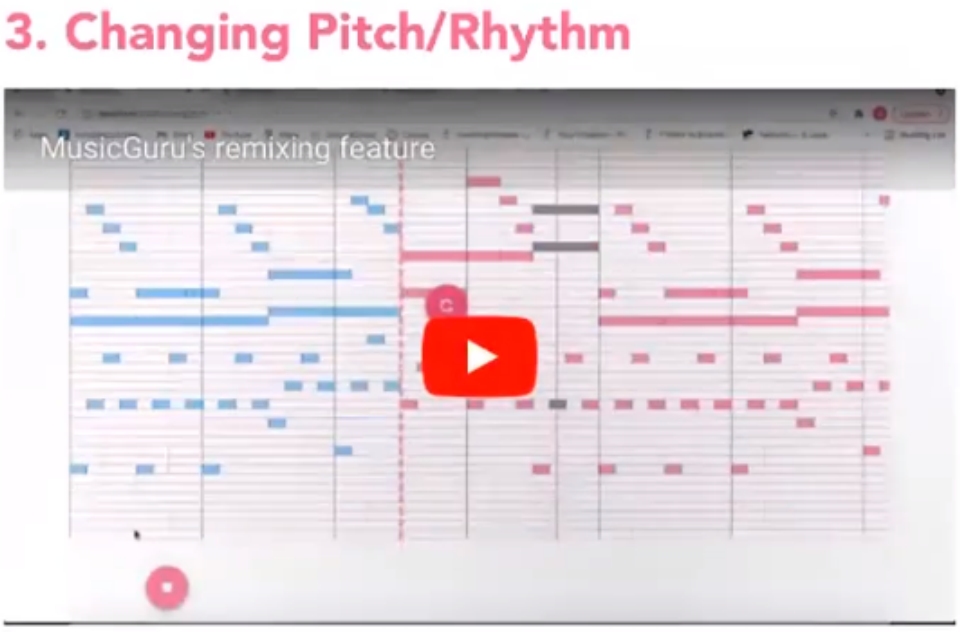
APPENDIX I: MusicGuru Demo Video, Melody Autocomplete



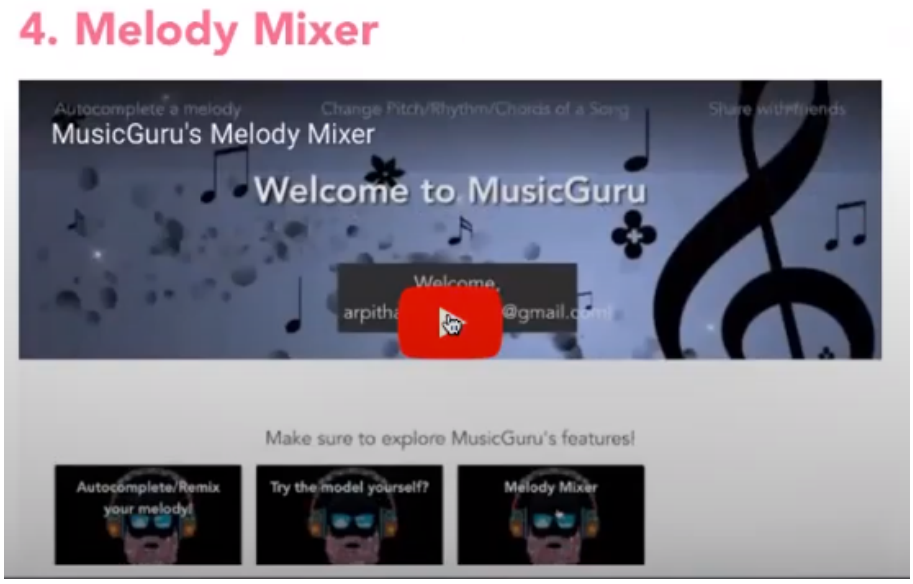
APPENDIX J: MusicGuru Demo Video, Melody to Chords & Chords to Melody



APPENDIX K: MusicGuru Demo Video, Changing Pitch/ Rhythm



APPENDIX L: MusicGuru Demo Video, Melody Mixer



APPENDIX M:

Project GitHub link: https://github.com/bhuvanabasapur/TeamInvinsibles