Experiment No:2

Aim:-Design UML Diagrams for the selected Case Study.

Theory:-

Unified Modeling Language (UML):
UML is a standardized visual language used to model the structure and behavior of software systems. It provides a way to visualize a system's design and architecture, helping developers and stakeholders understand, communicate, and document the structure and functionality of complex software. UML consists of various types of diagrams, each serving a different purpose in representing different aspects of the system.

Use Case Diagram:A Use Case Diagram represents the functional requirements of a system. It illustrates the interactions between external actors (users or other systems) and the system itself through use cases (specific functionalities or services the system provides). This diagram helps in identifying the system's primary functions and its users.
Components:
- **Actors:** Represent entities (human users or other systems) that interact with the system.
- **Use Cases:** Depict the actions or services the system performs for the actors.
- **System Boundary:** Defines the scope of the system being modeled.

Class Diagram:
A Class Diagram provides a static view of the system, representing its structure by showing the system's classes, their attributes, methods, and the relationships between them. It is essential for modeling the object-oriented structure of a system.
Classes: Represent the blueprint of objects in the system. Each class includes:
- **Name:** The identifier of the class.
- **Attributes:** Variables or properties of the class.
- **Methods:** Functions or operations the class can perform.

Advantages:
- **Clear Structure:** Provides a clear and detailed representation of the system's structure.
- **Design Blueprint:** Serves as a blueprint for system development.
- **Reusability:** Helps in identifying reusable components.

Disadvantages:
- **Complexity:** Can become complex with large systems.
- **Maintenance:** Requires continuous updating as the system evolves.

Sequence Diagram:
A Sequence Diagram models the flow of logic within a system in a visual manner. It shows how objects interact with each other over time, focusing on the sequence of messages exchanged between them to carry out a specific functionality or use case.
Advantages:
- **Clarity in Communication:** Clearly depicts the interaction flow, making it easier to understand the sequence of operations.
- **Problem Detection:** Helps in identifying potential issues in the sequence of interactions.

- **Documentation:** Acts as a valuable document for understanding the dynamic behavior of a system.

Disadvantages:

- **Complexity in Large Systems:** Can become cluttered and hard to follow for complex interactions.
- **Time-Consuming:** Creating detailed sequence diagrams for every use case can be time-consuming.
- **Static Nature:** Does not effectively represent all aspects of the system's behavior.

Topic:-Music recommendation system based on facial expressions

1.Use Case Diagram

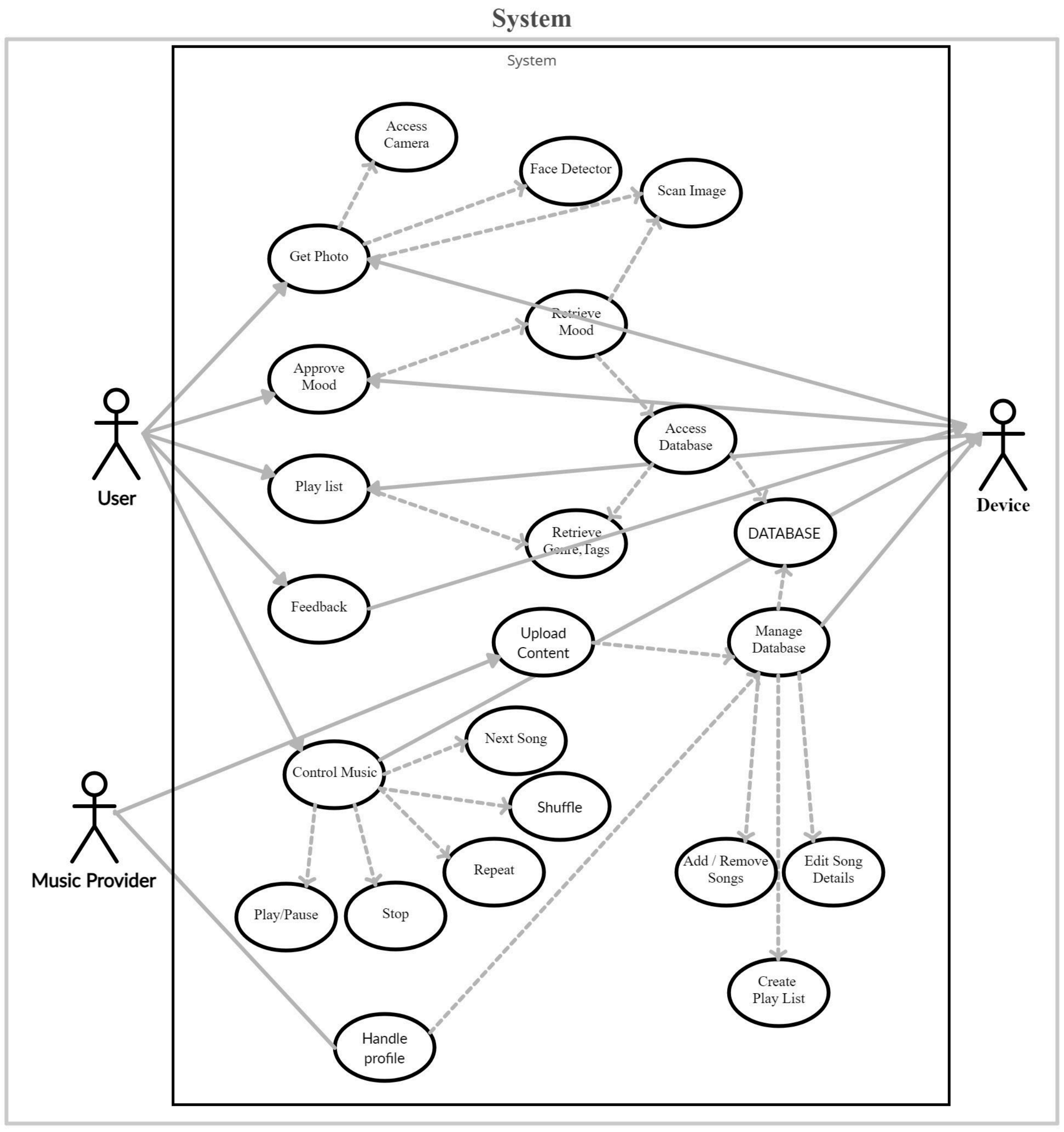
User: The primary actor who interacts with the system to have music recommended based on their facial expressions.
Device: Represents the hardware, likely including the camera and possibly the device that runs the system, such as a smartphone or computer.
Music Provider: An actor responsible for uploading and managing music content within the system.
Database: A subsystem that stores music, mood tags, and other relevant data for the recommendation system.

Use Cases

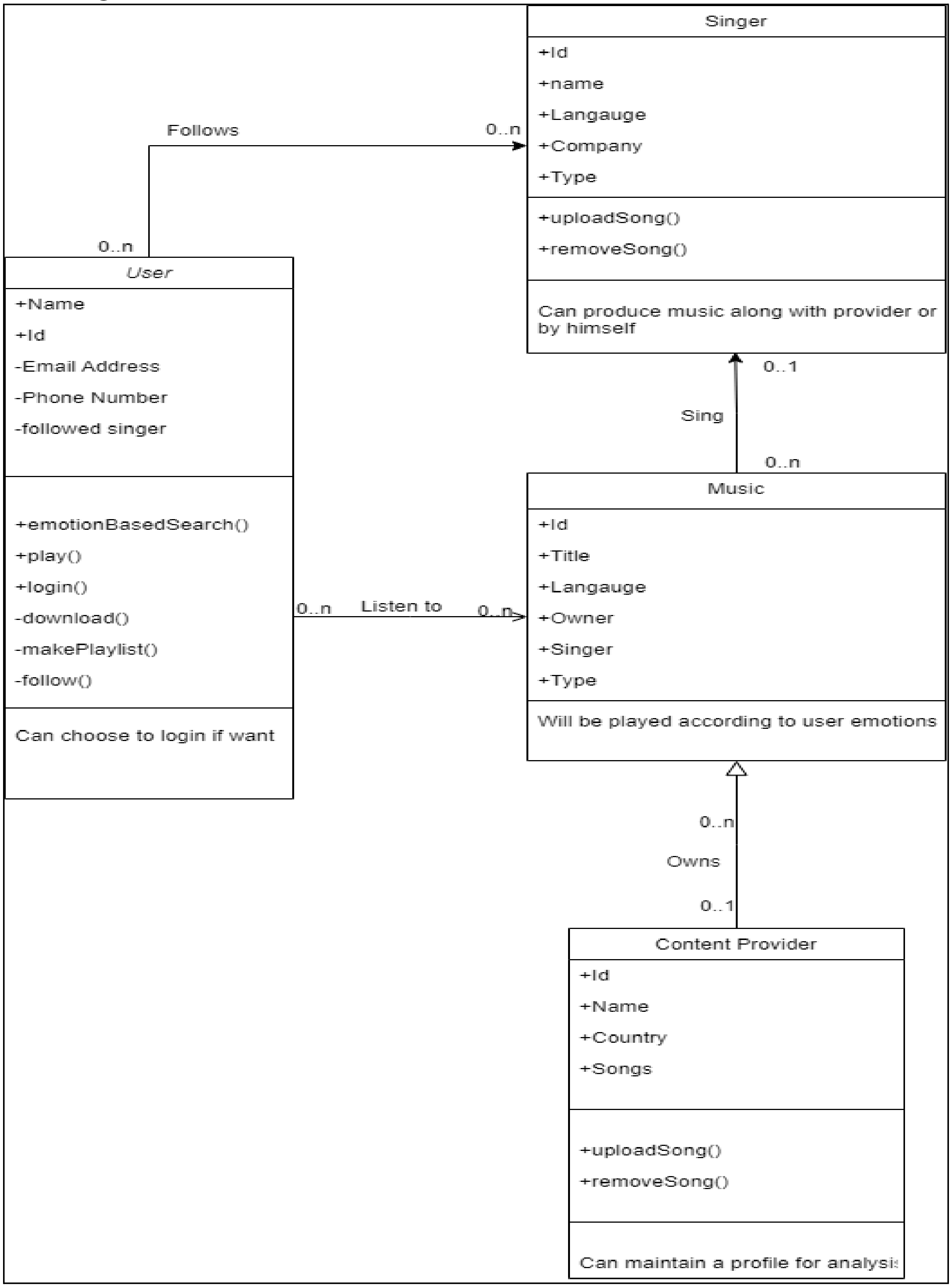
1. Access Camera: The system accesses the device's camera to capture the user's facial image.
2. Get Photo: The system retrieves the captured photo for further processing.
3. Face Detector: The system analyzes the captured image to detect facial features.
4. Scan Image: This use case likely involves processing the image to extract facial expressions.
5. Retrieve Mood: Based on the analysis, the system determines the user's mood.
6. Approve Mood: The user can confirm the mood detected by the system, allowing for more accurate recommendations.
7. Access Database: The system accesses the database to retrieve music that matches the detected mood.
8. Retrieve Genre/Tags: The system retrieves relevant music genres or tags associated with the detected mood.
9. Upload Content: The music provider uploads new music content to the system, which can be recommended to users.
10. Manage Database: Involves maintaining the database, including updating mood tags associated with different songs.
11. Add/Remove Songs: The music provider can add or remove songs from the database, influencing the pool of available recommendations.
12. Edit Song Details: The music provider can edit the details of songs, such as their associated mood tags.
13. Create Play List: The system or music provider creates playlists that can be mood-specific, allowing for curated listening experiences.
14. Play List: The user can play songs from a playlist, possibly one recommended based on

their mood.

15. Feedback: The user provides feedback on the recommendations, which can be used to improve the system's accuracy.
16. Control Music: The user can control the playback of music, including actions like play/pause, stop, shuffle, repeat, and next song.
17. Handle Profile: The music provider manages their profile, which might include preferences that influence content uploads.



Class Diagram:



1. User Class

- ➔ Attributes:
  - ◆ +Name, +Id, +Email Address, -Phone Number, -followed singer
- ➔ Methods:
  - ◆ +emotionBasedSearch(), +play(), +login(), -download(), -makePlaylist(), -follow()
- ➔ Relationship:
  - ◆ The user can follow multiple singers (0..n relationship) and listen to multiple songs.

2. Singer Class

- ➔ Attributes:
  - ◆ +Id, +name, +Language, +Company, +Type
- ➔ Methods:
  - ◆ +uploadSong(), +removeSong()
- ➔ Relationship:
  - ◆ Singers can produce songs independently or with a content provider.
  - ◆ The singer can sing multiple songs (0..n relationship).

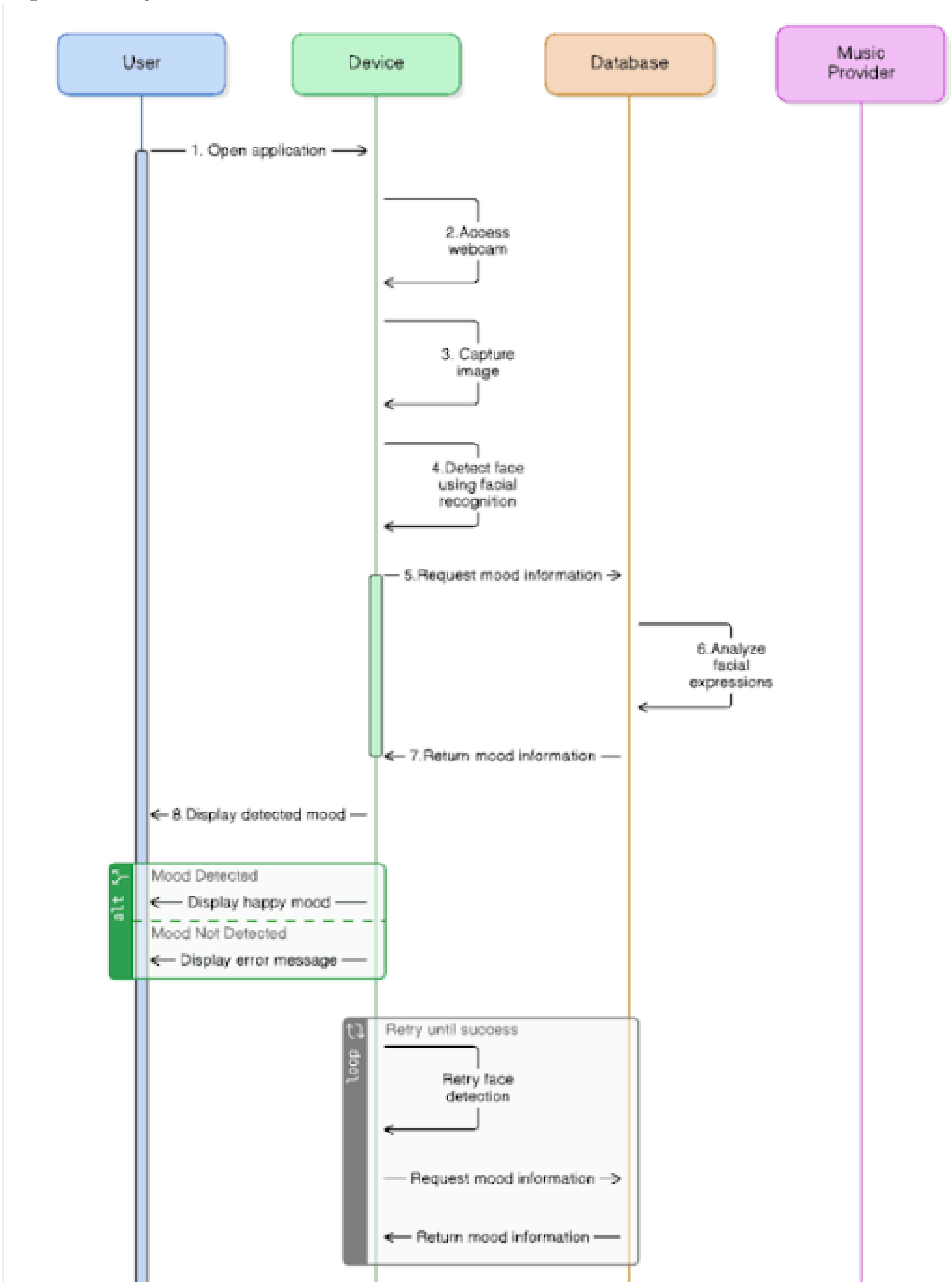
3. Music Class

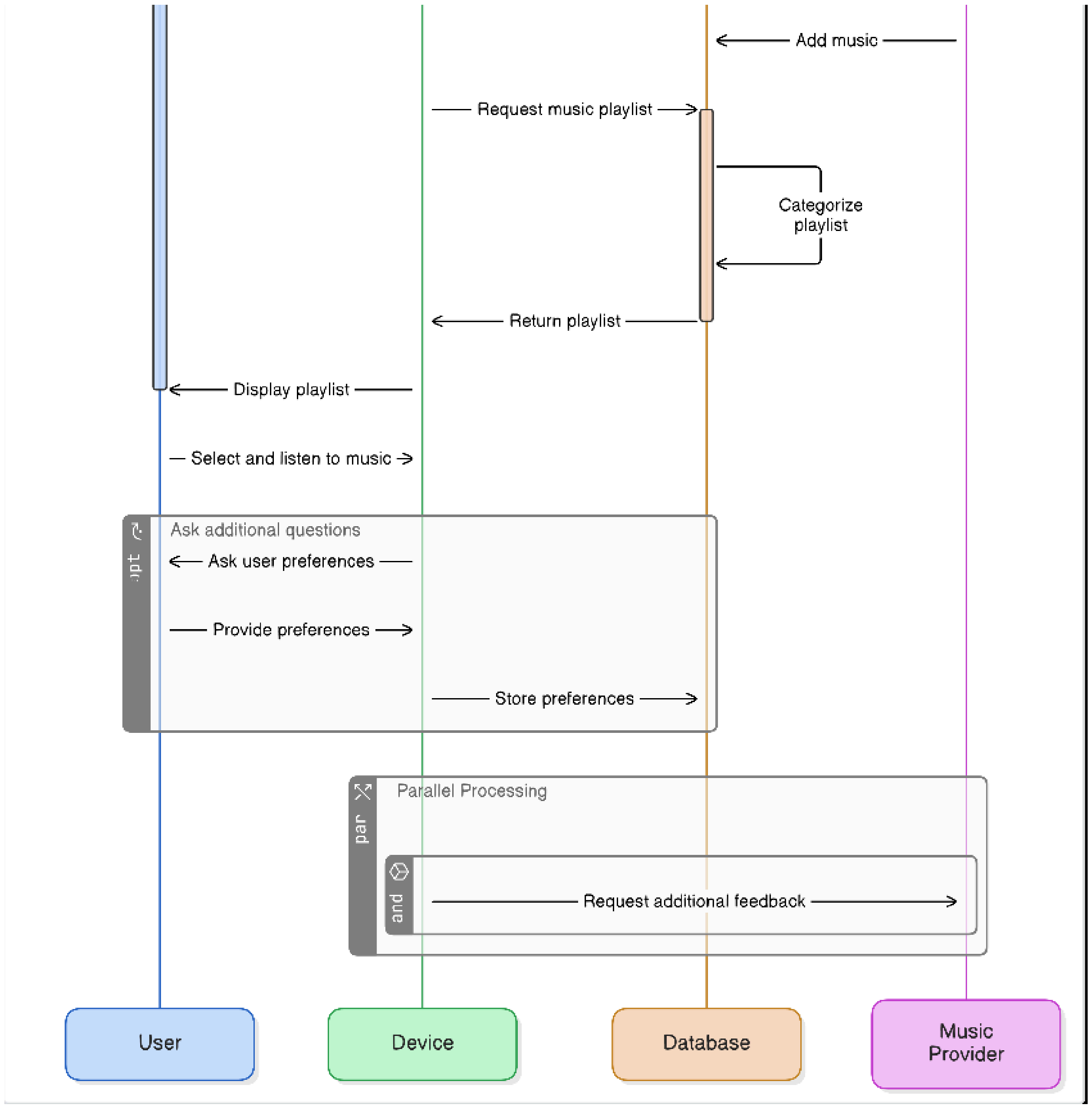
- ➔ Attributes:
  - ◆ +Id, +Title, +Language, +Owner, +Singer, +Type
- ➔ Description:
  - ◆ The music will be played according to the user's emotions.
- ➔ Relationship:
  - ◆ Each song can be performed by multiple singers (0..n relationship).
  - ◆ Songs can be owned by a content provider (0..1 relationship).

4. Content Provider Class

- ➔ Attributes:
  - ◆ +Id, +Name, +Country, +Songs
- ➔ Methods:
  - ◆ +uploadSong(), +removeSong()
- ➔ Relationship:
  - ◆ The content provider owns multiple songs (0..n relationship).
  - ◆ The content provider can maintain profiles for analysis.

Sequence Diagram:-





1. User Interaction:
    - The user starts by opening the application.
2. Device Interaction:
    - The device accesses the webcam.
    - The device captures the user's image.
    - Facial recognition is used to detect the user's face.
3. Database Interaction:
    - The device requests mood information from the database.
    - The database analyzes the user's facial expressions.

4. Returning Results:
    - The database returns the mood information to the device.
5. Conditional Logic:
    - If a mood is detected, the application displays the detected mood (e.g., happy mood).
    - If no mood is detected, an error message is displayed.
6. Retry Mechanism:
    - If face detection fails, the system retries until successful, repeating the steps to request and return mood information.
7. Music Playlist:
    - The device requests a music playlist from the database based on the detected mood.
    - The database categorizes and returns the playlist.
8. User Interaction:
    - The playlist is displayed to the user.
    - The user selects and listens to music.
9. User Preferences:
    - The application may ask the user additional questions to refine preferences.
    - The user provides preferences, which are then stored in the database.
10. Parallel Processing:
    - The system can process multiple tasks simultaneously, such as requesting additional feedback while interacting with the user.

DFD (Data Flow Diagram):

1. Definition:

- A Data Flow Diagram (DFD) is a graphical representation of the flow of data within a system.
- It visually illustrates how data moves through processes, external entities, and data stores in a system.

2. Purpose:

- Helps in understanding the flow of information in the system.
- Aids in identifying inefficiencies, redundancies, and potential improvements in the system.
- Used to model both physical and logical data flow within a system.

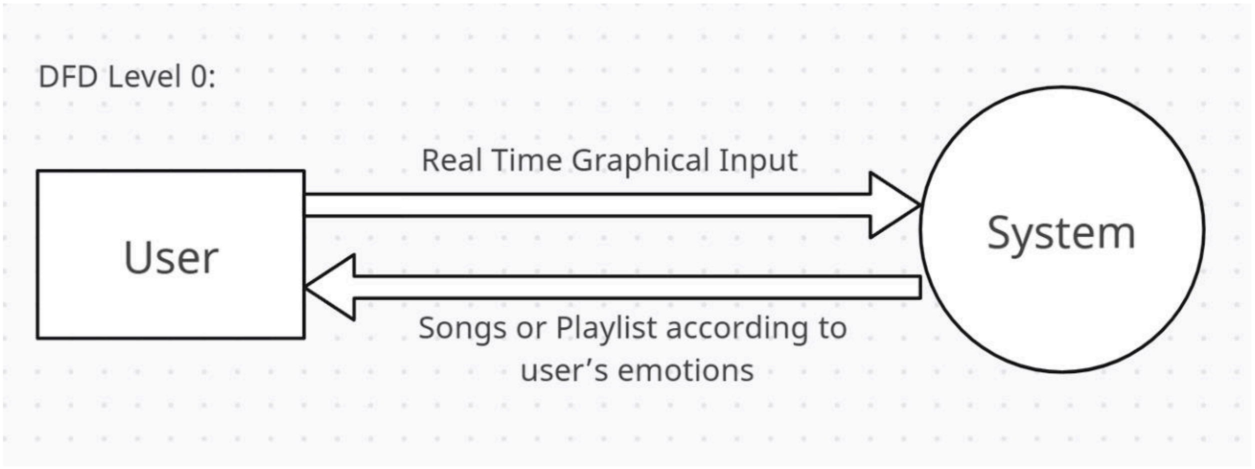
3. Levels of DFD:

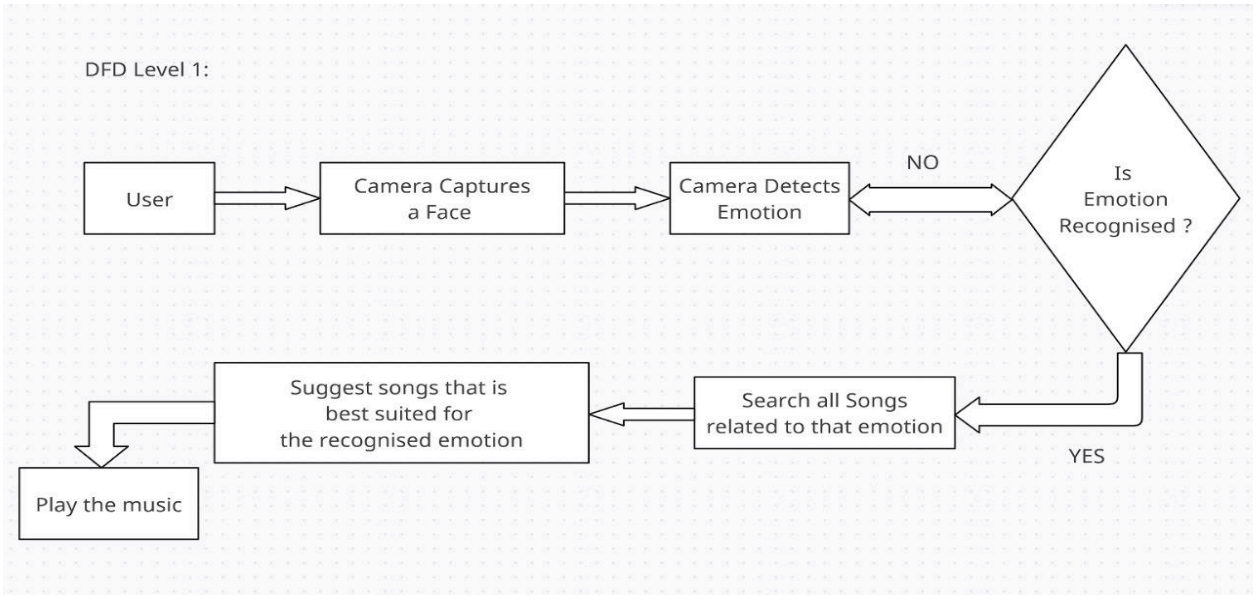
- **Level 0 DFD (Context Diagram)**:
    - Represents the entire system as a single process.
    - Shows the interactions between external entities and the system.
- **Level 1 DFD**:

    - Breaks down the main process from the context diagram into sub-processes.
    - Provides more detail about data flow within the system.
- **Level 2 (and beyond)**:
    - Further decomposes the processes into finer levels of detail.
    - Each level provides more granularity about data flow and system structure.

4. Key Components:

- **Processes**: Represented by circles or ovals, they indicate how data is transformed or processed.
- **Data Stores**: Represented by open-ended rectangles, they show where data is stored within the system.
- **Data Flow**: Represented by arrows, they show the direction of data movement between processes, entities, and data stores.
- **External Entities**: Represented by rectangles, they show entities outside the system that interact with it.


DFD Level 0:
Real Time Graphical Input
User
System
Songs or Playlist according to user's emotions



Conclusion:

In this experiment, we created UML diagrams to model a Music Recommendation System based on facial expressions. The Use Case Diagram clarified system interactions and functional requirements, while the Class Diagram depicted the static structure, showing key entities and their relationships. The Sequence Diagram illustrated the dynamic behavior, highlighting the flow of interactions from facial recognition to music recommendation. These diagrams collectively provided a comprehensive understanding of the system's design, functionality, and user interactions. They serve as a valuable tool for guiding development, ensuring clear communication, and supporting system maintenance and enhancements.