APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 6 No. 1 (2025) 2076-2092
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

Musical Chords Recognition System Using Artificial Intelligence Techniques

Mohammed AbdulWahab[1], Abd Samad Hasan Basari[1*]

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: abdsamad@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2025.06.01.115

Article Info

Received: 29 July 2024
Accepted: 11 June 2025
Available online: 30 June 2025

Keywords

Music Chords Recognition, AI-Powered System, NNLS Chroma Algorithm, Audio Input Recognition, Music Technology

Abstract

In response to the evolving landscape of music creation and technology, the ChordWizard AI-powered Music Chords Recognition Web System addresses the challenges of accurately recognizing music chords. The primary objective is to develop and evaluate the effectiveness of ChordWizard in improving music chord recognition. The project examines modules such as Register and Login, Audio Input Recognition, Music Theory, MIDI File Sharing, Forum, Update Profile, and Administrative Tasks. Utilizing the advanced NNLS Chroma algorithm, Python Audio and chords libraries, and the Linux operating system, the system ensures precision and reliability. Key findings show that ChordWizard significantly improves the accuracy and speed of music chord recognition. The Audio Input Recognition module effectively understands diverse musical compositions, from single instruments to genres like pop, classical, and EDM. These results highlight the transformative potential of ChordWizard in music technology, addressing current challenges and paving the way for future enhancements. Future research may explore AI advancements for creating original chord compositions across different music genres.

1. Introduction

Manual chord transcription, often time-consuming and error-prone, hinders the creative flow for beginner musicians and composers [1]. ChordWizard emerges as a solution to this, a beacon of innovation set to transform how musicians interact with musical compositions. By harnessing the power of AI algorithms and machine learning models, ChordWizard aims to analyze audio recordings swiftly and accurately, providing musicians with a tool to expedite their creative processes and composers with an ally in composing intricate harmonies.

The main problems identified are inaccuracies, time inefficiencies, and a lack of creative inspiration in existing chord identification systems. ChordWizard aims to provide precise chord identification, streamline processes through automation, and offer a comprehensive repository of chord inspirations. These improvements are expected to enhance user satisfaction and reduce operational costs.

The subsequent sections detail specific modules such as Register and Login, Audio Input Recognition, Chords Course, Midi Sharing, Update Profile, and Administrative Tasks. Each module addresses essential aspects, from user authentication to chord analysis and collaborative features.

The expected result is a comprehensive online environment fostering collaboration and education. ChordWizard aims to be a go-to platform for both chord recognition and a thriving community of music enthusiasts and creators.

The significance of this project lies in its ability to address challenges in chord identification, offering a reliable method to enhance musical understanding, foster creativity, and promote collaboration. The journey through the chapters of literature review, methodology, analysis, design, implementation, and conclusion aims to present a coherent exploration of the project's facets, laying the foundation for future enhancements to the proposed system.

The objectives of ChordWizard are to design an intuitive interface for easy uploads and chord notations, develop an AI website for accurate chord recognition, test the system across diverse music genres, and bridge the gap between technology and music. The system targets a broad user base, including Music Producers, Musicians, Songwriters, Enthusiasts, Students, and Teachers of Music Theory. The integration of the Autochord library in Python and the use of Windows Subsystem for Linux (WSL) and the Django framework are key technical aspects of this project.

2. Related Work

Chord recognition, a pivotal aspect of music analysis, has spurred significant research and innovation in the development of various algorithms and platforms. This section presents a review of existing studies and systems, culminating in the emergence of ChordWizard—an advanced Music Chords Recognition System.

2.2 Previous Studies

The landscape of chord recognition algorithms has seen diverse approaches aimed at enhancing accuracy and efficiency. Noteworthy among these is the work by Sumarno (2018), who introduced a novel technique for feature extraction in chord recognition using Segment Averaging with Simplified Harmonic Product Spectrum and logarithmic scaling, achieving high efficiency for recognizing guitar chords with minimal coefficients [2].

Sigtia et al. (2015) contributed with an innovative framework incorporating a hybrid recurrent neural network (RNN) for audio chord recognition. Their model, which replaced hidden Markov models (HMMs) with RNNs, demonstrated the capacity of feed-forward deep neural networks (DNNs) for direct feature learning, performing comparably to existing state-of-the-art approaches [3]. Similarly, Park et al. (2019) introduced a bi-directional transformer for musical chord recognition (BTC), leveraging a self-attention mechanism to efficiently capture long-term dependencies in audio sequences, showing competitive performance with a single-phase training approach [4].

Weiss and Müller (2015), although not directly focused on chord recognition, explored tonal complexity features for the style classification of classical music, highlighting the significance of extracting meaningful characteristics for musical analysis [5]. Mauch and Dixon (2010) challenged traditional chord transcription methods by identifying chroma features better suited for musically motivated models, employing approximate transcription using non-negative least squares (NNLS), and achieving an 80% accuracy rate, which was a significant improvement over previous results [6].

To provide a comprehensive overview of these diverse chord extraction algorithms, Table 1, Chord Extraction Algorithm, is introduced below, summarizing key aspects of each method, including the author, year, main approach, and notable outcomes. This table serves as a reference point for readers seeking a comparative understanding of the discussed chord recognition methodologies.

Table 1 Chord Extraction Algorithm

Chord Extraction Algorithm	Efficient Feature Extraction	Specific Focus	Handling Long-Term Dependencies	Performance Comparison	Chord Extraction Algorithm

Segment Averaging	High	Guitar Chords	Low	High	Segment Averaging
Hybrid RNN	High	General Chord Recognition	High	Mid	Hybrid RNN
Bi-Directional Transformer	High	General Chord Recognition	High	Mid-High	Bi-Directional Transformer
Tonal Complexity Features	Low	Classical Music Style Classification	High	Not Directly Applicable (Style Classification)	Tonal Complexity Features
Approximate Note Transcription	High	General Chord Recognition	Low	High	NNLS

2.3 Existing System in Chords Recognition

Chordify, a notable existing system, provides automated chord charts for music. It analyzes audio tracks to identify and display chords in real time, offering musicians quick and accurate references. While Chordify boasts a user-friendly interface and clear chord charts, its limitations include reliance on existing songs for chord identification and a lack of dedicated features for MIDI file sharing. Figure 2 shows the interface of chordify.

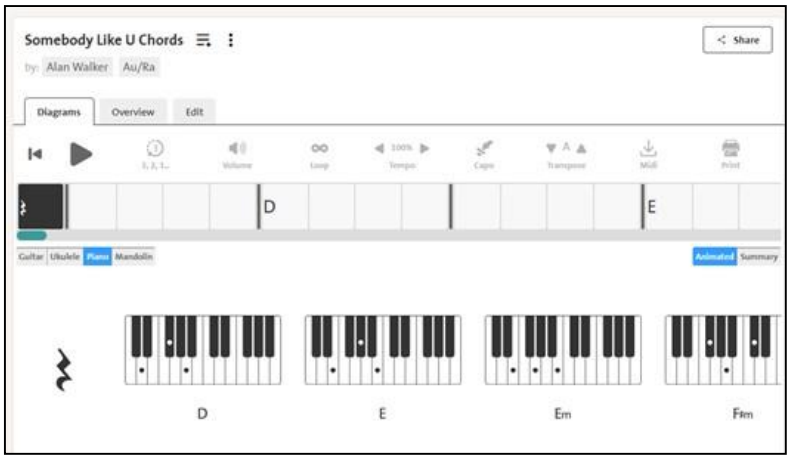


Fig. 1 *Chord Charts [7]*

2.4 ChordWizard: Advancements and Unique Features

ChordWizard, our proposed Music Chords Recognition System, distinguishes itself from existing platforms by offering a versatile and collaborative approach. Unlike Chordify, ChordWizard allows users to upload any song or sound file, expanding the scope of chord recognition. Moreover, ChordWizard features a dedicated Midi Sharing module, facilitating global collaboration among users and administrators.

This unique aspect positions ChordWizard as more than a chord identification tool—it transforms into a dynamic hub fostering a collaborative musical community, as it is shown in table 2. The system's innovative approach, drawing inspiration from previous studies, aligns with the evolving landscape of chord recognition, promising a comprehensive and user-centric platform for musicians and enthusiasts.

Table 2 Comparison with the Existing Systems

Feature	ChordWizard	Chordify
Chord Recognition Approach	AI-Powered with TensorFlow	-
Upload Flexibility	Any Audio File	Existing Songs in Database

MIDI Sharing Module	Yes	No
Collaborative Features	Yes (MIDI Sharing)	Limited
Interface User-Friendliness	Yes	Yes
Focus on Specific Instruments	No (Versatile)	No (General)
Performance Comparison	Anticipated High	Real-Time (May Vary)

3. Methodology/Framework

ChordWizard development rely on the Prototype Methodology, as shown in figure 2 follows a structured process encompassing key stages: define, sketch and design, create and develop, test and feedback, refine, and implement and maintain. This iterative approach allows for continuous refinement based on user feedback, ensuring that the system evolves in alignment with user expectations.

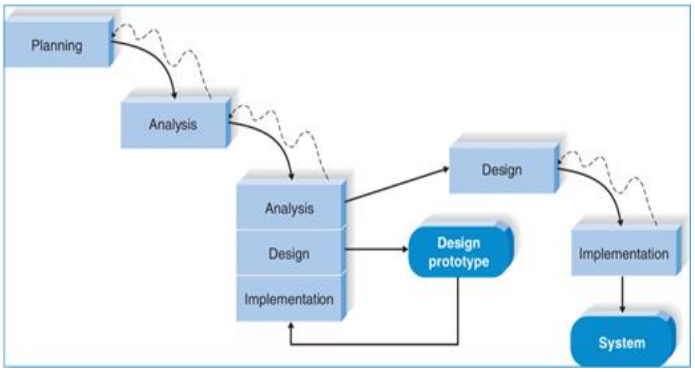


Fig. 2 *prtotype methodology*

Figure 2 illustrates the key phases involved in the Prototyping Model for the development of ChordWizard.

3.1 Planning Phase

In the Planning Phase, the project's problem, scope, and objectives are defined through activities such as acquiring and analyzing information, producing a proposal, and creating a Gantt chart for project management [8]. The proposal outlines challenges, objectives, approaches, and projected outcomes. Figure 3 displays the Gantt chart, illustrating the project timetable and task dependencies. This visual representation serves as a concise and effective tool for project planning and management.

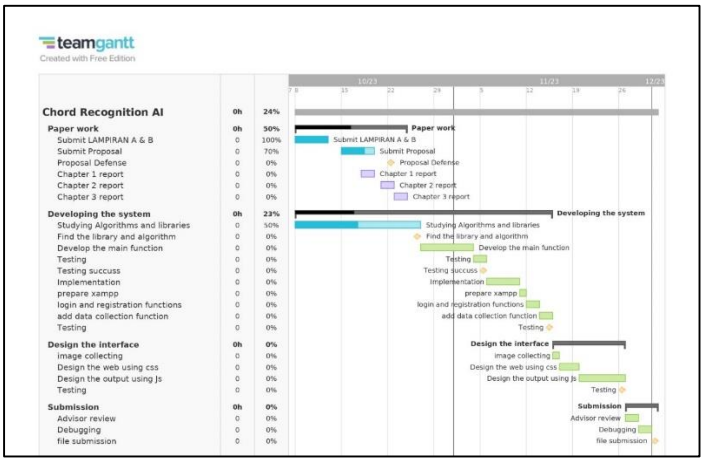


Fig. 3 *Gantt Chart*

3.2 Analysis Phase

In the analysis phase of the prototype methodology, the focus is on extracting clear project objectives. This stage involves a detailed examination of user requirements, system functionalities, and potential challenges. By

scrutinizing user stories and specifications, the analysis phase establishes a solid foundation for subsequent design and implementation, ensuring alignment with user needs and efficient solution development [6].

3.2.1 System Requirement Analysis

System Requirement Analysis is a crucial step in understanding what the ChordWizard system needs to accomplish. It involves both functional and non-functional requirements as shown in table 3 and 4.

Table 3 *Functional Requirements for Chord Wizard*

Modules	Function	User
Register and Login Module	• The system should allow the new user to register before login. • The system should show an error when the duplicate email is entered. • The system should display an error message when an empty field is found. • The system should allow the user to input a valid email and password to log in as a user. • The system should alert the user for invalid input. • The system should redirect to the dashboard once the user successfully login. • The system should allow the user to log in. • The system should show an error when the email and password are wrong. • The system should display an error message when an empty field is found.	User and Admin
Audio Input Recognition	• The chords recognition system identifies and displays chords for musical compositions. • Users can input a song or chord progression, and the system will visually represent the chords used.	User and Admin
Forum	• Users can create new posts by filling out a title and message. • The forum lists all existing posts, showing the email of the user who created each post. • When a post is selected, its detailed view appears, displaying the full title, and user information. • Users can add messages to a selected post by typing in an input field and submitting.	
Music Theory	• It offers detailed information on chord structures. • The module assists musicians in learning and playing music by explaining common chord formations.	User and Admin
Midi Sharing	• Offer the user and admin to share MIDI files with all users all around the world	User and Admin
Administrative Tasks	• Create, edit, and delete existing entries • Answer user inquiries and provide support	Admin
Update Profile	• The system should allow the user to update their profile. • The system should save the change.	User and Admin

Table 4 Non-Functional Requirements

No	Requirement	Description
1	Performance	The system is expected to maintain a consistently high level of performance, ensuring that users can interact with it seamlessly and without any noticeable delays
2	Operational	Users expect that the website's content and features load within one minute, providing a satisfactory and timely user experience.
3	Security	A user-friendly system enhances overall user satisfaction and efficiency in performing tasks.

4	Cultural and political	The requirement for compatibility across web browsers ensures accessibility for users worldwide, accommodating diverse cultural preferences and varying political contexts.

3.2.2 UseCase Diagram

The use case diagram in Figure 4 highlights the interactions within the ChordWizard application between the User and the Admin. The User can identify chords, create new posts, select and add messages to posts, view all posts, and access chord information. The Admin can upload MIDI files, update user information, provide support, and manage the database, including creating, editing, and deleting entries. The diagram employs "include" and "extend" relationships to detail dependencies and optional functionalities, ensuring a clear understanding of the system's user and admin interactions.

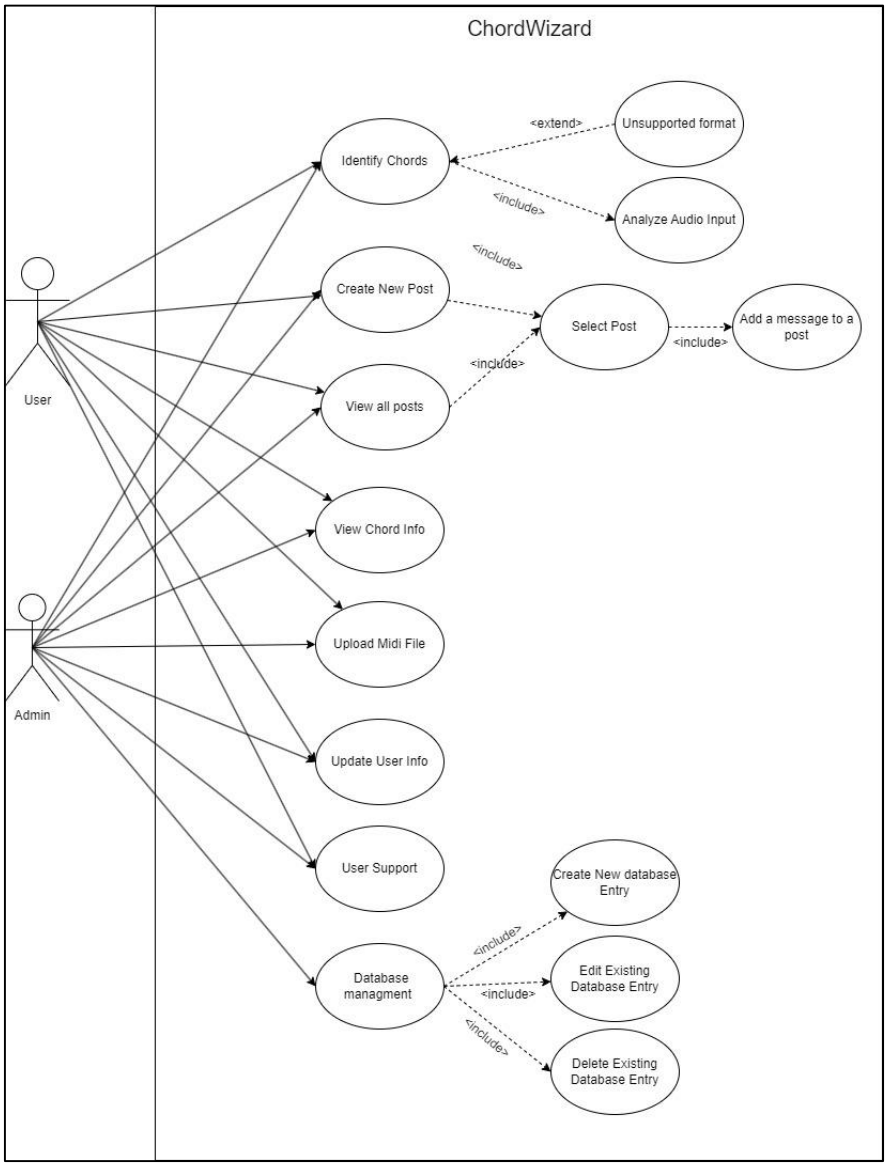


Fig. 4 *ChordWizard use case diagram*

3.2.3 Sequence Diagram

The sequence diagram in Figure 5 illustrates the interactions between the User/Admin, Home View, File Storage, Chord Recognizer, and Context components. The user requests the home page, uploads a file, and the system processes the file to recognize chords, updating the context with chords and file path. The final step involves rendering the home page with the recognized chords.

Applied Information Technology and Computer Science Vol. 6 No. 1 (2025) p. 2076-2092

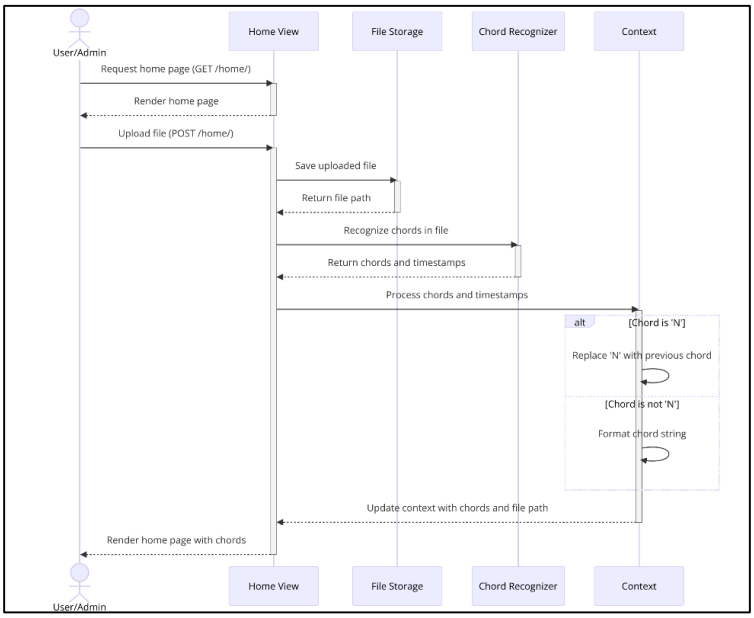


Fig. 5 *Chord Recognition sequence diagram*

3.2.4 Activity Diagram

The activity diagram in Figure 6 shows the flow of actions taken when a user uploads a file or simply requests the home page. If a file is uploaded, the system saves the file, processes it for chord data, processes the chord data, and then renders the home page with the updated context. If no file is uploaded, the system directly renders the home page.

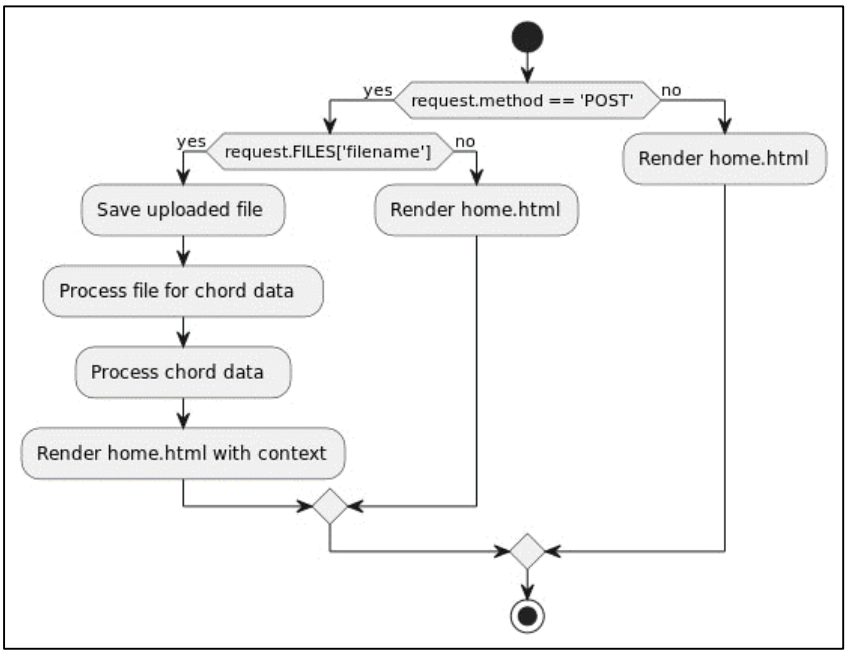
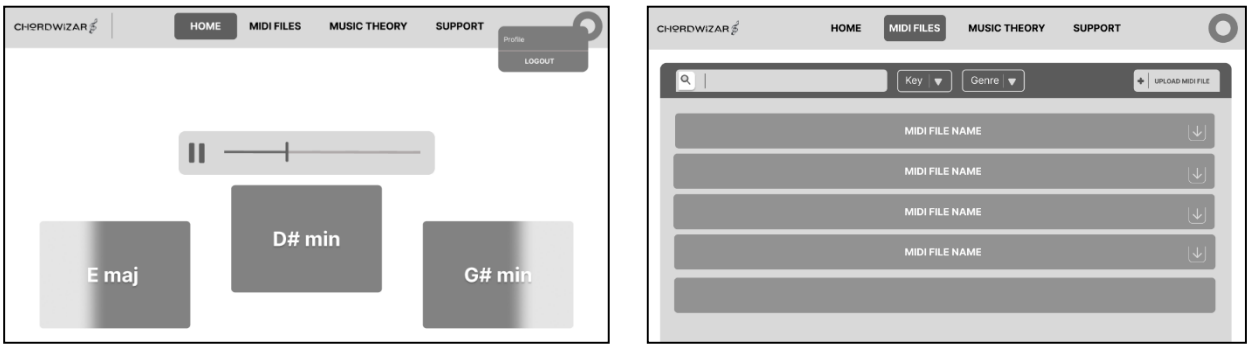


Fig. 6 *Chord Recognition Activity diagram*

3.2 Design Phase

The project will proceed with the design phase after the user's requirements have been met. During this phase, an interface and database were created to help visualize the system. before the implementation phase. Figures 7– 12 show the interface that has been designed for the system.



(a) **(b)**

Fig. 13 *(a) Home Page and chord representation, (b) Midi Share Page*

3.2.1 System Architecture

Figure 13 illustrates the user structure for the Chord recognition system, known as ChordWizard, consisting of two main user categories: Musicians and Admin. Each user is assigned a role based on their specific responsibilities within the system.

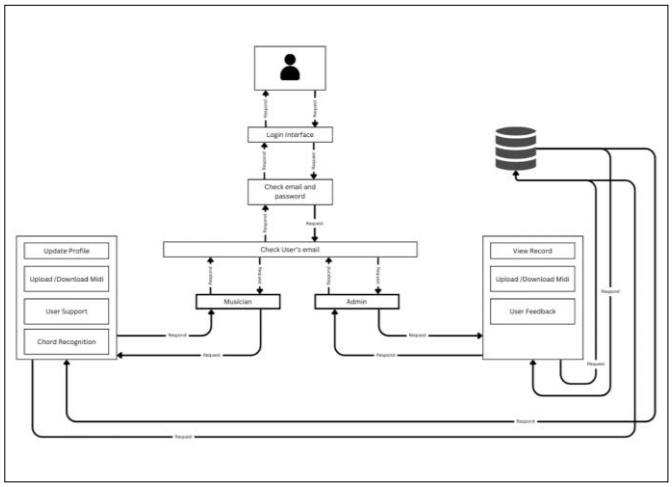


Fig. 13 *System Architecture*

3.3 Implementation Phase

The Implementation Phase for the Music Chords Recognition System involves critical activities. The user interface is constructed using HTML, CSS, JavaScript, and Django for front-end and back-end development. The NNLS Chroma algorithm, implemented in Python, analyzes audio for chord recognition, while Autochord streamlines transcription. The system is developed in a Windows Subsystem for Linux (WSL) environment.

3.3.1 Chord Recognition

The provided code snippet in Figure 14 – 17 is designed to recognize musical chords from audio files using a combination of chroma feature extraction and machine learning models. It sets up the necessary environment by importing relevant libraries and configuring parameters. The NNLS-chroma VAMP plugin is used for extracting chroma vectors from audio files. The system downloads and loads a pre-trained chord model for inference. Core functions generate chroma vectors from raw audio, predict chord labels, and recognize chords in audio files, optionally saving the results in a specified

format. The process involves downloading the model, initializing the module, and processing audio to output recognized chords with timestamps.

```python
import os
from shutil import copy
import pkg_resources
import numpy as np
from scipy.signal import resample
import gdown
import librosa
import vamp
import lazycats.np as catnp
from tensorflow import keras
_CHROMA_VAMP_LIB = pkg_resources.resource_filename('autochord', 'res/nnls-chroma.so')
_CHROMA_VAMP_KEY = 'nnls-chroma:nnls-chroma'
_CHORD_MODEL_URL = 'https://drive.google.com/uc?id=1XBn7FyYjF8Ff6EuC7PjwwPzFBLRXGP7n'
_EXT_RES_DIR = os.path.join(os.path.expanduser('~'), '.autochord')
_CHORD_MODEL_DIR = os.path.join(_EXT_RES_DIR, 'chroma-seq-bilstm-crf-v1')
_CHORD_MODEL = None

_SAMPLE_RATE = 44100            # operating sample rate for all audio
_SEQ_LEN = 128                  # LSTM model sequence length
_BATCH_SIZE = 128               # arbitrary inference batch size
_STEP_SIZE = 2048/_SAMPLE_RATE  # chroma vectors step size

_CHROMA_NOTES = ['C','Db','D','Eb','E','F','Gb','G','Ab','A','Bb','B']
_NO_CHORD = 'N'
_MAJMIN_CLASSES = [_NO_CHORD, *[f'{note}:maj' for note in _CHROMA_NOTES],
                   *[f'{note}:min' for note in _CHROMA_NOTES]]

def _setup_chroma_vamp():
    # pylint: disable=c-extension-no-member
    vamp_paths = vamp.vampyhost.get_plugin_path()
    vamp_lib_fn = os.path.basename(_CHROMA_VAMP_LIB)
    for path in vamp_paths:
        try:
            if not os.path.exists(os.path.join(path, vamp_lib_fn)):
                os.makedirs(path, exist_ok=True)
                copy(_CHROMA_VAMP_LIB, path)
            # try to load to confirm if configured correctly
            vamp.vampyhost.load_plugin(_CHROMA_VAMP_KEY, _SAMPLE_RATE,
                                       vamp.vampyhost.ADAPT_NONE)
            print(f'autochord: Using NNLS-Chroma VAMP plugin in {path}')
            return
        except Exception as e:
            continue

    print(f'autochord WARNING: NNLS-Chroma VAMP plugin not setup properly. '
          f'Try copying `{_CHROMA_VAMP_LIB}` in any of following directories: {vamp_paths}')
```

Fig. 14 *Configuration & Initializers Setup*

```python
def _download_model():
    os.makedirs(_EXT_RES_DIR, exist_ok=True)
    model_zip = os.path.join(_EXT_RES_DIR, 'model.zip')
    gdown.download(_CHORD_MODEL_URL, model_zip, quiet=False)

    model_files = gdown.extractall(model_zip)
    model_files.sort()
    os.remove(model_zip)
    print(f'autochord: Chord model downloaded in {model_files[0]}')
    return model_files[0]

def _load_model():
    global _CHORD_MODEL_DIR, _CHORD_MODEL
    try:
        if not os.path.exists(_CHORD_MODEL_DIR):
            _CHORD_MODEL_DIR = _download_model()

        _CHORD_MODEL = keras.models.load_model(_CHORD_MODEL_DIR)
        print(f'autochord: Loaded model from {_CHORD_MODEL_DIR}')
    except Exception as e:
        raise Exception(f'autochord: Error in loading model: {e}')

def _init_module():
    print('autochord: Initializing...')
    _setup_chroma_vamp()
    _load_model()

_init_module()


################
# Core Functions
################
def generate_chroma(audio_fn, rollon=1.0):
    """ Generate chroma from raw audio using NNLS-chroma VAMP plugin """

    samples, fs = librosa.load(audio_fn, sr=None, mono=True)
    if fs != _SAMPLE_RATE:
        samples = resample(samples, num=int(len(samples)*_SAMPLE_RATE/fs))

    out = vamp.collect(samples, _SAMPLE_RATE, 'nnls-chroma:nnls-chroma',
                       output='bothchroma', parameters={'rollon': rollon})

    chroma = out['matrix'][1]
    return chroma
```

Fig. 15 *Load Model & Chroma Generation*

```python
def predict_chord_labels(chroma_vectors):
    """ Predict (numeric) chord labels from sequence of chroma vectors """

    chordseq_vectors = catnp.divide_to_subsequences(chroma_vectors, sub_len=_SEQ_LEN)
    pred_labels, _, _, _ = _CHORD_MODEL.predict(chordseq_vectors, batch_size=_BATCH_SIZE)
    pred_labels = pred_labels.flatten()
    if len(chroma_vectors) < len(pred_labels): # remove pad
        pad_st = len(pred_labels)-_SEQ_LEN
        pad_ed = pad_st+len(pred_labels)-len(chroma_vectors)
        pred_labels = np.append(pred_labels[:pad_st], pred_labels[pad_ed:])

    assert len(pred_labels)==len(chroma_vectors)
    return pred_labels

def recognize(audio_fn, lab_fn=None):
    """
    Perform chord recognition on provided audio file. Optionally,
    you may dump the labels on a LAB file (MIREX format) through `lab_fn`.
    """

    chroma_vectors = generate_chroma(audio_fn)
    pred_labels = predict_chord_labels(chroma_vectors)

    chord_labels = catnp.squash_consecutive_duplicates(pred_labels)
    chord_lengths = [0] + list(catnp.contiguous_lengths(pred_labels))
    chord_timestamps = np.cumsum(chord_lengths)

    chord_labels = [_MAJMIN_CLASSES[label] for label in chord_labels]
    out_labels = [(_STEP_SIZE*st, _STEP_SIZE*ed, chord_name)
                  for st, ed, chord_name in
                  zip(chord_timestamps[:-1], chord_timestamps[1:], chord_labels)]

    if lab_fn: # dump labels to file
        str_labels = [f'{st}\t{ed}\t{chord_name}'
                      for st, ed, chord_name in out_labels]
        with open(lab_fn, 'w') as f:
            for line in str_labels:
                f.write("%s\n" % line)

    return out_labels
```

Fig. 16 *Recognizing Function*

```
(0.0, 1.0216780045351475, 'N')
(1.0216780045351475, 12.538775510204083, 'F:min')
(12.538775510204083, 15.650249433106577, 'Db:maj')
(15.650249433106577, 16.857687074829933, 'Bb:min')
(16.857687074829933, 18.250884353741498, 'Eb:maj')
(18.250884353741498, 24.241632653061224, 'F:min')
(24.241632653061224, 28.699863945578233, 'Db:maj')
(28.699863945578233, 29.72154195011338, 'Eb:maj')
(29.72154195011338, 34.87637188208617, 'F:min')
(34.87637188208617, 35.944489795918365, 'Eb:maj')
(35.944489795918365, 38.963083900226756, 'Db:maj')
(38.963083900226756, 40.35628117913832, 'Bb:min')
(40.35628117913832, 41.61015873015873, 'Eb:maj')
(41.61015873015873, 46.300589569161, 'F:min')
(46.300589569161, 47.55446712018141, 'Eb:maj')
(47.55446712018141, 48.34394557823129, 'Bb:min')
(48.34394557823129, 48.99410430839002, 'Eb:maj')
(48.99410430839002, 49.73714285714286, 'Ab:maj')
(49.73714285714286, 50.758820861678004, 'Db:maj')
(50.758820861678004, 51.78049886621315, 'Bb:min')
(51.78049886621315, 53.498775510204084, 'N')
(53.498775510204084, 57.12108843537415, 'F:maj')
(57.12108843537415, 58.096326530612245, 'F:min')
(58.096326530612245, 59.44308390022676, 'Eb:maj')
(59.44308390022676, 61.11492063492064, 'N')
(61.11492063492064, 65.38739229024944, 'C:maj')
```

Fig. 17 *Model Output*

3.3.2 Login and Registration Module

Figure 18 and Figure 19 illustrate the login functionality of a web application. Figure 18 shows the backend code for handling the login process using Django, a Python web framework. Figure 19 shows the corresponding frontend login form, where users enter their email address and password. The form is simple and user-friendly, with a clear call-to-action button for logging in and a link to create a new account if the user is not already registered.

```python
def login_page(request):
    # user = request.user
    domain_name = request.build_absolute_uri('/')[:-1]
    if request.method == "POST":
        email = request.POST['email']
        password = request.POST['password']

        user = authenticate(request, username=email, password=password)

        if user is not None:
            login(request, user)
            return redirect("/")

        else:
            messages.error(request, f'The email or password is incorrect')

    return render(request, "accounts/login.html")
```

Fig. 18 *Login Function*

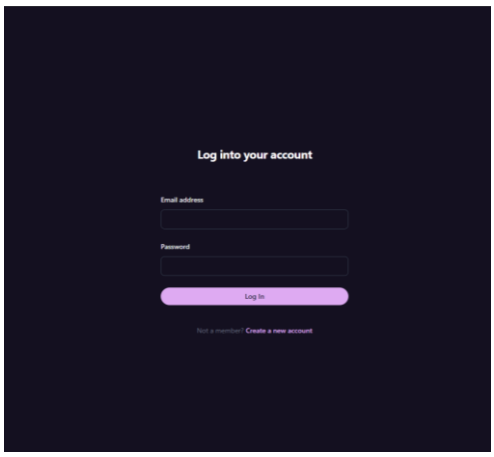


Fig. 19 *Login Page Interface*

Figures 20 and 21 depict the registration functionality of a web application. Figure 20 shows the backend code for the registration process using Django. Figure 21 presents the frontend registration form, where users can enter their email address, and password, and confirm their password. The form is designed to be clean and straightforward, with a clear call-to-action button for registration and a link to log in for users who already have an account.

```python
def register_page(request):
    domain_name = request.build_absolute_uri('/')[:-1]

    if request.method == "POST":
        form = UserAccountForm(request.POST)

        if form.is_valid():
            user = form.save()

            return redirect('/accounts/login/')

        else:
            for field, errors in form.errors.items():
                for error in errors:
                    messages.error(request, f"{error}")

            return render(request, "accounts/register.html", { 'form': form })

    user_form = UserAccountForm()

    context = {
        'form': user_form,
    }

    return render(request, "accounts/register.html", context)
```

Fig. 20 *Registration Function*

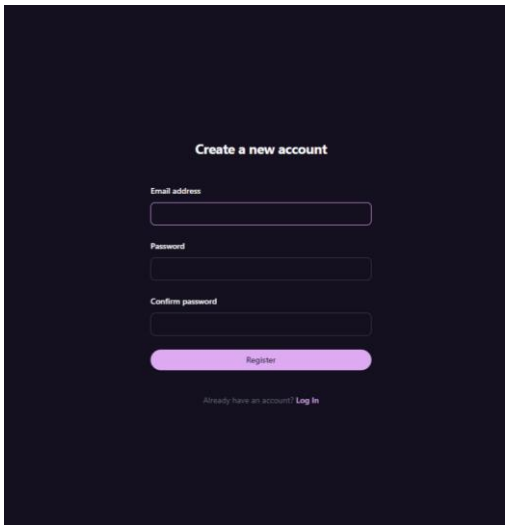


Fig. 21 *Registration Page Interface*

3.3.3 Audio Recognition Module

Figures 22 and 23 illustrate the chord detection functionality of a web application. Figure 22 shows the backend code for processing uploaded audio files and recognizing chords using the `home` function in Django. The function handles file uploads, calls the `recognize` function to identify chords in the uploaded audio file, and processes the chord data. It then updates the context with the chords and the file path before rendering the "home.html" template. Figure 23 displays the frontend interface, where users can see the detected chords visualized in sequence as they upload and play their audio files. The interface provides a user-friendly experience for discovering and viewing song chords in real-time.

Fig. 22 *Audio Input Function*

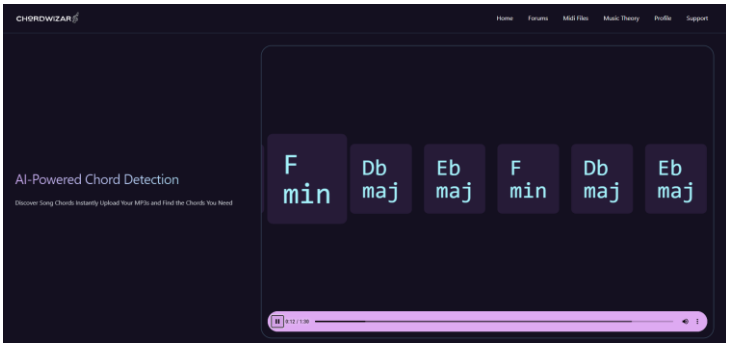


Fig. 23 *Home Page Interface*

3.3.4 Forum Module

Figures 24 and 25 showcase the forum functionality of a web application. Figure 24 presents the backend code for handling forum posts and messages using Django. The `forum` function processes GET and POST requests, allowing users to view selected posts and add new messages. It validates and saves the form data, updating the context with the post and message lists before rendering the "forum.html" template. Figure 25 shows the frontend forum interface, where users can view and post messages in a selected thread. The interface is designed to be clean and interactive, facilitating easy communication among users.

Fig. 24 *Forum Function*

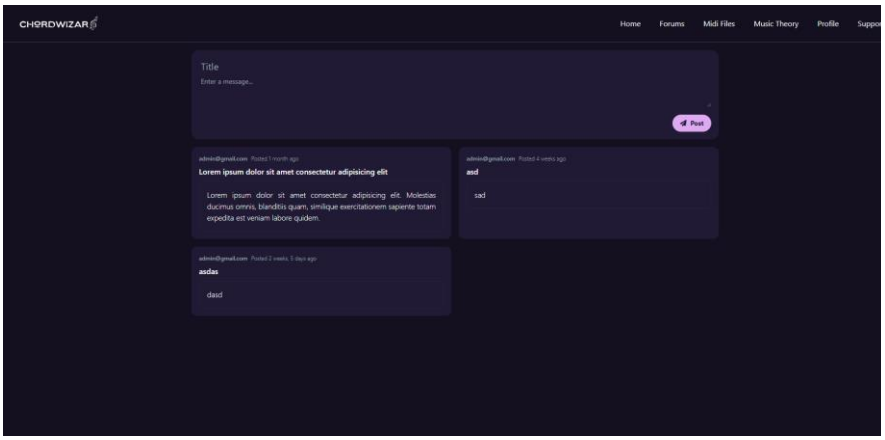


Fig. 25 *Forum Page Interface*

3.3.5 Music Theory Module

Figures 26 and 27 illustrate the music theory section of a web application. Figure 26 shows the backend code for displaying chapters related to music theory. The `music_theory` function retrieves all chapters from the database, adds them to the context, and renders the "music_theory.html" template. Figure 27 presents the frontend interface, where users can explore various music theory topics. The interface includes a sidebar with links to different chapters and a main content area displaying detailed information about the selected topic.

```python
def music_theory(request):

    chapter_list =  Chapter.objects.all()

    context = {
        'chapter_list': chapter_list,
    }

    return render(request, "music_theory.html", context)
```

Fig. 26 *Music Theory Function*

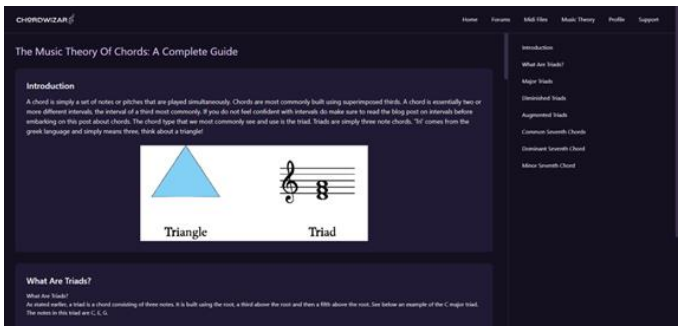


Fig. 27 *Music Theory Page Interface*

3.3.6 Midi Share Module

Figures 28 and 29 depict the MIDI files management section of a web application. Figure 28 shows the backend code for handling MIDI file uploads and filtering using Django. The `midi_files` function processes POST requests to add new MIDI files or filter existing files based on various criteria such as name, genre, key, and emotion. It saves valid file uploads and filters the list of MIDI files accordingly, updating the context with the form and the filtered list before rendering the "midi_files.html" template. Figure 29 illustrates the frontend interface, where users can upload new MIDI files and filter existing ones using different attributes. The interface provides options to download the filtered MIDI files, offering a user-friendly way to manage and access a collection of MIDI files.

```python
@login_required
def midi_files(request):
    midi_files_list = MidiFile.objects.all()

    if request.method == "POST":
        if "add_post" in request.POST:
            form = MidiFileForm(request.POST, request.FILES)

            if form.is_valid():
                file = form.save(commit=False)
                file.user = request.user
                file.save()
            else:
                print(form.errors)

        elif "filter" in request.POST:
            name = request.POST['name']
            genre = request.POST['genre']
            key = request.POST['key']
            emotion = request.POST['emotion']

            if name:
                midi_files_list = midi_files_list.filter(name=name)
            if genre:
                midi_files_list = midi_files_list.filter(genre=genre)
            if key:
                midi_files_list = midi_files_list.filter(key=key)
            if emotion:
                midi_files_list = midi_files_list.filter(emotion=emotion)

    form = MidiFileForm()

    context = {
        'form': form,
        'midi_files_list': midi_files_list
    }
    return render(request, "midi_files.html", context)
```

Fig. 28 *Midi Share Function*

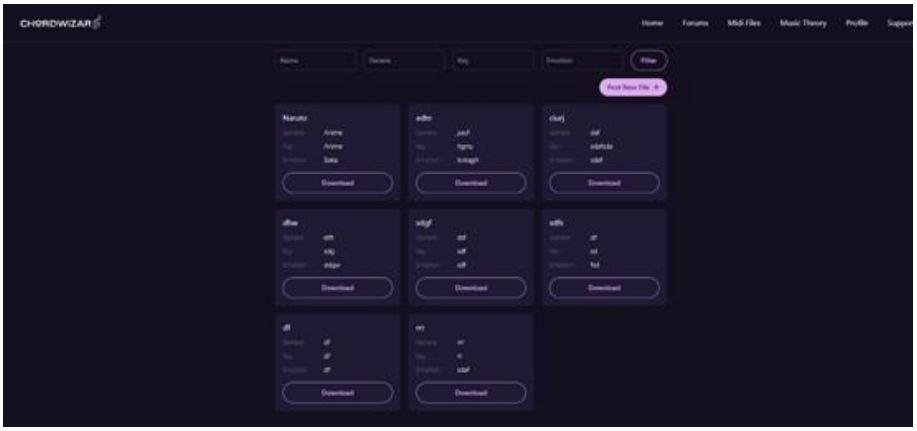


Fig. 29 *Midi Share Page Interface*

3.3.7 Administrative Tasks Module

Figures 30 and 31 display the admin panel functionality of a web application. Figure 30 shows the backend code for managing the admin panel using Django. The code imports necessary modules and defines utility functions to get model classes and create model forms dynamically. These functions assist in managing different models within the admin panel. Figure 31 illustrates the frontend admin interface, where an admin can view and edit chapters of a music theory guide. The interface is designed with a sidebar for navigation, providing access to different models and functionalities such as posts, MIDI files, testimonials, and chapters. This admin panel allows for efficient content management, ensuring that administrators can easily update and maintain the application's data.

```python
from django.shortcuts import render, HttpResponse, redirect
from .admin import models
# from accounts.models import UserAccount
from django.forms import ModelForm
from django.contrib import messages
from django.contrib.auth.decorators import user_passes_test, login_required
from django.http import JsonResponse


import datetime
today = datetime.date.today()


# UTILITY FUNCTIONS
def get_model(name:str):
    '''Returns the registered array from the admin.py. '''
    for TableClass in models:
        if TableClass.name == name:
            return TableClass

    raise ValueError("Table Not Found")

def create_form(model):
    '''Create and return a django ModelForm for the given model'''
    model_obj = model
    class AutoForm(ModelForm):
        class Meta:
            model = model_obj
            fields = "__all__"

    return AutoForm

def is_admin(user):
    return user.is_superuser
```

Fig. 30 *Administrative Tasks Function*

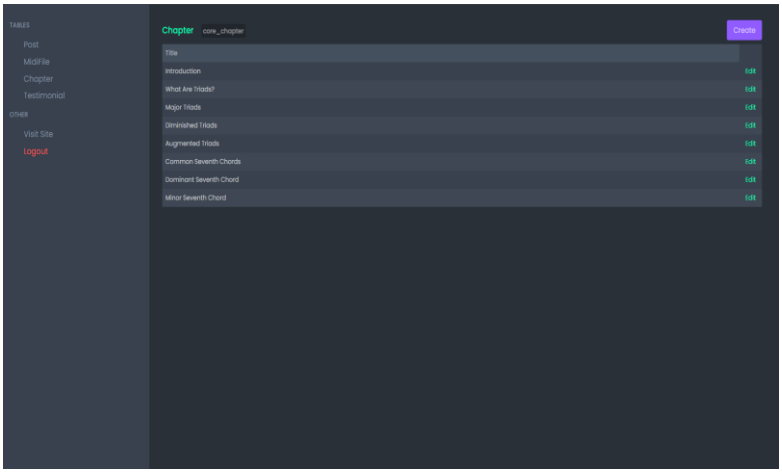


Fig. 31 *Administrative Tasks Function*

3.3.8 Update Profile Module

Figures 32 and 33 illustrate the profile management section of a web application. Figure 31 shows the backend code for handling various profile-related actions using Django. The `profile` function processes POST requests to delete forum posts, delete MIDI files, save user information, and update profile photos. It performs the necessary operations based on the request data and updates the context with the user's forum list and MIDI files list before rendering the "profile.html" template. Figure 33 displays the frontend interface, where users can manage their profile details, view their forum posts, and access their uploaded MIDI files. The interface allows users to update their email, change their password, and manage their uploaded content, providing a comprehensive and user-friendly profile management experience.

```python
@login_required
def profile(request):
    if request.method == "POST":
        if "delete_forum" in request.POST:
            forum_id = request.POST['forum_id']
            forum = Post.objects.get(id=forum_id)
            forum.delete()

        elif "delete_file" in request.POST:
            file_id = request.POST['file_id']
            file = MidiFile.objects.get(id=file_id)
            file.delete()

        if "save_info" in request.POST:
            email = request.FILES['email']
            request.user.email = email
            request.user.save()

        else:
            photo = request.FILES['photo']
            request.user.img.delete()
            request.user.img = photo
            request.user.save()

    forum_list = Post.objects.filter(user=request.user)
    midi_files_list = MidiFile.objects.filter(user=request.user)

    context = {
        "forum_list": forum_list,
        "midi_files_list": midi_files_list,
    }

    return render(request, "profile.html", context)
```

Fig. 32 *Update Profile Function*

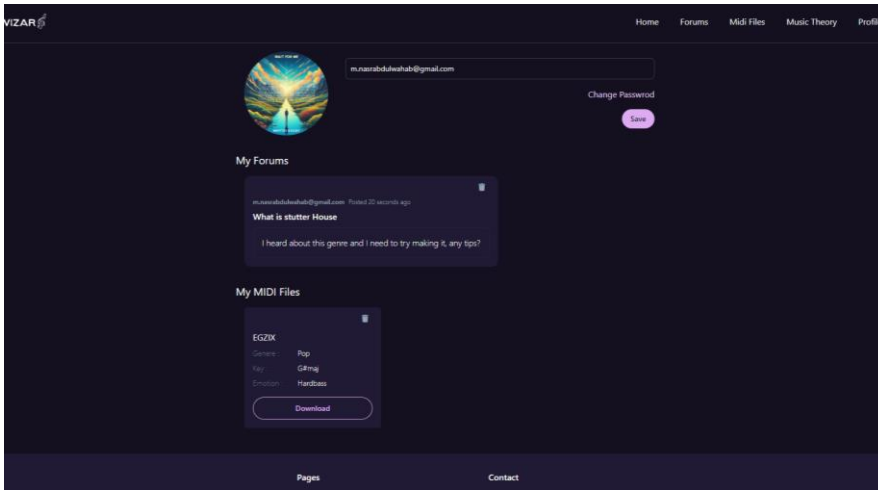


Fig. 33 *Update Profile Page Interface*

3.4 Testing Phase

The Testing Phase for the Music Chords Recognition System is a critical step to ensure that its functionality, performance, and reliability align with the defined requirements. During this phase, human testers engage in manual testing by following carefully designed test cases. After the prototype is completed, real-user feedback becomes vital to further refine the system. The functional testing for each module is detailed in specific tables: Table 5.1 covers the Login and Registration module; Table 5.2 covers the Audio Recognition module; Table 5.3 covers the Forum module; Table 5.4 covers the Music Theory module; Table 5.5 covers the MIDI Share module; Table 5.6 covers the Admin Support module; and Table 5.7 covers the User Profile module. This comprehensive

testing approach ensures that all aspects of the system are thoroughly evaluated to meet user expectations and operational requirements.

Table 5 Test Case for Login and Registration Module

Test Case ID	Description	Expected Result	Actual
LR-1	To check whether the user can register for an account.	The user should be able to create an account.	The user has successfully created an account.
LR-2	To check whether an administrator or user can log into the system.	The user or administrator should be able to log into the system.	The user has successfully logged into the system.
LR-3	To check whether the system will restrict login whenever a wrong credential is entered.	The system should restrict login when incorrect credentials have been entered.	The system restricted the login when incorrect or no credentials had been entered.

Table 6 Test Case for Audio Recognition Module

Test Case ID	Description	Expected Result	Actual
AR-1	To check whether the user can upload an audio file.	The user should be able to upload an audio file.	The user has successfully uploaded an audio file.
AR-2	To check whether the system can generate chroma features from the audio file.	The system should generate chroma features from the uploaded audio file.	The system has successfully generated chroma features.
AR-3	To check whether the system can predict chord labels from chroma features.	The system should predict chord labels accurately from the generated chroma.	The system has successfully predicted chord labels.
AR-4	To check whether the system can handle unsupported audio formats.	The system should provide an error message for unsupported audio formats.	The system provided an error message for unsupported formats.
AR-5	To check whether the system can display the recognized chords and timestamps.	The system should display recognized chords and their corresponding timestamps.	The system has successfully displayed chords and timestamps.

Table 7 Test Case for Forum Module

Test Case ID	Description	Expected Result	Actual
F-1	To check whether the user can view all forum posts.	The user should be able to view all forum posts.	The user has successfully viewed all forum posts.
F-2	To check whether the user can create a new forum post.	The user should be able to create a new forum post.	The user has successfully created a new forum post.
F-3	To check whether the user can view a specific forum post.	The user should be able to view a specific forum post.	The user has successfully viewed the specific forum post.
F-4	To check whether the user can add a message to a forum post.	The user should be able to add a message to a forum post.	The user has successfully added a message to the post.
F-5	To check whether the user can delete a forum post.	The user should be able to delete a forum post.	The user has successfully deleted a forum post.

Table 8 Test Case for Music Theory Module

Test Case ID	Description	Expected Result	Actual
MT-1	To check whether the user can view all music theory chapters.	The user should be able to view all music theory chapters.	The user has successfully viewed all chapters.
MT-2	To check whether the user can view details of a specific chapter.	The user should be able to view details of a specific chapter.	The user has successfully viewed the chapter details.
MT-3	To check whether the user can search for a music theory chapter.	The user should be able to search for a chapter by title or content.	The user has successfully searched for a chapter.
MT-4	To check whether the user can filter chapters by difficulty level.	The user should be able to filter chapters by difficulty level.	The user has successfully filtered chapters.
MT-5	To check whether the user can access multimedia content in chapters.	The user should be able to access multimedia content in chapters.	The user has successfully accessed multimedia content.

Table 9 Test Case for Music Theory Module

Test Case ID	Description	Expected Result	Actual
MS-1	To check whether the user can view all shared MIDI files.	The user should be able to view all shared MIDI files.	The user has successfully viewed all shared MIDI files.
MS-2	To check whether the user can upload a new MIDI file.	The user should be able to upload a new MIDI file.	The user has successfully uploaded a new MIDI file.
MS-3	To check whether the user can delete a MIDI file.	The user should be able to delete a MIDI file.	The user has successfully deleted a MIDI file.
MS-4	To check whether the user can filter MIDI files by genre.	The user should be able to filter MIDI files by genre.	The user has successfully filtered MIDI files by genre.
MS-5	To check whether the user can download a MIDI file.	The user should be able to download a MIDI file.	The user has successfully downloaded a MIDI file.

Table 10 Test Case for Music Theory Module

Test Case ID	Description	Expected Result	Actual
AT-1	To check whether the admin can retrieve a model by name.	The admin should be able to retrieve a model by its name without errors.	The admin has successfully retrieved a model by its name.
AT-2	To check whether the admin can create a form for a given model.	The admin should be able to create a form for a model including all fields.	The admin has successfully created a form for the model with all fields included.
AT-3	To check whether the admin can verify if a user has superuser status.	The admin should be able to check and confirm a user's superuser status.	The admin has successfully checked and confirmed the user's superuser status.

Table 11 Test Case for User Profile Module

Test Case ID	Description	Expected Result	Actual
UP-1	To check whether the user can view their profile information.	The user should be able to view their profile information.	The user has successfully viewed their profile.
UP-2	To check whether the user can update their email address.	The user should be able to update their email address.	The user has successfully updated their email address.
UP-3	To check whether the user can change their profile photo.	The user should be able to change their profile photo.	The user has successfully changed their profile photo.
UP-4	To check whether the user can delete a forum post	The user should be able to delete a forum post	The user has successfully deleted a forum post.
UP-5	To check whether the user can delete a MIDI file.	The user should be able to delete a MIDI file.	The user has successfully deleted a MIDI file.

4. Conclusion

In conclusion, the development and evaluation of the ChordWizard AI-powered Music Chords Recognition Web System marks a significant stride in addressing challenges associated with music chord recognition. Leveraging advanced techniques such as the NNLS Chroma algorithm and incorporating user-friendly modules, ChordWizard has demonstrated notable improvements in accuracy and speed. The system's versatility, allowing users to upload any audio file and fostering global collaboration through the Midi Sharing module, sets it apart from existing platforms like Chordify. The project not only provides a solution to current issues but also lays the groundwork for future enhancements, including the potential integration of artificial intelligence advancements. ChordWizard stands as a promising tool in the ever-evolving landscape of music technology, promising a transformative impact on music creation and collaboration.

Acknowledgment

I extend my sincere gratitude to my supervisor, PROF. Dr. ABD SAMAD BIN HASAN BASARI, for his unwavering support, invaluable guidance, and encouragement throughout the completion of this work. His expertise and mentorship have played a pivotal role in shaping my understanding and enhancing the quality of this endeavor. I also want to express my appreciation to all my professors and lecturers for the valuable learning experiences provided during this academic journey. Their dedication to education has been a source of inspiration and growth for which I am truly thankful.

References

[1] Y. Wu and Y. A. Wu, “A Unified Generative and Discriminative Approach to Automatic Chord Estimation for Music Audio Signals(Abstract_要旨),” 2021, doi: 10.14989/doctor.k23540.

[2] L. Sumarno, “Chord Recognition using Segment Averaging Feature Extraction with Simplified Harmonic Product Spectrum and Logarithmic Scaling,” *International Journal on Electrical Engineering and Informatics*, vol. 10, no. 4, 2018, doi: 10.15676/ijeei.2018.10.4.9.

[3] S. Sigtia, N. Boulanger-Lewandowski, and S. Dixon, “AUDIO CHORD RECOGNITION WITH A HYBRID RECURRENT NEURAL NETWORK”.

[4] J. Park, K. Choi, S. Jeon, D. Kim, and J. Park, “A BI-DIRECTIONAL TRANSFORMER FOR MUSICAL CHORD RECOGNITION”, Accessed: Dec. 27, 2023. [Online]. Available: https://github.com/jayg996/BTC-ISMIR19

[5] C. Weiss and M. Muller, “Tonal complexity features for style classification of classical music,” *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2015-August, pp. 688–692, Aug. 2015, doi: 10.1109/ICASSP.2015.7178057.

[6] M. Mauch, Q. Mary, and S. Dixon, "APPROXIMATE NOTE TRANSCRIPTION FOR THE IMPROVED IDENTIFICATION OF DIFFICULT CHORDS," 2010, Accessed: Dec. 27, 2023. [Online]. Available: https://www.researchgate.net/publication/220723830

[7] "Strategic Planning for Project Management Using a Project Management ... - Harold Kerzner - Google Books." Accessed: Dec. 27, 2023. [Online]. Available: https://books.google.com.my/books?hl=en&lr=&id=AkFpbYbJMEsC&oi=fnd&pg=PR5&dq=In+the+Planning+Phase,+the+project%27s+problem,+scope,+and+objectives+are+defined+through+activities+such+as+acquiring+and+analyzing+information,+producing+a+proposal,+and+creating+a+Gantt+chart+for+project+management.+&ots=IogoMdVGds&sig=ekKVJNebqipAWeRT_kxdb6Uibbs&redir_esc=y#v=onepage&q&f=false

[8] "Instant chords for any song," Chordify, https://chordify.net/ (accessed Jul. 30, 2024).