Module Guide for SFWRENG 4G06A

Team #25, RapidCare
Pranav Kalsi
Gurleen Rahi
Inreet Kaur
Moamen Ahmed

January 12, 2025

1 Revision History

Date	Version	Notes
Jan 9, 2025	1.1	Initial Document

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
SFWRENG 4G06A	Explanation of program name
UC	Unlikely Change

Contents

List of Tables

List of Figures

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 (**?**). We advocate a decomposition based on the principle of information hiding (**?**). 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 layed out by **?**, 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 (**?**). 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 system should support different authentication methods, such as password-based, biometric login etc.

AC2: The transcription module needs to be optimized to enhance the transcription speed from audio data to written text in real-time.

AC3: The diagnosis prediction and medicine prediction modules should be updated with the latest medical knowledge based on the new medical research.

AC4: The system shall enable real-time streaming of audio input to written text to ensure immediate transcription without delays.

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: The requirement to secure patient's profile ensures confidentiality will remain a constant priority in the system.

UC2: The ability of the system to be compatible with the latest versions of different operating systems such as Windows, Linux and macOS will remain a constant requirement.

UC3: The requirement to exclude the background noise while using transcription module is unlikely to change for accurately classification of the medical data.

UC4: The system's ease of use is anticipated to remain a constant requirement to allow them to focus on patient's care instead of struggling with the technology.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: App Module

M2: User Authentication Module

M3: API Module

M4: Administrator View Module

M5: Client View Module

M6: Report Generating Module

M7: Transcription Module

M8: Classification Module

M9: Diagnosis Prediction Module

M10: Medicine Prediction Module

M11: Administrator Account Management Module

M12: Patient Account Management Module

M13: Diagnosis Data Module

M14: Medicine Data Module

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 4.

Level 1	Level 2
Hardware-Hiding Module	None
Behaviour-Hiding Module	M2
	M3
	M4
	M5
	M13
	M14
Software Decision Module	M1
	M6
	M7
	M8
	M9
	M10
	M11
	M12

Table 1: Module Hierarchy

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. 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. SFWRENG 4G06A means the module will be implemented by the SFWRENG 4G06A 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 (M??)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

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 User Authentication Module (M2)

Moamen

Secrets:

Services:

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.2.2 API Module (M3)

Moamen

Secrets:

Services:

Implemented By: []

Type of Module: []

7.2.3 Administrator View Module (M4)

Secrets: User interface customization based on the administrative tools and functionality required by healthcare networks.

Services: Provide healthcare network adminstrators with tools to onboard, update, and remove thier network on the system. Provide healthcare network adminstrators with tools to add healthcare professionals under a healthcare network to the system.

Implemented By: TypeScript, React

Type of Module: Abstract Object

7.2.4 Client View Module (M5)

Secrets: User interface customization based on the tools and functionality required by healthcare professionals.

Services: Provides healthcare professionals with tools to login, create, update, and delete patient records, provide diagnostic suggestions, and medication suggestions.

Implemented By: TypeScript, React

Type of Module: Abstract Object

7.2.5 Diagnosis Data Module (M13)

Secrets: Database schema for indexing strategies, and data caching mechanisms for efficient storage and retrieval.

Services: Manages secure storage, retrieval, and updates of data related to diagnosis records. The ultimate purpose for this is to provide training data for the ciagnosis prediction module.

Implemented By: MongoDB, Spring

Type of Module: Record

7.2.6 Medicine Data Module (M14)

Secrets: Database schema for indexing strategies, and data caching mechanisms for efficient storage and retrieval.

Services: Manages secure storage, retrieval, and updates of data related to drug records. The ultimate purpose for this is to provide training data for the medicine prediction module.

Implemented By: MongoDB, Spring

Type of Module: Record

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 App Module (M1)

Secrets: The algorithm used to control and coordinate the flow of data between different modules.

Services: Facilitates communication between different independent modules and manages the state of the application.

Implemented By: React

Type of Module: Abstract Object

7.3.2 Report Generation Module (M6)

Secrets: The algorithm to generate accurate reports based on the transcribed data from the conversation.

Services: Extracts the important aspects of medical data and compiled into the report.

Implemented By: Python

Type of Module: Library

7.3.3 Transcription Module (M7)

Secrets: The algorithm used to convert audio data into written text.

Services: Accurately converts the audio data from the conversation into written text.

Implemented By: Python

Type of Module: Library

7.3.4 Classification Module (M8)

Secrets: The algorithm used to classify the medical data after the transcription module.

Services: Accurately classifies the medical data received from the transcription module into relevant categories.

Implemented By: Python

Type of Module: Library

7.3.5 Diagonsis Prediction Module (M9)

Secrets: The algorithms used to predict possible diagnoses.

Services: Predicts a set of applicable diagnoses for a patient based on patient characteristics, symptoms, and past medical history.

Implemented By: Python

Type of Module: Abstract Object

7.3.6 Medicine Prediction Module (M10)

Secrets: The algorithms used to give possible medicines applicable.

Services: Predicts a set of applicable medicines for a patient based on patient characteristics, symptoms, and past medical history.

Implemented By: Python

Type of Module: Abstract object

7.3.7 Administrator Account Management Module (M11)

Secrets: Database schema for indexing strategies, and data caching mechanisms for efficient storage and retrieval.

Services: Manages secure storage, retrieval, and updates of data related to healthcare networks and healthcare professionals.

Implemented By: MongoDB, Spring

Type of Module: Record

7.3.8 Patient Account Management Module (M12)

Secrets: Database schema for indexing strategies, and data caching mechanisms for efficient storage and retrieval.

Services: Manages secure storage, retrieval, and updates of data related to patient records.

Implemented By: MongoDB, Spring

Type of Module: Record

8 Traceability Matrix

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

Req.	**Modules**
FR1	M1, M3, M4, M11
FR2	M1, M2, M3, M4, M11
FR3	M1, M2, M3, M4, M11
FR4	M1, M2, M3, M4, M11
FR5	M1, M2, M3, M4, M11
FR6	M1, M2, M3, M4, M11
FR7	M1, M2, M3, M11
FR8	M1, M2, M3, M5, M12
FR9	M1, M2, M3, M5, M12
FR10	M1, M2, M3, M5, M12
FR11	M1, M2, M3, M5, M6, M7, M8, M12
FR12	M1,M2, M3, M5, M6, M9, M12, M13
FR13	M1, M2, M3, M5, M6, M10, M12, M14
FR14	M1, M2, M3, M4, M12

Table 2: Trace Between Functional Requirements and Modules

Req.	**Modules**
NFR1	M1, M4, M5
NFR2	M1, M4, M5
NFR3	M1, M7
NFR4	M1, M11, M12, M13, M14
NFR5	All
NFR6	M1, M2, M12
NFR7	M1, M5
NFR8	All
NFR9	All

Table 3: Trace Between Non-Functional Requirements and Modules

Req.	Modules
AC1	M3
AC2	M1, M2, M3, M4, M11
IR1	M2, M3, M4, M11
IR2	All
IR3	M9, M10
IR4	M11, M12
IR5	M9, M10
IR6	M6, M8
IR7	M6, M7

Table 4: Trace Between Safety and Security Requirements and Modules

AC	Modules
AC1	M1, M2
AC2	M7
AC3	M9, M10
AC4	M1, M3, M5, M7, M??

Table 5: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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]

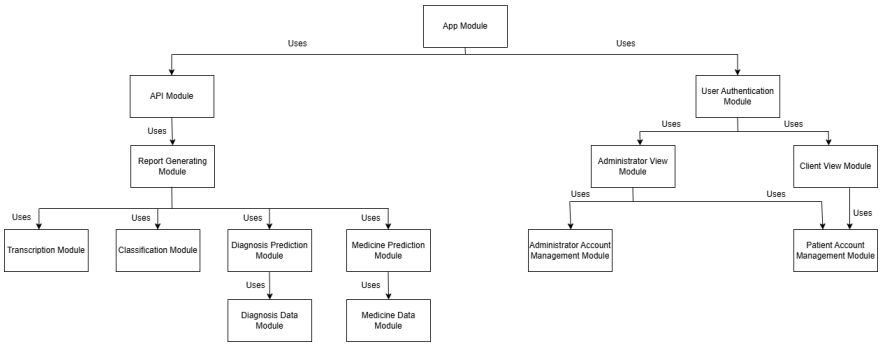

Figure 1: Use hierarchy among modules

10 User Interfaces

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS]

11 Design of Communication Protocols

Due to the the broker architectures connecting with various services the communication protocols differ slightly. For most of the communication as they will be simply 'POST' or 'GET'. Examples if this communication being used would be fetching Patient data from the database, classifying the transcription, or employee retrieval, etc... HTTP communication will be used in **all** but one use case.

The one use case where HTTP communication will not be used is for the transcription service. For this service we will be using web socket communication as this will provide streaming capabilities for real time transcription. The frontend will act as the client who will send audio chunks to the backend to be transcribed. This will facilitate bidirectional communcation as well.

12 Timeline

[Schedule of tasks and who is responsible —SS]

[You can point to GitHub if this information is included there —SS]