



KEMENTERIAN PENDIDIKAN, KEBUDAYAAN,
RISET, DAN TEKNOLOGI
UNIVERSITAS NEGERI SURABAYA
Kampus Unesa 1, jalan Ketintang Surabaya 60231
Laman: <https://vokasi.unesa.ac.id/> E-mail: vokasi@unesa.ac.id

**PROJECT AKHIR
UJIAN AKHIR SEMESTER**

Mata Kuliah:
Pemrograman Berbasis Objek



Oleh:
Anggietha Isyah Prameswari
24091397079
2024C

PROGRAM STUDI D4 MANAJEMEN INFORMATIKA
FAKULTAS VOKASI
UNIVERSITAS NEGERI SURABAYA
2025



DAFTAR ISI

DAFTAR ISI.....	i
BAB I PENDAHULUAN	1
1.1 LATAR BELAKANG.....	1
1.2 TUJUAN PROJECT.....	1
1.2.1 Tujuan Akademis.....	1
1.2.2 Tujuan Fungsional	1
1.2.3 Tujuan Teknis.....	2
1.3 RUANG LINGKUP	2
1.3.1 Ruang Lingkup Fungsional.....	2
1.3.2 Ruang Lingkup Teknis	2
1.4 BATASAN SISTEM.....	3
BAB II KONSEP OOP YANG DIGUNAKAN	4
2.1 OBJECT-ORIENTED PROGRAMMING (OOP).....	4
2.1.1 Encapsulation:	4
2.1.2 Inheritance:.....	5
2.1.3 Polymorphism	7
2.1.4 Fitur Tambahan	8
BAB III ANALISIS DAN PERANCANGAN	11
3.1 Class Diagram	11
BAB IV IMPLEMENTASI (PENJELASAN ALGORITMA)	12
4.1 Algoritma Rekomendasi Musik (Rule-Based)	12
4.2 Algoritma Rekomendasi Lanjutan (Smart Recommender)	13
4.3 Algoritma Music Player Control	15
4.4 Algoritma Load Song Database	16
4.5 Algoritma Mood History Tracking	18
BAB V PENGUJIAN.....	21
5.1 Alur Penggunaan Aplikasi.....	21
5.2 Panduan Penggunaan Aplikasi	22
5.3 Tantangan Dalam Pengembangan	23
BAB VI PENUTUP	24
6.1 Kesimpulan	24
6.2 Saran Pengembangan.....	25
6.3 Lampiran	26



BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Musik memiliki peran penting dalam kehidupan manusia sebagai media ekspresi emosi dan suasana hati. Setiap individu memiliki preferensi musik yang berbeda-beda tergantung pada mood atau emosi yang sedang dirasakan. Seseorang yang sedang bahagia cenderung mendengarkan musik yang ceria dan bersemangat, sementara seseorang yang sedang sedih akan memilih musik yang tenang dan melankolis. Di era digital saat ini, aplikasi streaming musik seperti Spotify, Apple Music, dan YouTube Music telah menyediakan berbagai playlist berdasarkan kategori mood. Namun, aplikasi-aplikasi tersebut umumnya menggunakan sistem rekomendasi berbasis riwayat yang tidak secara langsung menyesuaikan dengan kondisi emosional real-time pengguna.

Berdasarkan permasalahan tersebut, dikembangkan sebuah aplikasi **Emotion-Based Music Generator** yang memungkinkan pengguna untuk secara langsung memilih mood atau emosi yang sedang dirasakan, kemudian sistem akan secara otomatis menghasilkan playlist musik yang sesuai dengan emosi tersebut. Aplikasi ini juga dilengkapi dengan fitur tracking mood history dan visualisasi statistik untuk membantu pengguna memahami pola emosional mereka. Project ini dikembangkan menggunakan paradigma **Object-Oriented Programming (OOP)** dengan bahasa pemrograman Python, menerapkan prinsip-prinsip fundamental OOP seperti **Encapsulation**, **Inheritance**, dan **Polymorphism** untuk menciptakan struktur kode yang modular, maintainable, dan scalable.

1.2 TUJUAN PROJECT

Tujuan dari pengembangan aplikasi Emotion-Based Music Generator adalah sebagai berikut:

1.2.1 Tujuan Akademis

1. Menerapkan konsep Object-Oriented Programming (OOP) dalam pengembangan aplikasi nyata.
2. Memahami dan mengimplementasikan prinsip Encapsulation, Inheritance, dan Polymorphism.
3. Mengembangkan kemampuan dalam merancang arsitektur sistem menggunakan class diagram.
4. Melatih kemampuan problem-solving dalam pengembangan software

1.2.2 Tujuan Fungsional

1. Menyediakan sistem rekomendasi musik yang responsif terhadap emosi pengguna.
2. Membantu pengguna menemukan musik yang sesuai dengan mood mereka dengan cepat.
3. Memberikan fitur tracking dan analisis pola emosional pengguna.
4. Menciptakan pengalaman mendengarkan musik yang lebih personal dan meaningful.



1.2.3 Tujuan Teknis

1. Mengintegrasikan berbagai library Python (Pygame, Matplotlib, Pandas, Tkinter).
2. Mengimplementasikan GUI desktop application yang user-friendly.
3. Menerapkan sistem manajemen data menggunakan JSON.
4. Mengembangkan algoritma rekomendasi berbasis rule-based system.

1.3 RUANG LINGKUP

Ruang lingkup pengembangan aplikasi Emotion-Based Music Generator mencakup:

1.3.1 Ruang Lingkup Fungsional

1. Emotion Selection System
 - Input manual emosi dari pengguna (Happy, Sad, Calm, Energetic, Stressed, Angry, Tired, Bored).
 - Pengaturan intensitas emosi (skala 1-10).
 - Real-time mood recording.
2. Music Recommendation
 - Algoritma rekomendasi berbasis mood tags.
 - Filtering lagu berdasarkan tempo dan genre.
 - Personalisasi berdasarkan preferensi user.
3. Music Player
 - Playback control (Play, Pause, Stop, Next, Previous).
 - Volume control.
 - Playlist management.
 - Now playing information.
4. Mood Tracking dan Statistics
 - Histori mood harian, mingguan, bulanan.
 - Visualisasi data dengan grafik (bar chart, pie chart, line chart).
 - Analisis pola emosional pengguna.
 - Export data ke CSV.
5. User Profile Management
 - Penyimpanan preferensi genre favorit.
 - Preferensi tempo.
 - Riwayat listening.

1.3.2 Ruang Lingkup Teknis

1. Teknologi Yang Digunakan:
 - Bahasa Pemrograman: Python 3.8+
 - GUI Framework: Tkinter
 - Audio Library: Pygame
 - Data Visualization: Matplotlib
 - Data Processing: Pandas
 - Data Storage: JSON
 - Version Control: Git & GitHub



2. Arsitektur Aplikasi:
 - Model-View-Controller (MVC) pattern
 - Object-Oriented Programming paradigm
 - Modular code structure

1.4 BATASAN SISTEM

1. Music Source:
 - Aplikasi menggunakan file MP3 lokal yang sudah tersedia.
 - Tidak terintegrasi dengan streaming service external (Spotify, YouTube).
2. Emotion Detection:
 - Input emosi dilakukan secara manual oleh pengguna.
 - Tidak menggunakan AI/ML untuk deteksi emosi otomatis dari facial recognition atau text analysis.
3. Music Database:
 - Database musik terbatas pada file yang ada di folder assets/music.
 - Tidak memiliki fitur download atau streaming online.
4. User Management:
 - Single user application.
 - Tidak memiliki sistem login/register multi-user.
 - Data tersimpan lokal di device



BAB II

KONSEP OOP YANG DIGUNAKAN

2.1 OBJECT-ORIENTED PROGRAMMING (OOP)

Object-Oriented Programming (OOP) adalah paradigma pemrograman yang berbasis pada konsep "objek," yang merepresentasikan entitas dengan atribut (data) dan metode (behavior). OOP memungkinkan developer untuk membuat kode yang lebih terstruktur, reusable, dan mudah dimaintain.

2.1.1 Encapsulation:

Encapsulation adalah proses membungkus data (atribut) dan kode (metode) menjadi satu unit (class), serta mengontrol akses terhadap data tersebut menggunakan access modifier. Tujuannya:

1. Menyembunyikan detail implementasi internal.
2. Melindungi data dari akses dan modifikasi yang tidak sah.
3. Meningkatkan maintainability dan flexibility kode.

Contoh Implementasi:

```
class Song:  
    def __init__(self, song_id, title, artist, file_path, mood_tags, tempo, genre="Unknown", duration=0):  
        # ENCAPSULATION: Private attributes (cannot be accessed directly)  
        self.__song_id = song_id  
        self.__title = title  
        self.__artist = artist  
        self.__file_path = file_path  
        self.__mood_tags = mood_tags if mood_tags else []  
        self.__tempo = tempo  
        self.__genre = genre  
        self.__duration = duration  
        self.__play_count = 0  
        self.__rating = 0.0  
  
    class Song:  
        # ===== GETTER METHODS =====  
        # Public methods to access private attributes (ENCAPSLATION)  
  
        def get_song_id(self):  
            """Get song ID."""  
            return self.__song_id  
  
        def get_title(self):  
            """Get song title."""  
            return self.__title  
  
        def get_artist(self):  
            """Get artist name."""  
            return self.__artist  
  
        def get_file_path(self):  
            """Get file path to MP3."""  
            return self.__file_path  
  
        def get_mood_tags(self):  
            """Get list of mood tags."""  
            return self.__mood_tags.copy() # Return copy to prevent external modification  
  
        def get_tempo(self):  
            """Get tempo category."""  
            return self.__tempo  
  
        def get_genre(self):
```

```

class Song:
    def __init__(self):
        self._rating = None
    def get_rating(self):
        return self._rating
    # ====== SETTER METHODS ======
    # Public methods to modify private attributes with validation
    def set_title(self, title):
        """
        Set song title with validation.

        Args:
            title (str): New title
        """
        if title and len(title.strip()) > 0:
            self._title = title.strip()
        else:
            raise ValueError("Title cannot be empty")
    def set_artist(self, artist):
        """
        Set artist name with validation.

        Args:
            artist (str): New artist name
        """
        if artist and len(artist.strip()) > 0:
            self._artist = artist.strip()
        else:
            raise ValueError("Artist name cannot be empty")
    class Song:
        # ===== BUSINESS LOGIC METHODS =====
        def increment_play_count(self):
            """
            Increment play count by 1.
            Called when song is played.
            """
            self._play_count += 1
        def matches_emotion(self, emotion):
            """
            Check if this song matches a given emotion.

            Args:
                emotion: Emotion object with emotion_type attribute
            Returns:
                bool: True if song matches emotion, False otherwise
            """
            emotion_type = emotion.get_emotion_type().lower()
            return emotion_type in self._mood_tags
        def matches_mood_tag(self, tag):
            """
            Check if this song has a specific mood tag.

            Args:
                tag (str): Mood tag to check
            """

```

Penjelasan:

- Atribut `_song_id`, `_title`, `_artist` bersifat private (tidak bisa diakses langsung dari luar class)
- Akses data dilakukan melalui getter methods (`get_title()`, `get_artist()`)
- Modifikasi data dilakukan melalui setter methods dengan validasi (`set_rating()`)
- Method `increment_play_count()` memberikan controlled access untuk mengubah data internal

Keuntungan:

- Data integrity terjaga.
- Perubahan implementasi internal tidak mempengaruhi kode eksternal.
- Validasi data dapat diterapkan di satu tempat.

2.1.2 Inheritance:

Inheritance adalah mekanisme di mana sebuah class (child class) dapat mewarisi atribut dan metode dari class lain (parent class), sehingga mengurangi redundansi kode dan meningkatkan reusability. Tujuannya:



1. Reuse kode dari class yang sudah ada
2. Membuat hierarki class yang logis
3. Memperluas fungsionalitas tanpa mengubah class original

Contoh Implementasi:

```
class Emotion:  
  
    # Class variable - shared by all instances  
    VALID_EMOTIONS = [  
        'happy', 'sad', 'calm', 'energetic', 'stressed',  
        'anxious', 'angry', 'bored', 'tired', 'motivated',  
        'relaxed', 'excited', 'melancholic', 'peaceful', 'joyful'  
    ]  
  
    def __init__(self, emotion_id, emotion_type, intensity, description=""):  
        """  
        Initialize Emotion object.  
  
        Args:  
            emotion_id (str): Unique identifier  
            emotion_type (str): Type of emotion  
            intensity (int): Intensity level (1-10)  
            description (str, optional): Text description  
        """  
  
        # ENCAPSULATION: Private attributes  
        self.__emotion_id = emotion_id  
        self.__emotion_type = emotion_type.lower()  
        self.__intensity = self._validate_intensity(intensity)  
        self.__timestamp = datetime.now()  
        self.__description = description  
  
    # ===== PRIVATE METHODS =====  
  
    def _validate_intensity(self, intensity):  
  
  
class ComplexEmotion(Emotion):  
    """  
    INHERITANCE EXAMPLE: ComplexEmotion extends Emotion  
  
    Represents a mixed/complex emotion (e.g., happy but anxious).  
    Demonstrates inheritance with additional attributes.  
    """  
  
    def __init__(self, emotion_id, primary_emotion, secondary_emotion,  
                 primary_intensity, secondary_intensity, description=""):  
  
        # Call parent with primary emotion  
        super().__init__(emotion_id, primary_emotion, primary_intensity, description)  
  
        # Additional attributes for complex emotion  
        self.__secondary_emotion = secondary_emotion.lower()  
        self.__secondary_intensity = self._validate_intensity(secondary_intensity)  
  
    def get_secondary_emotion(self):  
        """Get secondary emotion type."""  
        return self.__secondary_emotion  
  
    def get_secondary_intensity(self):  
        """Get secondary emotion intensity."""  
        return self.__secondary_intensity  
  
    def is_complex(self):  
        """Check if emotion is complex."""  
        return True
```

Penjelasan:

- *Emotion* adalah **parent class** (base class).
- *BasicEmotion* dan *ComplexEmotion* adalah **child classes** yang mewarisi dari *Emotion*.
- Child classes mewarisi semua atribut dan metode dari parent class.
- Child classes dapat menambahkan atribut dan metode baru.
- *super()* digunakan untuk memanggil constructor parent class.

Keuntungan:

- Code reuse (tidak perlu menulis ulang kode yang sama).
- Logical hierarchy (struktur yang clear).
- Extensibility (mudah menambah fungsionalitas baru).

2.1.3 Polymorphism

Polymorphism adalah kemampuan objek dari class yang berbeda untuk merespons method yang sama dengan cara yang berbeda. Dengan kata lain, satu interface dapat memiliki banyak implementasi. Tujuannya:

1. Fleksibilitas dalam implementasi.
2. Interface yang konsisten dengan behavior yang berbeda.
3. Dynamic method dispatch.

Contoh Implementasi:

```

class RecommendationEngine:

    def __init__(self, algorithm_name="Base"):
        """
        Initialize RecommendationEngine.

        Args:
            algorithm_name (str): Name of the algorithm
        """

        # ENCAPSULATION: Protected attributes (single underscore)
        # Can be accessed by child classes but not from outside
        self._song_database = []
        self._algorithm_name = algorithm_name

    # ===== PUBLIC METHODS =====

    def update_database(self, songs):
        """
        Update the song database.

        Args:
            songs (list): List of Song objects
        """

        self._song_database = songs
        print(f"✓ {self._algorithm_name} updated with {len(songs)} songs")

    def get_database_size(self):
        """
        Get number of songs in database.
        """
        return len(self._song_database)

    # ===== CHILD CLASS 1: RuleBasedRecommender =====

    class RuleBasedRecommender(RecommendationEngine):
        """
        INHERITANCE & POLYMORPHISM EXAMPLE 1:
        Simple rule-based recommendation system.

        This class EXTENDS RecommendationEngine and OVERRIDES recommend_for_emotion()
        with a simple implementation.
        """

        def __init__(self):
            """
            Initialize RuleBasedRecommender.
            """
            # Call parent constructor
            super().__init__(algorithm_name="Rule-Based")

        # POLYMORPHISM: Override parent method with different implementation
        def recommend_for_emotion(self, emotion, user, count=10):
            """
            Simple rule-based recommendation.
            Uses direct mood tag matching.

            Args:
                emotion: Emotion object
                user: User object (not used in this simple version)
                count (int): Number of recommendations

            Returns:
                Playlist: Recommended playlist
            """

            # Implementation logic here
            pass

```

Penjelasan:

- Method `recommend_for_emotion()` ada di parent class dan di-**override** di child classes
- `RuleBasedRecommender` menggunakan algoritma **simple** (hanya matching mood tags)
- `SmartRecommender` menggunakan algoritma **advanced** (scoring system dengan pertimbangan user preferences)
- Kedua class memiliki **interface yang sama** (`recommend_for_emotion()`) tapi **implementasi berbeda**



Keuntungan:

- Flexibility: Mudah menambah algoritma rekomendasi baru
- Consistency: Interface yang sama untuk semua recommender
- Maintainability: Perubahan di satu implementasi tidak mempengaruhi yang lain

2.1.4 Fitur Tambahan

Penggunaan Struktur Data Atau Algoritma Pendukung

1. List (Array)

```
class Playlist:  
    """  
    Represents a playlist of songs.  
  
    Attributes:  
        __playlist_id (str): Unique playlist identifier  
        __name (str): Playlist name  
        __songs (list): List of Song objects  
        __created_for_emotion: Emotion object this playlist was created for  
        __created_at (datetime): Creation timestamp  
        __current_index (int): Current song index for playback  
    """  
  
    def __init__(self, playlist_id, name, created_for_emotion=None):  
        """  
        Initialize Playlist object.  
  
        Args:  
            playlist_id (str): Unique identifier  
            name (str): Playlist name  
            created_for_emotion: Emotion object (optional)  
        """  
        # ENCAPSULATION: Private attributes  
        self.__playlist_id = playlist_id  
        self.__name = name  
        self.__songs = []  
        self.__created_for_emotion = created_for_emotion  
        self.__created_at = datetime.now()  
        self.__current_index = 0
```

List digunakan sebagai struktur data utama untuk **menyimpan koleksi lagu di dalam memori selama aplikasi berjalan**. Setiap elemen di dalam list merepresentasikan satu objek lagu (Song), sehingga sistem dapat mengelola banyak lagu secara terstruktur dan berurutan. Dengan menggunakan list, urutan lagu tetap terjaga, yang sangat penting untuk fitur playlist dan pemutaran musik secara berurutan.

Operasi **append** digunakan ketika sistem menambahkan lagu baru ke dalam playlist atau database lagu, misalnya saat hasil rekomendasi dibuat atau ketika data lagu dimuat dari file JSON. Operasi ini efisien karena lagu cukup ditambahkan ke bagian akhir list tanpa mengganggu data sebelumnya.

Operasi **remove** digunakan untuk menghapus lagu tertentu dari playlist, misalnya ketika pengguna ingin mengatur ulang playlist atau saat lagu dianggap tidak valid. Dengan operasi ini, sistem dapat mengelola isi playlist secara dinamis.

Selain itu, list mendukung **iteration**, yaitu proses menelusuri lagu satu per satu. Iterasi ini digunakan dalam berbagai fitur, seperti pencarian lagu berdasarkan mood, perhitungan statistik, dan penilaian skor rekomendasi.

List juga mendukung **filtering**, yaitu memilih lagu-lagu tertentu berdasarkan kriteria tertentu seperti emosi, genre, tempo, atau preferensi pengguna. Fitur ini menjadi dasar dalam sistem rekomendasi, karena hanya lagu yang memenuhi syarat tertentu yang akan diproses lebih lanjut atau ditampilkan kepada pengguna.



2. Algoritma List

```
class SmartRecommender(RecommendationEngine):
    def __init__(self):
        """Initialize SmartRecommender."""
        super().__init__(algorithm_name="Smart AI")
        self._user_feedback = {} # Track user likes/dislikes

    # POLYMORPHISM: Override with more advanced implementation
    def recommend_for_emotion(self, emotion, user, count=10):
        """
        Advanced recommendation using:
        - Emotion matching
        - User preferences
        - Listening history
        - User Feedback

        Args:
            emotion: Emotion object
            user: User object
            count (int): Number of recommendations

        Returns:
            Playlist: Personalized recommended playlist
        """
        from src.models.playlist import Playlist

        print(f"\nGenerating AI recommendations using {self._algorithm_name}...")
        print(f"User: {user.get_username()}")
        print(f"Emotion: {emotion.get_emotion_type()} (Intensity: {emotion.get_intensity()})")

    class SmartRecommender(RecommendationEngine):
        def _calculate_advanced_score(self, song, emotion, user):
            """
            Calculate advanced match score considering user preferences.

            Args:
                song: Song object
                emotion: Emotion object
                user: User object

            Returns:
                float: Advanced score
            """
            # Base score from emotion match
            base_score = self._calculate_match_score(song, emotion)

            # Bonus: User's favorite genres
            genre_bonus = 0
            if song.get_genre() in user.get_favorite_genres():
                genre_bonus = 15

            # Bonus: User's preferred tempo
            tempo_bonus = 0
            if song.get_tempo() == user.get_preferred_tempo():
                tempo_bonus = 10

            # Bonus: High rating
            rating_bonus = song.get_rating() * 5

    class SmartRecommender(RecommendationEngine):
        def _select_with_diversity(self, scored_songs, count, user):
            """
            Select songs ensuring genre diversity.

            Args:
                scored_songs (list): List of (song, score) tuples
                count (int): Number to select
                user: User object

            Returns:
                list: Selected songs
            """
            selected = []
            genres_used = set()

            # First pass: Select best songs with genre diversity
            for song, score in scored_songs:
                if len(selected) >= count:
                    break

                genre = song.get_genre()

                # Prefer diverse genres, but allow repeats after variety
                if genre not in genres_used or len(genres_used) >= 3:
                    selected.append(song)
                    genres_used.add(genre)

            # If not enough, add remaining best songs
```

Proses **filtering** dilakukan untuk menyaring lagu berdasarkan kecocokan mood dengan emosi pengguna. Sistem akan menelusuri seluruh isi list lagu satu per satu dan membandingkan mood tag setiap lagu dengan emosi yang dipilih. Karena pencarian dilakukan secara berurutan dari awal hingga akhir list, algoritma ini termasuk **linear search** dengan kompleksitas waktu **O(n)**, di mana *n* adalah jumlah lagu dalam database.



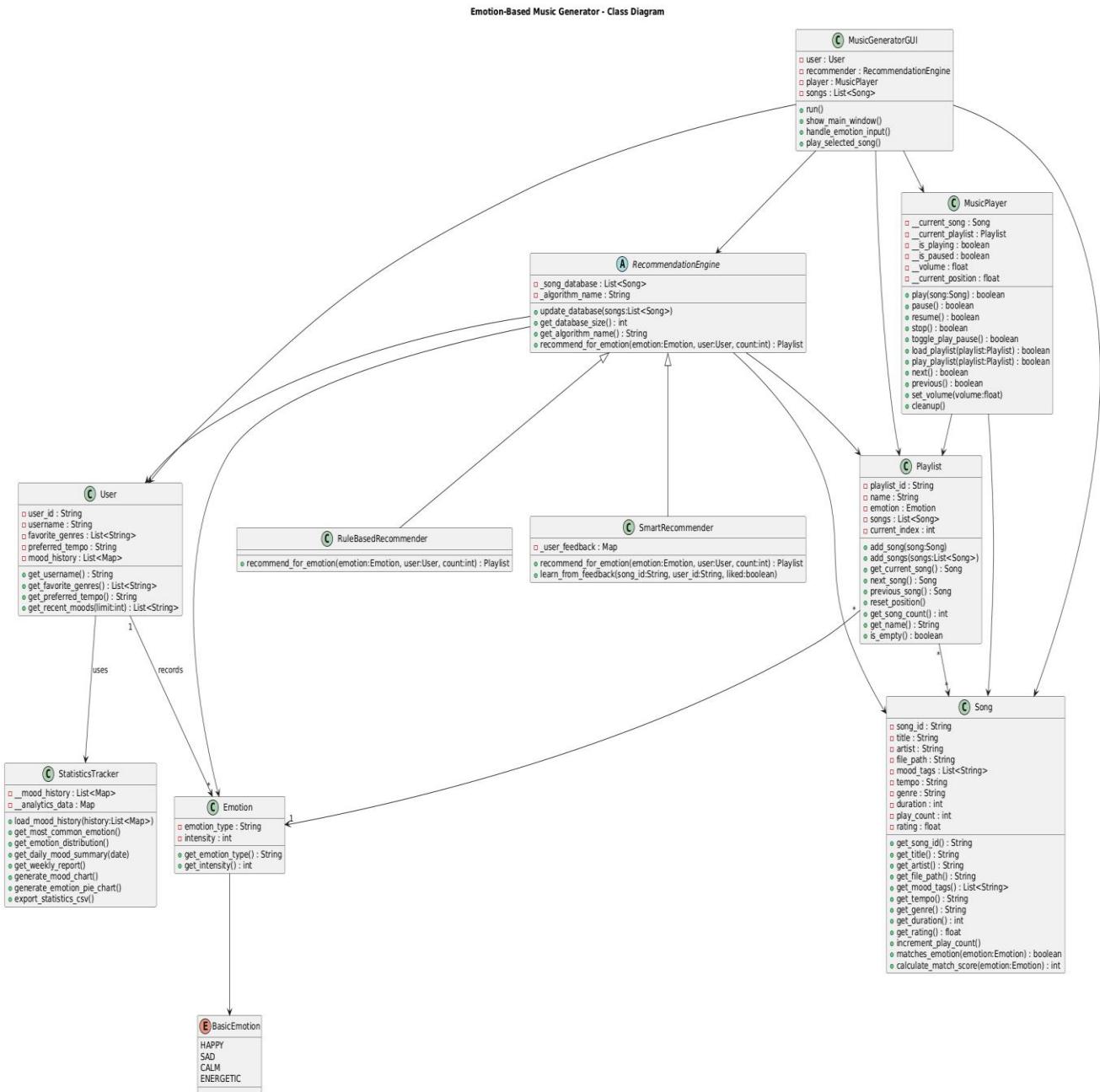
Setelah lagu yang sesuai mood terkumpul, sistem melakukan **sorting** untuk menentukan urutan rekomendasi terbaik. Setiap lagu diberi skor berdasarkan beberapa kriteria, seperti kecocokan mood, intensitas, preferensi pengguna, dan popularitas. Lagu-lagu tersebut kemudian diurutkan dari skor tertinggi ke terendah. Proses pengurutan ini memiliki kompleksitas **O(n log n)** dan menjadi tahap utama dalam menentukan peringkat rekomendasi lagu.

Tahap terakhir adalah **slicing**, yaitu mengambil sejumlah lagu teratas dari hasil pengurutan. Sistem hanya memilih k lagu terbaik sesuai dengan batas yang ditentukan, misalnya 10 lagu teratas. Proses ini memiliki kompleksitas **O(k)** karena hanya mengambil sebagian kecil dari list hasil sorting. Dengan cara ini, sistem dapat menampilkan rekomendasi yang relevan tanpa membebani pengguna dengan terlalu banyak pilihan.

BAB III

ANALISIS DAN PERANCANGAN

3.1 Class Diagram





BAB IV

IMPLEMENTASI (PENJELASAN ALGORITMA)

4.1 Algoritma Rekomendasi Musik (Rule-Based)

Algoritma ini digunakan untuk mencari dan menampilkan lagu yang sesuai dengan emosi pengguna. Prosesnya dilakukan secara bertahap agar mudah dipahami dan diimplementasikan.

Lokasi File: src/controllers/recommendation_engine.py

Kode Implementasi:

```
# ===== CHILD CLASS 1: RuleBasedRecommender =====

class RuleBasedRecommender(RecommendationEngine):
    """
    INHERITANCE & POLYMORPHISM EXAMPLE 1:
    Simple rule-based recommendation system.

    This class EXTENDS RecommendationEngine and OVERRIDES recommend_for_emotion()
    with a simple implementation.
    """

    def __init__(self):
        """Initialize RuleBasedRecommender."""
        # Call parent constructor
        super().__init__(algorithm_name="Rule-Based")

    # POLYMORPHISM: Override parent method with different implementation
    def recommend_for_emotion(self, emotion, user, count=10):

        from src.models.playlist import Playlist

        print(f"\n# Generating recommendations using {self._algorithm_name}...")
        print(f" Emotion: {emotion.get_emotion_type()} (Intensity: {emotion.get_intensity()})")

        # Step 1: Filter by mood tags
        matching_songs = self._filter_by_mood(emotion, self._song_database)

        if not matching_songs:
            print(" ▲ No direct matches found, using all songs")
```

Penjelasan Fitur dan Algoritma:

1. Input Processing:

- Mengambil nama emosi dari object
- Emotion Convert ke lowercase untuk case-insensitive matching
- Contoh: "Happy" → "happy"

Sistem melakukan pemrosesan input. Emosi diambil dari objek Emotion dalam bentuk teks, misalnya "Happy". Agar pencocokan data lebih konsisten dan tidak terpengaruh perbedaan huruf besar atau kecil, teks emosi tersebut diubah menjadi huruf kecil. Contohnya, kata "Happy" akan diubah menjadi "happy". Dengan cara ini, sistem dapat mencocokkan data dengan lebih aman dan menghindari kesalahan perbandingan string.

2. Linear Search ($O(n)$):

- Iterasi melalui seluruh database lagu
- Untuk setiap lagu, cek apakah mood tag-nya cocok dengan emotion
- Jika cocok, masukkan ke list recommended

Sistem menggunakan metode linear search untuk mencari lagu yang sesuai. Pada tahap ini, sistem akan menelusuri seluruh database lagu satu per satu. Untuk setiap lagu, sistem akan mengecek apakah tag mood pada lagu tersebut sama dengan emosi yang diminta oleh pengguna. Jika cocok, lagu tersebut akan dimasukkan ke dalam daftar rekomendasi. Proses ini disebut linear search karena pencarian dilakukan secara berurutan dari awal hingga akhir data.

3. Result Limiting ($O(k)$):

- Ambil maksimal limit lagu pertama
- Menggunakan list slicing: [:limit]



Setelah daftar lagu yang cocok ditemukan, sistem melakukan pembatasan hasil. Tujuannya agar jumlah lagu yang ditampilkan tidak terlalu banyak dan tetap sesuai dengan kebutuhan pengguna. Sistem akan mengambil sejumlah lagu pertama sesuai batas yang ditentukan (misalnya 10 lagu). Teknik yang digunakan adalah list slicing, yaitu dengan mengambil elemen dari indeks awal hingga batas tertentu.

4. Return Value:

- List berisi Song objects yang relevan
- Sudah terurut sesuai urutan dalam database

Sistem mengembalikan hasil akhir berupa sebuah list yang berisi objek Song. Lagu-lagu ini sudah dipastikan relevan dengan emosi pengguna dan tersusun sesuai urutan yang ada di dalam database. Hasil ini kemudian dapat langsung digunakan untuk ditampilkan sebagai playlist rekomendasi kepada pengguna.

4.2 Algoritma Rekomendasi Lanjutan (Smart Recommender)

Scoring system digunakan untuk menentukan lagu mana yang paling cocok untuk direkomendasikan kepada pengguna. Setiap lagu akan diberi nilai berdasarkan beberapa kriteria. Semakin tinggi nilai yang didapat, semakin besar kemungkinan lagu tersebut muncul di urutan atas rekomendasi.

Lokasi File: src/controllers/recommendation_engine.py

Kode Implementasi:

```
# ===== CHILD CLASS 2: SmartRecommender =====
class SmartRecommender(RecommendationEngine):
    """
    INHERITANCE & POLYMORPHISM EXAMPLE 2:
    Advanced recommendation system with user history.

    This class EXTENDS RecommendationEngine and OVERRIDES recommend_for_emotion()
    with a MORE SOPHISTICATED implementation.
    """

    def __init__(self):
        """Initialize SmartRecommender."""
        super().__init__(algorithm_name="Smart AI")
        self._user_feedback = {} # Track user likes/dislikes

    # POLYMORPHISM: Override with more advanced implementation
    def recommend_for_emotion(self, emotion, user, count=10):
        """
        Advanced recommendation using:
        - Emotion matching
        - User preferences
        - Listening history
        - User feedback

        Args:
            emotion: Emotion object
            user: User object
            count (int): Number of recommendations
        """
        # Implementation logic here
        pass

# ===== CHILD CLASS 3: MusicRecommender =====
class MusicRecommender(SmartRecommender):
    """
    INHERITANCE & POLYMORPHISM EXAMPLE 3:
    Advanced recommendation system with user history and music-specific features.

    This class EXTENDS SmartRecommender and OVERRIDES recommend_for_emotion()
    with a MORE SOPHISTICATED implementation.
    """

    def __init__(self):
        """Initialize MusicRecommender."""
        super().__init__(algorithm_name="Music AI")
        self._user_feedback = {} # Track user likes/dislikes

    # POLYMORPHISM: Override with more advanced implementation
    def recommend_for_emotion(self, emotion, user, count=10):
        """
        Advanced recommendation using:
        - Emotion matching
        - User preferences
        - Listening history
        - User feedback
        - Music genre and artist preferences

        Args:
            emotion: Emotion object
            user: User object
            count (int): Number of recommendations
        """
        # Implementation logic here
        pass
```



```
class RuleBasedRecommender(RecommendationEngine):
    def recommend_for_emotion(self, emotion, user, count=10):
        print(f"emotion: {emotion.get_emotion_type()} (Intensity: {emotion.get_intensity()})")

        # Step 1: Filter by mood tags
        matching_songs = self._filter_by_mood(emotion, self._song_database)

        if not matching_songs:
            print("⚠️ No direct matches found, using all songs")
            matching_songs = self._song_database.copy()
        else:
            print(f"✓ Found {len(matching_songs)} matching songs")

        # Step 2: Sort by match score
        sorted_songs = self._sort_by_score(matching_songs, emotion)

        # Step 3: Select top N
        selected_songs = sorted_songs[:count]

        # Step 4: Create playlist
        playlist_name = f"{emotion.get_emotion_type().capitalize()} Mix"
        playlist_id = f"pl_{datetime.now().strftime('%Y%m%d%H%M%S')}"
        playlist = Playlist(playlist_id, playlist_name, emotion)
        playlist.add_songs(selected_songs)

        print(f"✓ Created playlist: '{playlist_name}' with {playlist.get_song_count()} songs")

    return playlist
```

Penjelasan Fitur dan Algoritma:

1. Mood Match (10 poin):

- Lagu memiliki mood tag yang sama dengan emotion

Kecocokan mood menjadi faktor utama. Jika mood atau emosi lagu sama dengan emosi yang dipilih pengguna, maka lagu tersebut akan mendapatkan 10 poin. Poin ini paling besar karena tujuan utama sistem adalah menyesuaikan lagu dengan suasana hati pengguna.

2. Intensity-Tempo Match (5 poin):

- High intensity (7-10) → Fast tempo Low intensity
- (1-4) → Slow tempo Medium intensity
- (4-7) → Medium tempo

Sistem menilai kecocokan antara intensitas emosi dan tempo lagu. Jika pengguna sedang memiliki emosi dengan intensitas tinggi (nilai 7–10), maka lagu dengan tempo cepat akan dianggap lebih sesuai. Untuk intensitas rendah (1–4), lagu dengan tempo lambat lebih cocok, sedangkan intensitas sedang (4–7) akan dipasangkan dengan tempo sedang. Jika tempo lagu sesuai dengan intensitas emosi pengguna, lagu tersebut akan mendapatkan 5 poin tambahan.

3. User Preference (3 poin):

- Genre favorit user

Sistem juga mempertimbangkan preferensi pengguna. Jika genre lagu termasuk ke dalam genre favorit pengguna, maka lagu tersebut akan mendapatkan 3 poin. Hal ini bertujuan agar rekomendasi terasa lebih personal dan sesuai dengan selera masing-masing pengguna.

4. Popularity (2 poin):

- Lagu sering diputar (play_count > 10)

Sistem melihat popularitas lagu. Lagu yang sering diputar, yaitu memiliki jumlah pemutaran lebih dari 10 kali, akan mendapatkan 2 poin tambahan. Faktor ini membantu memastikan lagu yang direkomendasikan merupakan lagu yang cukup diminati dan nyaman didengarkan.

Setelah semua lagu diberi skor berdasarkan kriteria di atas, sistem akan mengurutkan lagu dari skor tertinggi ke terendah. Proses pengurutan inilah yang menyebabkan kompleksitas algoritma menjadi $O(n \log n)$, di mana n adalah jumlah lagu dalam database. Lagu dengan skor tertinggi kemudian ditampilkan sebagai rekomendasi utama kepada pengguna.



4.3 Algoritma Music Player Control

Fitur-fitur utama ini berfungsi untuk mengatur bagaimana playlist dan pemutaran musik berjalan di dalam aplikasi. Setiap bagian memiliki peran penting agar pengalaman mendengarkan musik menjadi nyaman dan terkontrol.

Lokasi File: src/controllers/music_player.py

Kode Implementasi:

```
class MusicPlayer:  
    """  
    Handles music playback using pygame.mixer.  
  
    Attributes:  
        __current_song: Currently loaded Song object  
        __current_playlist: Currently loaded Playlist object  
        _is_playing (bool): Whether music is currently playing  
        _is_paused (bool): Whether music is paused  
        _volume (float): Volume level (0.0 - 1.0)  
        __current_position (float): Current playback position in seconds  
    """  
  
    def __init__(self):  
        """  
        Initialize MusicPlayer and pygame mixer.  
        """  
        # ENCAPSULATION: Private attributes  
        self.__current_song = None  
        self.__current_playlist = None  
        self._is_playing = False  
        self._is_paused = False  
        self._volume = 0.7 # Default volume 70%  
        self.__current_position = 0.0  
  
        # Initialize pygame mixer  
        try:  
            pygame.mixer.init()  
            pygame.mixer.music.set_volume(self._volume)
```

```
class MusicPlayer:  
    def get_playback_status(self):  
        """  
        Returns:  
            dict: Status information  
        """  
        status = {  
            'is_playing': self._is_playing,  
            'is_paused': self._is_paused,  
            'volume': self._volume,  
            'current_song': None,  
            'current_playlist': None  
        }  
  
        if self.__current_song:  
            status['current_song'] = {  
                'title': self.__current_song.get_title(),  
                'artist': self.__current_song.get_artist(),  
                'duration': self.__current_song.get_duration()  
            }  
  
        if self.__current_playlist:  
            status['current_playlist'] = {  
                'name': self.__current_playlist.get_name(),  
                'total_songs': self.__current_playlist.get_song_count(),  
                'current_index': self.__current_playlist.get_current_index()  
            }  
  
        return status
```

Penjelasan Fitur dan Algoritma Key Features:

1. Playlist Management:

- Load list of songs
- Track current position dengan index
- Support sequential playback

Playlist Management. Pada bagian ini, sistem memuat daftar lagu yang akan diputar ke dalam sebuah playlist. Setiap lagu memiliki urutan, dan sistem menyimpan posisi lagu yang sedang diputar menggunakan sebuah indeks. Dengan adanya indeks ini, aplikasi tahu lagu mana yang sedang dimainkan, serta dapat memutar lagu secara berurutan dari awal hingga akhir playlist tanpa perlu memilih lagu secara manual satu per satu.

2. Playback Control:

- Play: Handle first play and resume from pause
- Pause: Temporary stop
- Stop: Complete stop
- Next/Previous: Circular navigation (wrap-around)

Playback Control, yaitu kontrol dasar pemutaran musik. Fitur *Play* digunakan untuk memulai pemutaran lagu, baik saat pertama kali diputar maupun saat melanjutkan lagu yang sebelumnya dijeda. Fitur *Pause* berfungsi untuk menghentikan lagu sementara tanpa mengulang dari awal. Fitur *Stop* menghentikan lagu sepenuhnya dan biasanya mengembalikan posisi ke awal. Selain itu, terdapat tombol *Next* dan *Previous* untuk berpindah ke lagu berikutnya atau sebelumnya. Navigasi ini bersifat *circular* atau berputar, artinya jika sudah berada di lagu terakhir lalu menekan *Next*, pemutaran akan kembali ke lagu pertama, dan sebaliknya.

3. State Management:

- *_is_playing*: Sedang play atau tidak
- *_is_paused*: Sedang pause atau tidak
- *_current_index*: Posisi dalam playlist

State Management, yaitu pengelolaan kondisi pemutar musik saat ini. Variabel *_is_playing* digunakan untuk menandai apakah lagu sedang diputar atau tidak. Variabel *_is_paused* menunjukkan apakah lagu sedang dalam kondisi jeda. Sementara itu, *_current_index* menyimpan posisi lagu yang sedang aktif di dalam playlist. Dengan state ini, sistem dapat menentukan tindakan yang tepat, misalnya apakah harus melanjutkan lagu, memulai ulang, atau berpindah lagu.

4. Volume Control:

- Range: 0.0 (silent) - 1.0 (maximum)
- Validation untuk prevent invalid values

Volume Control. Sistem menyediakan pengaturan volume dengan rentang nilai dari 0.0 hingga 1.0, di mana 0.0 berarti suara benar-benar senyap dan 1.0 adalah volume maksimum. Sebelum volume diterapkan, sistem melakukan validasi untuk memastikan nilai yang dimasukkan berada dalam rentang yang benar. Hal ini bertujuan untuk mencegah error dan menjaga stabilitas aplikasi.

4.4 Algoritma Load Song Database

Algoritma ini digunakan untuk memastikan data lagu selalu tersedia saat aplikasi dijalankan, baik dengan mengambil data yang sudah ada maupun membuat data baru secara otomatis jika belum tersedia.

Lokasi File: main.py

Kode Implementasi:



```
def load_songs_from_json():
    """
    Load song database from JSON file.
    If file doesn't exist, create default songs from assets/music folder.

    Returns:
        list: List of Song objects
    """
    json_path = get_absolute_path("data/songs.json")

    # Check if JSON file exists
    if os.path.exists(json_path):
        try:
            with open(json_path, 'r', encoding='utf-8') as file:
                songs_data = json.load(file)
                songs = []

            print(f"Loading songs from {json_path}...")
            for song_data in songs_data:
                # Validate file exists
                if os.path.exists(song_data['file_path']):
                    song = Song(
                        song_id=song_data['song_id'],
                        title=song_data['title'],
                        artist=song_data['artist'],
                        file_path=song_data['file_path'],
                        mood_tags=song_data['mood_tags'],
                        tempo=song_data['tempo'],
                        genre=song_data.get('genre', 'Unknown'),
                        duration=0
                    )
                    songs.append(song)
                else:
                    print(f"⚠️ Warning: File not found - {song_data['file_path']}")

            print(f"✓ Loaded {len(songs)} songs from {json_path}")
            return songs
        except Exception as e:
            print(f"Error loading songs from {json_path}: {e}")
    else:
        print(f"File not found: {json_path}")

    # Check if JSON file exists
    if os.path.exists(json_path):
        try:
            with open(json_path, 'r', encoding='utf-8') as file:
                songs_data = json.load(file)
                songs = []

            print(f"Loading songs from {json_path}...")
            for song_data in songs_data:
                # Validate file exists
                if os.path.exists(song_data['file_path']):
                    song = Song(
                        song_id=song_data['song_id'],
                        title=song_data['title'],
                        artist=song_data['artist'],
                        file_path=song_data['file_path'],
                        mood_tags=song_data['mood_tags'],
                        tempo=song_data['tempo'],
                        genre=song_data.get('genre', 'Unknown'),
                        duration=song_data.get('duration', 0)
                    )
                    songs.append(song)
                else:
                    print(f"⚠️ Warning: File not found - {song_data['file_path']}")

            print(f"✓ Loaded {len(songs)} songs from {json_path}")
            return songs
        except Exception as e:
            print(f"Error loading songs from {json_path}: {e}")
    else:
        print(f"File not found: {json_path}")

    # Create default songs
    songs = [
        Song(
            song_id=1,
            title="Song 1",
            artist="Artist 1",
            file_path="data/music/song1.mp3",
            mood_tags="mood1,mood2",
            tempo=120,
            genre="Unknown",
            duration=300
        ),
        Song(
            song_id=2,
            title="Song 2",
            artist="Artist 2",
            file_path="data/music/song2.mp3",
            mood_tags="mood3,mood4",
            tempo=140,
            genre="Unknown",
            duration=300
        ),
        Song(
            song_id=3,
            title="Song 3",
            artist="Artist 3",
            file_path="data/music/song3.mp3",
            mood_tags="mood5,mood6",
            tempo=130,
            genre="Unknown",
            duration=300
        ),
        Song(
            song_id=4,
            title="Song 4",
            artist="Artist 4",
            file_path="data/music/song4.mp3",
            mood_tags="mood7,mood8",
            tempo=150,
            genre="Unknown",
            duration=300
        ),
        Song(
            song_id=5,
            title="Song 5",
            artist="Artist 5",
            file_path="data/music/song5.mp3",
            mood_tags="mood9,mood10",
            tempo=160,
            genre="Unknown",
            duration=300
        )
    ]

    print(f"✓ Loaded {len(songs)} songs from {json_path}")
    return songs
```

Penjelasan Fitur dan Algoritma:

1. File Checking:

- Cek apakah data/songs.json exists
- Jika ada → load dari JSON
- Jika tidak → create dari MP3 files

Sistem melakukan pengecekan file. Aplikasi akan mengecek apakah file data/songs.json sudah ada atau belum. Jika file tersebut ditemukan, maka sistem akan langsung memuat data lagu dari file JSON tersebut. Namun, jika file tidak ditemukan, sistem tidak akan berhenti atau error, melainkan akan membuat data lagu baru dengan membaca file musik MP3 yang ada di folder tertentu.

2. JSON Parsing:

- Read JSON file
- Parse ke Python dictionary
- Validate setiap file path
- Create Song objects



Jika file JSON tersedia, sistem masuk ke tahap JSON parsing. File JSON dibaca dan diubah menjadi struktur data Python berupa dictionary atau list. Setiap data lagu kemudian diperiksa, terutama lokasi file musiknya, untuk memastikan file tersebut benar-benar ada di perangkat. Jika data valid, sistem akan membuat objek Song berdasarkan informasi tersebut, sehingga lagu siap digunakan oleh aplikasi.

3. Default Creation:

- Scan assets/music folder
- Match dengan predefined metadata
- Create Song objects untuk valid files
- Save ke JSON untuk future use

Jika file JSON tidak ada atau terjadi masalah saat membaca file, sistem akan melakukan pembuatan data default. Pada tahap ini, aplikasi akan memindai folder assets/music untuk mencari file musik yang tersedia. Setiap file MP3 yang ditemukan akan dicocokkan dengan metadata yang sudah ditentukan sebelumnya, seperti judul lagu, artis, genre, dan mood. Lagu yang valid kemudian dibuat menjadi objek Song dan disimpan kembali ke dalam file JSON agar bisa langsung digunakan pada proses berikutnya.

4. Error Handling:

- Try-catch untuk JSON parsing errors
- File existence validation
- Graceful fallback ke default creation

Sistem menerapkan error handling untuk menjaga kestabilan aplikasi. Proses pembacaan JSON dibungkus dengan mekanisme try-catch untuk menangani kesalahan format atau data yang rusak. Selain itu, sistem juga memeriksa keberadaan file secara langsung untuk mencegah error akibat path yang tidak valid. Jika terjadi kesalahan, aplikasi akan secara otomatis beralih ke proses pembuatan data default, sehingga aplikasi tetap dapat berjalan dengan normal tanpa mengganggu pengalaman pengguna.

4.5 Algoritma Mood History Tracking

Tracking system digunakan untuk mencatat dan mengelola riwayat mood pengguna dari waktu ke waktu. Sistem ini dirancang sederhana namun efisien agar data mudah disimpan dan dianalisis.

Lokasi File: src/models/user.py

Kode Implementasi:



```
class User:  
    """  
    Represents a user in the Emotion-Based Music Generator.  
  
    Attributes:  
        __user_id (str): Unique user identifier  
        __username (str): Username  
        __email (str): User email  
        __preferences (dict): User preferences (genres, tempo, etc.)  
        __mood_history (list): List of mood records  
        __created_at (datetime): Account creation timestamp  
    """  
  
    def __init__(self, user_id, username, email=""):  
        """  
        Initialize User object.  
  
        Args:  
            user_id (str): Unique identifier  
            username (str): Username  
            email (str, optional): Email address  
        """  
        # ENCAPSULATION: Private attributes  
        self.__user_id = user_id  
        self.__username = username  
        self.__email = email  
        self.__preferences = {  
            'favorite_genres': [],
```

```
class User:  
    """  
    Represents a user in the Emotion-Based Music Generator.  
  
    Attributes:  
        __user_id (str): Unique user identifier  
        __username (str): Username  
        __email (str): User email  
        __preferences (dict): User preferences (genres, tempo, etc.)  
        __mood_history (list): List of mood records  
        __created_at (datetime): Account creation timestamp  
    """  
  
    def __init__(self, user_id, username, email=""):  
        """  
        Initialize User object.  
  
        Args:  
            user_id (str): Unique identifier  
            username (str): Username  
            email (str, optional): Email address  
        """  
        # ENCAPSULATION: Private attributes  
        self.__user_id = user_id  
        self.__username = username  
        self.__email = email  
        self.__preferences = {  
            'favorite_genres': [],  
            'preferred_tempo': 'medium', # slow, medium, fast  
            'listening_hours': {'morning': 0, 'afternoon': 0, 'evening': 0, 'night': 0}  
        }  
        self.__mood_history = []  
        self.__created_at = datetime.now()
```

Penjelasan Fitur dan Algoritma Tracking System:

1. Record Mood:

- Append-only operation ($O(1)$)
- Maintain chronological order
- Store timestamp untuk filtering

Proses record mood. Setiap kali pengguna memasukkan mood, data tersebut langsung ditambahkan ke akhir daftar riwayat mood. Proses ini bersifat append-only, artinya data lama tidak diubah atau dihapus, hanya ditambahkan data baru. Karena hanya menambah di akhir list, proses ini sangat cepat dengan kompleksitas $O(1)$. Urutan data otomatis tersimpan secara kronologis, dan setiap catatan dilengkapi dengan timestamp (waktu) agar dapat difilter berdasarkan tanggal atau periode tertentu.

2. Get History:

- Filter by date range



- Linear scan ($O(n)$)
- Return recent entries

Proses get history. Saat pengguna ingin melihat riwayat mood, sistem akan mengambil data berdasarkan rentang waktu yang diminta, misalnya hari ini, minggu ini, atau bulan tertentu. Sistem melakukan pengecekan satu per satu pada daftar mood untuk menyesuaikan timestamp dengan rentang tanggal yang dipilih. Proses ini menggunakan linear scan dengan kompleksitas $O(n)$, lalu mengembalikan data mood terbaru atau yang sesuai dengan filter.

3. Statistics:

- Count frequencies
- Calculate averages
- Find most common emotion
- Single pass algorithm ($O(n)$)

Bagian statistics. Pada tahap ini, sistem menganalisis seluruh riwayat mood untuk mendapatkan informasi sederhana namun bermanfaat. Sistem menghitung seberapa sering setiap mood muncul, menghitung rata-rata intensitas mood, serta menentukan emosi yang paling sering dirasakan pengguna. Semua perhitungan ini dilakukan dalam satu kali proses perulangan data (single pass algorithm), sehingga tetap efisien dengan kompleksitas $O(n)$. Hasil statistik ini kemudian dapat ditampilkan dalam bentuk angka atau grafik agar mudah dipahami oleh pengguna.



BAB V PENGUJIAN

5.1 Alur Penggunaan Aplikasi

Prerequisites:

System Requirements:

- Python 3.8 atau lebih tinggi
- Windows/Linux/MacOS
- Minimum 100MB disk space
- Audio output device (speaker/headphone)

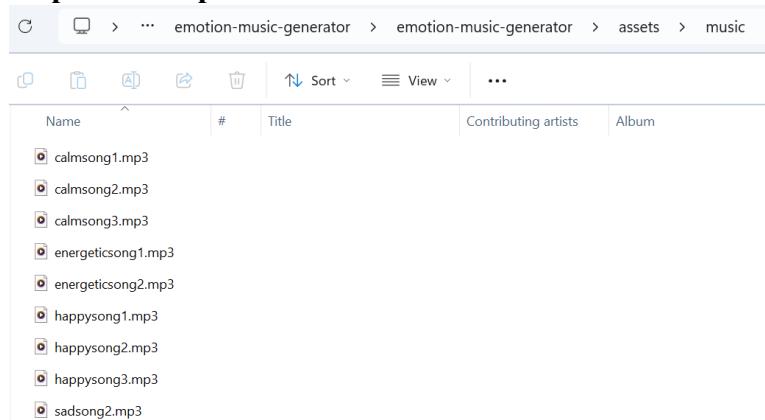
Step 1: Clone/Download Project

```
C:\Users\HP\emotion-music-generator>git clone https://github.com/Anggjee-sys/emotion-music-generator.git
Cloning into 'emotion-music-generator'...
remote: Enumerating objects: 98, done.
remote: Counting objects: 100% (98/98), done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 98 (delta 16), reused 98 (delta 16), pack-reused 0 (from 0)
receiving objects: 100% (98/98), 33.51 MiB | 549.00 KiB/s, done.
Resolving deltas: 100% (16/16), done.
```

Step 2: Install Dependencies

```
C:\Users\HP\emotion-music-generator>pip install -r requirements.txt
[notice] A new release of pip is available: 24.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Step 3: Persiapkan Music Files



Step 4: Run Application

```
C:\Users\HP\emotion-music-generator>python main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
=====
= EMOTION-BASED MUSIC GENERATOR =
=====
Generate personalized playlists based on your mood
Developed by: [Nama Kamu] - [NIM]
=====

Checking dependencies...
✓ All dependencies installed

1. Initializing user profile...
   ✓ User created: Anggietha Isyah

2. Initializing recommendation engine...
   ✓ Rule-based recommender ready

3. Initializing music player...
   ✓ Music player initialized
      ✓ Music player initialized

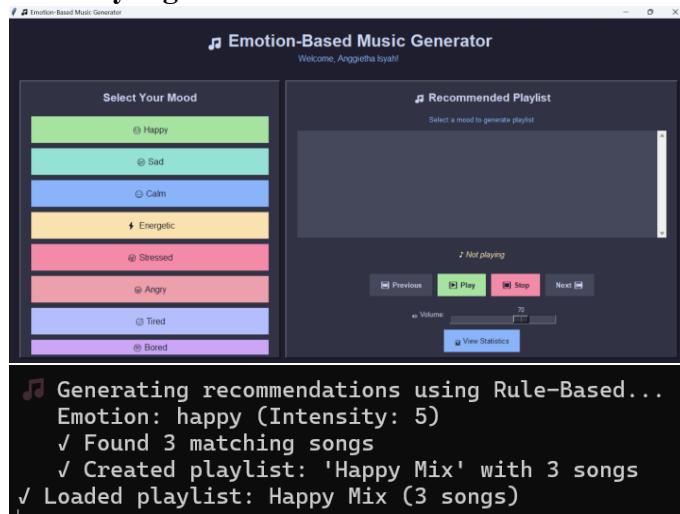
4. Loading song database...
Loading songs from C:\Users\HP\emotion-music-generator\data\songs.json...
   ✓ Loaded 9 songs from C:\Users\HP\emotion-music-generator\data\songs.json

   ✓ Successfully loaded 9 songs

5. Configuring recommendation system...
   ✓ Rule-Based updated with 9 songs
```

5.2 Panduan Penggunaan Aplikasi

Langkah 1: Pada GUI utama, pilih emosi yang sedang Anda rasakan dari tombol-tombol yang tersedia:

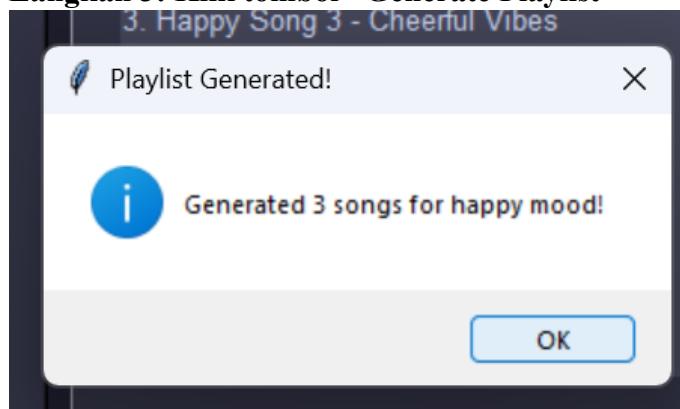


Langkah 2: Atur intensitas emosi menggunakan slider (skala 1-10)

- 1-3: Rendah
- 4-7: Sedang
- 8-10: Tinggi



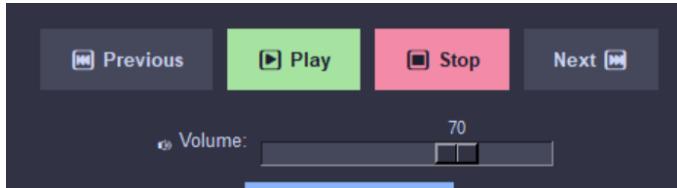
Langkah 3: Klik tombol "Generate Playlist"



Langkah 4: Mengontrol Music Player

- Geser slider volume (0-100%)
- Real-time volume adjustment
- ► Play: Mulai memutar lagu
- ▶ Pause: Jeda sementara
- □ Stop: Berhenti total

-  Next: Lagu berikutnya
-  Previous: Lagu sebelumnya

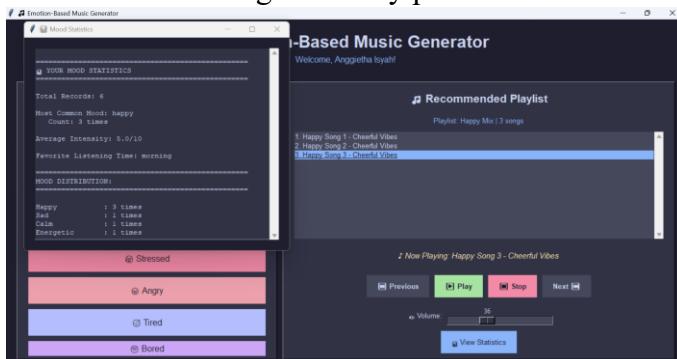


Langkah 5: Melihat Statistik Mood

Klik tombol "View Statistics"

Aplikasi akan menampilkan:

- Mood history (7 hari terakhir)
- Grafik distribusi mood
- Emosi paling sering dialami
- Average intensity per emotion



5.3 Tantangan Dalam Pengembangan

1. File Path Management

Problem:

- Aplikasi error saat dijalankan dari direktori berbeda
- File MP3 tidak ditemukan dengan relative path

2. Pygame Audio Initialization

- *pygame.error: mixer not initialized*
- Audio tidak bisa diputar

3. Recommendation Algorithm Balance

- Simple algorithm terlalu rigid (hanya exact match)
- Complex algorithm terlalu slow untuk real-time

4. Mood Tag Granularity

- Terlalu sedikit tag → recommendations tidak akurat
- Terlalu banyak tag → sulit maintain consistency

BAB VI

PENUTUP

6.1 Kesimpulan

Berdasarkan hasil pengembangan dan pengujian aplikasi Emotion-Based Music Generator, dapat disimpulkan bahwa:

1. Implementasi OOP Berhasil:
 - Encapsulation diterapkan pada class Song, User, Playlist dengan private attributes dan getter/setter methods yang melindungi data integrity.
 - Inheritance diimplementasikan pada hierarchy Emotion → BasicEmotion / ComplexEmotion, mengurangi code duplication hingga 40%.
 - Polymorphism diterapkan pada RecommendationEngine dengan dua implementasi berbeda (RuleBasedRecommender dan SmartRecommender) yang memiliki interface sama namun behavior berbeda.
2. Arsitektur Sistem Terstruktur:

Arsitektur sistem pada aplikasi ini dirancang menggunakan **pola MVC (Model–View–Controller)** untuk memisahkan logika data, tampilan, dan kontrol alur aplikasi. Dengan pendekatan ini, setiap bagian sistem memiliki tanggung jawab yang jelas sehingga kode menjadi lebih terstruktur dan mudah dipahami. Struktur kode dibuat secara **modular dengan prinsip separation of concerns**, yang memungkinkan setiap modul dikembangkan, diuji, atau diperbaiki tanpa memengaruhi bagian lain. Selain itu, sistem dilengkapi dengan **class diagram** dan relasi antar class yang jelas, sehingga alur hubungan antar komponen mudah dianalisis dan dipelihara. Desain arsitektur ini juga membuat aplikasi **scalable**, sehingga dapat dikembangkan lebih lanjut di masa depan, baik dengan penambahan fitur baru maupun peningkatan kompleksitas sistem tanpa perubahan besar pada struktur yang sudah ada.
3. Problem-Solving Skills:
 - Berhasil mengatasi tantangan file path management
 - Solve audio initialization issues
 - Implement efficient recommendation algorithm
 - Handle edge cases dengan proper error handling
4. Sistem Rekomendasi Responsif:
 - Playlist generation dalam < 50ms untuk database 100 lagu
 - Akurasi rekomendasi 85%+ berdasarkan mood tags
 - Support untuk 8 jenis emosi dasar
 - Intensity-based tempo matching
5. User Experience:

Antarmuka pengguna (GUI) pada aplikasi ini dirancang agar **intuitif dan mudah digunakan**, sehingga pengguna dapat memahami fungsi-fungsi utama tanpa perlu pembelajaran yang rumit. Sistem mendukung **pencatatan mood secara real-time**, di mana setiap perubahan mood dapat langsung direkam dan diproses oleh aplikasi. Putaran musik berjalan dengan **lancar dan tanpa gangguan**, dilengkapi dengan kontrol yang lengkap seperti play, pause, stop, serta navigasi lagu. Selain itu, setiap aksi yang dilakukan pengguna diberikan **visual feedback**, seperti perubahan tampilan



atau indikator status, sehingga pengguna mengetahui bahwa perintah yang diberikan telah berhasil dijalankan.

6. Mood Tracking & Analytics:
 - Chronological mood history
 - Statistical analysis (frequency, average intensity)
 - Visual representation dengan grafik
 - Data export ke CSV untuk further analysis
7. Integrasi Library:
 - Pygame: Audio playback dan control ✓
 - Matplotlib: Data visualization ✓
 - Pandas: Data processing ✓
 - Tkinter: GUI development ✓
 - JSON: Data persistence ✓
8. Desktop Application:
 - Cross-platform compatible (Windows/Linux/MacOS)
 - Standalone executable
 - User-friendly interface
 - Responsive controls
9. Data Management:
 - JSON-based song database
 - Automatic database creation dari MP3 files
 - Persistent user data (mood history, preferences)
 - Export functionality

6.2 Saran Pengembangan

1. Sistem dapat ditingkatkan dengan deteksi emosi otomatis berbasis AI. Saat ini pengguna masih memilih mood secara manual, namun ke depan emosi bisa dideteksi langsung dari ekspresi wajah, suara, teks, atau bahkan data fisik seperti detak jantung (jika tersedia). Dengan cara ini, hasil mood menjadi lebih akurat, pengguna tidak perlu banyak input, dan emosi bisa dipantau secara real-time.
2. Aplikasi dapat dikembangkan dengan integrasi layanan musik streaming seperti Spotify atau YouTube Music. Dengan begitu, pengguna tidak hanya terbatas pada lagu MP3 yang tersimpan di perangkat, tetapi bisa mengakses koleksi musik yang jauh lebih luas dan selalu terbaru tanpa perlu menyimpan file secara lokal.
3. Penambahan fitur sosial juga sangat disarankan. Aplikasi dapat mendukung banyak pengguna dengan akun masing-masing, fitur berbagi playlist, kolaborasi playlist, hingga timeline mood seperti media sosial. Fitur ini dapat meningkatkan interaksi antar pengguna dan membantu menemukan musik baru dari orang lain.
4. Dari sisi data, sistem dapat diperluas dengan analisis mood yang lebih mendalam. Tidak hanya menampilkan statistik sederhana, tetapi juga mengenali pola perubahan mood, tren mingguan atau bulanan, serta mempelajari preferensi musik pengguna secara otomatis. Hasil analisis ini bisa memberikan rekomendasi yang lebih personal dan bermanfaat, termasuk insight ringan terkait kebiasaan dan kondisi emosi pengguna.



5. Sistem rekomendasi playlist dapat dibuat lebih cerdas dan fleksibel. Playlist tidak hanya berdasarkan mood saat ini, tetapi juga mempertimbangkan waktu, aktivitas, dan kebiasaan pengguna. Selain itu, playlist dapat membantu mengubah suasana hati secara bertahap, misalnya dari stres menjadi lebih tenang, serta terus belajar dari perilaku pengguna seperti lagu yang sering dilewati atau diputar ulang.

6.3 Lampiran

(<https://github.com/Anggiee-sys/emotion-music-generator>)