ISSN 2319-5991 www.ijerst.com
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

MACHINE LEARNING DRIVEN MOOD-TUNES MUSIC RECOMMENDATION VIA FACIAL EMOTION RECOGNISATION

NAMMI MOHANA VAMSI KRISHNA[1], E. SATEESH[2]

[1]M. Tech Student, Department of Computer Science and Engineering, Amrita Sai Institute of Science & Technology (Autonomous) Paritala, Kanchikacherla Mandal, Krishna District – 521180, India

[2]Assistant Professor, Department of Computer Science and Engineering, Amrita Sai Institute of Science & Technology (Autonomous), Paritala, Kanchikacherla Mandal, Krishna District – 521180, India

Abstract: ***Emotions provide valuable insight into a person's psychological state, while facial expressions offer a convenient non-invasive method for emotion recognition. MoodTunes is a full-stack Django-based web application that uses facial emotion recognition to recommend music according to the user's detected mood. The system captures a single webcam image, processes it using the DeepFace framework, and identifies six emotions: happy, sad, angry, neutral, surprise, and fear. The detected emotion is mapped to an offline catalogue of 46 songs, from which suitable recommendations are generated through a Django-based recommendation module. The system provides REST APIs for emotion detection and music recommendations, with detection results stored in a relational database for basic analytics. To improve portability, MoodTunes includes a fallback mechanism using OpenCV Haar-Cascade face detection and a lightweight simulated emotion response when DeepFace or TensorFlow is unavailable. The application uses Django's ORM with SQLite for persistence and a vanilla JavaScript interface for webcam capture and API communication. Its three-tier architecture separates the presentation, application, and data layers, making the system lightweight, modular, and extensible. The proposed architecture can be further enhanced by integrating external music services, user authentication, personalized playlists, and advanced recommendation algorithms.***

Keywords: ***Facial Emotion Recognition, DeepFace, OpenCV, Django REST Framework, Music Recommendation, Computer Vision, WebRTC, Affective Computing, Human-Computer Interaction, SQLite.***

1. Introduction

The development of artificial intelligence (AI) and machine learning (ML) has enabled computer systems to understand human behavior and provide personalized services. Facial emotion recognition has become an important research area in affective computing because facial expressions provide useful visual information about a person's emotional state [1], [2]. The development of deep neural networks has further improved the ability of computer vision systems to extract meaningful features directly from facial images, reducing the dependence on manually designed features [3].

Music is closely associated with human emotions and can influence mood, relaxation, concentration, and entertainment. Conventional music recommendation systems generally depend on listening history, ratings, playlists, or predefined user preferences. However, these approaches may not adequately represent the user's current emotional state. Emotion-aware recommendation systems attempt to overcome this limitation by incorporating affective information into the recommendation process [4], [5].

Deep-learning-based facial analysis provides an effective approach for automatically identifying facial expressions. DeepFace demonstrated the capability of deep neural networks to learn highly discriminative facial representations from large-scale facial datasets [6]. Subsequent research has explored convolutional neural networks (CNNs) and other deep-learning architectures for facial

expression recognition, demonstrating their applicability to automated emotion classification [1], [7]. These developments provide a foundation for integrating facial emotion recognition with intelligent recommendation systems.

This paper presents MoodTunes, a machine-learning-driven music recommendation system that connects facial emotion recognition with mood-based song selection. The proposed system captures the user's facial image through a webcam and analyzes it using a deep-learning-based facial emotion recognition framework. The detected emotion is categorized into six classes: happy, sad, angry, neutral, surprise, and fear. Based on the identified emotion, the recommendation module selects appropriate songs from an emotion-associated music catalogue. Recent research has also demonstrated the feasibility of combining facial emotion detection with music recommendation, supporting the relevance of this approach [8], [9].

The proposed system is implemented as a web-based application to provide an accessible and interactive user experience. The architecture integrates facial image acquisition, emotion analysis, recommendation processing, database management, and a graphical interface. The system is designed to minimize manual interaction by automatically generating mood-based recommendations after facial analysis. Furthermore, its modular architecture can be extended to incorporate personalized user profiles, larger music databases, streaming platforms, and advanced recommendation algorithms.

2. Literature Review

Facial emotion recognition (FER) and music recommendation are two important research areas in artificial intelligence, computer vision, and affective computing. FER enables machines to estimate human emotional states from facial characteristics, while music recommendation systems identify suitable songs according to user preferences and contextual information. Their integration provides an opportunity to develop intelligent systems capable of recommending music according to a user's current emotional state.

Deep-learning techniques have significantly improved the performance of FER systems. Taigman *et al.* proposed DeepFace, which demonstrated the effectiveness of deep neural networks for extracting robust facial representations [1]. Subsequent studies have employed convolutional neural networks (CNNs) for facial expression classification using datasets such as FER2013 and AffectNet. These approaches commonly perform face detection followed by emotion classification into categories such as happy, sad, angry, fear, surprise, disgust, and neutral [2]. However, variations in illumination, head pose, facial appearance, and occlusion can affect recognition accuracy.

Music recommendation systems have traditionally used collaborative filtering and content-based approaches. Collaborative filtering relies on user-item interactions, whereas content-based techniques consider characteristics of songs and user preferences [3]. Although these methods can provide personalized recommendations, they generally depend on historical listening behavior and may not adequately reflect the user's current mood.

To overcome this limitation, emotion-aware music recommendation systems incorporate affective information into the recommendation process. Kim *et al.* investigated music emotion recognition and demonstrated the relationship between musical characteristics and perceived emotional states [4]. More recent studies have explored deep-learning-based approaches for emotion-aware music recommendation, including CNN-based systems that associate detected emotions with suitable music selections [5]. Facial emotion has also been investigated as a convenient input for music recommendation because it can be obtained using a standard camera without requiring specialized physiological sensors [6].

Despite these developments, many existing studies concentrate primarily on improving emotion classification accuracy, while complete integration with a practical recommendation platform receives comparatively less attention. Issues such as recommendation diversity, real-time processing, error handling, data storage, and user interaction are important for practical deployment. Therefore, the proposed MoodTunes system combines facial emotion recognition and mood-based music recommendation within a Django web application. The system captures facial images through a webcam, determines the dominant emotion using a deep-learning-based framework, and dynamically recommends songs associated with the detected emotional state. This integrated approach provides a practical foundation for developing personalized and

context-aware music recommendation applications.

3. Problem Statement

Conventional music recommendation systems mainly rely on listening history, ratings, and predefined user preferences, which may not reflect the user's current emotional state. Facial emotion recognition provides an opportunity to identify real-time mood through facial expressions, but its integration with personalized music recommendation is still challenging due to variations in facial appearance, lighting, and user preferences. Therefore, this work proposes MoodTunes, a machine-learning-based system that recognizes facial emotions using a webcam and automatically recommends suitable music based on the detected mood, providing a simple, interactive, and personalized listening experience.

4. Existing System

Existing music recommendation approaches generally provide mood-related playlists through manual selection or automatically generated recommendations based on user activity. In manual playlist selection, users browse predefined playlists such as happy, sad, or relaxing music and select one according to their perceived mood. In contrast, conventional algorithmic recommendation systems primarily analyze listening history, previously played songs, user preferences, and collaborative filtering information to generate personalized music suggestions. Although these approaches are useful for general music discovery, they do not directly consider the user's real-time emotional state. Mood identification is therefore dependent on the user's own judgment in manual selection, while automated systems mainly optimize for long-term listening preferences. Furthermore, conventional approaches generally do not use facial expressions or other real-time behavioral signals as recommendation inputs and provide limited information about the reason behind a particular recommendation.

5. Proposed System

The proposed MoodTunes system is a camera-driven, machine-learning-based music recommendation platform that automatically identifies the user's facial emotion and provides suitable music without requiring login credentials, listening history, or manual mood selection. The system captures a facial image through the webcam and analyzes it using the DeepFace framework to determine the user's dominant emotional state. Based on the detected emotion, the recommendation module selects a curated set of genre-appropriate songs and presents them through a web interface.

The system provides automatic emotion inference from a single webcam frame, eliminating the need for manual mood selection. Its recommendation process is stateless, with the song pool being reshuffled for every request, thereby providing varied recommendations without depending on previous user interactions. The emotion-to-song mapping is maintained in a human-readable catalogue, making the recommendation process transparent and easy to modify. In addition, the system incorporates a graceful fallback mechanism that allows the application to remain operational when the DeepFace or deep-learning environment is unavailable. Thus, MoodTunes provides a lightweight, transparent, and practical approach to real-time emotion-aware music recommendation.

6. Workflow Analysis

The complete workflow of the proposed MoodTunes system consists of a sequence of coordinated operations between the web browser, Django application, machine-learning module, database, and recommendation engine. The process begins when the user accesses the application and continues through webcam activation, facial image acquisition, emotion detection, result storage, and music recommendation.

Step 1—Page Loading: The user accesses the MoodTunes web application, and the Django server renders the main HTML interface along with the required JavaScript and CSS resources.

Step 2—Camera Activation: The user selects the Start Camera option. The browser requests webcam permission and displays the live video stream. The mood-scanning function becomes available once the camera is successfully activated.

Step 3—Frame Capture: When the user selects Scan My Mood, the current webcam frame is copied to a hidden HTML canvas. The captured frame is converted into a base64-encoded JPEG data string for transmission to the server.

Step 4—Emotion Detection Request: The frontend sends the encoded image to the Django REST endpoint together with the required CSRF token. The server validates the received request before initiating emotion analysis.

Step 5—Machine Learning Inference: The image is decoded into an OpenCV-compatible NumPy array and processed by the emotion detection module. When DeepFace is available, the facial image is analyzed to determine the dominant emotion and confidence distribution. If the deep-learning framework is unavailable, the system uses a fallback face-detection mechanism and generates a compatible emotion response.

Step 6—Data Persistence: When a face is successfully detected, the system stores the detection information in the database, including the identified emotion, confidence value, emotion-score distribution, detection method, and session information.

Step 7—Result Rendering: The detected emotion and confidence scores are returned to the frontend in JSON format. The interface then displays the mood report using confidence indicators and an emotion-wise score breakdown.

Step 8—Music Recommendation: Following successful emotion detection, the frontend sends a recommendation request to the Django recommendation endpoint. The recommendation module identifies the corresponding emotion category, randomly shuffles its available songs, and selects the required number of tracks.

Step 9—Playlist Rendering: The recommended songs are displayed as interactive cards on the webpage. The interface also presents an emotion-specific message and visual theme. The Shuffle Playlist option allows the user to request another randomly selected set of songs without repeating the complete workflow.

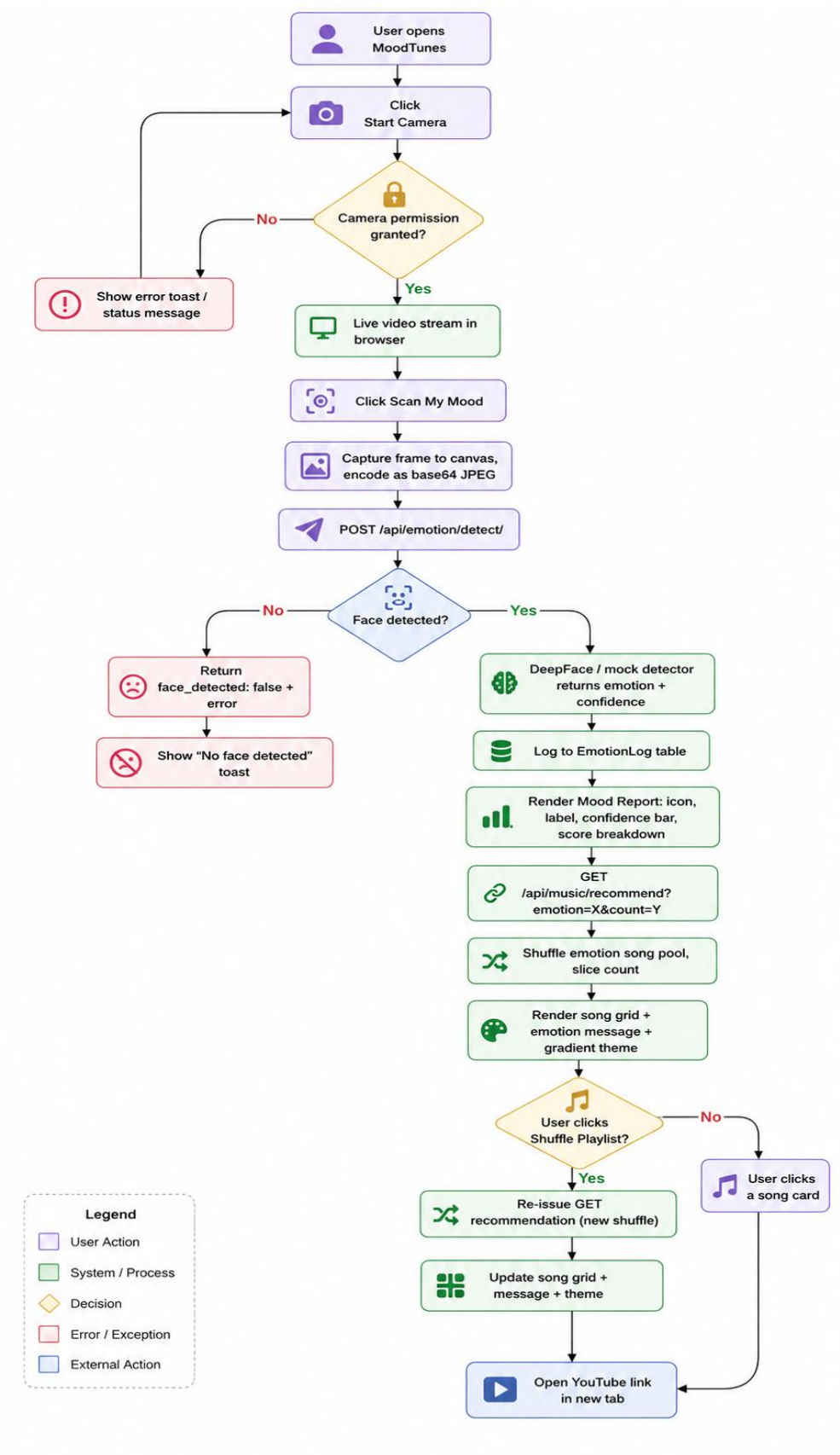


Fig.1 Flowchart of workflow analysis

7. System Design

7.1 Data Flow Diagrams

Data Flow Diagrams (DFDs) are used to represent the flow of information within the MoodTunes system. They provide a clear view of how data enters the system, how it is processed, and how the resulting information is stored or presented to the user. The system is represented using two levels: the Level 0 Context Diagram and the Level 1 Process Decomposition.

7.1.1 Level 0 — Context Diagram

The Level 0 DFD represents MoodTunes as a single high-level process and illustrates its interactions with the external entities associated with the system.

The primary external entities are:

- **End User:** Interacts with the system through a web browser and webcam to capture facial images and obtain emotion-based music recommendations.
- **Django Admin Operator:** Manages and inspects application data through the Django administration interface.
- **YouTube:** Acts as an outbound link target for recommended songs. In the default API-key-free configuration, the application does not retrieve data from YouTube at request time.

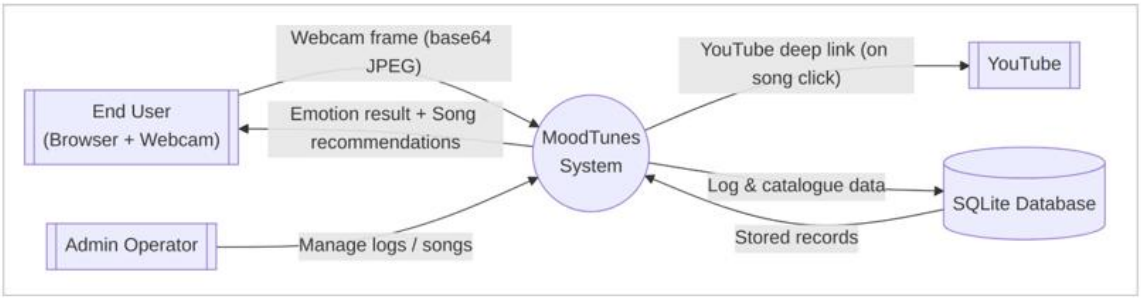


Fig.2 — Level 0 Data Flow Diagram

7.1.2 Level 1 — Process Decomposition

The Level 1 DFD decomposes the MoodTunes system into its major functional processes and illustrates the flow of data between the user interface, application components, and data stores.

The major processes are:

1. **Frame Capture and Submission** – Captures the user's facial image through the browser webcam and submits the frame to the Django application.
2. **Emotion Detection** – Processes the submitted image using the emotion-detection model and identifies the user's detected emotion.
3. **Detection Logging** – Stores the detected emotion and related information in the database.
4. **Music Recommendation** – Selects suitable songs based on the detected emotion.
5. **Recommendation Logging** – Records the generated recommendation information for later inspection and analysis.
6. **Song Catalogue Seeding** – Populates the song catalogue with the predefined curated music data.
7. **Admin Data Management** – Allows the administrator to inspect and manage stored application data through Django Admin.

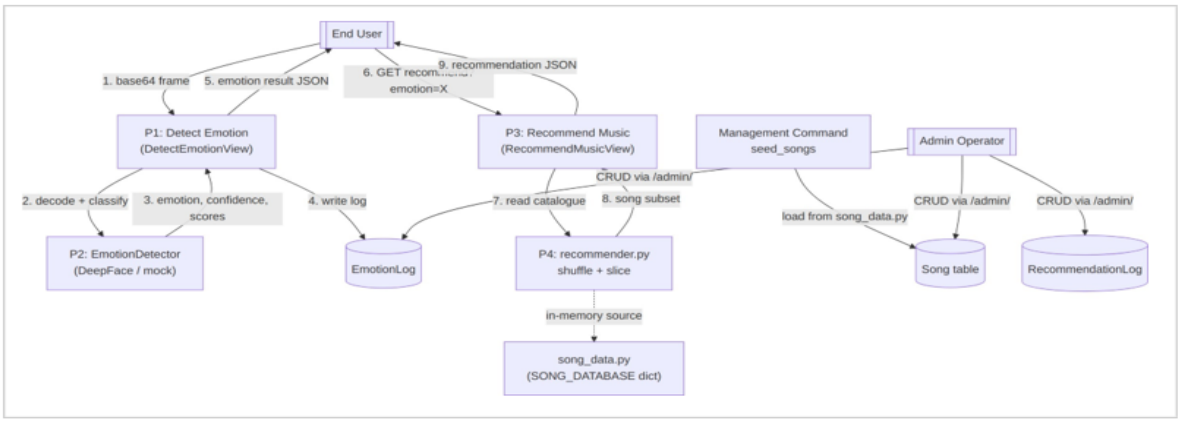


Fig.3 — Level 1 Data Flow Diagram

Data Source Duality

The MoodTunes application currently maintains the curated song information in two forms: a database table populated through the song-seeding mechanism and the in-memory SONG_DATABASE dictionary defined in song_data.py. Both sources contain the same curated song information.

The live recommendation function reads directly from the in-memory SONG_DATABASE dictionary. Therefore, the recommendation functionality can operate correctly even if the song catalogue seeding process has not yet been executed. The database table primarily provides support for Django Admin-based inspection and management and can also support future features such as filtering, searching, and database-driven recommendations.

7.2 UML Diagrams

Unified Modeling Language (UML) diagrams are used to document the structural and behavioral aspects of the MoodTunes application. The Class Diagram represents the static structure of the system, while the Sequence Diagram, Activity Diagram, and Use Case Diagram describe its dynamic behavior, interactions, workflows, and functional relationships.

These diagrams are designed to closely correspond to the actual Django models, views, service classes, and modules implemented in the application.

7.2.1 Class Diagram

The Class Diagram represents the core structural components of the MoodTunes application. It reflects the Django data models, including EmotionLog, together with the principal service class EmotionDetector and the relevant functions implemented within the application modules.

The relationships between these components illustrate how facial-expression data is processed, how detected emotions are recorded, and how emotion information is used to generate music recommendations. The module-level functions are represented as a conceptual class for clarity and to provide a simplified view of their functional relationships with the other system components.

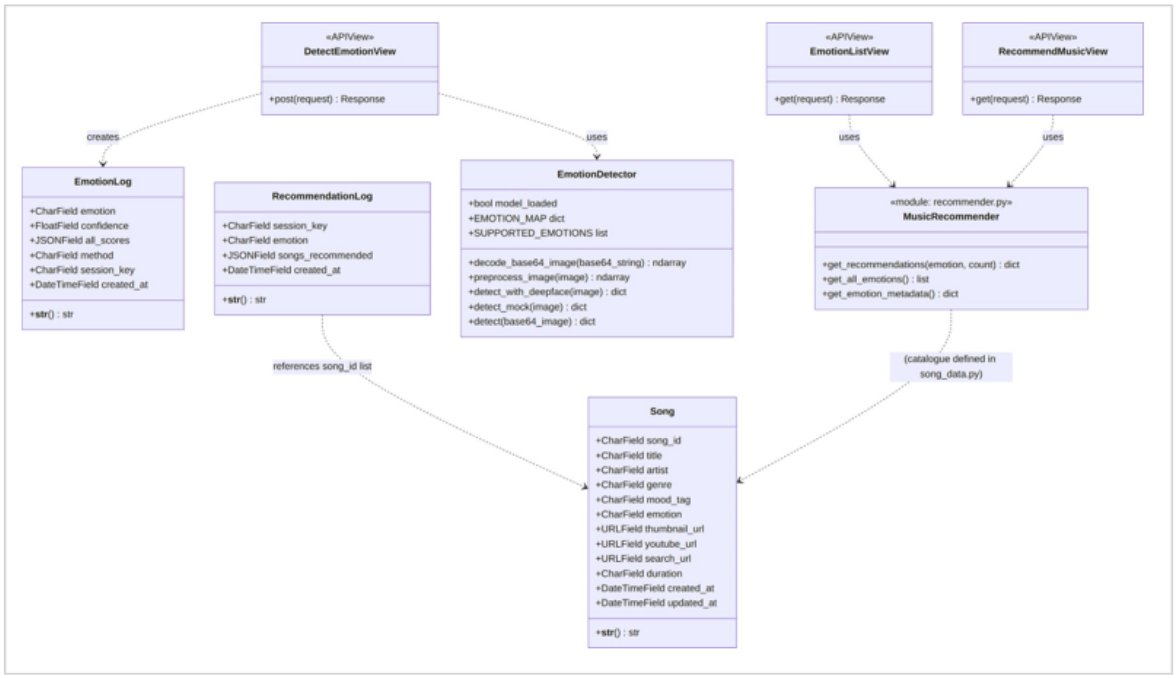


Fig. 4 — Class Diagram

7.2.2 Sequence Diagram

The Sequence Diagram illustrates the chronological flow of a complete scan-to-recommendation interaction. It shows the communication between the user's browser, Django views, the emotion-detection component, and the database.

The interaction begins when the user captures and submits a facial image. The Django application forwards the image to the emotion-detection component, which analyzes the image and returns the detected emotion. The detected result is then logged in the database, after which the recommendation process selects suitable music based on the identified emotion and returns the recommendations to the user.

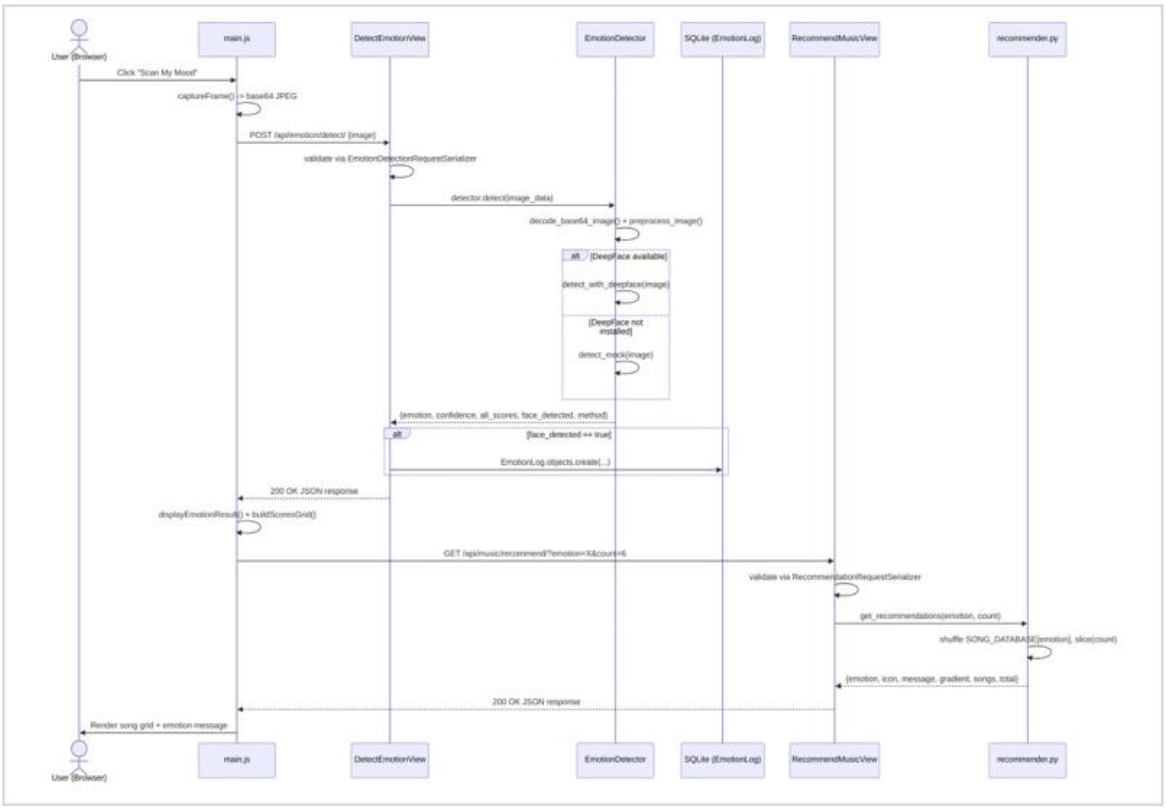


Fig. 5 — Sequence Diagram

7.2.3 Activity Diagram

The Activity Diagram represents the control flow of the emotion-detection request handler. It describes the sequence of activities performed from image submission through emotion detection and recommendation generation.

The diagram highlights two important decision points:

- **Face Detection:** Determines whether a valid face is detected in the submitted image. If no face is detected, an appropriate response is returned to the user.
- **ML Dependency Availability:** Determines whether the required machine-learning dependencies and model components are available. If the required dependencies are unavailable, the system follows the corresponding fallback or error-handling path.

This representation provides a clear understanding of the decision-making and error-handling mechanisms implemented in the application.

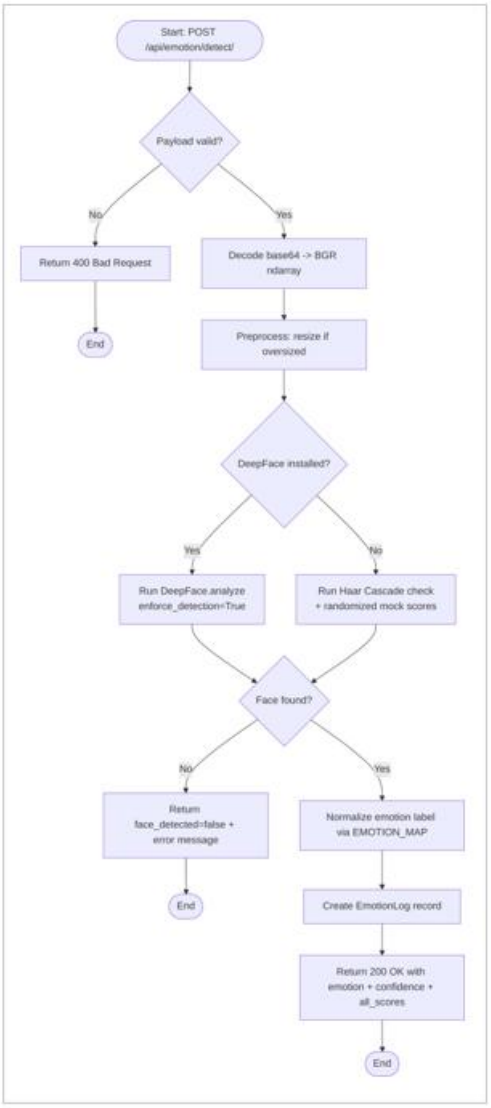


Fig. 6 — Activity Diagram

7.2.4 Use Case Diagram

The Use Case Diagram identifies the principal actors interacting with the MoodTunes system and the functions available to each actor.

The two primary actors are:

- **End User:** Captures a facial image, submits it for emotion detection, views the detected emotion, and receives music recommendations based on the detected emotional state.
- **Admin Operator:** Uses the Django Admin interface to inspect and manage application data, including stored emotion logs and song catalogue information.

The diagram provides a high-level representation of the functional requirements and interactions between the actors and the MoodTunes system.

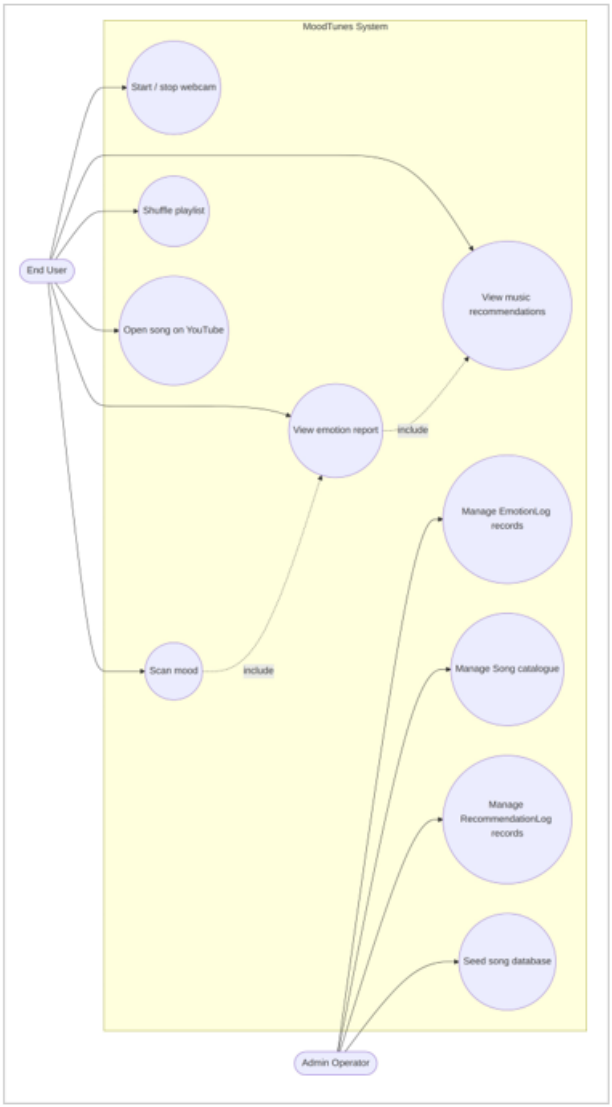


Fig. 7— Use Case Diagram

8. Results

8.1 Modules

MoodTunes is implemented as five cohesive functional modules distributed across two Django applications. Each module performs a specific responsibility, allowing the system to maintain a clear separation between emotion detection, music recommendation, data management, API communication, and frontend interaction.

8.1.1 Emotion Detection Module

Users: End User and System

The Emotion Detection Module is responsible for processing facial images submitted by the user and identifying the corresponding emotional state. Its major features include Base64 image decoding, DeepFace-based emotion classification, automatic fallback to a mock OpenCV-based detector, structured error handling when a face cannot be detected, and detection-history logging.

The EmotionDetector class forms the main machine-learning boundary of the application. A single instance of the detector is created at the module level and reused across requests. This approach avoids repeatedly initializing the DeepFace and TensorFlow components, thereby reducing unnecessary processing overhead.

The detector exposes a common detection interface whose return structure remains consistent for both the real DeepFace implementation and the OpenCV mock implementation. Consequently, the remaining components of the application do not need to know which detection backend is currently being used.

This design improves modularity and simplifies fallback handling.

8.1.2 Music Recommendation Module

Users: End User and System

The Music Recommendation Module is responsible for selecting songs according to the emotion detected by the system. Its primary features include emotion-to-song-pool mapping, randomized song selection for each request, emotion-specific display information, and fallback handling for unsupported emotion labels.

The module maintains the SONG_DATABASE as a dictionary organized according to emotion categories. Each emotion is associated with a collection of song records containing information such as song ID, title, artist, genre, mood tag, thumbnail, YouTube URL, search URL, and duration.

This framework-independent data structure keeps the music catalogue simple and easy to maintain. It can be reviewed, compared, version-controlled, and modified without requiring changes to Django models or database migrations.

8.1.3 Data Persistence and Admin Module

Users: Admin Operator

The Data Persistence and Admin Module manages the persistent application data used by MoodTunes. It contains the database models for emotion-detection history and music recommendations, together with the song catalogue model that mirrors the predefined static catalogue.

The module also provides an idempotent catalogue-seeding mechanism. The loader uses update_or_create with the song ID as the key, allowing existing records to be updated without creating duplicate entries. An optional command-line flag can be used to clear the existing catalogue before seeding.

The Django Admin interface is configured for the relevant models, enabling administrators to inspect and manage emotion logs, songs, and recommendation records efficiently.

8.1.4 REST API Module

Users: Frontend JavaScript and External API Clients

The REST API Module provides the communication layer between the frontend interface and the Django backend. It exposes three primary Django REST Framework endpoints for emotion detection, music recommendation, and retrieving supported emotions.

Request validation is performed using dedicated serializer classes to ensure that incoming data follows the expected structure. Corresponding response structures are used to maintain consistency in the information returned by the API.

This module provides a clear interface between the frontend and backend, allowing the frontend application to communicate with the system without directly accessing internal implementation details.

8.1.5 Frontend and User Experience Module

Users: End User

The Frontend and User Experience Module provides the interactive web interface through which users interact with MoodTunes. It manages webcam access, image capture, API communication, and presentation of emotion and music results.

The module uses WebRTC for webcam capture and the HTML canvas for capturing frames. Horizontal mirroring is corrected before images are submitted to the backend. API requests are implemented with CSRF-token support to maintain secure communication with the Django application.

The user interface follows a three-step workflow:

1. Webcam – Capturing the user's facial image.
2. Mood Report – Displaying the detected emotional state and confidence information.
3. Soundtrack – Presenting music recommendations based on the detected emotion.

Additional interface features include an animated loading overlay, rotating status messages, an animated confidence indicator, emotion-score visualization, a shuffleable song grid, toast notifications, and a dark glassmorphism-based visual design implemented using CSS custom properties.

8.2 Algorithms

The MoodTunes system uses a sequence of algorithms to convert the webcam image into an emotion classification and subsequently generate suitable music recommendations.

8.2.1 Image Decoding Algorithm

Purpose: Convert a Base64-encoded image received from the browser into an OpenCV-compatible NumPy array.

Procedure:

1. Remove the data-URI prefix from the image string, if present.
2. Decode the remaining Base64 data into raw image bytes.
3. Open the decoded bytes using PIL and convert the image to RGB format.
4. Convert the PIL image into a NumPy array.
5. Convert the image representation from RGB to BGR to match the OpenCV convention.

This process ensures that images captured by the browser can be processed correctly by the computer-vision components of the system.

8.2.2 Image Preprocessing Algorithm

Purpose: Normalize the image dimensions to provide consistent and efficient processing.

Procedure:

1. Determine the height and width of the input image.
2. Check whether the largest image dimension exceeds 640 pixels.
3. If necessary, calculate an appropriate scaling factor.
4. Resize the image using the selected interpolation method.
5. Return the resized image or the original image if no resizing is required.

This preprocessing step reduces the computational workload while maintaining sufficient image quality for facial-emotion detection.

8.2.3 DeepFace Emotion Classification Algorithm

Purpose: Identify the dominant facial emotion and generate the complete emotion-confidence distribution.

Procedure:

1. Pass the preprocessed image to the DeepFace emotion-analysis function.
2. If no face is detected, capture the resulting exception and convert it into a user-facing detection error.
3. Extract the emotion scores from the first detected face.
4. Normalize the DeepFace emotion labels to the six emotion categories supported by MoodTunes.
5. Map unsupported categories to an appropriate supported category when required by the recommendation system.
6. Round the emotion scores to two decimal places.
7. Return the detected emotion, confidence value, complete score distribution, and detection method in a structured response.

This algorithm provides a consistent output format that can be directly consumed by the recommendation and frontend components.

8.2.4 Mock Detection Algorithm

Purpose: Provide a lightweight fallback implementation when the DeepFace-based detector is unavailable, particularly during development and testing.

Procedure:

1. Convert the input image into grayscale.
2. Apply OpenCV's bundled face classifier.
3. Check whether any faces are detected.
4. If no face is detected, generate the same type of detection failure used by the primary detection path.
5. If a face is detected, generate six random emotion scores within the defined range.
6. Normalize the scores so that their total is 100.
7. Select the emotion with the highest score as the dominant emotion.
8. Return the result using the same response structure as the DeepFace implementation.

Using the same response format for both implementations ensures that downstream components can operate independently of the selected detection backend.

8.2.5 Music Recommendation Selection Algorithm

Purpose: Select a randomized and size-limited collection of songs corresponding to the detected emotion.

Procedure:

1. Normalize the supplied emotion string by converting it to lowercase and removing unnecessary whitespace.
2. Check whether the emotion exists in SONG_DATABASE.
3. If the emotion is not recognized, log a warning and use the default emotion category.
4. Copy the corresponding song list so that the original catalogue is not modified.
5. Randomly shuffle the copied list.
6. Select the requested number of songs, with the supported count restricted to the defined range.
7. Add emotion-specific metadata such as the icon, message, and visual gradient.
8. Return the complete recommendation response.

This approach provides varied recommendations across requests while preserving the original song catalogue.

8.2.6 Detection Request Orchestration Algorithm

Purpose: Coordinate request validation, emotion detection, result logging, and response generation.

Procedure:

1. Validate the incoming request using the appropriate serializer.
2. If validation fails, return an HTTP 400 response containing the validation errors.
3. Pass the validated image to the emotion detector.
4. If a face and emotion are successfully detected, attempt to create an emotion-log record in the database.
5. If database logging fails, record a warning without interrupting the user-facing detection response.
6. Construct the final response containing the detection result, confidence, score distribution, detection method, and face-detection status.
7. If no face is detected, include an appropriate error message while still returning a normal HTTP response because failure to detect a face is treated as an expected user-level outcome rather than a server failure.

This orchestration keeps validation, detection, persistence, and response handling logically separated while providing graceful error handling.

8.3 Algorithm Diagram

The algorithm flow diagram illustrates the complete emotion-detection process, beginning with image submission and continuing through image preprocessing, face detection, emotion classification, result processing, and response generation.

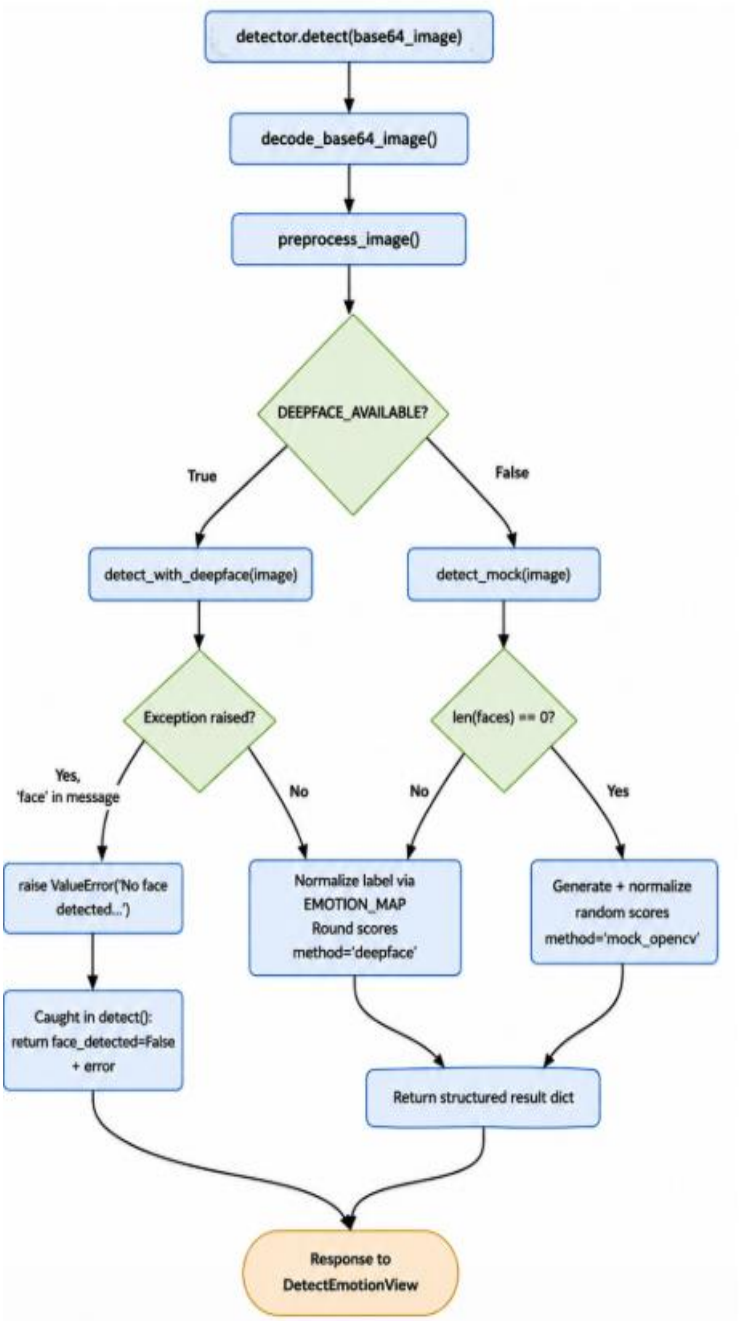


Fig. 8— Emotion Detection Algorithm Flow

8.4 Code

The following code excerpts represent the core implementation of the major MoodTunes components. They demonstrate the implementation of the emotion-detection entry point, REST API view, recommendation engine, song-catalogue loader, and frontend frame-capture process.

8.4.1 Emotion Detection — Detection Entry Point

The detection entry point acts as the interface to the EmotionDetector component. It receives the browser-supplied image, performs the required decoding and processing, and returns the standardized emotion-detection result.

```
def detect(self, base64_image: str) -> dict:
    """
    Main entry point: detect emotion from a base64-encoded image.
    """
    try:
        # Step 1: Decode the base64 image
        image = self.decode_base64_image(base64_image)

        # Step 2: Preprocess for better detection
        image = self.preprocess_image(image)

        # Step 3: Run detection (DeepFace or mock)
        if DEEPFACE_AVAILABLE:
            result = self.detect_with_deepface(image)
        else:
            result = self.detect_mock(image)

        logger.info(
            f"Emotion detected: {result['emotion']} "
            f"(confidence: {result['confidence']}%, method: {result['method']})"
        )
        return result

    except ValueError as e:
        logger.warning(f"Detection warning: {str(e)}")
        return {
            'emotion': None, 'confidence': 0, 'all_scores': {},
            'face_detected': False, 'method': 'failed', 'error': str(e)
        }
    except Exception as e:
        logger.error(f"Unexpected detection error: {str(e)}", exc_info=True)
        return {
            'emotion': None, 'confidence': 0, 'all_scores': {},
            'face_detected': False, 'method': 'failed',
            'error': "An unexpected error occurred during emotion detection."
        }
```

8.4.2 Emotion Detection — API View

The DetectEmotionView class implements the REST endpoint responsible for receiving webcam images and coordinating the detection process.

The API first validates the incoming request using EmotionDetectionRequestSerializer. After successful validation, the image is passed to the detector. When an emotion is successfully detected, the result is stored in the EmotionLog model together with the confidence score, complete emotion-score distribution, detection method, and session information.

Database logging errors are handled independently so that a failure to store a log does not prevent the user from receiving the detection result. The API returns the detection status, detected emotion, confidence, score distribution, face-detection status, and detection method.

```
class DetectEmotionView(APIView):
    """
    POST /api/emotion/detect/
    Accepts a base64 webcam image, runs the emotion detector,
    logs the result, and returns the detection + music recommendations.
    """
    def post(self, request, *args, **kwargs):
        serializer = EmotionDetectionRequestSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(
                {'success': False, 'errors': serializer.errors},
                status=status.HTTP_400_BAD_REQUEST
            )

        image_data = serializer.validated_data['image']
        result = detector.detect(image_data)

        if result.get('face_detected') and result.get('emotion'):
            try:
                EmotionLog.objects.create(
                    emotion=result['emotion'],
                    confidence=result['confidence'],
                    all_scores=result.get('all_scores', {}),
                    method=result.get('method', 'unknown'),
                    session_key=request.session.session_key or ''
                )
            except Exception as e:
                logger.warning(f"Failed to log emotion: {e}")

        response_data = {
            'success': result.get('face_detected', False),
            'emotion': result.get('emotion'),
            'confidence': result.get('confidence', 0),
            'all_scores': result.get('all_scores', {}),
            'face_detected': result.get('face_detected', False),
            'method': result.get('method', 'unknown'),
        }
        if not result.get('face_detected'):
            response_data['error'] = result.get('error', 'Detection failed.')
            return Response(response_data, status=status.HTTP_200_OK)

        return Response(response_data, status=status.HTTP_200_OK)
```

8.4.3 Music Recommendation — Recommendation Engine

The recommendation engine receives an emotion and the desired number of songs. It normalizes the emotion value, verifies that the emotion exists in the song database, and falls back to the default emotion when necessary.

The corresponding song list is copied and randomly shuffled before the requested number of songs is selected. The response also contains emotion-specific presentation information, including the emotion icon, message, and gradient, together with the selected songs and total count.

```
def get_recommendations(emotion: str, count: int = 6) -> dict:
    """
    Return a list of song recommendations for the given emotion.
    """
    emotion = emotion.lower().strip() if emotion else DEFAULT_EMOTION
    if emotion not in SONG_DATABASE:
        logger.warning(f"Unknown emotion '{emotion}', falling back to '{DEFAULT_EMOTION}'")
        emotion = DEFAULT_EMOTION

    all_songs = SONG_DATABASE[emotion]
    shuffled = all_songs.copy()
    random.shuffle(shuffled)
    selected = shuffled[:count]

    logger.info(f"Recommending {len(selected)} songs for emotion: {emotion}")

    return {
        "emotion": emotion,
        "emotion_icon": EMOTION_ICONS.get(emotion, "  "),
        "emotion_message": EMOTION_MESSAGES.get(emotion, "Here are some songs for you!"),
        "emotion_gradient": EMOTION_GRADIENTS.get(emotion, "linear-gradient(135deg,#667eea,
        "songs": selected,
        "total": len(selected),
    }
```

8.4.4 Song Catalogue — Idempotent Catalogue Loader

The catalogue loader is implemented as a Django management command that populates the database using the curated SONG_DATABASE information.

An optional --clear argument allows existing records to be removed before loading new data. The command iterates through each emotion and its associated songs and uses update_or_create with the song ID as the unique lookup key. Existing songs are therefore updated, while new songs are inserted without generating duplicates.

At the end of the process, the command reports the number of created and updated records together with the total number of songs stored in the database.

```
class Command(BaseCommand):
    help = 'Seed the database with curated songs from song_data.py'

    def add_arguments(self, parser):
        parser.add_argument('--clear', action='store_true',
                            help='Clear all existing songs before seeding')

    def handle(self, *args, **options):
        if options['clear']:
            count = Song.objects.count()
            Song.objects.all().delete()
            self.stdout.write(self.style.WARNING(f'Cleared {count} existing songs.'))

        created = 0
        updated = 0
        for emotion, songs in SONG_DATABASE.items():
            for song_data in songs:
                obj, was_created = Song.objects.update_or_create(
                    song_id=song_data['id'],
                    defaults={
                        'title': song_data['title'],
                        'artist': song_data['artist'],
                        'genre': song_data.get('genre', ''),
                        'mood_tag': song_data.get('mood_tag', ''),
                        'emotion': emotion,
                        'thumbnail_url': song_data.get('thumbnail', ''),
                        'youtube_url': song_data.get('youtube_url', ''),
                        'search_url': song_data.get('search_url', ''),
                        'duration': song_data.get('duration', ''),
                    }
                )
                created += 1 if was_created else 0
                updated += 0 if was_created else 1

        self.stdout.write(self.style.SUCCESS(
            f'Done! Created: {created} songs, Updated: {updated} songs. '
            f'Total: {Song.objects.count()} songs in database.'
        ))
```

8.4.5 Frontend — Frame Capture and API Call

The frontend frame-capture component obtains an image from the user's webcam through WebRTC, captures the current frame using a canvas element, and prepares the resulting image for submission to the emotion-detection API.

The captured image is transmitted to the Django backend using an API request that includes the required CSRF protection. The returned emotion-detection information is then used to update the mood-report and music-recommendation stages of the interface.

```
function captureFrame() {
   const ctx = canvasEl.getContext('2d');
   const width = videoEl.videoWidth || 640;
   const height = videoEl.videoHeight || 480;
   canvasEl.width = width;
   canvasEl.height = height;

   // Draw mirrored (flip back for correct orientation for the ML model)
   ctx.save();
   ctx.scale(-1, 1);
   ctx.drawImage(videoEl, -width, 0, width, height);
   ctx.restore();

   return canvasEl.toDataURL('image/jpeg', 0.85);
}

async function detectEmotion(imageBase64) {
   const response = await fetch('/api/emotion/detect/', {
     method: 'POST',
     headers: {
        'Content-Type': 'application/json',
        'X-CSRFToken': getCsrfToken(),
     },
     body: JSON.stringify({ image: imageBase64 }),
   });
   if (!response.ok) {
      const errorData = await response.json().catch(() => ({}));
      throw new Error(errorData.detail || `HTTP ${response.status}`);
   }
   return response.json();
}
```

8.5 API Reference

The MoodTunes REST API provides endpoints for rendering the application, detecting emotions, generating music recommendations, and retrieving supported emotion categories. All API endpoints are hosted under the Django development server's base URL.

Method	URL	Payload	Response
GET	/	—	Renders the main SPA (index.html)

Method	URL	Payload	Response
POST	/api/emotion/detect/	{ "image": "<base64 JPEG, with or without data-URI prefix>" }	{ "success": bool, "emotion": str|null, "confidence": float|null, "scores": {emotion: score, ...}, "face_detected": bool, "method": "deepface"|"mock_opencv|", "message": str|null }
GET	/api/music/recommendd/ ?emotion=<happy|sad|angry| neutral|surprise|fear> &count=<1-20, default 6>	– (query params)	{ "emotion": str, "emotion_icon": str, "emotion_message": str, "gradient": str, "songs": [{"song_id", "title", "artist", "genre", "mood_tag", "thumbnail", "youtube_url", "search_url", "duration"}, ...], "total": int }
GET	/api/music/emotions/	—	{ "emotions": [str, ...], "metadata": { emotion: {icon, message, gradient} } }
GET / POST	/admin/	Django admin auth	Django admin interface for EmotionLog, Song, Recommendation, etc.

8.5.6 Example Recommendation Response

A typical recommendation response contains the detected emotion, associated presentation metadata, the selected song records, and the total number of recommendations returned. For example, when the detected emotion is sad, the response contains the corresponding emotion message and visual metadata together with the recommended songs.

The song records include information such as the song ID, title, artist, genre, and other catalogue attributes.

Overall, the implementation demonstrates a modular Django-based architecture in which facial-image processing, emotion classification, recommendation generation, data persistence, REST communication, and frontend interaction are separated into clearly defined components. This structure improves maintainability while allowing the system to operate with either the primary DeepFace detector or the lightweight fallback mechanism.

```
// Request
{
  "image": "..."
}

// Response (200 OK, face detected)
{
  "success": true,
  "emotion": "happy",
  "confidence": 87.5,
  "all_scores": {
    "happy": 87.5, "neutral": 8.2, "sad": 2.1,
    "angry": 1.0, "surprise": 0.8, "fear": 0.4
  },
  "face_detected": true,
  "method": "deepface"
}

// Response (200 OK, no face found)
{
  "success": false,
  "emotion": null,
  "confidence": 0,
  "all_scores": {},
  "face_detected": false,
  "method": "failed",
  "error": "No face detected in the image. Please look directly at the camera."
}
```

```
{
  "emotion": "sad",
  "emotion_icon": "  ",
  "emotion_message": "It's okay to feel sad. Let these soothing songs hold you gently.   '
  "emotion_gradient": "linear-gradient(135deg, #a1c4fd 0%, #c2e9fb 100%)",
  "songs": [
    {"song_id": "s001", "title": "...", "artist": "...", "genre": "Acoustic / Folk", "..."
  ],
  "total": 3
}
```


Conclusion

MoodTunes demonstrates that an effective AI-driven personalization system can be developed as a compact, self-contained Django application without requiring heavyweight machine-learning infrastructure, paid inference APIs, or client-side frameworks. By combining the facial-emotion recognition capabilities of DeepFace with a transparent, manually curated recommendation catalogue, the system avoids the complexity of opaque recommendation models and resource-intensive ML environments. The graceful-degradation mechanism is a major strength of the design, as the mock emotion detector follows the same response contract as the real DeepFace-based detector. This allows the application to remain functional for development and demonstration even when the complete deep-learning environment is unavailable. Functionally, MoodTunes satisfies all the objectives defined in Chapter 1. The system detects emotion from a single webcam frame without manual mood input, maps the detected emotion to a suitable music genre and playlist, provides these capabilities through a documented REST API, and remains usable in a dependency-light development environment. The modular separation between emotion detection and music recommendation also improves maintainability and extensibility, as both components communicate through a simple emotion-label interface. This allows future versions to replace the emotion classifier or integrate a live music-streaming catalogue without requiring significant changes to the remaining system. Overall, MoodTunes demonstrates a practical, transparent, and flexible approach to implementing AI-based personalization in a lightweight web application.

References

[1] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2014, pp. 1701–1708, doi: 10.1109/CVPR.2014.220.

[2] E. Sariyanidi, H. Gunes, and A. Cavallaro, "Automatic analysis of facial affect: A survey of registration, representation, and recognition," IEEE Trans. Pattern Anal. Mach. Intell., vol. 37, no. 6, pp. 1113–1133, Jun. 2015, doi: 10.1109/TPAMI.2014.2366127.

[3] C. A. Corneanu, M. Oliu, J. F. Cohn, and S. Escalera, "Survey on RGB, 3D, thermal, and multimodal approaches for facial expression recognition: History, trends, and affect-related applications," IEEE Trans. Pattern Anal. Mach. Intell., vol. 38, no. 8, pp. 1548–1568, Aug. 2016, doi: 10.1109/TPAMI.2016.2515606.

[4] B. Martinez, M. F. Valstar, B. Jiang, and M. Pantic, "Automatic analysis of facial actions: A survey," IEEE Trans. Affect. Comput., vol. 10, no. 3, pp. 325–347, Jul. 2019, doi: 10.1109/TAFFC.2017.2731763.

[5] O. S. Ekundayo and S. Viriri, "Facial expression recognition: A review of trends and techniques," IEEE Access, vol. 9, pp. 136944–136973, 2021, doi: 10.1109/ACCESS.2021.3113464.

[6] S. Li and W. Deng, "Deep facial expression recognition: A survey," IEEE Trans. Affect. Comput., vol. 13, no. 3, pp. 1195–1215, 2022, doi: 10.1109/TAFFC.2020.2981446.

[7] P. J. Bota, C. Wang, A. L. N. Fred, and H. P. da Silva, "A review, current challenges, and future possibilities on emotion recognition using machine learning and physiological signals," IEEE Access, vol. 7, pp. 140990–141020, 2019, doi: 10.1109/ACCESS.2019.2944001.

[8] R. W. Picard, Affective Computing. Cambridge, MA, USA: MIT Press, 1997.

[9] P. Ekman and W. V. Friesen, Facial Action Coding System: A Technique for the Measurement of Facial Movement. Palo Alto, CA, USA: Consulting Psychologists Press, 1978.

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

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural

networks," in Advances in Neural Information Processing Systems, vol. 25, 2012, pp. 1097–1105.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[13] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 2, pp. 423–443, Feb. 2019, doi: 10.1109/TPAMI.2018.2798607.

[14] F. Chollet, Deep Learning with Python, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2021.

[15] F. Ricci, L. Rokach, and B. Shapira, Eds., Recommender Systems Handbook, 3rd ed. New York, NY, USA: Springer, 2022.

[16] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," IEEE Trans. Knowl. Data Eng., vol. 17, no. 6, pp. 734–749, Jun. 2005, doi: 10.1109/TKDE.2005.99.

[17] P. Resnick and H. R. Varian, "Recommender systems," Commun. ACM, vol. 40, no. 3, pp. 56–58, Mar. 1997.

[18] B. Amiri, N. Shahverdi, A. Haddadi, and Y. Ghahremani, "Beyond the trends: Evolution and future directions in music recommender systems research," IEEE Access, vol. 12, pp. 51500–51522, 2024, doi: 10.1109/ACCESS.2024.3386684.

[19] M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, and M. Elahi, "Current challenges and visions in music recommender systems research," Int. J. Multimedia Data Eng. Manage., vol. 7, no. 2, pp. 1–17, 2016.

[20] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," Adv. Artif. Intell., vol. 2009, pp. 1–19, 2009.

[21] R. Burke, "Hybrid recommender systems: Survey and experiments," User Model. User-Adapt. Interact., vol. 12, no. 4, pp. 331–370, 2002.

[22] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.

[23] Django Software Foundation, "Django documentation," 2026. Django Documentation

[24] Encode OSS Ltd., "Django REST framework documentation," 2026. Django REST Framework Documentation

[25] Encode OSS Ltd., "Documenting your API—Django REST framework," 2026. Django REST Framework API Documentation