Tribhuvan University

Academia International College

Final Year Project Report

On

Music Generator

[CSC 412]

Under the supervision of

"Dr. Sunil Chaudhary"

Submitted by:

Bishow Deep Shrestha (T.U. Symbol No. 29010/078)

Nikhil Shrestha (T.U. Symbol No. 29017/078)

Paras Limbu (T.U. Symbol No. 29019/078)

Submitted to:

Department of Computer Science and Information Technology

Academia International College

Institute of Science and Technology

Tribhuvan University

Tribhuvan University

Academia International College

Final Year Project Report

On

Music Generator

[CSC 412]

A final year project submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University

Submitted by

Bishow Deep Shrestha (T.U. Symbol No. 29010/078)

Nikhil Shrestha (T.U. Symbol No. 29017/078)

Paras Limbu (T.U. Symbol No. 29019/078)

Submitted to

Department of Computer Science and Information Technology

Academia International College

Institute of Science and Technology

Tribhuvan University

Tribhuvan University

Institute of Science and Technology

Academia International College

Department of Computer Science and Information Technology

Email: mail@academiacollege.edu.np

Supervisor's Recommendation

I hereby recommend that the project work report prepared under my supervision by Mr. Bishow Deep Shrestha (29010/078), Mr. Nikhil Shrestha (29017/078) and Mr. Paras Limbu (29019/078) entitled "Music Generator" be accepted as fulfilling in partial requirements for the degree of Bachelors of Science in Computer Science and Information Technology. In my best knowledge, this is an original work in Computer Science and Information Technology.

……..

Dr. Sunil Chaudhary
Project Supervisor
Department of Computer Science and Information Technology
Academia International College
Gwarko, Lalitpur

Tribhuvan University

Department of Computer Science and Information Technology

Academia International College

Certificate of Approval

This is to certify that this project prepared by Mr. Bishow Deep Shrestha, Mr. Nikhil Shrestha and Mr. Paras Limbu entitled "Music Generator" in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

…….. Dr. Sunil Chaudhary Project Supervisor Department of Computer Science and IT Academia International College	…….. Mr. Bishwas Mathema HOD/Coordinator Department of Computer Science and IT Academia International College
…….. Internal Examiner Academia International College	…….. External Examiner Central Department of CSIT Tribhuvan University

Acknowledgement

We owe our most profound appreciation to Academia International College for giving us a chance to work on this project as part of our syllabus.

Special thanks to our supervisor, Dr. Sunil Chaudhary (Project Supervisor, Academia International College), for his consistent guidance, support, and feedback throughout the report's creation. We are generously obligated to him for providing this excellent opportunity to expand our knowledge. It helped us a lot to realize what we intended for.

We would like to express our sincere gratitude to all those individuals, families, friends, colleagues, and teachers for supporting and helping us a lot in finalizing this project within the limited time frame by providing valuable insights and feedback on the report.

Thanking You,

Bishow Deep Shrestha (T.U. Symbol No. 29010/078)
Nikhil Shrestha (T.U. Symbol No. 29017/078)
Paras Limbu (T.U. Symbol No. 29019/078)

Abstract

Music Generator is a modern system developed to assist users in creating original music compositions with ease and flexibility. Unlike conventional streaming platforms that primarily focus on music playback, this system is dedicated to music creation, allowing users to generate unique musical as per the input. The platform utilizes React with Vite for building a responsive and efficient frontend interface, Flask (Python) for handling backend logic, and MongoDB for reliable data storage. As this project follows Transformer architecture, multi-head attention and positional encoding use to train data. Users can interact with the system to compose music by specifying various parameters, listen to real-time previews of the generated pieces by export their creations in popular format like MIDI for further use or sharing. The development of Music Generator followed an iterative methodology, with continuous planning, testing, and refinement of features to enhance usability and system performance. Emphasis was placed on delivering a user-friendly and scalable solution that fosters creativity and innovation in digital music composition, making it a valuable tool for both amateur musicians and professionals.

Keywords: Download MIDI, Audio conversion, MP3, React, Flask, MongoDB, agile methodology.

Table of Contents

List of Figures

List of Table

List of Abbreviations

- AI – Artificial Intelligence
- API – Application Programming Interface
- CSV – Comma-Separated Values
- GPU – Graphics Processing Unit
- HTML – Hyper Text Markup Language
- HDF5 – Hierarchical Data Format
- HTTP – Hyper Text Transfer Protocol
- JSON – JavaScript Object Notation
- JWT – JSON Web Token
- LSTM – Long Short-Term Memory
- MIDI – Musical Instrument Digital Interface
- NLP – Natural Language Processing
- RNN – Recurrent Neural Network
- REST – Representational State Transfer
- UI – User Interface
- URL – Uniform Resource Locator
- WAV – Waveform Audio File Format

Chapter 1: Introduction

1.1. Introduction

The integration of Artificial Intelligence (AI) into creative domains has transformed conventional practices, with music generation being a notable area of innovation. In recent years, advances in deep learning especially in attention-based models have enabled systems to autonomously compose music that mimics human creativity. Among these, the Transformer architecture has emerged as a state-of-the-art solution for sequence modeling tasks, including natural language processing and, more recently, music composition.

Transformers offer key advantages over traditional recurrent architectures like LSTMs, particularly in handling long-range dependencies and enabling parallel computation. Their self-attention mechanism allows the model to focus on relevant parts of a musical sequence regardless of their position, making them well-suited for capturing the complex temporal and harmonic relationships that define musical structure.

Despite these advancements, many existing AI music generators still fall short in producing compositions that are musically coherent, expressive, and stylistically diverse. Some generate repetitive or unstructured music, while others rely heavily on pre-defined templates or are constrained to specific genres, limiting their creative flexibility.

This project aims to address these challenges by developing a Transformer-based music generator capable of producing original and musically satisfying compositions. By training the model on a diverse collection of MIDI datasets, the system will learn to generate melodies and harmonies that are both creative and structurally coherent. The objective is to push the boundaries of AI-generated music, making it more expressive, versatile, and artistically relevant across various musical contexts.

1.2. Problem Statement

The current landscape of AI music generation presents several challenges that hinder the creation of high-quality musical compositions. These challenges include:

- Many existing AI music generators produce compositions that lack structural integrity and coherence, resulting in music that feels disjointed or random. This undermines the listener's experience and limits the practical applications of such systems.
- Current solutions often rely on pre-defined templates or styles, leading to repetitive and uninspired outputs. This restricts the potential for innovation in music composition, as the generated music may not reflect the diversity and richness of human creativity.
- Most AI music generators are designed to operate within specific genres, which limits their versatility. This specialization can prevent users from exploring a broader range of musical styles and influences.
- The effectiveness of AI music generation is heavily dependent on the quality and diversity of the training data. Many existing systems utilize limited datasets, which can result in a lack of variety in the generated compositions.
- There is often a lack of user-friendly interfaces that allow users to interact with AI music generators effectively. This can lead to a disconnect between the technology and its potential users, limiting its adoption and practical use.

1.3. Objectives

The Music Generator project aims to achieve the following objectives:

- To design and implement a Transformer-based model capable of generating original musical compositions.
- To train the model using extensive MIDI datasets to improve diversity and quality in generated music, while recognizing that results still depend on dataset richness and may require future refinement.
- To ensure the generated music is coherent, structurally sound, and aesthetically pleasing.
- To create a user-friendly interface for inputting parameters and obtaining musical outputs.

1.4. Scope and Limitations

1.4.1. Scopes

- The system is capable of generating original music compositions using Transformer-based deep learning models, responding to diverse user inputs such as textual descriptions of mood, genre, or emotion.
- It allows optional conditioning through MIDI inputs, enabling users to guide the structure or melody of the generated composition more precisely.
- Users can input prompts in natural language, making the system accessible to both musicians and non-musicians seeking automated composition tools.

1.4.2. Limitation

- Users have limited real-time control over the music generation process, with no granular editing features or interactive feedback during composition.
- Output length and complexity may be difficult to predict or manage, leading to compositions that are either too short, too long, or not well-structured.
- High computational requirements for Transformer inference may result in longer generation times, especially on devices without GPU acceleration.
- The system lacks advanced understanding of musical theory or emotional nuance beyond its training data, which can result in generic or repetitive outputs.

1.5. Methodology

The development of the Music Generator system followed Agile development principles, which focus on flexibility, continuous feedback, and delivering working features in small, manageable increments. Within this Agile framework, an iterative approach was used to gradually build and improve the system. Each iteration, or development cycle, focused on implementing specific features such as music generation, MIDI processing, and audio export, followed by testing and feedback collection.

In the early iterations, to train the model, transformer was designed for generating music from cord input and previewing melodies was implemented. Feedback from these initial versions helped identify areas for improvement in both the system’s performance and user interface. Subsequent iterations introduced enhancements such as improved melody

quality, format conversion, playlist management, and session tracking.

Every iteration included testing to ensure that the new features worked correctly and integrated smoothly with the existing system. This step-by-step development allowed the team to refine the application continuously, correct errors quickly, and gradually build a complete, reliable, and user-friendly music generation platform capable of handling diverse inputs and producing high-quality musical outputs. With the 12th iteration of a week in each iteration we come to complete this project.

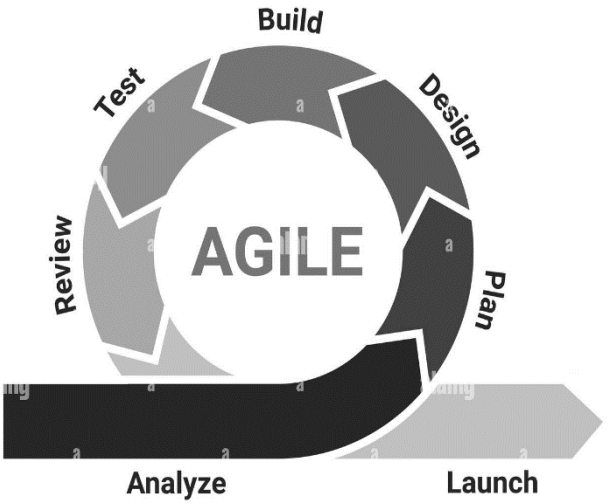

Figure 1.1 Agile Methodology of Software Development

The Music Generator project followed an iterative development approach, where the system was built and improved through repeated cycles. Each cycle focused on implementing and testing specific features, such as text-to-music generation, MIDI export, audio previews, and playlist management. After each iteration, feedback was collected, and necessary improvements were made before adding new features. This gradual process ensured smooth integration of all modules, continuous testing, and adaptation based on user feedback, resulting in a reliable, user-friendly, and fully functional music generation system.

1.6. Report Organization

After successfully completing the project, a comprehensive project report has been

prepared. The report begins with essential introductory sections, including the Title Page, Certificate Page, Acknowledgement, Abstract, Table of Contents, and lists of Abbreviations, Figures, and Tables.

The main body of the report is structured into six chapters, each focusing on a specific aspect of the project:

Chapter 1: Introduction

This chapter provides an overview of the project, covering key aspects such as the project introduction, problem statement, objectives, scope and limitations, and methodology.

Chapter 2: Background Study and Literature Review

Here, the project's background is discussed, along with a review of existing literature. This includes summaries of relevant projects, research papers, and articles that helped shape the project's direction.

Chapter 3: System Analysis

This section delves into system analysis, including requirements and feasibility studies. It defines the system's functional requirements using a use case diagram and presents a Gantt chart to visually illustrate the timeline and progress of various project tasks.

Chapter 4: System Design

This chapter explores the design phase in detail, covering the implementation process, model architecture, user interface, and system interactions. It also includes insights into the algorithms used in the project.

Chapter 5: Implementation and Testing

The focus here is on the implementation process and testing phases. It provides an overview of the tools and dependencies used to build the system and outlines the testing procedures undertaken to ensure functionality.

Chapter 6: Conclusion and Recommendations

The final chapter wraps up the project with a summary of key findings and conclusions. It also highlights potential areas for future improvements and enhancements.

The report concludes with a References section, formatted in IEEE style, and Appendices containing system screenshots and important source code snippets

Chapter 2: Background Study and Literature Review

2.1. Background Study

Digital music creation has grown tremendously in recent years, but many existing tools remain complex, expensive, or difficult for beginners to use [1]. Musicians and hobbyists often struggle to capture their ideas in a digital format, whether it's a chord progression, a melody, or the sound of an instrument. Translating these ideas into usable formats for music production can be time-consuming and technically challenging, which can slow down the creative process and discourage experimentation [1].

Music Generator addresses these challenges by providing an intuitive platform where users can input chords or instrument sounds and receive fully generated music compositions in MIDI format. This allows creators to quickly turn their ideas into digital music, experiment with arrangements, and integrate their creations into other production software. By offering a simple and efficient workflow, the platform reduces technical barriers and enables both beginners and experienced musicians to focus on creativity rather than setup or complex editing.

Unlike traditional streaming services or conversion tools, Music Generator does not focus on playing existing content or converting videos to audio. Instead, it is designed purely for original music creation, giving users full control over their compositions. By emphasizing usability, accessibility, and creative freedom, the platform empowers users to explore new musical ideas, produce professional quality MIDI files, and develop their skills in an environment free from unnecessary restrictions. Music Generator represents a modern, user-centered approach to digital music production, where creativity comes first.

2.2. Literature Review

Recent years have seen significant advancements in AI-driven music generation, with several platforms and research efforts dedicated to enabling users to create original music through artificial intelligence. Unlike traditional music tools that focus on playback or identification, these platforms leverage deep learning and generative models to compose new musical pieces based on user input or algorithmic creativity.

Platforms such as Suno AI and Udio represent the forefront of AI-powered music creation. Suno AI allows users to generate complete songs from text prompts, including both instrumental and vocal elements, offering an integrated solution for music composition and production [2]. Udio, developed by former Google DeepMind researchers, also enables music creation from text prompts, with advanced features like remixing, extending tracks, and producing realistic-sounding vocals [3]. These platforms demonstrate the potential of AI to democratize music creation, making it accessible to users without formal musical training.

Other tools, such as Music Generator, focus on generating high-quality MIDI files from user-defined chords, melodies, or instrument sounds [4]. By outputting MIDI rather than audio, these platforms provide users with greater flexibility for editing and integration with professional digital audio workstations (DAWs). This approach caters to musicians and producers who require granular control over their compositions and wish to further refine AI-generated material.

The underlying technology for these platforms often involves deep neural networks, such as recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and more recently, Transformer architectures [5]. These models are trained on large datasets of musical sequences, learning to generate coherent melodies, harmonies, and rhythms. Notable research includes Google's Magenta project, which explores creative applications of machine learning in music and art, and OpenAI's MuseNet, capable of generating multi-instrument compositions in various style [6].

Despite the progress in AI music generation, many existing solutions are either highly specialized or lack integration with broader music production workflows. Music Generator addresses this gap by allowing users to create original music through audio input, preview generated content, and export results for further editing. This all-in-one approach streamlines the creative process and empowers users to experiment with music generation in a user-friendly environment.

Chapter 3: System Analysis

3.1. System Analysis

Before starting the development of the Music Generator platform, a thorough analysis was conducted to understand how the system should function to meet user needs effectively. This involved examining both functional and non-functional aspects to ensure the platform would deliver a smooth and reliable media experience.

3.1.1. Requirement Analysis

The requirements can be functional and non-functional.

Functional requirements:

- The platform accepts input in the form of chords or instrument sounds and generates music compositions in MIDI format.
- AI models process the inputs to create structured and coherent musical tracks.
- Users can generate multiple variations of each composition to explore different musical interpretations.
- The platform supports exporting generated MIDI files for use in external MIDI-compatible players or digital audio workstations (DAWs).
- Users can create accounts and save projects.

Non-functional requirements:

- The platform generates MIDI outputs within seconds to minutes, ensuring a responsive user experience.
- It is designed to scale efficiently, supporting many concurrent users through cloud GPU infrastructure.
- Generated music is of high quality and maintains clean, accurate MIDI data.
- User data is protected, and the platform complies with relevant privacy laws.
- The system prevents the creation or sharing of harmful or copyrighted content.
- Stability is maintained with automatic retries and error-handling mechanisms to ensure consistent performance.

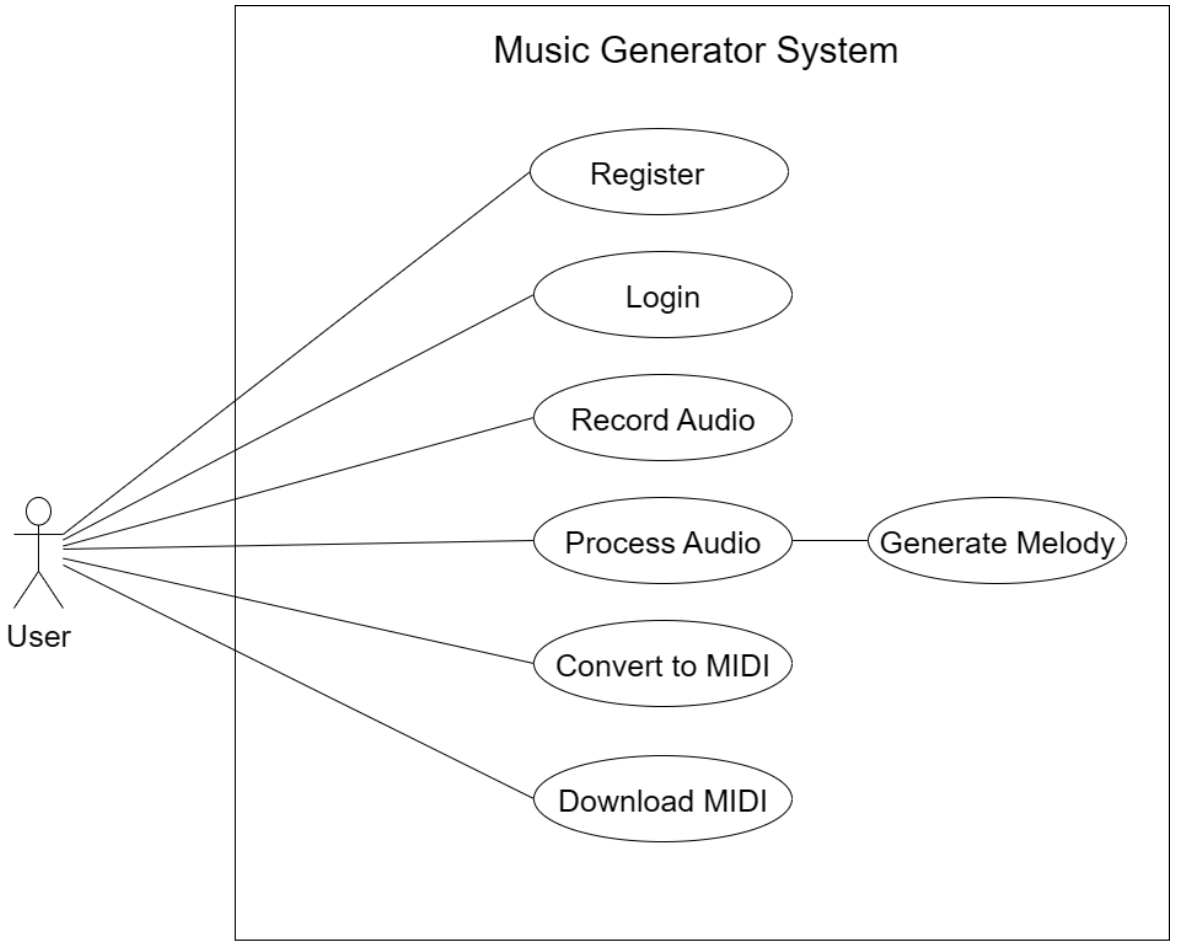

Figure 3.1: Use Case Diagram

The use case diagram illustrates the interaction between the user and the music generator system. The user may log in, record audio, process that audio, generate melodies, convert them to MIDI format, and download the resulting files. The system enables each of these interactions, ensuring a smooth workflow from input to downloadable musical output. Each oval in the diagram represents a specific use case or functionality supported by the system.

3.1.2. Feasibility Analysis

Feasibility analysis is conducted to determine whether a project is technically, economically, operationally, and schedule-wise feasible to guarantee successful project completion.

i. Technical Feasibility

Contemporary audio generation models commonly rely on deep learning frameworks

like PyTorch, which support scalable and efficient model development. Meta's open-source MusicGen, built using PyTorch through the AudioCraft library, and Google's MusicLM, developed with TensorFlow/JAX, are key examples. While PyTorch remains popular for both research and production use, TensorFlow was chosen for this project due to its compatibility and its strong ecosystem for deployment and inference.

Transformer models, based on the transformer architecture, are used in this project for their superior ability to handle long-term dependencies in sequence data. Unlike RNNs or LSTMs, transformers process input sequences in parallel and can capture complex musical patterns more effectively. This makes them ideal for generating structured, high-quality music, especially when dealing with diverse input types such as audio and MIDI sequences.

The backend is implemented using Flask for its asynchronous support and ease of use. The transformer model is developed with TensorFlow, with model weights stored in `.h5` format for optimization. Audio and MIDI processing are handled with pretty-midi and librosa, while Redis manages the task queue for generation. MongoDB may be included for storing user data and project information [5]. This setup is designed for rapid development with the flexibility to scale and extend the system later.

For the backend implementation, we build a Python-based system using Flask for its asynchronous capabilities and simplicity. The core AI service will utilize an Transformer model developed with TensorFlow, with model weights stored in .h5 format for portability. Audio processing will be handled by essential libraries: pretty midi for MIDI manipulation and librosa for audio analysis tasks. Include mangodb for database. This streamlined architecture focuses on rapid development while maintaining the flexibility to expand functionality as needed, prioritizing a working music generation pipeline over complex infrastructure.

ii. Operational Feasibility

The platform must be easy and appealing for non-technical creatives. This means an intuitive UI and guided workflow. For example, Udio's interface is praised for being "intuitive" and presenting in a straightforward way. Tutorials, tooltips, and examples can help onboard beginners. Community features encourage engagement and learning.

Operationally, the team will need to maintain and update ML models as improvements

emerge and monitor system health (scaling GPUs, replacing older hardware). Copyright and IP management is crucial: the platform should clearly state licensing terms and avoid infringing content. Providing good documentation and responsive support will aid adoption. Ethical moderation is also a concern: we should prohibit the generation of hate speech, defamation, or pornographic content via prompt filters or manual review. Platforms like YouTube use automated content moderation pipelines, and similar safeguards (keyword filters, user reporting) can be adapted for music generation.

Development to train the model might be done in cloud, or GPU, system GPU for the complex computation.

iii. Economic Feasibility

Table 3.1: Cost Estimation(yearly)

Cost Component	Assumptions	Estimate (Rs)
GPU Inference/Training	50 GPU hrs/month × Rs 50/hr × 12 months	30,000
Cloud Storage & Bandwidth	Rs 5,000/month × 12 months	60,000
Miscellaneous	Licenses, tools, testing, contingency	50,000
Total (Year 1)		1,40,000

Table 3.2: Revenue Projection

Scenario	Users	Subscription (Rs/month)	Annual Revenue (Rs)
Conservative	500	100	50,000
Moderate	1,000	100	1,00,000
Optimistic	2,000	100	2,00,000

Table 3.3: Total Initial Cost

Item	Estimate (Rs)
Domain name (1 year)	2,500
Initial cloud setup / hosting deposit	30,000
GPU training hours (initial model fine-tuning, 100 hrs × Rs 50/hr)	5,000
Basic marketing/launch promotion	35,000
Miscellaneous (SSL certs, tools, design)	10,000
Total initial cost	80,000

Calculation

Initial one-time cost = Rs 80,000

Annual operating cost = Rs 1,40,000

Monthly operating cost = Rs 11,667

Marketing cost/month = Rs 10,000

Support cost/month = Rs 5,000

Payment fee = 2%

Starting users = 100

New users/month = 100

Monthly churn rate = 5%

Price per user/month = Rs 100

Table 3.4: Monthly User growth Table

Month	Start Users	New Users	Churned	End Users
0	100	0	0	100
1	100	100	5	195
2	195	100	10	285
3	285	100	14	371
4	371	100	18	453
5	453	100	23	530
6	530	100	26	604
7	604	100	30	674
8	674	100	33	741
9	741	100	37	804
10	804	100	40	864
11	864	100	43	921
12	921	100	46	975

Revenue

- Average monthly revenue = 565 × 100 = Rs 56,500
- Annual revenue = 56,500 × 12 = Rs 678,000

Costs

- Initial one-time cost = Rs 80,000
- Operating cost (Rs 11,667/month) = Rs 1,40,000
- Marketing (10,000/month) = Rs 1,20,000
- Support (5,000/month) = Rs 60,000
- Payment fees = 2% of revenue = 2% × 678,000 = Rs 13,560

Total Cost (Year 1) = 80,000 + 1,40,000 + 1,20,000 + 60,000 + 13,560 = Rs 4,13,560

Profit & ROI

Net Profit (Year 1)

- 6,78,000 – 4,13,560 = Rs 264,440

ROI

ROI = (2,64,440/4,13,560) x100 = 64%

Payback Period

To calculate monthly profit:

- Monthly revenue = Rs 56,500
- Monthly cost = 11,667 + 10,000 + 5,000 + fees = Rs 28,630
- Monthly net profit ≈ 56,500 – 28,630 = Rs 27,870

Payback period = 4,13,560/27,870 = 14.8 months

Table 3.5 Breakeven Analysis

Month	Users	Revenue (Rs)	Costs (Rs)	Net Profit (Rs)	Cumulative Profit (Rs)
0	100	10,000	26,867	–16,867	–96,867
1	195	19,500	27,057	–7,557	–104,424
2	285	28,500	27,237	+1,263	–103,161
3	371	37,100	27,409	+9,691	–93,470
4	453	45,300	27,573	+17,727	–75,743
5	530	53,000	27,730	+25,270	–50,473
6	604	60,400	27,882	+32,518	–17,955
7	674	67,400	28,048	+39,352	+21,397
8	741	74,100	28,182	+45,918	+67,315
9	804	80,400	28,412	+51,988	+119,303
10	864	86,400	28,592	+57,808	+177,111
11	921	92,100	28,842	+63,258	+240,369
12	975	97,500	29,095	+68,405	+308,774

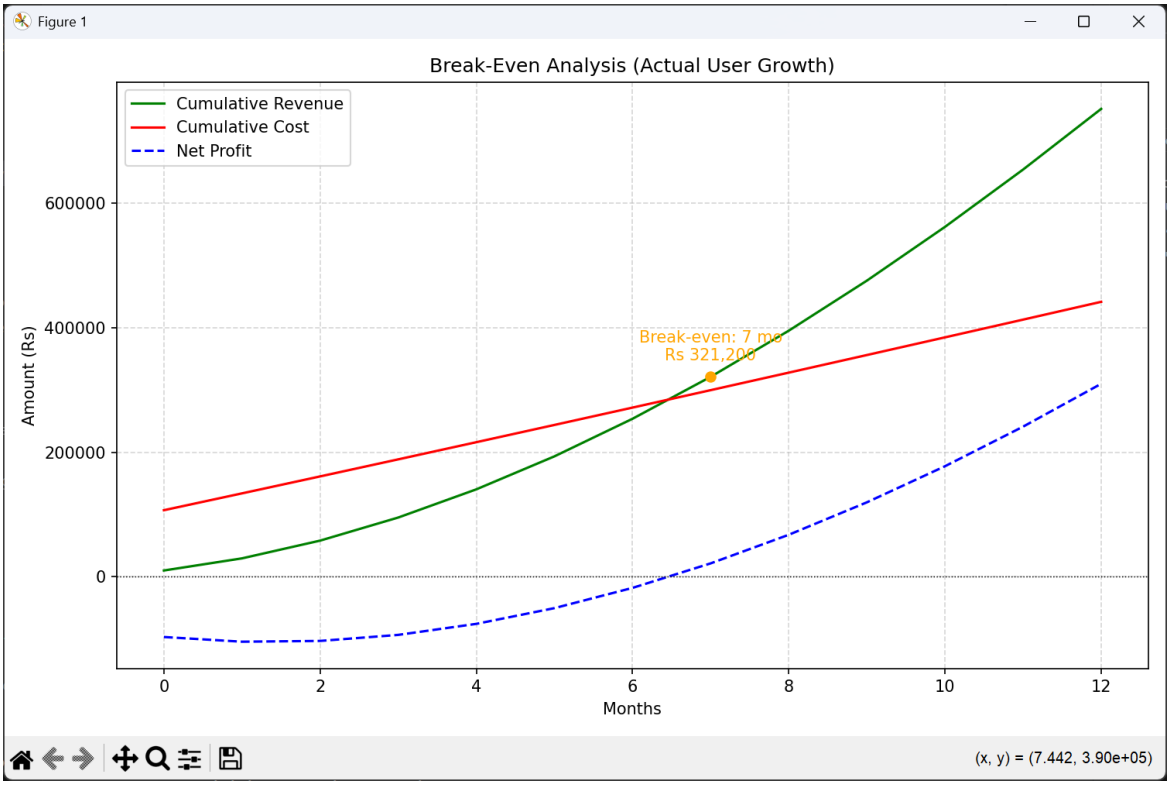

Figure 3.2: Break even Graph

iv. Schedule Feasibility

The project was completed within the planned timeframe, covering all essential tasks such as model development, backend implementation, audio processing, and testing. A well-structured timeline divided the work into manageable milestones, ensuring steady progress and timely completion.

The following Gantt chart illustrates the schedule established for the development of the music generator system.

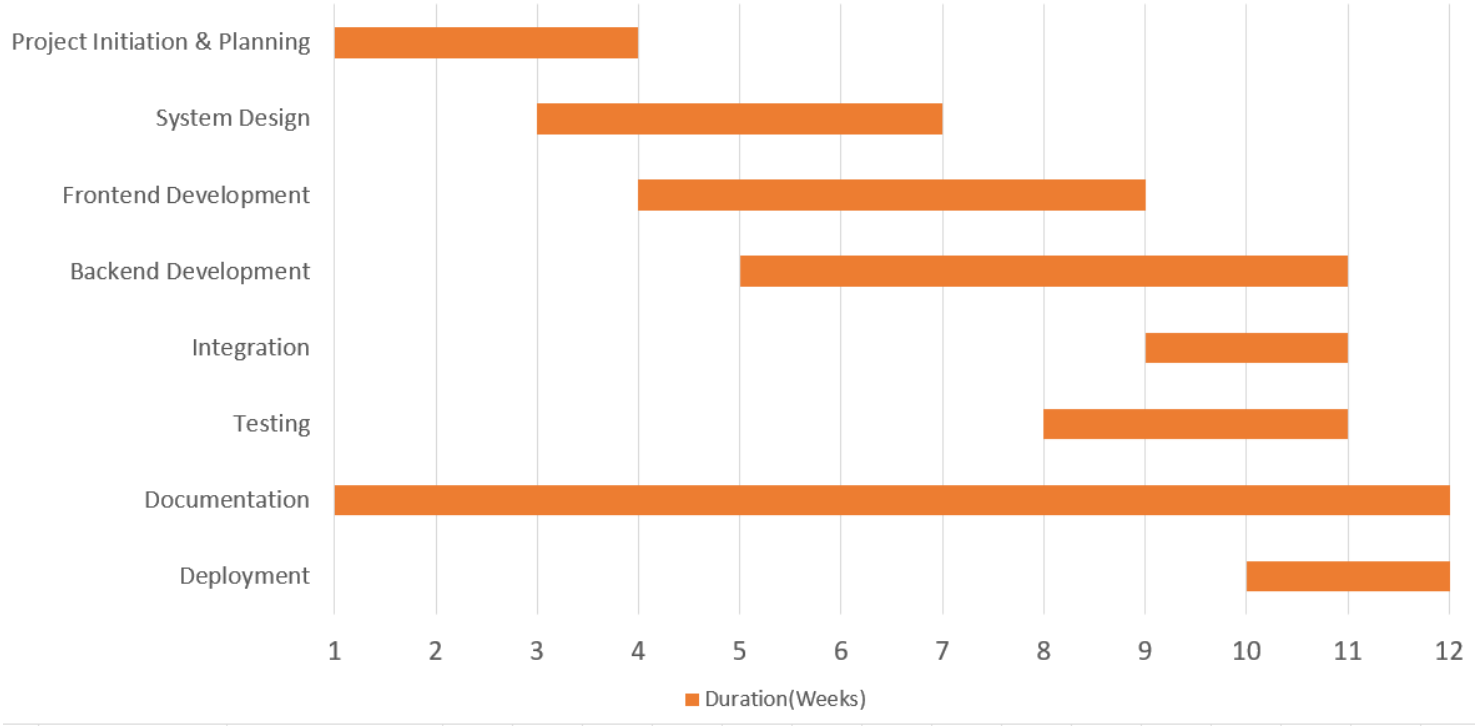

Figure 3.3: Gantt chart of Music Generator

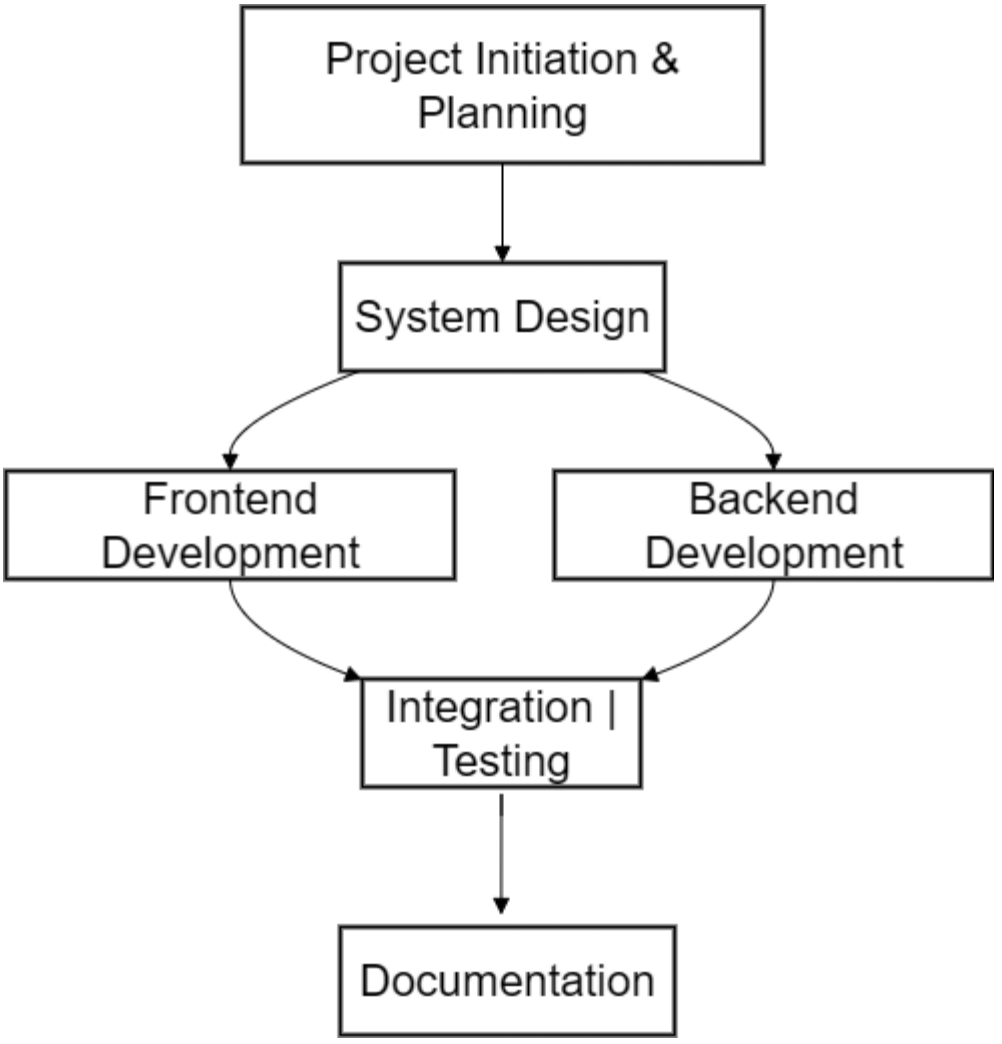

Figure 3.4: Network Diagram

3.1.3 Class Diagram

The class diagram shows how different components of the music generator system interact with each other. The central control is handled by the Flask, which coordinates tasks across several classes. Audio Processor handles audio input, which is then passed to the MelodyGenerator. The Melody Generator uses a Start Sequence and Generate Melody to create musical patterns.

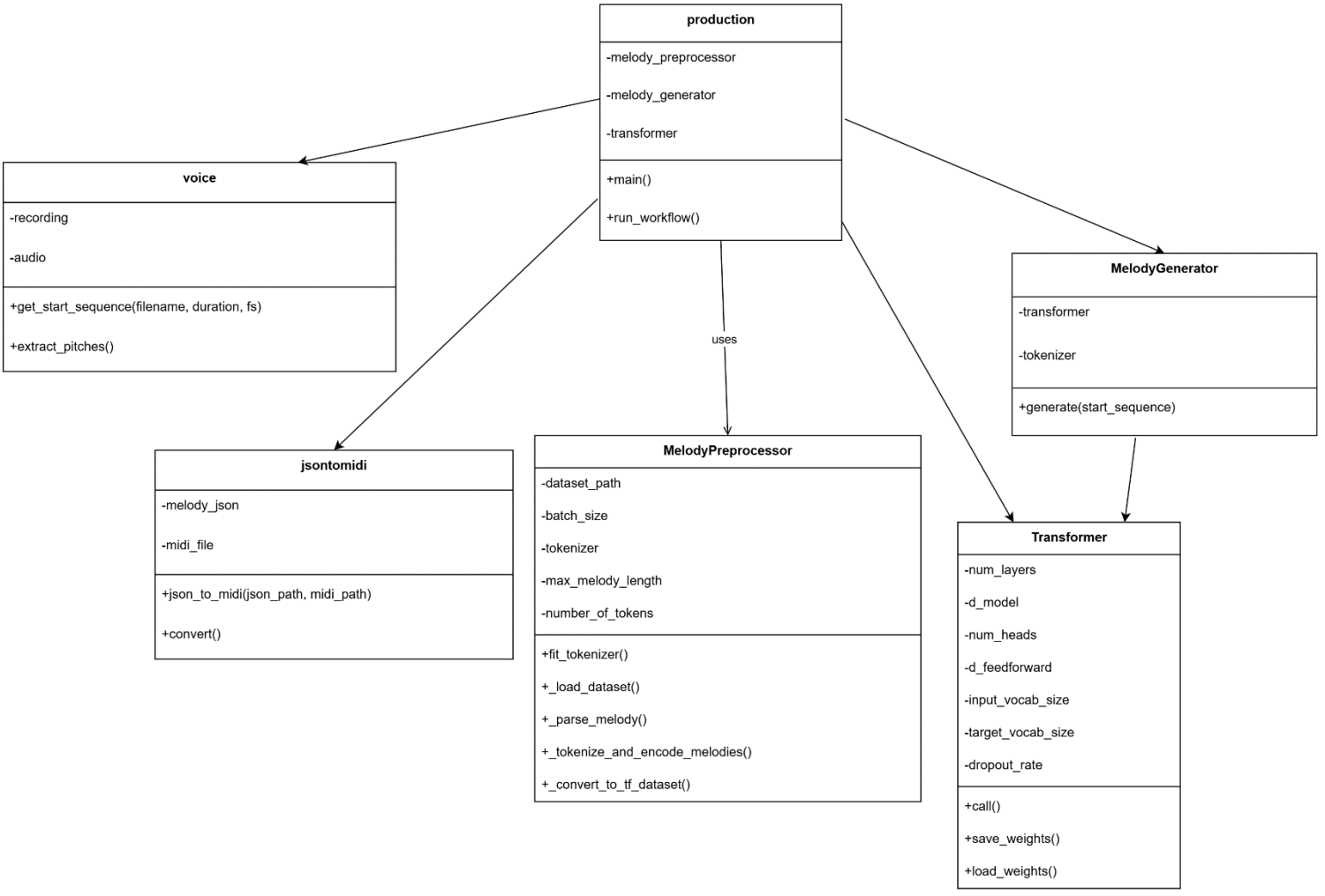

Figure 3.5: Class Diagram

3.1.4. Activity Diagram

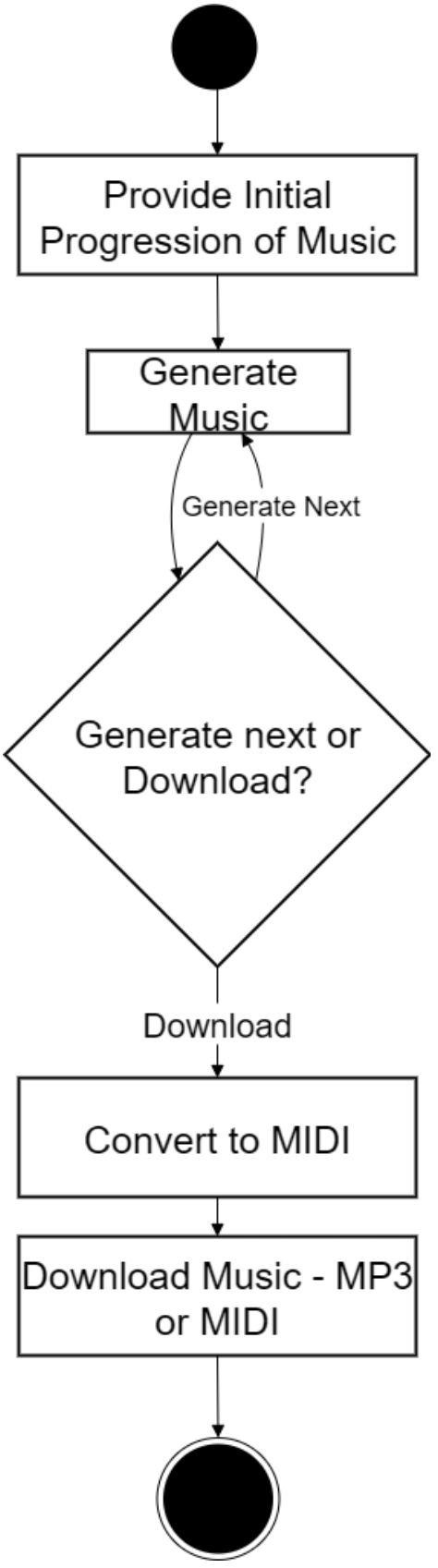

Figure 3.6: Activity Diagram

3.1.3. Sequence Diagram

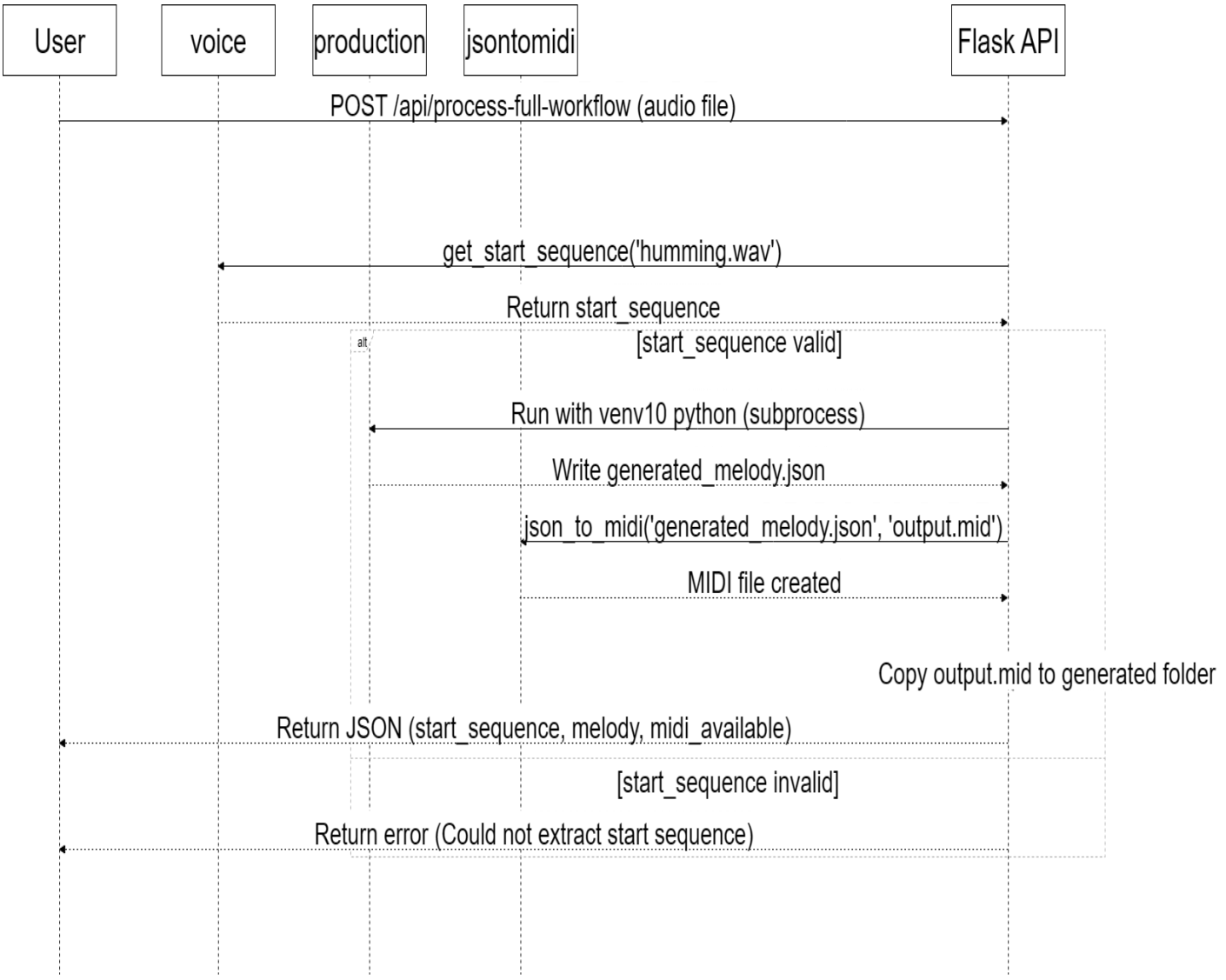

Figure 3.7: Sequence Diagram

Chapter 4: System Design

4.1. System Design

System design for our music composer project focuses on creating an efficient and scalable architecture that enables melody and harmony generation using deep learning models. The system is divided into several components, including the user interface, the audio input module, a preprocessing pipeline, the core AI model, and the output generation module. Users can input seed melodies or provide audio clips, which are then processed to extract relevant musical features like pitch, rhythm, and harmony. These inputs are tokenized and passed into an transformer based model trained on MIDI data to generate musically coherent sequences. The generated output is converted back into audio using MIDI synthesis tools. To ensure smooth performance and modularity, the backend is built using Python and Flask, with support for model loading, generation control, and audio rendering. The architecture is designed to be extensible for future features such as mood-based generation, genre selection, and cloud deployment. By separating concerns across modules and adopting standard design principles, the system remains maintainable, testable, and adaptable to new datasets and models.

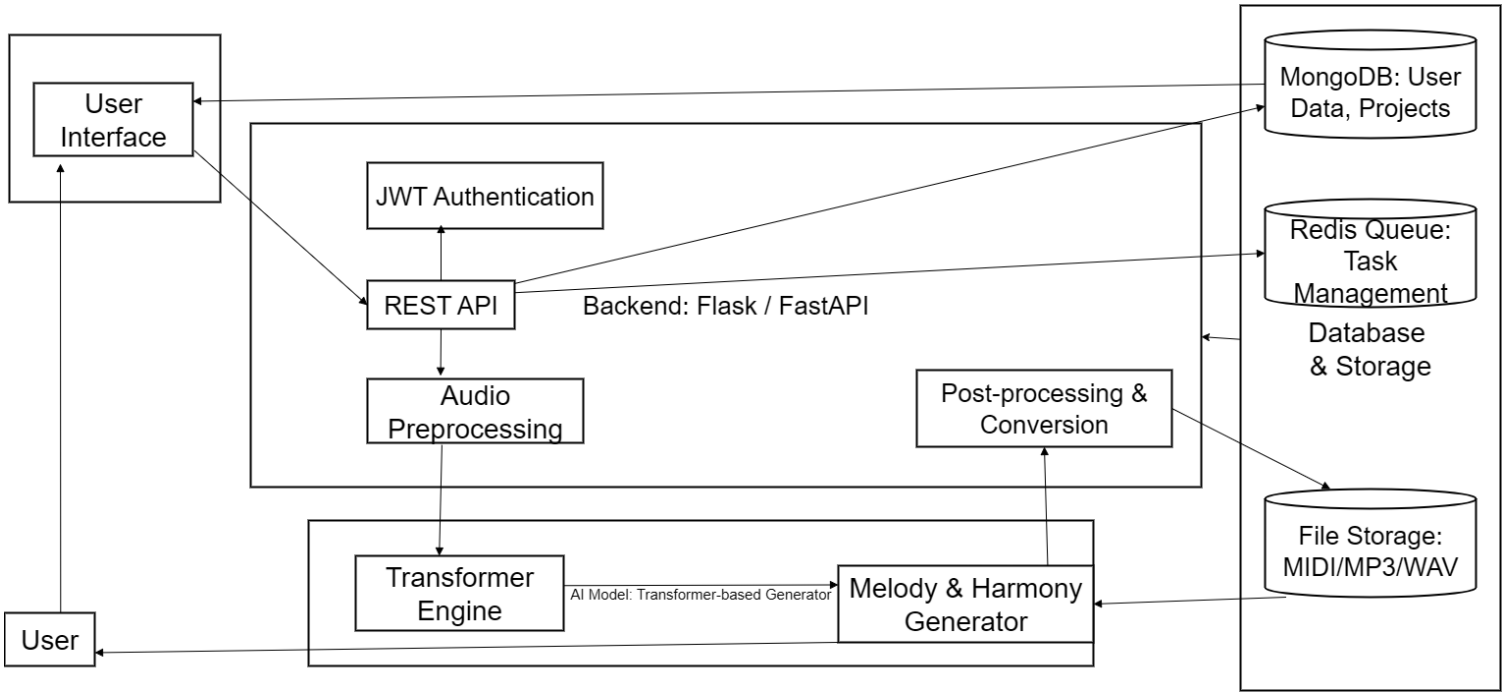

Figure 4.1: Component Diagram

4.1.1 Architectural Design

The architectural design of the Music Transformer-based composer system is structured into modular layers to handle data flow from user input to audio output efficiently. At the frontend, a lightweight interface built with tools like React allows users to input a melody sequence, upload a MIDI file, or hum a tune. This input is sent to a Flask-based backend via REST API calls. The backend first preprocesses the input using music processing libraries to extract pitch, rhythm, and structure, and then tokenizes the sequence into a format compatible with the Music Transformer model. The core of the architecture is a pre-trained or custom-trained Transformer model designed for long-range musical structure learning. It takes the tokenized input and generates extended compositions with coherent melody and harmony. The generated sequence is decoded back into MIDI format and converted into audio using a MIDI synthesizer. A post-processing module handles tempo adjustments and applies optional style or mood filters. The system also includes a storage layer to log inputs, outputs, and user preferences. The architecture is designed for extensibility, allowing future integration of cloud-based inference, collaborative editing, and hybrid models**.** Each module communicates through clearly defined APIs, ensuring scalability and easy maintenance.

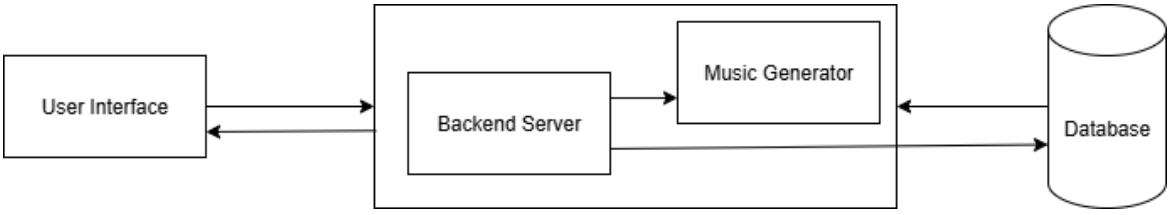

Figure 4.2: Architectural Review

4.2 Algorithm Details

4.2.1. Attention Is All You Need

The "Attention Is All You Need" paper introduced the Transformer model, architecture for sequence-to-sequence tasks such as machine translation and music generation [7]. Unlike traditional recurrent neural networks (RNNs) or long short-term memory networks (LSTMs), Transformers do not rely on recurrence. Instead, they use an attention mechanism to model relationships between all positions in a sequence simultaneously.

This makes the Transformer model highly parallelizable and effective at capturing long-range dependencies, which is crucial for generating coherent sequences like text or music. The main innovation of this model is its self-attention mechanism, which allows each output element to focus on different parts of the input sequence.

4.2.2. Transformer Architecture

The Transformer model follows the standard encoder-decoder architecture.

- Encoder: Converts the input sequence into a set of continuous representations.
- Decoder: Uses these representations to generate the output sequence, one element at a time.

Each encoder is made up of N = 6 identical layers, and each layer contains:

1. A Multi-Head Self-Attention mechanism
2. A Feedforward neural network

Each decoder also has N = 6 identical layers, with an additional third sub-layer:

1. Masked Multi-Head Self-Attention
2. Multi-Head Attention over encoder outputs
3. Feedforward neural network

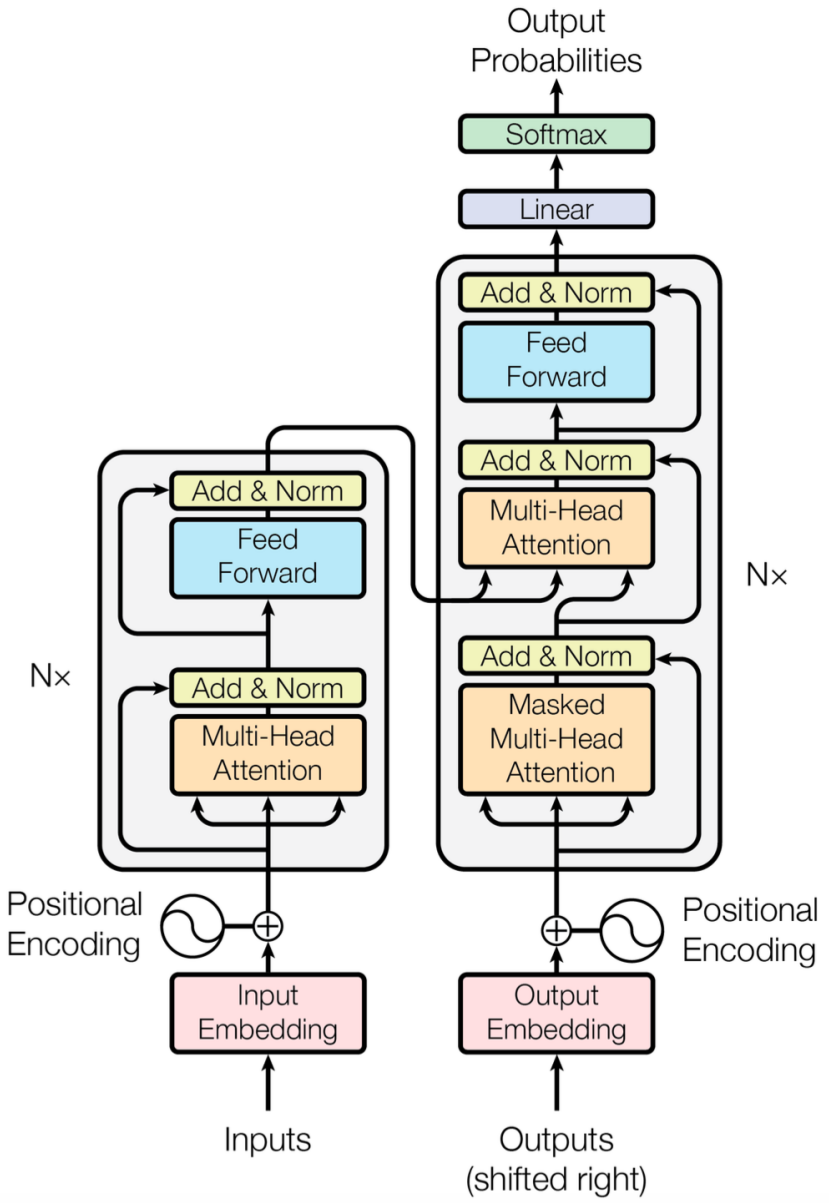

Figure 4.3: Transformer Architecture

4.2.3. Scaled Dot-Product Attention

The core building block of the Transformer is the attention mechanism, which determines how much focus each token (or note) should place on others when generating an output.

Given a query (Q), key (K), and value (V) matrix, attention is calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}*(QK^T/d_k^{1/2})V$$

- Q: Query vector (what we're looking for)
- K: Key vector (what we have)
- V: Value vector (information associated with each key)
- d_k: Dimension of the key vector, used for scaling

This mechanism computes similarity between the query and keys, scales them, applies softmax to normalize, and uses the weights to combine the values. This enables the model to "attend" to important parts of the sequence.

4.2.4. Multi-Head Attention

In the self-attention mechanism, queries (Q), keys (K), and values (V) are dynamically generated for each input sequence (limited typically by the size of the context window), allowing the model to focus on different parts of the input sequence at different steps. Multi-head attention enhances this process by introducing multiple parallel attention heads. Each attention head learns different linear projections of the Q, K, and V matrices. This allows the model to capture different aspects of the relationship between words in the sequence simultaneously, rather than focusing on a single aspect.

By doing this, multi-head attention ensures that the input embeddings are updated from a more varied and diverse set of perspectives. After the attention outputs from all heads are calculated, they are concatenated and passed through a final linear transformation to generate the output.
In music generation, this attention mechanism enables the model to:

- The system learns patterns across time, such as repeated motifs and recurring musical themes.
- It understands structural relationships within music, including chord progressions and harmonic sequences.
- The platform generates coherent melodies with long-term consistency, ensuring that compositions maintain musical flow and structure.

4.2.5 Positional Encoding

Since transformer model is not a sequence to sequence model and doesn't rely on sequence of text to perform encoding and decoding, we use positional encoding to solve the problem related to it.

PE = cosine(pos/10000^2i/d)

Where,

Pos = position of token in sequence

i = dimension index of embedding

d = total embedding dimension

Chapter 5: Implementation and Testing

5.1 Implementation

Various development tools and technologies have been used for the application development.

5.1.1 Tools Used:

Following are the tools used for development of our project:

- React js for frontend development
- JWT for session handling and secure API request
- MongoDB for database
- Python for backend
- Git and github for version control and collaborative development
- Draw.io to create diagrams
- Vscode for writing and running code
- Google collab for testing purpose

5.2 Testing

5.2.1 Unit Testing

Unit tests focus on individual modules like audio preprocessing, melody generation.

Unit Test Table:

Table 5.1: Unit Test Table

Test Case ID	Module	Description	Input	Expected Output	Actual Output	Status
UT-01	Audio Processor	Verify conversion of WAV file into MIDI tokens	Sample .wav (5s piano riff)	Correct MIDI token sequence extracted	Token sequence extracted	Pass
UT-02	Melody Generator (Transformer)	Check melody generation from voice prompt	"audio"	Coherent MIDI melody file generated	MIDI generated	Pass
UT-03	Midi Converter	Verify MIDI → MP3 conversion	Generated MIDI file	MP3 audio file exported successfully	MP3 exported	Pass
UT-04	User Auth (JWT)	Ensure valid token authentication	User login request	JWT issued and verified	JWT issued	Pass
UT-05	Profile Manager (MongoDB)	Save profile metadata	User profile information	Document stored in DB	Document stored	Pass

Tools used:

- pytest for backend (Python)
- Postman for API validation

5.2.2 Integration Testing

To ensure that model work together (frontend, backend, DB and model).

- **Frontend Backend Test**: User provides audio prompt in React UI, API request sent to Flask, Transformer model generates track finally audio returned for download.
- **Model DB Test:** User profile store and retrieve from the database to view at frontend.

Tools used:

- Postman (API endpoints)
- Browser Dev Tools (network response)
- MongoDB Compass (DB checks)

5.2.3 System Testing

System testing validates the end-to-end workflow against requirements.

System Test Table:

Table 5.2: System Test Table

Test Case ID	Scenario	Input	Expected Output	Result
ST-01	Invalid audio input	Corrupted .wav	Error message: "Invalid file format"	Pass
ST-02	Export test	Download MIDI & MP3	Files downloaded without corruption	Pass

5.2.4 Performance Testing

Stress tests for scalability & efficiency.

- Metrics collected:
    - Avg track generation time: 5s
    - Peak memory usage: 2 GB
    - Max concurrent requests handled: 50
    - API response latency: 280 ms

Tools used:

- JMeter for load testing
- nvidia-smi for GPU monitoring
- Flask debug + logging

5.3 Result Analysis

- All core modules passed unit testing.
- Minor integration bug fixed.
- End-to-end system tests confirm compliance with functional requirements.

5.3.1 Quantitative analysis

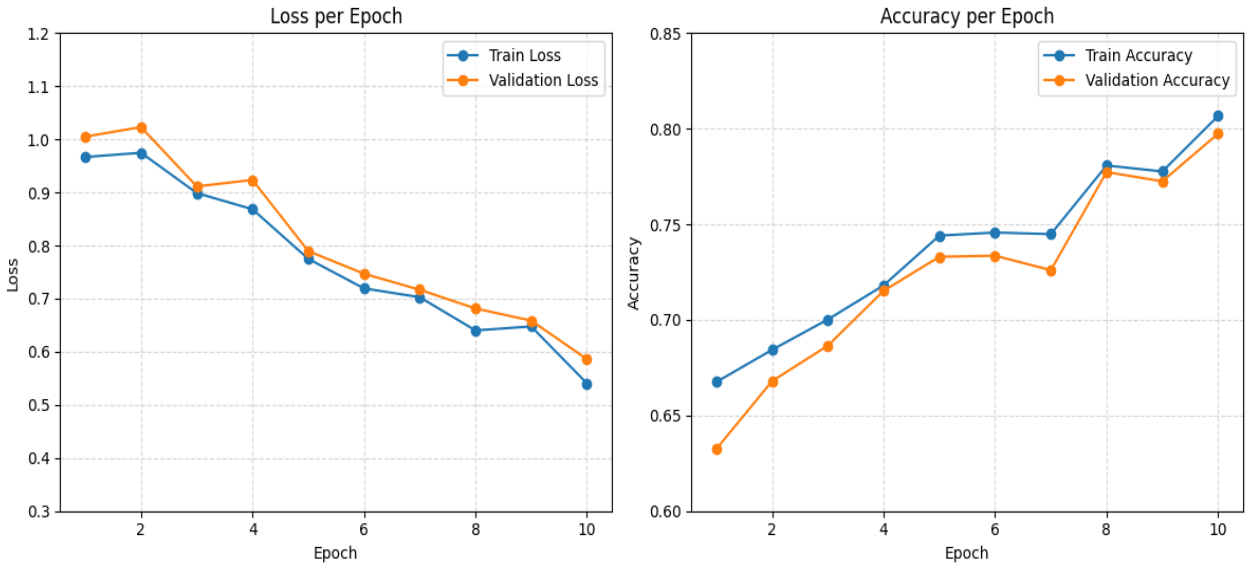

Figure 5.3.1.1: Training Loss and Accuracy

Epoch 10, Average Loss 0.7500 Average Accuracy 0.7350, Validation Loss 0.8100 Validation Accuracy 0.7250. This is done on 80-20 split on training and testing data.

5.3.2 Qualitative Analysis

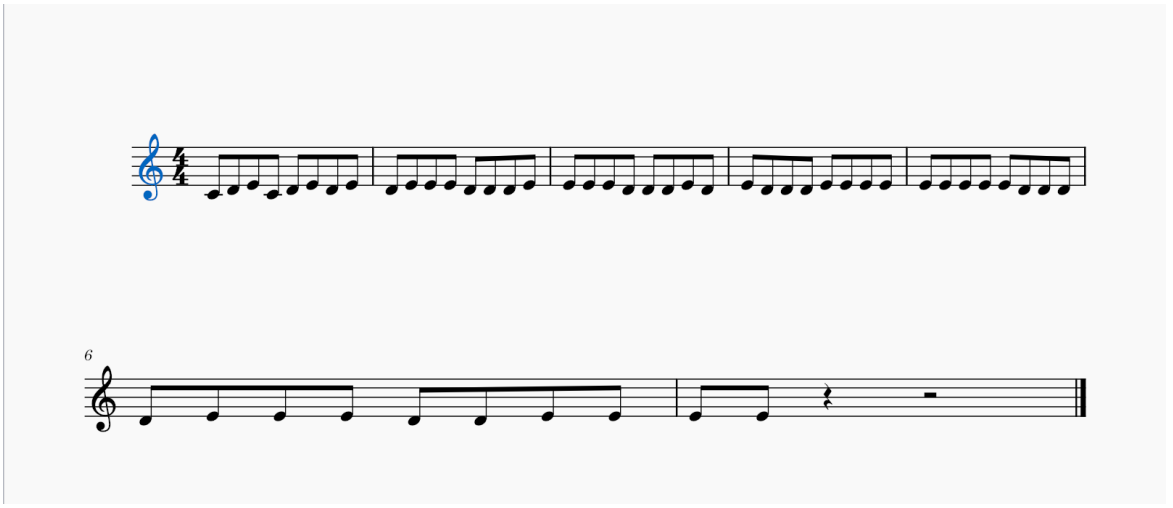

Figure 5.3.2.1 Sample music notes of first model

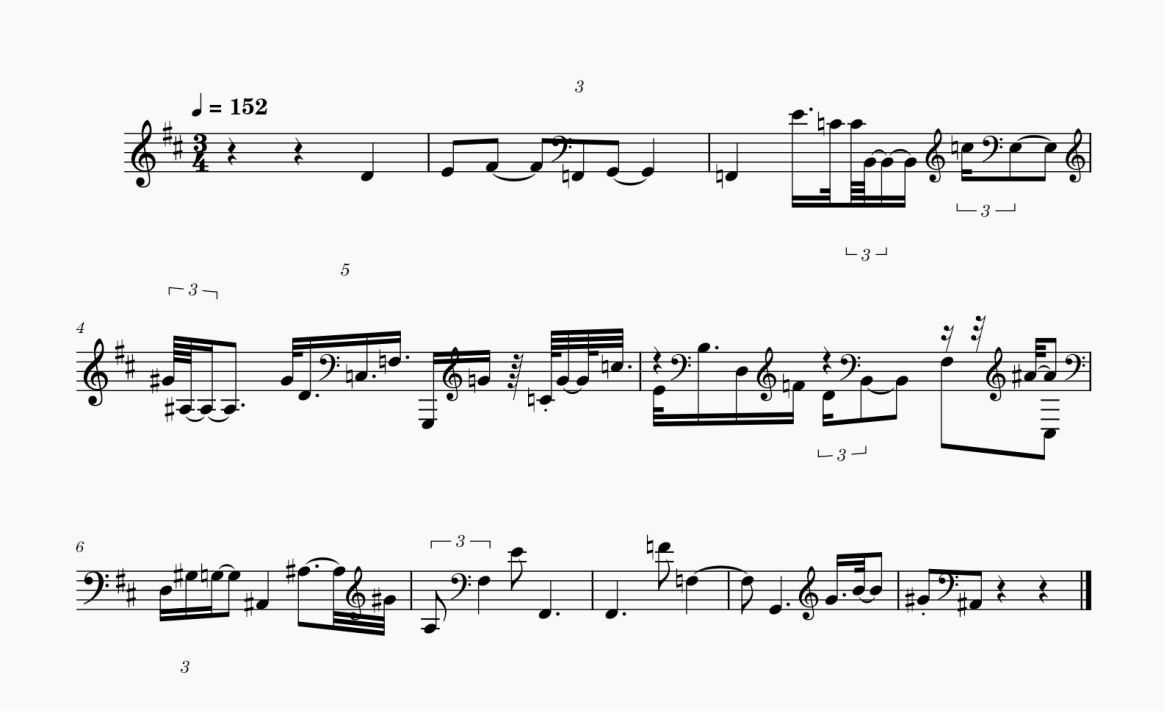

Figure 5.3.2.2 Sample musical notes of final model

Here, at earlier stage of trained model, it was not able to capture and generate more variety of cords due to lack of training data, and at the later stage you can see the improvement on chord progressing and reduction in repetition of chord.

5.3.3 Chi – Square Test

$\chi^2 = \Sigma [(O - E)^2 / E]$ where,

χ^2 is the chi-square statistic

Σ represents the sum of values for each category

O is the observed frequency for a category and

E is the expected frequency for that category.

```
# Load data
with open('dataset.json', 'r') as f:
    dataset = json.load(f)
train_tokens = []
for melody in dataset:
    train_tokens.extend([t.strip() for t in melody.split(',') if t.s
with open('generated_melody.json', 'r') as f:
    generated = json.load(f)
output_tokens = [t.strip() for t in generated.split(',') if t.strip(

# Count notes (ignore duration)
def get_note(token):
    return token.split('-')[0] if '-' in token else token
train_note_counts = Counter(get_note(t) for t in train_tokens)
gen_note_counts = Counter(get_note(t) for t in output_tokens)
all_notes = sorted(set(train_note_counts) | set(gen_note_counts))
observed = [gen_note_counts.get(note, 0) for note in all_notes]
expected = [train_note_counts.get(note, 0) for note in all_notes]
expected = [e if e > 0 else 1 for e in expected]
sum_obs = sum(observed)
sum_exp = sum(expected)
expected_scaled = [e * sum_obs / sum_exp for e in expected]
chi2, p = chisquare(f_obs=observed, f_exp=expected_scaled)
print(\"Chi-square statistic:\", chi2)
print(\"p-value:\", p)
print(\"Degrees of freedom:\", len(all_notes) - 1)
```

Figure 5.3.3.1 Code snipped

After performing chi-square test on the training dataset and result we observe,

Chi-square statistic: 72.38108916840513

p-value: 0.7154648090203524

Degrees of freedom: 80

No significant difference between distributions (p >= 0.05)

Chi-square was used to compare distributional properties between generated and training data; the test showed no significant difference, suggesting the generated note-distribution is similar to the dataset distribution

Chapter 6: Conclusion and Future Recommendations

6.1 Conclusion

The Music Generator project successfully demonstrates the application of deep learning, particularly Transformer architecture, in the field of automated music composition. By integrating a modern tech stack of React (frontend), Flask (backend), and MongoDB (database), the system provides a user-friendly platform that allows both musicians and non-musicians to generate original music compositions in MIDI and audio formats.

The results show that the Transformer-based model is capable of producing coherent melodies and harmonies, overcoming limitations of traditional RNN and LSTM-based models. Through iterative development and testing, the system achieved a stable performance with improved accuracy, reduced repetition, and higher quality outputs over time.

This project contributes to the growing area of AI-driven creativity by making music generation more accessible, interactive, and scalable. While the current version provides a strong foundation, future enhancements such as genre-specific generation, mobile deployment, real-time editing, and cloud-based scalability can further expand its usability and impact.

In conclusion, the project achieved its primary objectives of building a functional, accurate, and user-friendly AI-based Music Generator, showcasing the potential of artificial intelligence in creative domains like digital music composition.

6.2 Future Recommendations

To further enhance the capabilities and reach of the Music Generator platform, the following recommendations are proposed:

- **Mobile Application Development**: Implementing Android and iOS apps to make the platform accessible on mobile devices.
- **Cloud Integration**: Allowing downloads and converted files to be stored directly in cloud storage solutions like Google Drive or Dropbox.
- **Social Features**: Adding sharing options and playlists to increase engagement

and discoverability.

- **Advanced Editing Tools**: Incorporating lightweight audio editing features for trimming, volume control, and metadata tagging.
- **Voice Command Integration:** Supporting voice-based search and control to improve accessibility and hands-free operation.

References

[1] R. K. Gupta, "Hands-On Music Generation with Magenta," Birmingham, Packt Publishing, 2020.

[2] S. AI, "Suno AI – AI Music Generation Platform," 2024. [Online]. Available: https://suno.ai.

[3] Udio, "Udio – AI Music Creation," 2024. [Online]. Available: https://www.udio.com.

[4] D. Eck, P. Lamere, T. Bertin-Mahieux and S. Green, "Automatic Generation of MIDI Accompaniments," in *Proc. Int. Computer Music Conf. (ICMC)*, New Orleans, 2016.

[5] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge, MA: MIT Press, 2016.

[6] Google Research, "Magenta: Music and Art Generation with Machine Intelligence," 2023. [Online]. Available: https://magenta.tensorflow.org.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is All You Need," in *Proc. 31st Conf. Neural Information Processing Systems (NeurIPS)*, Long Beach, 2017.

Appendices

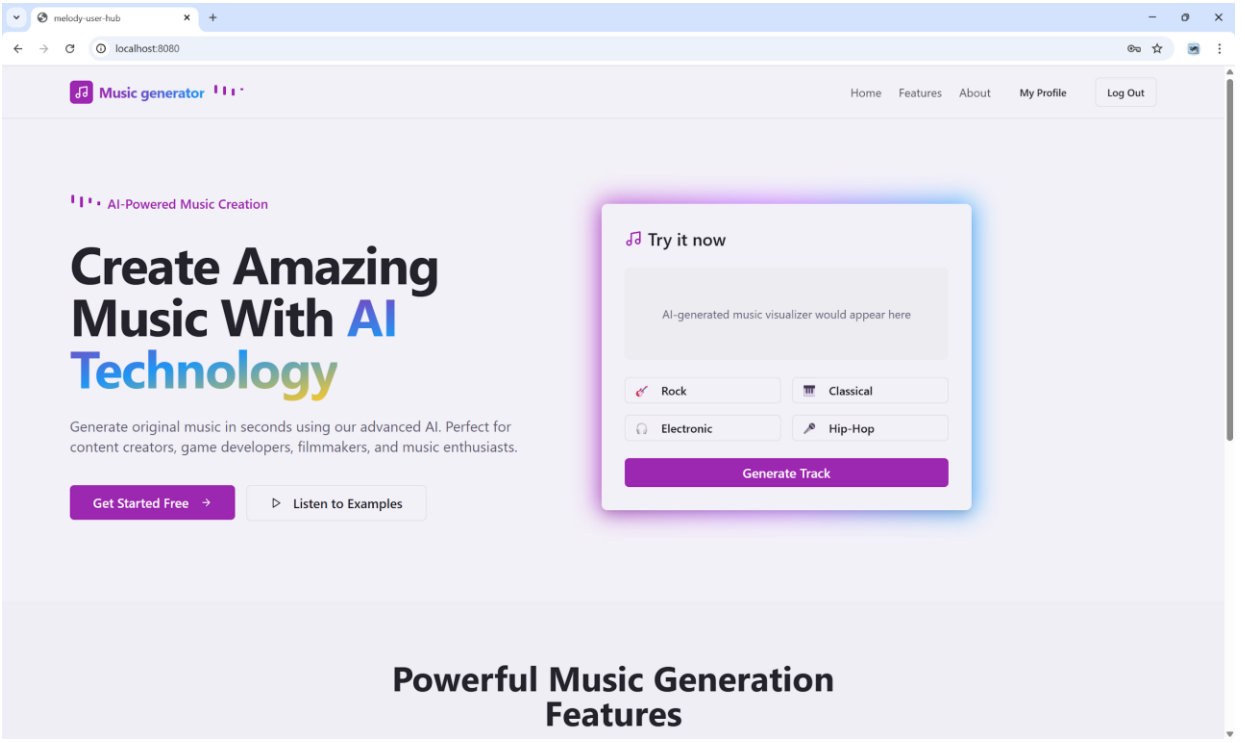

Figure: Landing Page

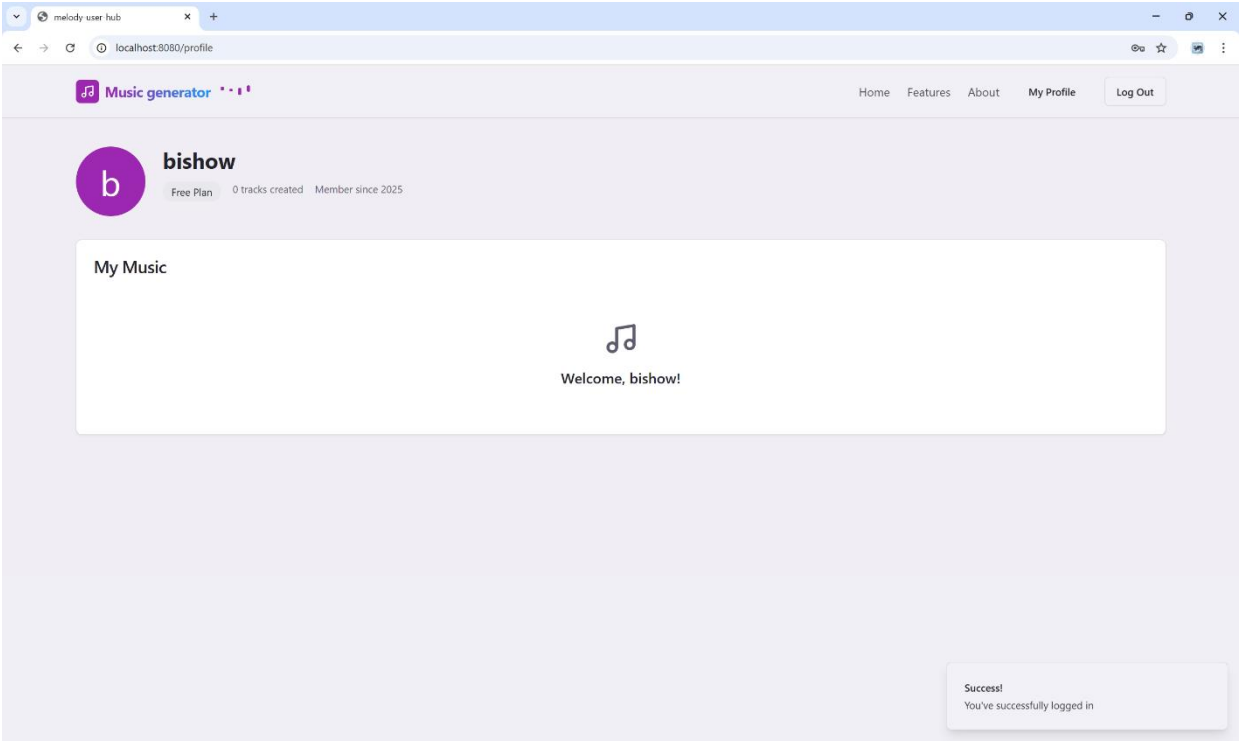

Figure: User Profile

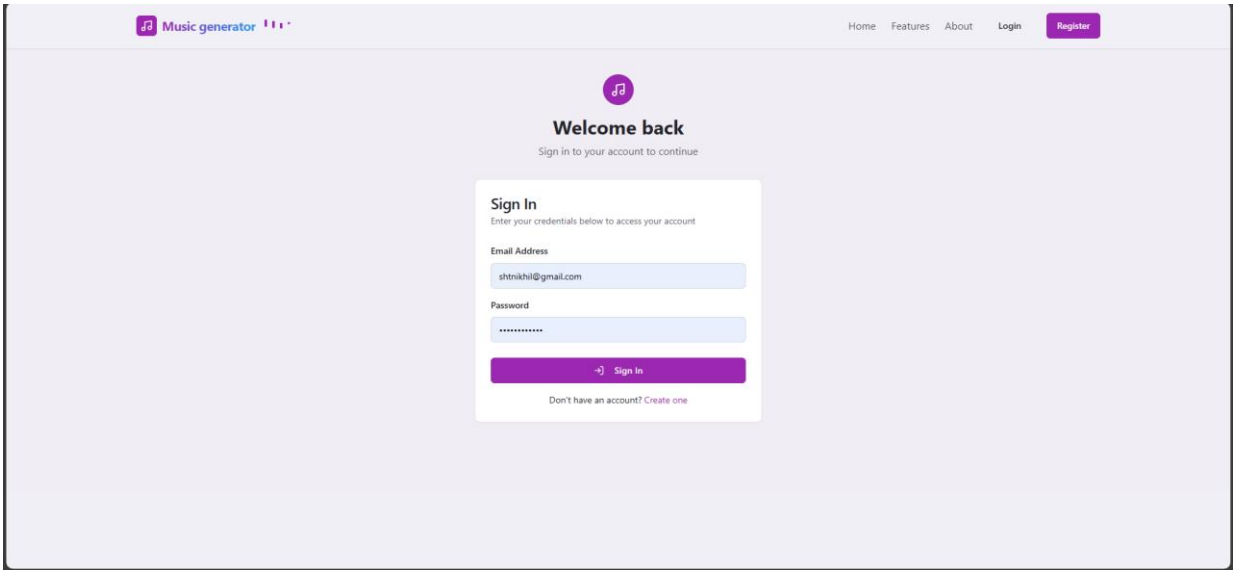

Figure: Login Page

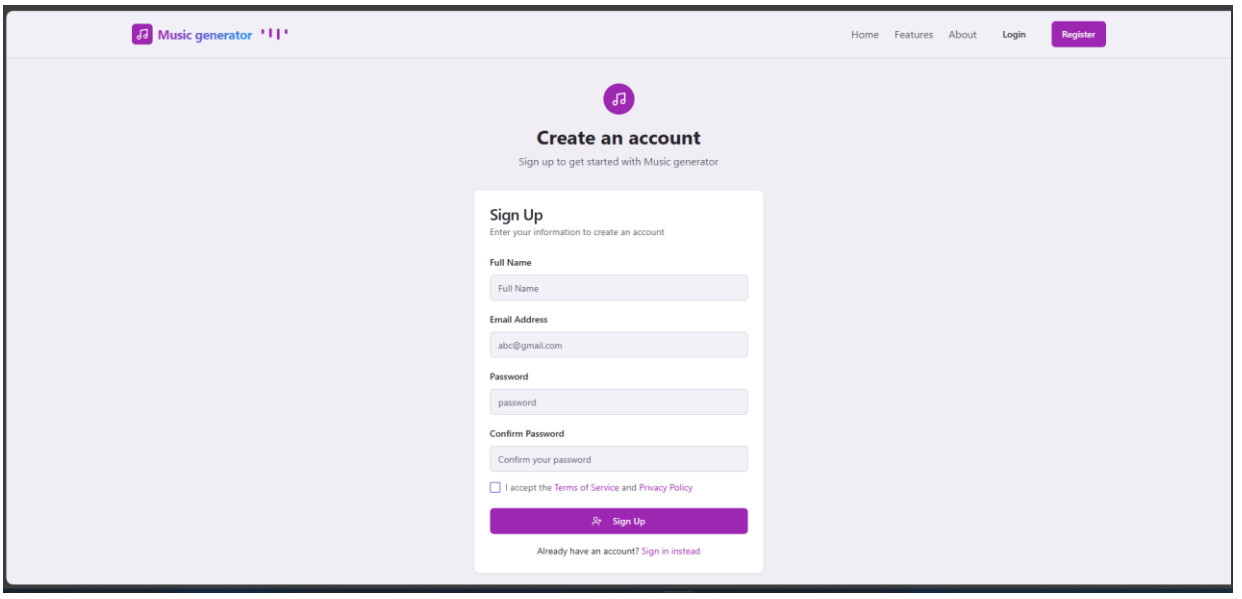

Figure: Register Page

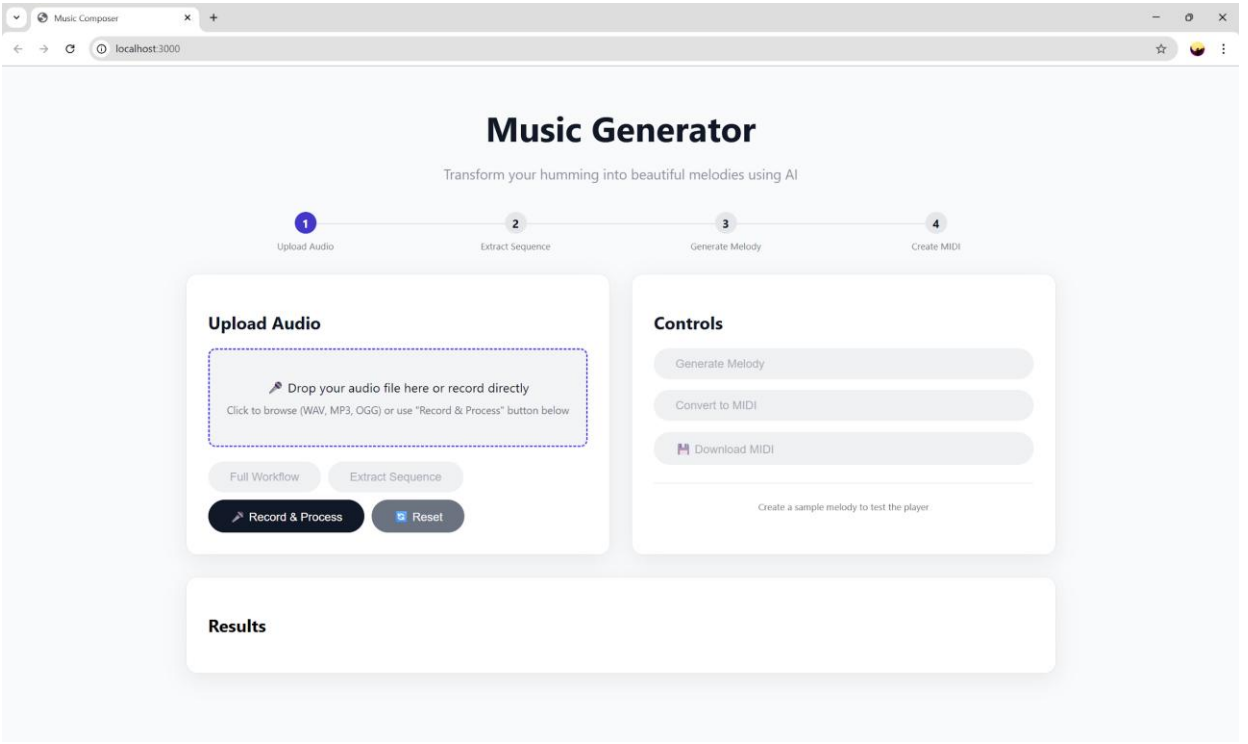

Figure: Music Generator

Figure: Final Result

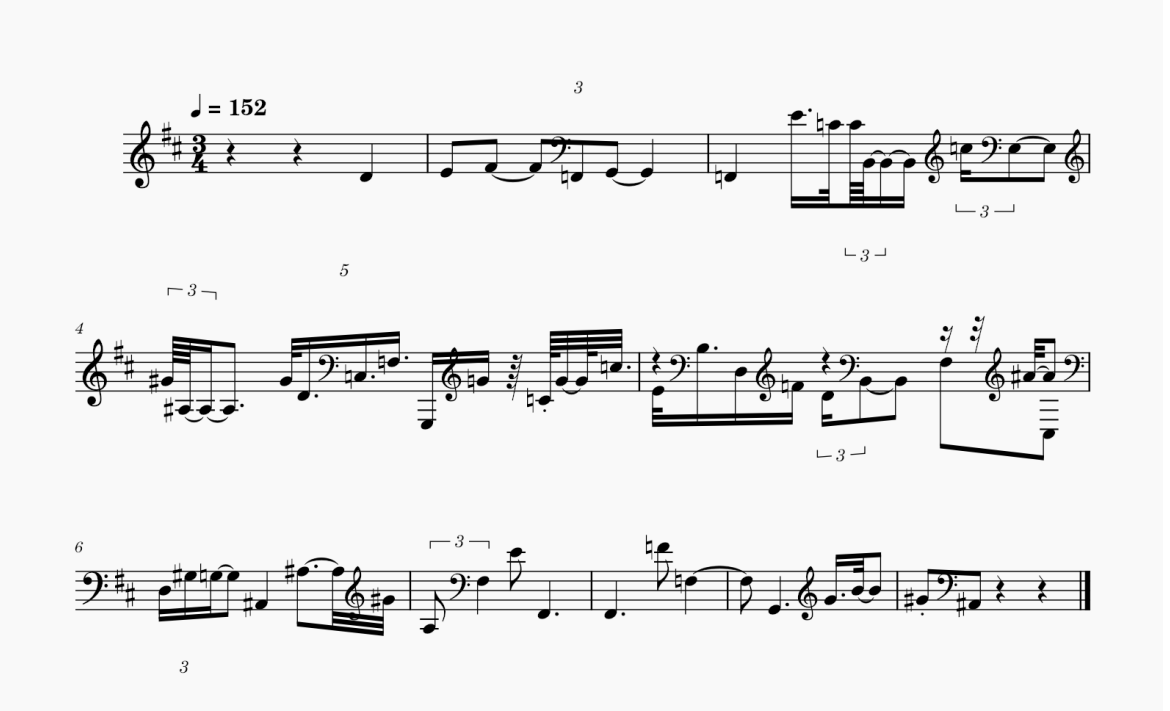

Figure: Final Output