

# Module Guide for Software Engineering

Team 8 – Rhythm Rangers

Ansel Chen

Muhammad Jawad

Mohamad-Hassan Bahsoun

Matthew Baleanu

Ahmed Al-Hayali

January 20, 2025

# 1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

## 2 Reference Material

This section records information for easy reference.

### 2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

# Contents

<b>1</b>	<b>Revision History</b>	<b>i</b>
<b>2</b>	<b>Reference Material</b>	<b>ii</b>
2.1	Abbreviations and Acronyms . . . . .	ii
<b>3</b>	<b>Introduction</b>	<b>1</b>
<b>4</b>	<b>Anticipated and Unlikely Changes</b>	<b>2</b>
4.1	Anticipated Changes . . . . .	2
4.2	Unlikely Changes . . . . .	2
<b>5</b>	<b>Module Hierarchy</b>	<b>2</b>
<b>6</b>	<b>Connection Between Requirements and Design</b>	<b>3</b>
<b>7</b>	<b>Module Decomposition</b>	<b>3</b>
7.1	Hardware Hiding Modules (M1) . . . . .	3
7.2	Behaviour-Hiding Module . . . . .	4
7.2.1	GUI Module (M??) . . . . .	5
7.2.2	Audio File Upload Module (M??) . . . . .	5
7.2.3	Search Query Module (M??) . . . . .	5
7.2.4	Client Communication Module (M??) . . . . .	5
7.2.5	Server Communication Module (M??) . . . . .	6
7.2.6	Driver Module (M??) . . . . .	6
7.2.7	Tempo (BPM) Feature Extraction Module (M??) . . . . .	6
7.2.8	Key and Scale Feature Extraction Module (M??) . . . . .	6
7.2.9	Instrument Type Feature Extraction Module (M??) . . . . .	6
7.2.10	Vocal Gender Feature Extraction Module (M??) . . . . .	7
7.2.11	Dynamic Range Feature Extraction Module (M??) . . . . .	7
7.2.12	Instrumentalness Feature Extraction Module (M??) . . . . .	7
7.2.13	Contour Feature Extraction Module (M??) . . . . .	7
7.2.14	Mood Feature Extraction Module (M??) . . . . .	7
7.2.15	Recommendation Module (M??) . . . . .	8
7.2.16	Program Results Interface Module (M??) . . . . .	8
7.3	Software Decision Module . . . . .	8
7.3.1	Database Module (M??) . . . . .	8
7.3.2	Spotify API (M??) . . . . .	8
7.3.3	Deezer API (M??) . . . . .	9
7.3.4	Genre Feature Module (M??) . . . . .	9
<b>8</b>	<b>Traceability Matrix</b>	<b>9</b>

<b>9</b>	<b>Use Hierarchy Between Modules</b>	<b>10</b>
<b>10</b>	<b>User Interfaces</b>	<b>11</b>
<b>11</b>	<b>Design of Communication Protocols</b>	<b>12</b>
<b>12</b>	<b>Timeline</b>	<b>13</b>

## List of Tables

1	Module Hierarchy . . . . .	3
2	Module Hierarchy . . . . .	4
3	Trace Between Requirements and Modules . . . . .	9
4	Trace Between Anticipated Changes and Modules . . . . .	10

## List of Figures

1	Use hierarchy among modules . . . . .	10
2	Home Page - The initial screen where users start their interaction. . . . .	11
3	Search Results - Shows the top matches for a user's search query. . . . .	11
4	Select Genre - Allows users to specify the genre for uploaded audio files. . . . .	12
5	Loading Page - Indicates that the application is processing a request. . . . .	12
6	Results Page - Displays the recommendation results. . . . .	13

### 3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the “secrets” that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module’s data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers’ understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

## 4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

### 4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

**AC1:** The specific hardware on which the software is running.

**AC2:** The format of the initial input data.

...

[Anticipated changes relate to changes that would be made in requirements, design or implementation choices. They are not related to changes that are made at run-time, like the values of parameters. —SS]

### 4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

**UC1:** Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

...

## 5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

**M1:** Hardware-Hiding Module

...

Level 1	Level 2
Hardware-Hiding Module	
	?
	?
	?
Behaviour-Hiding Module	?
	?
	?
	?
	?
	?
Software Decision Module	?
	?

Table 1: Module Hierarchy

## 6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

## 7 Module Decomposition

Modules are decomposed according to the principle of “information hiding” proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Software Engineering* means the module will be implemented by the Software Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

### 7.1 Hardware Hiding Modules (M1)

N/A



Level 1	Level 2
Hardware-Hiding	
Behaviour-Hiding	GUI Module Audio File Upload 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 Module
Software Decision	Database Spotify API Deezer API Genre Feature Module

Table 2: Module Hierarchy

## 7.2 Behaviour-Hiding Module

**Secrets:** The contents of the required behaviours.

**Services:** Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

**Implemented By:** –

### 7.2.1 GUI Module (M??)

**Secrets:** The format and structure of the input data.

**Services:** Handles user interaction, calling of the sub-input modules.

**Implemented By:** Frontend UI development framework

**Type of Module:** Abstract Data Type

### 7.2.2 Audio File Upload Module (M??)

**Secrets:** The format and structure of the input data.

**Services:** Takes in an audiofile and grabs metadata from the user.

**Implemented By:** Frontend framework file upload widget

**Type of Module:** Abstract Data Type

### 7.2.3 Search Query Module (M??)

**Secrets:** The format and structure of the input data.

**Services:** Takes in a user input, query searches it and then allows a user to select one of the query results.

**Implemented By:** Frontend Framework text input widget

**Type of Module:** Abstract Data Type

### 7.2.4 Client Communication Module (M??)

**Secrets:** The format and structure of the input data.

**Services:** Takes in a user input, query searches it and then allows a user to select one of the query results.

**Implemented By:** Python File

**Type of Module:** Abstract Data Type

### 7.2.5 Server Communication Module (M??)

**Secrets:** The format and structure of the input data.

**Services:** Takes in a user input, query searches it and then allows a user to select one of the query results.

**Implemented By:** Python File

**Type of Module:** Abstract Data Type

### 7.2.6 Driver Module (M??)

**Secrets:** The format and structure of the input data.

**Services:** Takes in a user input, query searches it and then allows a user to select one of the query results.

**Implemented By:** Python File

**Type of Module:** Abstract Data Type

### 7.2.7 Tempo (BPM) Feature Extraction Module (M??)

**Secrets:** The algorithm for calculating the tempo of the track.

**Services:** Extracts the Tempo (BPM) from the audio signal.

**Implemented By:** Python Function

**Type of Module:** Abstract Data Type

### 7.2.8 Key and Scale Feature Extraction Module (M??)

**Secrets:** The algorithm for calculating the key and scale of the track.

**Services:** Extracts the **key** and **scale** from the audio signal.

**Implemented By:** Python Function

**Type of Module:** Abstract Data Type

### 7.2.9 Instrument Type Feature Extraction Module (M??)

**Secrets:** The algorithm(s) for detecting the type(s) of instruments present within the track.

**Services:** Detects the presence of different instrument types within the input audio signal.

**Implemented By:** Python Function

**Type of Module:** Abstract Data Type

#### **7.2.10 Vocal Gender Feature Extraction Module (M??)**

**Secrets:** The algorithm for detecting the overall vocal gender of the track.

**Services:** Extracts the overall vocal gender of the input audio signal.

**Implemented By:** Python Function

**Type of Module:** Abstract Data Type

#### **7.2.11 Dynamic Range Feature Extraction Module (M??)**

**Secrets:** The format and structure of the input data.

**Services:** Extracts the Dynamic Range (difference between peak and trough in decibels) of the input audio signal.

**Implemented By:** Python Function

**Type of Module:** Abstract Data Type

#### **7.2.12 Instrumentalness Feature Extraction Module (M??)**

**Secrets:** The algorithm for detecting the instrumentalness of the track.

**Services:** Detects the presence level of instrumentals from the input audio signal.

**Implemented By:** Python Function

**Type of Module:** Abstract Data Type

#### **7.2.13 Contour Feature Extraction Module (M??)**

**Secrets:** The algorithm to detect the musical contour of the track.

**Services:** Extracts the musical contour of the input audio signal.

**Implemented By:** Python Function

**Type of Module:** Abstract Data Type

#### **7.2.14 Mood Feature Extraction Module (M??)**

**Secrets:** The algorithm for detecting the mood of the track.

**Services:** Extracts the mood of the input audio signal.

**Implemented By:** Python Function

**Type of Module:** Abstract Data Type

#### **7.2.15 Recommendation Module (M??)**

**Secrets:** The recommendation machine learning algorithm.

**Services:** Generates a list of recommended songs.

**Implemented By:** Machine Learning Algorithm

**Type of Module:** Library

#### **7.2.16 Program Results Interface Module (M??)**

**Secrets:** The recommended songs widget and features display element.

**Services:** Generates widget for recommended song(s) with preview snippet and UI element containing associated features for input song.

**Implemented By:** Spotify API Calls,

**Type of Module:** Abstract Data Type

### **7.3 Software Decision Module**

**Secrets:** The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

**Services:** Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

**Implemented By:** –

#### **7.3.1 Database Module (M??)**

**Secrets:** The data contained in the tables.

**Services:** Storages for generated features.

**Implemented By:** Database Service

**Type of Module:** Abstract Data Object

#### **7.3.2 Spotify API (M??)**

**Secrets:** Spotify API methods, credentials and metadata.

**Services:** Song Metadata

**Implemented By:** Spotify

**Type of Module:** API Library

### 7.3.3 Deezer API (M??)

**Secrets:** Deezer API methods, credentials and metadata.

**Services:** Converts the input data into the data structure used by the input parameters module.

**Implemented By:** Deezer

**Type of Module:** API Library

### 7.3.4 Genre Feature Module (M??)

**Secrets:** Genre label for a song

**Services:** Fetches a genre label for a required song

**Implemented By:** Deezer API

**Type of Module:** Abstract Data Type

## 8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M1
R2	M1
R3	M1
R4	M2
R5	M2
R6	M3, M4
R7	M6, M7, M8, M9, M10, M11, M13, M14
R8	M6, M7, M8, M9, M10, M11, M13, M14
R9	M15
R10	M4, M5
R11	M4, M16

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 4: Trace Between Anticipated Changes and Modules

## 9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

Figure 1: Use hierarchy among modules

## 10 User Interfaces



Figure 2: Home Page - The initial screen where users start their interaction.



Figure 3: Search Results - Shows the top matches for a user's search query.



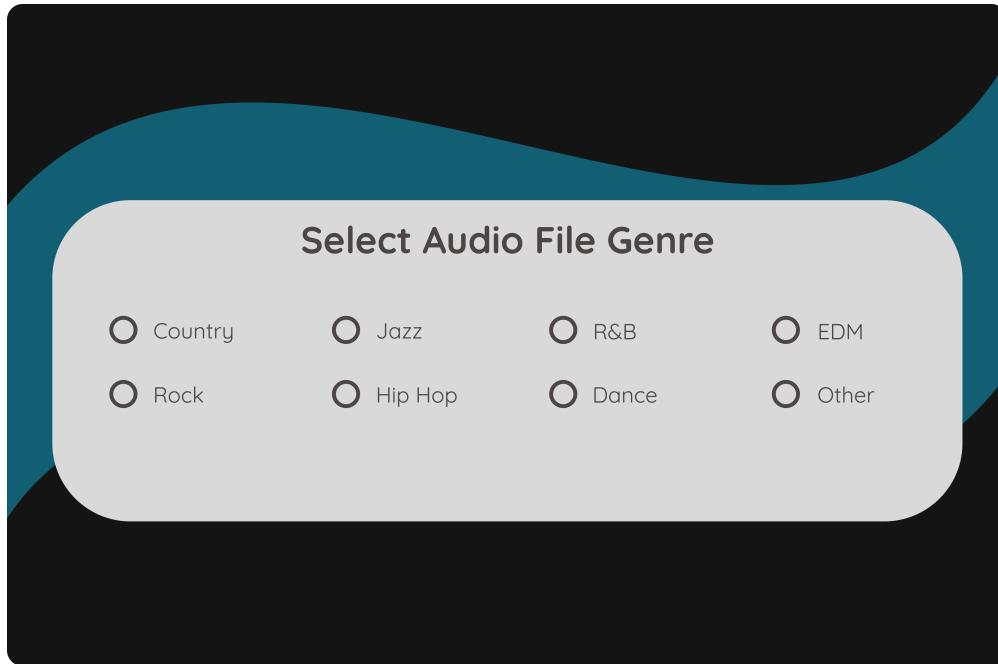


Figure 4: Select Genre - Allows users to specify the genre for uploaded audio files.

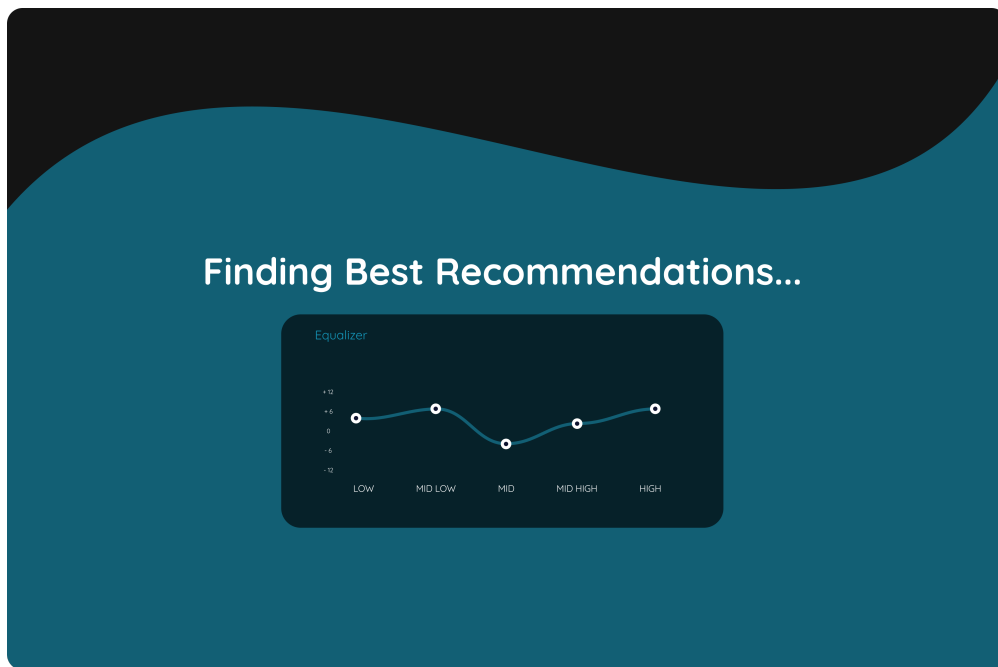


Figure 5: Loading Page - Indicates that the application is processing a request.

## 11 Design of Communication Protocols

Our design features 2 modules that have to do communication. They are listed as such:

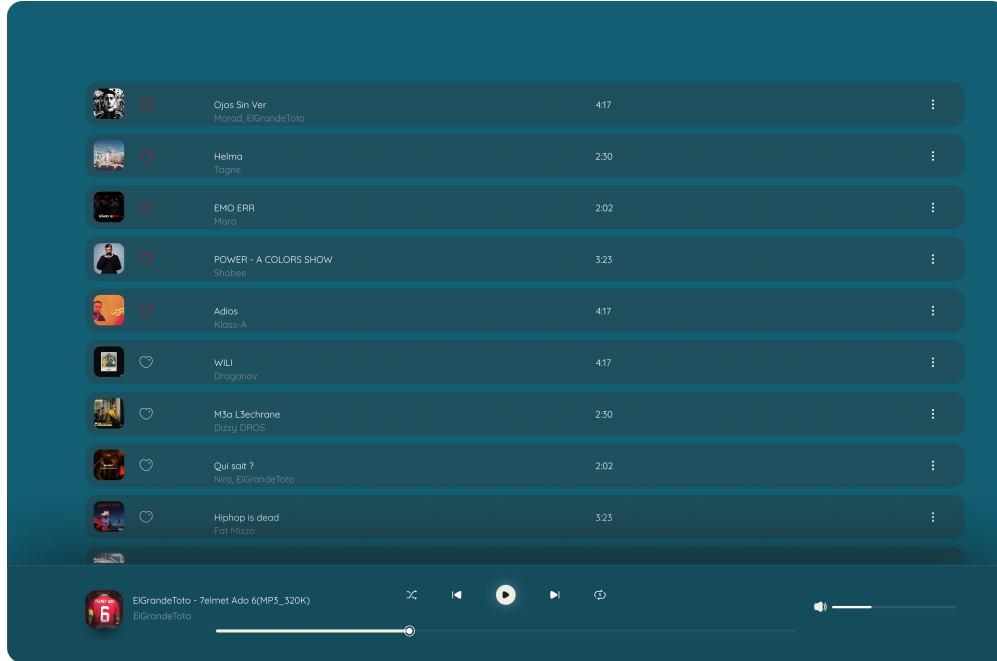


Figure 6: Results Page - Displays the recommendation results.

- Client communication module
- Server communication module

The communication between the frontend systems is done simply through function calls. In order to send data (input type, audio data, song ID) from the frontend the user interacts with to the backend (the server) this will be done through encrypted data sent to the hosted server. This would be done by using a RESTFUL interface as this is currently the most common method. Using standard REST API, the payload passed is a regular JSON object with a label for search or file input (the type of input) as that changes the behavior of the backend and the actual song ID itself.

## 12 Timeline

The following timeline outlines the key milestones and activities planned for the development and refinement process:

- **January 6th - January 17th:** Finalize the design document, decide on features and finalize on software architecture.
- **January 18th - January 20th:** Focus on refining the SRS, Hazard Analysis, and VnV documents.

- **January 20th - January 21st:** Conduct a meeting with Dr. MvM with a well-defined agenda to review progress and receive feedback.
- **January 22nd - January 23rd:** Engage with user groups to gather additional insights and validate design decisions.
- **January 23rd onwards:** Begin work on Rev 0 of the project deliverables, incorporating feedback from user groups and the advisor meeting.

All team members are responsible for the tasks described in this timeline breakdown. (These were agreed upon in the [Post-Winter break Check-in](<https://github.com/AhmedAl-Hayali/GenreGuru/issues/278>))

## References

- David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.
- David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.
- D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.