Module Interface Specification for Software Engineering

Team #22, TeleHealth Insights
Mitchell Weingust
Parisha Nizam
Promish Kandel
Jasmine Sun-Hu

January 13, 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 TeleHealth Insights. It is an at-home bilingual speech assessment system with video and audio analysis features. The system is designed to provide clear guidance to parents when administering the assessment to their children, in an environment where speech-language pathologists (SLPs) are unavailable. By streamlining the assessment process, the project aims to provide a convenient and comprehensive solution for SLPs to assess and support their patients' speech and language development remotely.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/parishanizam/TeleHealth

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	N/A
Behaviour-Hiding	Clinician GUI Parent GUI Authentication Module Result Storage Module Real-Time Feedback Module Report Generation Module Media Processing Module Video Processing Module Audio Processing Module Logging Module Question Bank Module Mandarian Question Bank English Question Bank Repetition Question Bank Module Matching Question Bacnk Module
Software Decision	APP Controller API Gateway

Table 1: Module Hierarchy

6 MIS of Authentication Module

6.1 Module

AuthenticationModule

6.2 Uses

N/A

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
signup	username: str, email: str, password: str, role: str	status: bool	UserAlreadyExistsException
login	username: str, password: str	sessionToken: str	InvalidCredentialsException
logout	sessionToken: str	status: bool	InvalidSessionException

6.4 Semantics

6.4.1 State Variables

- userList: Set(User) - maintains a set of all registered users.
- activeSessions: Map(sessionToken, User) - tracks active user sessions.

6.4.2 Environment Variables

N/A

6.4.3 Assumptions

- Usernames and emails are unique.
- Sessions are managed using session tokens.
- Role can be one of [’parent’, ’clinician’, ’admin’].

- Clinicians have given user their login token

6.4.4 Access Routine Semantics

signup():

- transition: Adds a new user to ‘userList‘ if the username and email are unique.
- output: Returns ‘true‘ if the user is successfully created, otherwise throws ‘UserAlreadyExistsException‘.
- exception: Throws ‘UserAlreadyExistsException‘ if the username or email already exists.

login():

- transition: Adds a new session to ‘activeSessions‘ if the credentials are valid.
- output: Returns a ‘sessionToken‘ for the logged-in user.
- exception: Throws ‘InvalidCredentialsException‘ if the username or password is incorrect.

logout():

- transition: Removes the ‘sessionToken‘ from ‘activeSessions‘.
- output: Returns ‘true‘ if the session is successfully ended.
- exception: Throws ‘InvalidSessionException‘ if the session token does not exist.

6.4.5 Local Functions

N/A

7 MIS of Result Storage Module

7.1 Module

ResultStorageModule

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**
storeResult	data: JSON object	status: bool	StorageException
retrieveResult	resultID: str	data: JSON object	NotFoundException
deleteResult	resultID: str	status: bool	NotFoundException

7.4 Semantics

7.4.1 State Variables

- resultStorage: Map(resultID, JSON object) - stores all processed results with unique IDs.

7.4.2 Environment Variables

N/A

7.4.3 Assumptions

- Each result is assigned a unique resultID.
- Results are stored as JSON objects for flexibility.
- Data is stored in MongoDB or an equivalent NoSQL database.

7.4.4 Access Routine Semantics

storeResult():

- transition: Adds the ‘data‘ to ‘resultStorage‘ with a unique ‘resultID‘.
- output: Returns ‘true‘ if the result is successfully stored.
- exception: Throws ‘StorageException‘ if there is an issue storing the data.

retrieveResult():

- transition: None
- output: Returns the result associated with the ‘resultID‘.

- exception: Throws ‘NotFoundException‘ if the ‘resultID‘ does not exist.

deleteResult():

- transition: Removes the result associated with the ‘resultID‘ from ‘resultStorage‘.
- output: Returns ‘true‘ if the result is successfully deleted.
- exception: Throws ‘NotFoundException‘ if the ‘resultID‘ does not exist.

7.4.5 Local Functions

N/A

8 MIS of Report Generation Module

8.1 Module

ReportGenerationModule

8.2 Uses

- Result Storage Module
- Media Processing Module
- Question Bank Module

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
generateReport	sessionID: str, metadata: JSON object	report: JSON	ReportGenerationException
getReport	reportID: str	report: JSON	NotFoundException

8.4 Semantics

8.4.1 State Variables

- reportStorage: Map(reportID, Report) - stores all generated reports with unique IDs.

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

- Each report is assigned a unique reportID.
- Reports are generated using data fetched from the Result Storage Module and other sources like the Question Bank Module.
- Reports can be retrieved in JSON format then converted to PDF format
- The clinician requesting the report has access to the session data.

8.4.4 Access Routine Semantics

generateReport():

- transition: Creates a new report using the ‘sessionID‘ and ‘metadata‘, and stores it in ‘reportStorage‘.
- output: Returns the generated report in the specified format (JSON or PDF).
- exception: Throws ‘ReportGenerationException‘ if there is an error during report generation.

getReport():

- transition: None
- output: Returns the report associated with the ‘reportID‘.
- exception: Throws ‘NotFoundException‘ if the ‘reportID‘ does not exist in ‘reportStorage‘.

8.4.5 Local Functions

N/A

9 MIS of Real-Time Feedback Module

9.1 Module

RealTimeFeedbackModule

9.2 Uses

- Media Processing Module

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
provideFeedback	sessionID: str, liveFeed: media stream	feedback: JSON object	FeedbackException
logFeedback	sessionID: str, feedback: JSON object	status: bool	LoggingException

9.4 Semantics

9.4.1 State Variables

- feedbackLogs: Map(sessionID, List(feedback)) - stores real-time feedback for sessions.

9.4.2 Environment Variables

N/A

9.4.3 Assumptions

- The module receives a continuous media stream (audio or video) during a session.
- Feedback is generated by analyzing live media streams using the Media Processing Module.
- Feedback is stored for each session to provide session summaries if needed.

- The module operates within acceptable latency constraints to ensure real-time performance.

9.4.4 Access Routine Semantics

provideFeedback():

- transition: Generates feedback from the ‘liveFeed‘ media stream and optionally logs it in ‘feedbackLogs‘.
- output: Returns actionable feedback in a structured JSON format (e.g., "Adjust microphone", "Increase lighting").
- exception: Throws ‘FeedbackException‘ if there is an issue processing the live feed.

logFeedback():

- transition: Adds the provided ‘feedback‘ to ‘feedbackLogs‘ for the corresponding ‘sessionID‘.
- output: Returns ‘true‘ if the feedback is successfully logged.
- exception: Throws ‘LoggingException‘ if there is an error while logging the feedback.

9.4.5 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.

10 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)