JOHANNES KEPLER
UNIVERSITÄT LINZ

Engineering of AI-Intensive Systems

SS 2025

Music Recommendation App: Brewtune

Team members:

- David Hauser, K01552420
- Lyna Hocine , K12449045

Table of Contents:

1. Goal of our system :

- **Non-AI goals :**

  - User Authentication: Allow users to sign up and log into their accounts.
  - Playlist Sharing: Allow users to share playlists via link.
  - Music control: Allow users to play, pause, skip and shuffle songs.
  - Interaction: Allow users to rate playlists and give feedback.
  - Dark/Light Mode: Provide theme customization.

- **AI goals :**

  Song recommendations based on a textual description of a mood, emotions, genre or style.

  - Justification for use of AI: Using an AI model is necessary because it can semantically understand natural language inputs, allowing it to recommend songs that closely match the user's text input, thereby supporting the user in music discovery.

2. Identifying the stakeholders :

- Listeners who want to discover new music (users)
- Music artists & producers
- Course supervisors
- Developers (us)

3. Documenting the requirements :

3.1. Functional Requirements :

- Req001: When the user inputs a text description of a mood, emotions, genre or style, then the system shall recommend songs based on this description.
- Req002: The system shall allow users to save recommended songs to a playlist.
- Req003: The system shall allow users to create a new playlist.
- Req004: The system shall allow users to edit or delete the playlists.
- Req005: The system shall allow users to reorder songs within the playlist.
- Req006: The system shall allow users to play, pause and skip songs.
- Req007: The system shall allow users to shuffle or play the playlist on loop
- Req008: The system shall allow users to share a playlist via a generated link.
- Req009: The system shall allow users to switch between light and dark mode.
- Req010: The system shall indicate how well the recommended songs fit to the users' text description.

3.2. Non-Functional Requirements :

External Interfaces:

- NfReq001: The system shall support multiple browsers including Firefox and Google Chrome.

Performance:

- NfReq003: The system shall generate playlists within 3 seconds.
- NfReq004: The system shall handle up to 1000 users at the same time.

Attributes:

- NfReq005: The system shall provide a responsive UI for both desktop and mobile devices.

- NfReq006: The system shall implement secure authentication.
- NfReq007: The system shall only recommend songs that have a cosine similarity of 0.5 or higher with the text description.

Constraints:

- NfReq008: The system shall use React for the frontend and Flask for the backend.
- NfReq009: The system shall store playlists in a database with a maximum size of 100MB per user.

Regulation:

- NfReq010: The system shall not violate any copyright laws.

3.3. Indicating AI-related requirements :

- Req001
  - Implementation: We plan to implement this by using the CLAP (Contrastive Language Audio Pretraining) model to compute the embedding vectors of audio files and store them in a vector database, before users use the system. When users input a text description we create the CLAP embedding of the text and use the cosine similarity between the input text and the stored audio embeddings to recommend the most similar songs.
- Req010
  - Implementation: We will display the cosine similarity between the text input of the user and the recommended songs, to indicate how well a recommended song matches to the users' input.
- NfReq007
  - The threshold of 0.5 was set, since we assume that any song with a lower similarity to the text will be significantly different and not fit to the users' input. However, we will need to verify and adjust the threshold once we have implemented the system.
  - Implementation: We will implement a threshold on the cosine similarity for recommending songs.
  - Category: Trust

4. Use cases Description :

4.1. Main Use Cases :

- **UC1 – Recommend Music (Implemented in the project):**

- ID : UC1
- Description: Recommends music based on textual description.
- Actors: User, Music Recommendation System.
- Stakeholders: Listeners, Music artists &Producers, Developers.
- Pre-Conditions: User is logged into the system.

- **UC2 – Create Playlist (Implemented in the project) :**

- ID : UC2
- Description: Allows users to create a new playlist.
- Actors: User.
- Stakeholders: Listeners, Developers.
- Pre-Conditions: User is logged into the system and has generated recommendations.

- **UC3 – Share Playlist (Implemented in the project) :**

- ID : UC3
- Description: Allows users to share a playlist via a link.
- Actors: User.
- Stakeholders: Listeners, Developers.
- Pre-Conditions: User has a playlist to share.

- **UC4 – Edit Playlist (Implemented in the project) :**

- ID : UC4
- Description: Allows users to edit or delete a playlist.
- Actors: User.
- Stakeholders: Listeners, Developers.
- Pre-Conditions: User has a playlist to edit or delete.

- **UC4 – Theme Customization (Implemented in the project) :**

- ID : UC5
- Description: Allows users to switch between light and dark mode.
- Actors: User.
- Stakeholders: Listeners, Developers.
- Pre-Conditions: User is logged into the system.

4.2. Supporting Use Cases :

- **SUC1 – Analyze Text Input (Implemented in the project) :**

- ID : SUC1
- Description: Analyzes the text input to determine mood, emotion, genre or style.
- Actors: Text AI Model.
- Stakeholders: Developers.
- Pre-Conditions: User has entered a text description.

- **SUC2 – Validate Playlist Name (Implemented in the project):**

- ID : SUC2
- Description: Validates the uniqueness of a playlist name.
- Actors: System.
- Stakeholders: Developers.
- Pre-Conditions: User has entered a playlist name.

- **SUC3 – Generate Shareable Link (Implemented in the project) :**

- ID : SUC3
- Description: Generates a shareable link for a playlist.
- Actors: System.
- Stakeholders: Developers.
- Pre-Conditions: User has selected a playlist to share.

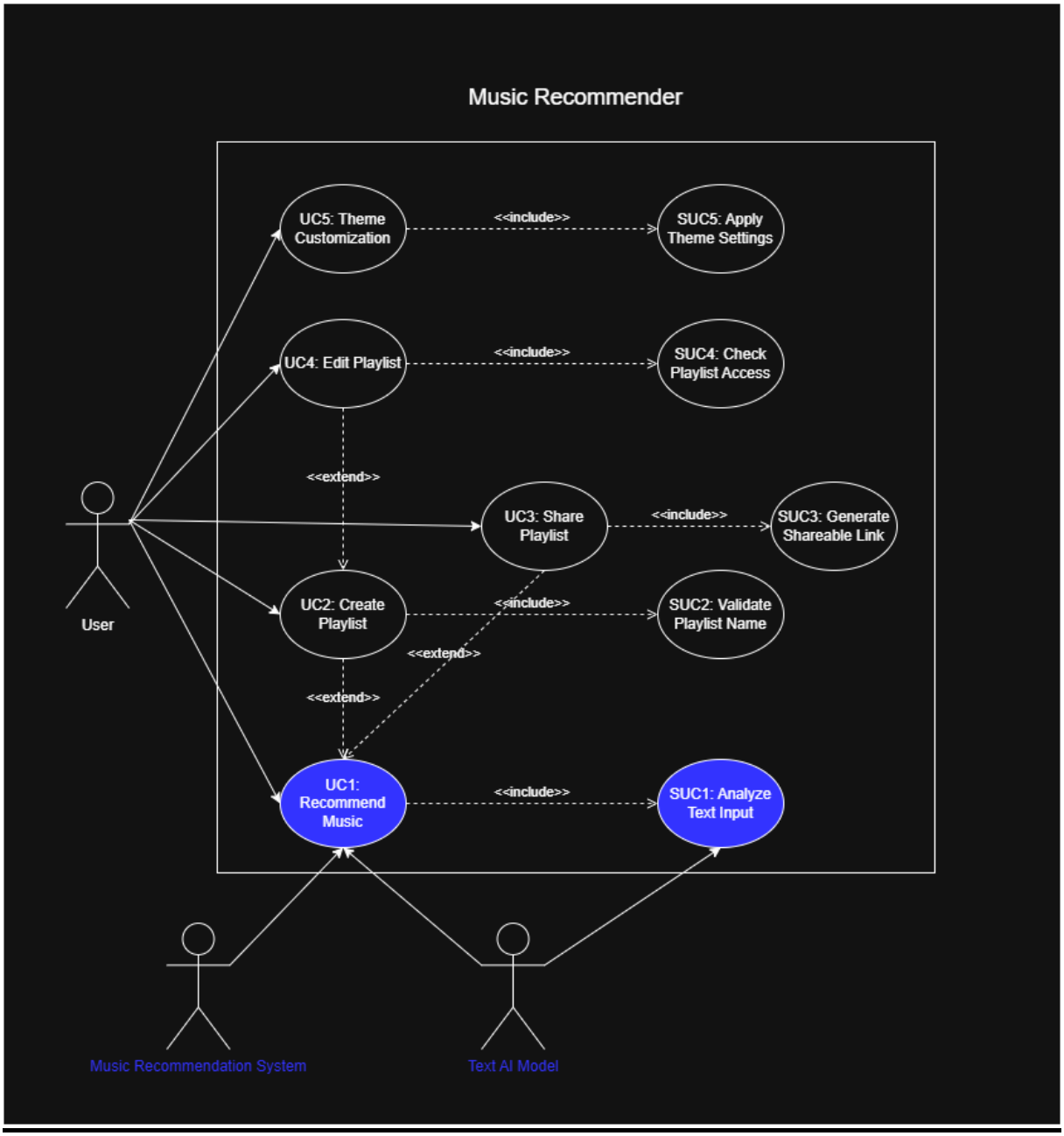
- **SUC4 – Check Playlist Access (Implemented in the project):**

- ID : SUC4
- Description: Checks if the user trying to edit a playlist is the creator of the playlist.
- Actors: User, System.
- Stakeholders: Developers.
- Pre-Conditions: User has selected a playlist to edit.

- **UC5 – Apply Theme Settings (Implemented in the project):**

- ID : SUC5
- Description: Applies the selected theme settings.
- Actors: System.
- Stakeholders: Developers.
- Pre-Conditions: User has selected a theme.

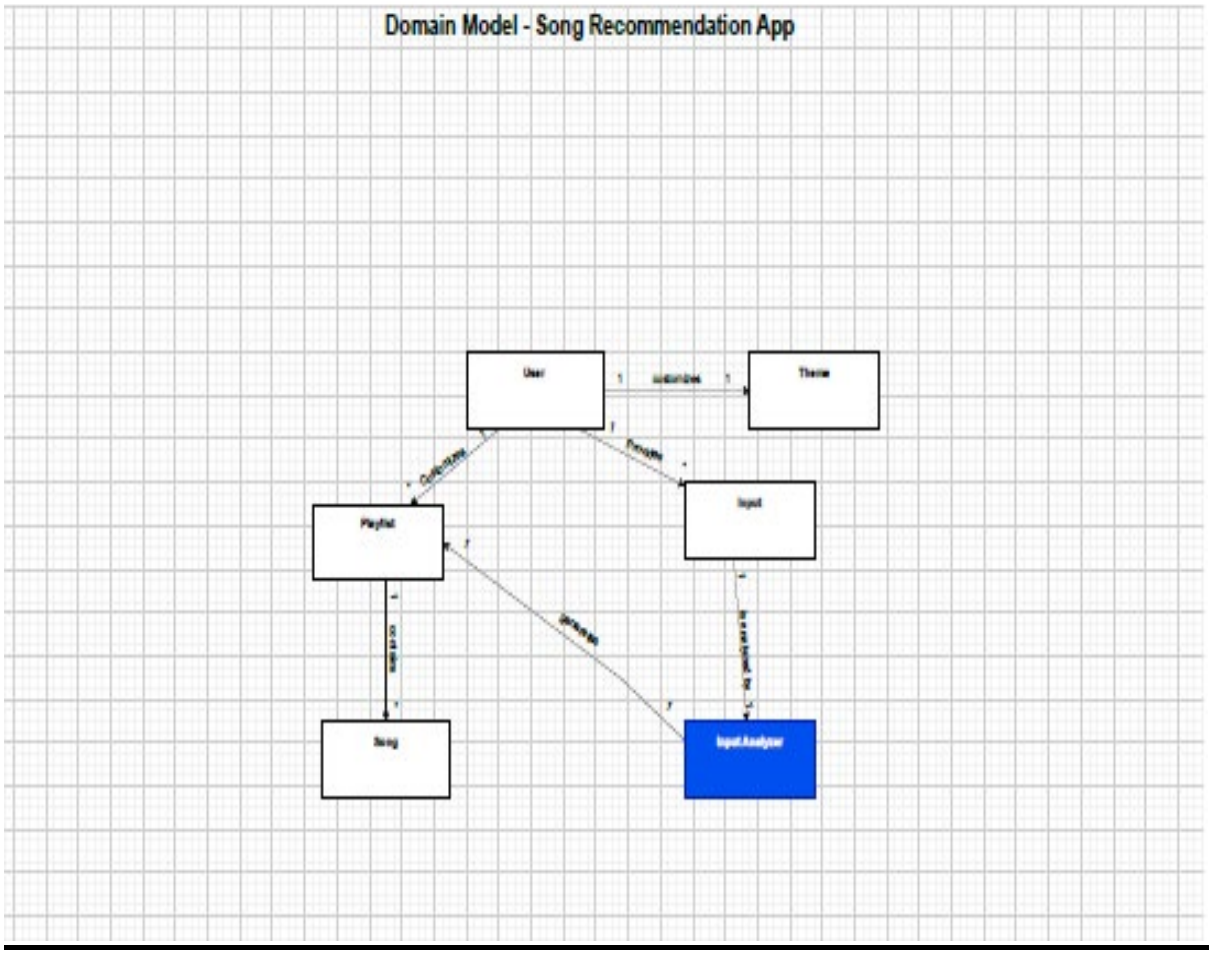
5. Use Case Diagram :



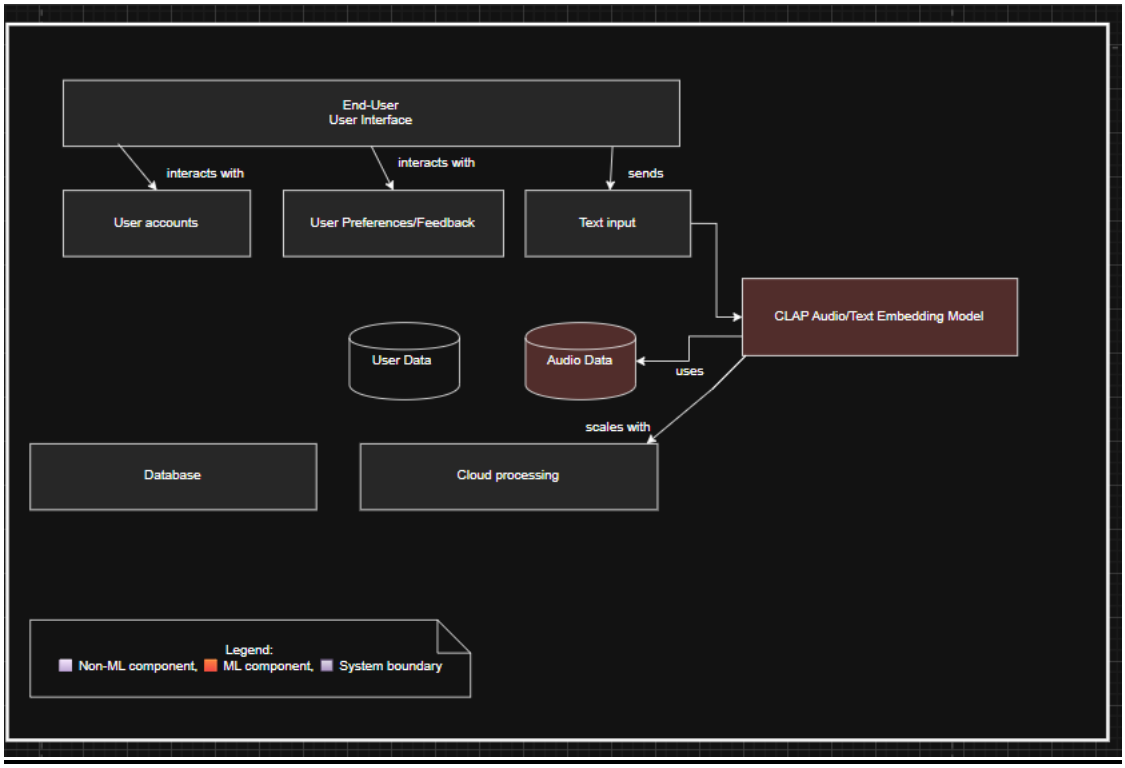
6. Traceability Matrix :

Use cases	UC1	UC2	UC3	UC4	UC5	SUC1	SUC2	SUC3	SUC4	SUC5
Requirements										
Req001	☑	☐	☐	☐	☐	☑	☐	☐	☐	☐
Req002	☐	☑	☐	☑	☐	☐	☑	☐	☑	☐
Req003	☐	☑	☐	☐	☐	☐	☑	☐	☐	☐
Req004	☐	☐	☐	☑	☐	☐	☑	☐	☑	☐
Req005	☐	☐	☐	☑	☐	☐	☐	☐	☑	☐
Req006	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐
Req007	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐
Req008	☐	☐	☑	☐	☐	☐	☐	☑	☑	☐
Req009	☐	☐	☐	☐	☑	☐	☐	☐	☐	☑
Req010	☑	☐	☐	☐	☐	☑	☐	☐	☐	☐
NfReq001	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
NfReq002	☑	☑	☐	☑	☐	☐	☐	☐	☐	☑
NfReq003	☑	☐	☐	☐	☐	☑	☑	☐	☑	☑
NfReq004	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
NfReq005	☑	☑	☑	☑	☐	☐	☑	☑	☑	☐
NfReq006	☑	☐	☐	☐	☐	☑	☐	☐	☐	☐
NfReq007	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
NfReq008	☐	☑	☑	☑	☐	☐	☑	☑	☑	☐
NfReq009	☑	☐	☐	☐	☐	☐	☐	☑	☑	☐

7. Domain Model :



8. Architecture Diagram :



9. Components Description :

- End-user interface: allows user to input text, view the playlist generated and control music playback. The requirements related to this component: Req001, Req002, Req003, Req006, Req007, Req009, Req010, and NfReq005.

- User accounts: deals with authentication and stores the user's preferences. The requirements related to this component: NfReq006 and NfReq009.

- User Preferences/Feedback: captures the likes/dislikes and the edits that the user makes to give precise recommendations in the future. The requirements related to this component: Req004 and Req005.

- Text input: collects text prompts from users' input. The requirements related to this component: Req001 and Req010.

- User Data: stores user profile, listening history and feedback to enable personalized music recommendations. The requirements related to this component: Req001, Req002, Req003, Req004, NfReq006 and NfReq009.

- Database: Stores user data, logins and playlists. The requirements related to this component: Req002, Req003, Req004, Req005, NfReq009 and NfReq006.

- Cloud processing: Handles backend API makes the communication between the frontend and the ML components easier. The requirements related to this component: Req001, Req008, NfReq003, NfReq004 and NfReq008.

- CLAP audio/text Embedding Model: Processes user input and returns a playlist of songs. The requirements related to this component: Req001, Req010, NfReq007 and NfReq008.

- Audio Data: Stores the audio database used to train the recommendation model. The requirements related to this component: Req001 and Req010.

10. Design questions and their answers :

- How can we ensure that user data is protected while still personalizing the music recommendations?

Answer: We ensure user data is protected by collecting only necessary information, using secure authentication and giving users control over their data. This allows us to personalize music recommendations without compromising privacy.

- How can we scale our recommendation system to support thousands of users without degrading its performance?

Answer: We can scale the recommendation system by using load balancing, caching frequent queries and deploying the model through a scalable cloud infrastructure.

- How do we ensure that the music recommendation app work seamlessly across multiple platforms?

Answer: We ensure the app works seamlessly across platforms by building a responsive web interface using React, testing on multiple browsers and devices, and following web standards.

- What should we do if a recommendation system fails?

Answer: If the recommendation system fails, we fall back to default playlists, show a user-friendly message, and log the issue for debugging. We also monitor system performance to quickly detect any problems.

- What metrics can we use to monitor the effectiveness of our recommendation system in real time?

Answer: We can monitor effectiveness using metrics like click-through rate, playlist save rate, skip rate and user feedback.

- How can we design the user interface to help users easily provide feedback on recommendations?

Answer: We can design the UI with simple feedback options like thumbs up/down, a quick emoji or star rating. We can place these near the songs to make it easy for users to give input without interrupting their listening experience.

- How do we prevent overloading the recommendation system with irrelevant data?

Answer: We prevent overloading by filtering out low quality or unrelated inputs, using data validation, and focusing only on relevant features like mood and genre.

- What should be the approach to limit data storage without compromising the effectiveness of personalized recommendations?

Answer: To limit data storage while maintaining effective recommendations, we can store only essential data like user preferences, playlist history and interactions metrics. We can also rely on real-time processing for dynamic recommendations instead of storing large datasets.

- Can our system adapt to live listening feedback (skipping a song immediately) in real time?

Answer: Yes, by tracking real-time actions and using this data to adjust the recommendations.

- Could our logs or models accidentally reveal private user preferences?

Answer: Yes if the sensitive data is not properly anonymized or stored securely. To prevent this, we can ensure that all personal data is anonymized and store only necessary information.

- What is the UX when there is no recommendation data for a specific request?

Answer: The UX should handle the situation by displaying a message like “No recommendations found for this request” or “try broadening your request “. It should then show alternative recommendations like popular songs.

- How do we detect recommendation loops, where the system keeps suggesting the same few songs?

Answer: We can track the diversity of suggested songs and monitoring user interaction patterns. If a user skips or ignores the same songs repeatedly, the system should log this behaviour and adjust the recommendations to introduce more variety.