Module Interface Specification for Software Engineering

Team 8 – Rhythm Rangers

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

January 15, 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	User Input Interface
	First-Match Text Field Input Module
	URL Input Module
	Audio File Input Module
	Spotify Query Search & Select
	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
	Pre-Processing Module
	Spotify API
	Deezer API
	Genre Fetching

Table 1: Module Hierarchy

6 MIS of User Input Interface

6.1 User Input Interface

6.2 Uses

- First-Match Text Field Input Module
- URL Input module
- Audio File Input Module
- Spotify Query Search & Select

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
Consolidate Inputs	Up to 4 collection(s) of reference(s) to track(s)	Merged collection of track references	-

6.4 Semantics

6.4.1 State Variables

- Collection of track reference(s)

6.4.2 Environment Variables

N/A

6.4.3 Assumptions

N/A

6.4.4 Access Routine Semantics

consolidate_inputs():

- output: parses the user input and returns the songs that are sent to be processed

6.4.5 Local Functions

- parse_wav_file(file)

  –

- parse_url(url)

  –

- parse_text(text)

  –

7 MIS of First-Match Text Field Input Module

7.1 First-Match Text Field Input Module

The text field input is turned into a multi-line string where each line is a spotify search query and a reference is generated for the first match of every query.

7.2 Uses

N/A

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
Input Button Press	Text Field/Multi-line string	Collection of song references corresponding to each line in the input	N/A

7.4 Semantics

7.4.1 State Variables

- Collection of reference(s) to track(s)

7.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

- Spotify Client ID
- Spotify Client Secret

7.4.3 Assumptions

-

7.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

7.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

8 MIS of URL Input Module

8.1 URL Input Module

User inputs a URL and a reference is generated through the input.

8.2 Uses

N/A

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
URL Input Request	URL	track reference(s)	Invalid URL

8.4 Semantics

8.4.1 State Variables

- Collection of track reference(s)

8.4.2 Environment Variables

- Spotify Client ID
- Spotify Client Secret

8.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

8.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

8.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

9 MIS of Audio File Input Module

9.1 Audio File Input Module

User inputs an audio file to the system to analyze.

9.2 Uses

N/A

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
On Input Button Press	Audio File	Collection of song reference(s)	Invalid File Type

9.4 Semantics

9.4.1 State Variables

- Collection of track reference(s)

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

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

9.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

9.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

10 MIS of Spotify Query Search & Select Input Module

10.1 Spotify Query Search & Select Input Module

User inputs a song and that is turned into a spotify search query where the top 10 matches are available for user to select

10.2 Uses

N/A

10.3 Syntax

10.3.1 Exported Constants

N/A

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
Search Query Request	text input	top 10 matches from spotify query search	N/A
Output result selection	user selection	Collection containing track reference	N/A

10.4 Semantics

10.4.1 State Variables

- Collection containing track reference

10.4.2 Environment Variables

- Spotify Client ID
- Spotify Client Secret

10.4.3 Assumptions

N/A

10.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

10.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

11 MIS of Tempo (BPM) Feature Extraction Module

11.1 Tempo (BPM) Feature Extraction Module

11.2 Uses

N/A

11.3 Syntax

11.3.1 Exported Constants

N/A

11.3.2 Exported Access Programs

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

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Environment Variables

N/A

11.4.3 Assumptions

Valid audio file with coherent song information.

11.4.4 Access Routine Semantics

`ExtractTempo`():

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

11.4.5 Local Functions

N/A

12 MIS of Key and Scale Feature Extraction Module

12.1 Key and Scale Feature Extraction Module

12.2 Uses

N/A

12.3 Syntax

12.3.1 Exported Constants

N/A

12.3.2 Exported Access Programs

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

12.4 Semantics

12.4.1 State Variables

N/A

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

Valid audio file with coherent song information.

12.4.4 Access Routine Semantics

`Extract_Key_Scale`():

- transition: N/A
- output: `SongKey, SongScale` : = `Extract_Key_Scale(Audio_Time_Series)`
- exception: N/A

12.4.5 Local Functions

N/A

13 MIS of Instrument Type Feature Extraction Module

13.1 Instrument Type Feature Extraction Module

13.2 Uses

N/A

13.3 Syntax

13.3.1 Exported Constants

N/A

13.3.2 Exported Access Programs

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

13.4 Semantics

13.4.1 State Variables

N/A

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

Valid audio file with coherent song information.

13.4.4 Access Routine Semantics

`Extract_Instrument_Type`():

- transition: N/A

- output: `InstrumentType` : = `Extract_Instrument_Type(Audio_Time_Series)`
- exception: N/A

13.4.5 Local Functions

N/A

14 MIS of Vocal Gender Feature Extraction Module

14.1 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.

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 Vocal Gender	Audio time series (`np.ndarray`)	Vocal Gender $\in \mathbb{R}$	N/A

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

`Extract_Vocal_Gender()`:

- transition: N/A
- output: `VocalGender` := `Extract_Vocal_Gender(Audio_Time_Series)`
- exception: N/A

14.4.5 Local Functions

N/A

15 MIS of Dynamic Range Feature Extraction Module

15.1 Dynamic Range Feature Extraction Module

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

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 Dynamic Range	Audio time series (`np.ndarray`)	Dynamic Range (decibels) $\in \mathbb{R}$	N/A

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

`Extract_Dynamic_Range`():

- transition: N/A
- output: `DynamicRange` : = `Extract_Dynamic_Range(Audio_Time_Series)`
- exception: N/A

15.4.5 Local Functions

N/A

16 MIS of Instrumentalness Feature Extraction Module

16.1 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.

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 Instrumentalness	Audio time series (`np.ndarray`)	Instrumentalness $\in \mathbb{R}$	N/A

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

`Extract_Instrumentalness`():

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

16.4.5 Local Functions

N/A

17 MIS of Contour Feature Extraction Module

17.1 Contour Feature Extraction Module

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 Melodic Contour	Audio time series (`np.ndarray`)	output	N/A

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

`Extract_Melodic_Contour()`:

- transition: N/A
- output: `variable name` : = `Extract_Melodic_Contour(Audio_Time_Series)`
- exception: N/A

17.4.5 Local Functions

N/A

18 MIS of Mood Feature Extraction Module

18.1 Mood Feature Extraction Module

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 Mood	Audio time series (`np.ndarray`)	Mood $\in \mathbb{Z}$	N/A

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

`Extract_Mood()`:

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

18.4.5 Local Functions

N/A

19 MIS of Recommendation Module

19.1 Recommendation Module

19.2 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

19.3 Syntax

19.3.1 Exported Constants

N/A

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
Generate Recommendations	List of Song Features (`np.ndarray[np.ndarray]`)	output	N/A

19.4 Semantics

19.4.1 State Variables

N/A

19.4.2 Environment Variables

N/A

19.4.3 Assumptions

N/A

19.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

19.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

20 MIS of Program Results Interface Module

20.1 Program Results Interface Module

20.2 Uses

- Recommendation Module

20.3 Syntax

20.3.1 Exported Constants

N/A

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
Display Results	input	output	N/A

20.4 Semantics

20.4.1 State Variables

N/A

20.4.2 Environment Variables

N/A

20.4.3 Assumptions

N/A

20.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

20.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

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.

21 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)