Module Interface Specification for Software Engineering

Team 8 – Rhythm Rangers

Ansel Chen
Muhammad Jawad
Mohamad-Hassan Bahsoun
Matthew Baleanu
Ahmed Al-Hayali

January 18, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Software Engineering.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	GUI Module Audio File Input Module Search Query Module Client Communication Module Server Communication Module Driver Module Tempo (BPM) Feature Extraction Module Key and Scale Feature Extraction Module Instrument Type Feature Extraction Module Vocal Gender Feature Extraction Module Dynamic Range Feature Extraction Module Instrumentalness Feature Extraction Module Contour Feature Extraction Module Mood Feature Extraction Module Recommendation Module Program Results Interface
Software Decision	Database Spotify API Deezer API Genre Feature Module

Table 1: Module Hierarchy

6 GUI Module

6.1 GUI Module

gui

6.2 Uses

- Audio File Input Module
- Search Query Module
- Spotify API Module

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
gui	N/A	N/A	-

6.4 Semantics

6.4.1 State Variables

- user_selection: Stores the track or audio file chosen by the user
- spotify_results: Stores the top 10 songs that best fit the search query
- recommendations: Stores the list of the recommended songs after feature extraction

6.4.2 Environment Variables

- Keyboard
- Mouse
- Screen

6.4.3 Assumptions

- User inputs are valid

6.4.4 Access Routine Semantics

gui

- transition: provides methods to build and deploy the GUI to the user

6.4.5 Local Functions

N/A

7 MIS of Audio File Input Module

7.1 Audio File Input Module

audioFileIM

7.2 Uses

- GUI Module
- Client Communication Module

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
audioFileIM	Audio File	Track reference	Invalid File Type

7.4 Semantics

7.4.1 State Variables

- user_af_input: path to the audio file currently being processed

7.4.2 Environment Variables

N/A

7.4.3 Assumptions

- User has a properly named Audio File.
- User audio file input is actually a song.

7.4.4 Access Routine Semantics

audioFileIM

- transition: if the provided file is not in the .wav, then after it is converted, the file is sent to the Client Communication Module

7.4.5 Local Functions

N/A

8 MIS of Search Query Module

8.1 Search Query Module

searchQuery

8.2 Uses

- GUI Module
- Client Communication Module

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
searchQuery	input_text	Spotify Query	-

8.4 Semantics

8.4.1 State Variables

- user_sq_input: stores the query being processed

8.4.2 Environment Variables

- Spotify Client ID
- Spotify Client Secret

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

searchQuery

- transition: Takes the text input and/or Spotify ID from the GUI Module, and builds the query to be sent to the Client Communication Module

8.4.5 Local Functions

N/A

9 MIS of Client Communication Module

9.1 Client Communication Module

The module that sends request to and receives responses from the server

9.2 Uses

- Audio File Input Module
- Search Query Module
- Server Communication Module

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
send_request	request (ADT)	-	-
await_response	-	response (ADT)	-

9.4 Semantics

9.4.1 State Variables

N/A

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

N/A

9.4.4 Access Routine Semantics

send_request():

- transition: sends the request to the server, where it is received by the server communication module

await_response():

- output: gets the response from the server communication module and sends it to the Program Results Interface Module

9.4.5 Local Functions

N/A

10 MIS of Server Communication Module

10.1 Server Communication Module

Sends requests to the server and receives responses from the server

10.2 Uses

- Server Driver Module
- Client Communication Module

10.3 Syntax

10.3.1 Exported Constants

N/A

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
send_response	response (ADT)	-	-
await_request	-	request (ADT)	-

10.4 Semantics

10.4.1 State Variables

N/A

10.4.2 Environment Variables

N/A

10.4.3 Assumptions

N/A

10.4.4 Access Routine Semantics

send_response():

- transition: sends the response to the client, where it is received by the Client Communication module

await_request():

- output: gets the request from the Client Communication module and sends it to the Server Driver Module

10.4.5 Local Functions

N/A

11 MIS of Driver Module

11.1 Driver Module

Controls all the functions of the server

11.2 Uses

- Featurizer Module
- Server Communication Module
- Database Module
- Recommendation Module
- Deezer API Module

11.3 Syntax

11.3.1 Exported Constants

N/A

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
-	-	-	-

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Environment Variables

- Deezer App ID
- Deezer Secret

11.4.3 Assumptions

N/A

11.4.4 Access Routine Semantics

main():

- transition: Connects all server-side modules together

11.4.5 Local Functions

N/A

12 MIS of Audio Lookup Module

12.1 Module

Audio Lookup Module

12.2 Uses

- Driver Module: Receives the International Standard Recording Code (ISRC) from the Driver Module. - Deezer API: Responsible for retrieving the audio file, genre, and associated metadata for the provided ISRC.

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
getAudioDetails	isrc: String	audioDetails: AudioDetails	AuthenticationFailure, APIRequestError

12.4 Semantics

12.4.1 State Variables

- `isrc`: The International Standard Recording Code for identifying the requested song. - `authToken`: The authentication token used for accessing the Deezer API. - `audioDetails`: A structure containing the audio file, genre, and other metadata.

12.4.2 Environment Variables

- The Audio Lookup Module interacts with the Deezer API over the internet to fetch the requested audio file, genre, and metadata.

12.4.3 Assumptions

- The ISRC provided by the Driver Module is valid and corresponds to an existing song. - The authentication token for the Deezer API is valid and not expired. - The Deezer API is available and operational at the time of the request.

12.4.4 Access Routine Semantics

getAudioDetails(isrc: String):

- **Transition:** - Authenticates with the Deezer API using `authToken`. - Sends a request to the Deezer API with the provided ISRC to retrieve the audio file, genre, and metadata.
- **Output:** - Returns the `audioDetails` structure, which includes:
  - `audioFile`: The retrieved audio file.
  - `genre`: The genre of the song.
  - `metadata`: Additional metadata such as song title, artist, and album information.
- **Exceptions:** - `AuthenticationFailure`: Raised if the API authentication fails (e.g., invalid or expired token). - `APIRequestError`: Raised if there is an issue with the API request, such as a network error or invalid ISRC.

12.4.5 Local Functions

authenticateWithDeezer:

- Purpose: Handles authentication with the Deezer API and retrieves a valid `authToken`.
- Input: None.
- Output: `authToken`.

fetchAudioFile:

- Purpose: Sends the ISRC to the Deezer API and retrieves the corresponding audio file.
- Input: `isrc`.
- Output: `audioFile`.

fetchGenreAndMetadata:

- Purpose: Retrieves the genre and metadata associated with the song from the Deezer API.
- Input: `isrc`.
- Output: `genre`, `metadata`.

13 MIS of Featurizer Module

13.1 Featurizer Module

The Featurizer Module is responsible for extracting 9 distinct feature values from audio files:

- Tempo
- Key and Scale
- Instrument Type
- Vocal Gender
- Dynamic Range
- Instrumentalness
- Contour
- Mood
- Genre

The module invokes sub-feature modules to compute these feature values. It consolidates the results into a single `FeatureValues` object and returns it to the Driver Module.

13.2 Uses

- **Driver Module**: Sends requests to the Featurizer Module and receives feature values.
- **Sub-Feature Modules**: Each responsible for computing a specific feature (e.g., Tempo, Key and Scale).

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
extractFeatures	audioFile: AudioFile	featureValues: FeatureValues	UnsupportedFileFormatException

13.4 Semantics

13.4.1 State Variables

- `audioFile`: The input audio file provided for feature extraction. - `featureValues`: An object containing the extracted values for all 9 features.

13.4.2 Environment Variables

None.

13.4.3 Assumptions

- Input audio files are in supported formats (e.g., WAV, MP3). - All sub-feature modules are functional and return valid outputs for their respective features.

13.4.4 Access Routine Semantics

extractFeatures:

- **Precondition:**
  - `audioFile` is a valid audio file in a supported format.
- **Postcondition:**
  - `featureValues` contains valid results for all 9 features:
    - Tempo
    - Key and Scale
    - Instrument Type
    - Vocal Gender
    - Dynamic Range
    - Instrumentalness
    - Contour
    - Mood
    - Genre
  - If the input file format is unsupported, an `UnsupportedFileFormatException` is raised.

13.4.5 Local Functions

invokeSubFeatureModule:

- Purpose: Calls a specific sub-feature module (e.g., for Tempo, Genre) and retrieves its computed value.

- Input: `audioFile`, `featureType`
- Output: Value of the requested feature.

aggregateFeatureValues:

- Purpose: Consolidates all feature values into a `FeatureValues` object.
- Input: A list of feature values retrieved from sub-feature modules.
- Output: `FeatureValues` object.

14 MIS of Tempo (BPM) Feature Extraction Module

14.1 Tempo (BPM) Feature Extraction Module

14.2 Uses

N/A

14.3 Syntax

14.3.1 Exported Constants

N/A

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
`Extract Tempo`	`Audio time series (np.ndarray)`	`Song Tempo` $\in \mathbb{R}$	-

14.4 Semantics

14.4.1 State Variables

N/A

14.4.2 Environment Variables

N/A

14.4.3 Assumptions

Valid audio file with coherent song information.

14.4.4 Access Routine Semantics

`ExtractTempo()`:

- transition: N/A
- output: `Song_Tempo` : = `ExtractTempo(Audio_Time_Series)`
- exception: N/A

14.4.5 Local Functions

N/A

15 MIS of Key and Scale Feature Extraction Module

15.1 Key and Scale Feature Extraction Module

15.2 Uses

N/A

15.3 Syntax

15.3.1 Exported Constants

N/A

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
`Extract Key & Scale`	`Audio time series (np.ndarray)`	`Song Key, Scale` $\in \mathbb{Z}^2$	-

15.4 Semantics

15.4.1 State Variables

N/A

15.4.2 Environment Variables

N/A

15.4.3 Assumptions

Valid audio file with coherent song information.

15.4.4 Access Routine Semantics

`ExtractKeyScale()`:

- transition: N/A
- output: `Song_Key, Song_Scale` : = `ExtractKeyScale(Audio_Time_Series)`
- exception: N/A

15.4.5 Local Functions

N/A

16 MIS of Instrument Type Feature Extraction Module

16.1 Instrument Type Feature Extraction Module

16.2 Uses

N/A

16.3 Syntax

16.3.1 Exported Constants

N/A

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
`Extract Instrument Type`	`Audio time series (np.ndarray)`	`Instrument Type` $\in \mathbb{Z}^k$	-

16.4 Semantics

16.4.1 State Variables

N/A

16.4.2 Environment Variables

N/A

16.4.3 Assumptions

Valid audio file with coherent song information.

16.4.4 Access Routine Semantics

`ExtractInstrumentType()`:

- transition: N/A
- output: `Instrument_Type` : = `ExtractInstrumentType(Audio_Time_Series)`
- exception: N/A

16.4.5 Local Functions

N/A

17 MIS of Vocal Gender Feature Extraction Module

17.1 MIS of Vocal Gender Feature Extraction Module

This feature seeks to quantify whether the voices features in the inputted audio file are largely more feminine or masculine sounding. This is represented by a float with a range between 0 and 1 where 0 means only "masculine" sound signatures are contained and 1 means only "feminine" sounds, where values in-between represent a blend.

17.2 Uses

N/A

17.3 Syntax

17.3.1 Exported Constants

N/A

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
`Extract Vocal Gender`	`Audio time series (np.ndarray)`	`Vocal Gender` $\in \mathbb{R}$	-

17.4 Semantics

17.4.1 State Variables

N/A

17.4.2 Environment Variables

N/A

17.4.3 Assumptions

Valid audio file with coherent song information.

17.4.4 Access Routine Semantics

`ExtractVocalGender()`:

- transition: N/A
- output: `Vocal_Gender` : = `ExtractVocalGender(Audio_Time_Series)`
- exception: N/A

17.4.5 Local Functions

N/A

18 MIS of Dynamic Range Feature Extraction Module

18.1 Dynamic Range Feature Extraction Module

Feature extracts the range of sounds (difference between peak and through) of the audio signal.

18.2 Uses

N/A

18.3 Syntax

18.3.1 Exported Constants

N/A

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
Extract Dynamic Range	Audio time series (np.ndarray)	Dynamic Range (decibels) $\in \mathbb{R}$	-

18.4 Semantics

18.4.1 State Variables

N/A

18.4.2 Environment Variables

N/A

18.4.3 Assumptions

Valid audio file with coherent song information.

18.4.4 Access Routine Semantics

ExtractDynamicRange():

- transition: N/A
- output: Dynamic_Range : = ExtractDynamicRange(Audio_Time_Series)
- exception: N/A

18.4.5 Local Functions

N/A

19 MIS of Instrumentalness Feature Extraction Module

19.1 Instrumentalness Feature Extraction Module

Extracts the how prominent instrumental sounds are within the song. Represented by a float variable where the range is between 0 and 1, where higher values mean more instrumental sounds and lower means less. Eg, 0 would mean an acapella piece of music, 1 would be something that purely features instruments.

19.2 Uses

N/A

19.3 Syntax

19.3.1 Exported Constants

N/A

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
Extract Instrumentalness	Audio time series (np.ndarray)	Instrumentalness $\in \mathbb{R}$	-

19.4 Semantics

19.4.1 State Variables

N/A

19.4.2 Environment Variables

N/A

19.4.3 Assumptions

Valid audio file with coherent song information.

19.4.4 Access Routine Semantics

ExtractInstrumentalness():

- transition: N/A
- output: Instrumentalness : = ExtractInstrumentalness(Audio_Time_Series)
- exception: N/A

19.4.5 Local Functions

N/A

20 MIS of Contour Feature Extraction Module

20.1 Contour Feature Extraction Module

20.2 Uses

N/A

20.3 Syntax

20.3.1 Exported Constants

N/A

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
`Extract Melodic Contour`	`Audio time series (np.ndarray)`	`Contour`	-

20.4 Semantics

20.4.1 State Variables

N/A

20.4.2 Environment Variables

N/A

20.4.3 Assumptions

Valid audio file with coherent song information.

20.4.4 Access Routine Semantics

`ExtractMelodicContour()`:

- transition: N/A
- output: `Contour` : = `ExtractMelodicContour(Audio_Time_Series)`
- exception: N/A

20.4.5 Local Functions

N/A

21 MIS of Mood Feature Extraction Module

21.1 Mood Feature Extraction Module

21.2 Uses

N/A

21.3 Syntax

21.3.1 Exported Constants

N/A

21.3.2 Exported Access Programs

Name	In	Out	Exceptions
`Extract Mood`	`Audio time series (np.ndarray)`	`Mood` $\in \mathbb{Z}$	-

21.4 Semantics

21.4.1 State Variables

N/A

21.4.2 Environment Variables

N/A

21.4.3 Assumptions

Valid audio file with coherent song information.

21.4.4 Access Routine Semantics

`ExtractMood()`:

- transition: N/A
- output: `Mood` : = `ExtractMood(Audio_Time_Series)`
- exception: N/A

21.4.5 Local Functions

N/A

22 MIS of Genre Feature Extraction Module

22.1 Module

Genre Feature Extraction Module

22.2 Uses

- Featurizer Module: Receives metadata from the Featurizer Module and extracts the genre attribute from it. - Metadata Structure: Utilizes the metadata structure to locate and retrieve the genre attribute.

22.3 Syntax

22.3.1 Exported Constants

None.

22.3.2 Exported Access Programs

Name	In	Out	Exceptions
extractGenre	metadata: Metadata	genre: String	MissingGenreException, InvalidMetadataException

22.4 Semantics

22.4.1 State Variables

- `metadata`: The metadata provided by the Featurizer Module, which contains the genre attribute.

22.4.2 Environment Variables

None.

22.4.3 Assumptions

- The metadata provided by the Featurizer Module is valid and includes the genre attribute.
- The genre attribute in the metadata is correctly formatted and accessible.

22.4.4 Access Routine Semantics

extractGenre(metadata: Metadata):

- **Transition:** - Extracts the genre attribute from the provided metadata.
- **Output:** - Returns the extracted genre as a string.
- **Exceptions:** - `MissingGenreException`: Raised if the genre attribute is not found in the metadata. - `InvalidMetadataException`: Raised if the provided metadata is improperly formatted or invalid.

22.4.5 Local Functions

validateMetadata:

- Purpose: Ensures the provided metadata is valid and contains the necessary attributes.
- Input: `metadata`.
- Output: Boolean (true if valid, false otherwise).

retrieveGenre:

- Purpose: Locates and retrieves the genre attribute from the metadata.
- Input: `metadata`.
- Output: `genre` (String).

23 MIS of Recommendation Module

23.1 Recommendation Module

23.2 Uses

- Tempo (BPM) Feature Extraction Module
- Key and Scale Feature Extraction Module
- Instrument Type Feature Extraction Module
- Vocal Gender Feature Extraction Module

- Dynamic Range Feature Extraction Module
- Instrumentalness Feature Extraction Module
- Contour Feature Extraction Module
- Mood Feature Extraction Module
- Driver Module
- Spotify API

23.3 Syntax

23.3.1 Exported Constants

N/A

23.3.2 Exported Access Programs

Name	In	Out	Exceptions
`Generate Recs`	`Song_Features (np.ndarray ∈ Feature)`	`Rec_Tracks np.ndarray ∈ Track`	-

23.4 Semantics

23.4.1 State Variables

N/A

23.4.2 Environment Variables

N/A

23.4.3 Assumptions

N/A

23.4.4 Access Routine Semantics

GenerateRecommendations():

- transition: N/A
- output: `Recommended_Songs` : = `GenerateRecommendations(Song_Features)`
- exception: N/A

23.4.5 Local Functions

N/A

24 MIS of Program Results Interface Module

24.1 Program Results Interface Module

24.2 Uses

- Spotify API

24.3 Syntax

24.3.1 Exported Constants

N/A

24.3.2 Exported Access Programs

Name	In	Out	Exceptions
`Generate Spotify Embed`	`Rec_Track (np.ndarray ∈ Track)`	`Tracks_Embed` (Spotify Embed Element)	-
`Display Features`	`Song Features (np.ndarray ∈ Feature)`	`Features_Display` (UI Image)	-

24.4 Semantics

24.4.1 State Variables

N/A

24.4.2 Environment Variables

N/A

24.4.3 Assumptions

N/A

24.4.4 Access Routine Semantics

`GenerateSpotifyEmbed()`:

- transition: N/A
- output: `Tracks_Embed_Widget`: = `GenerateSpotifyEmbed(Tracks)`
- exception: N/A

`DisplayFeatures()`:

- transition: N/A
- output: `Features_Display`: = `DisplayFeatures(Song_Features)`
- exception: N/A

24.4.5 Local Functions

N/A

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

25 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)