Module Interface Specification for Software Engineering

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

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 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 Clinician GUI

6.1 Module

clinicianGUI

6.2 Uses

- AppController

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayLoginPage	-	-	-
displayClinicianDashboardPage	-	-	-
displayAddClientPage	-	-	-
displayPatientOverviewPage	-	-	-
displayPatientAssessmentResultsAnalysisPage	-	-	-
displayBiasReviewPage	-	-	-
displayFlagBiasPage	-	-	-

6.4 Semantics

6.4.1 State Variables

N/A

6.4.2 Environment Variables

N/A

6.4.3 Assumptions

N/A

6.4.4 Access Routine Semantics

displayLoginPage():

- transition: Navigates to and displays the clinician login page for the system.

- output: N/A
- exception: N/A

displayClinicianDashboardPage():

- transition: Navigates to and displays the clinician dashboard page for accessing a clinician's list of clients.
- output: N/A
- exception: N/A

displayAddClientPage():

- transition: Navigates to and displays the add client page for adding a new client to a clinician's list.
- output: N/A
- exception: N/A

displayPatientOverviewPage():

- transition: Navigates to and displays the patient overview page for accessing all of the patient's previous assessments.
- output: N/A
- exception: N/A

displayPatientAssessmentResultsAnalysisPage():

- transition: Navigates to and displays the patient assessment results analysis page for accessing all of the results of a particular assessment.
- output: N/A
- exception: N/A

displayBiasReviewPage():

- transition: Navigates to and displays the bias review page for reviewing and removing bias from a particular question in an assessment.
- output: N/A
- exception: N/A

displayFlagBiasPage():

- transition: Navigates to and displays the flag bias page for reviewing and flagging bias on a particular question in an assessment.
- output: N/A
- exception: N/A

6.4.5 Local Functions

N/A

7 MIS of Parent GUI

7.1 Module

parentGUI

7.2 Uses

- AppController

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
displayLoginPage	-	-	-
displayCreateAccountPage	-	-	-
displayHomePage	-	-	-
displayAssessmentSelectionPage	-	-	-
displayParentChecklistPage	-	-	-
displayInputCheckPage	microphoneInput, cameraInput	-	-
displayAssessmentInstructionsPage	-	-	-
displayAssessmentQuestionsPage	-	-	-
displayCompletionPage	-	-	-

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

- microphoneInput
- cameraInput

7.4.3 Assumptions

N/A

7.4.4 Access Routine Semantics

displayLoginPage():

- transition: Navigates to and displays the parent login page for the system.
- output: N/A
- exception: N/A

displayCreateAccountPage():

- transition: Navigates to and displays the parent account creation page for creating a new account for the system.
- output: N/A
- exception: N/A

displayHomePage():

- transition: Navigates to and displays the homepage for a parent account, with the ability to start a new assessment.
- output: N/A
- exception: N/A

displayAssessmentSelectionPage():

- transition: Navigates to and displays the assessment selection page for selecting the type of assessment for the user.
- output: N/A
- exception: N/A

displayParentChecklistPage():

- transition: Navigates to and displays the parent checklist page for informing parents about the requirements of the assessment.
- output: N/A
- exception: N/A

displayInputCheckPage():

- transition: Navigates to and displays the input check page for testing input devices.
- output: N/A

- exception: N/A

displayAssessmentInstructionsPage():

- transition: Navigates to and displays the assessment instructions page for the child to read and engage with to learn how to interact with the assessment interface.
- output: N/A
- exception: N/A

displayAssessmentQuestionsPage():

- transition: Navigates to and displays the assessment questions page for displaying a question and its corresponding answers for the user to select.
- output: N/A
- exception: N/A

displayCompletionPage():

- transition: Navigates to and displays the completion page to confirm to the user that the assessment is complete and their results have been saved.
- output: N/A
- exception: N/A

7.4.5 Local Functions

N/A

8 MIS of App Controller

8.1 Module

AppController

8.2 Uses

- APIGateway

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
accessAPIGateway	-	-	-

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

accessAPIGateway():

- transition: Controller accesses the API Gateway.
- output: N/A
- exception: N/A

8.4.5 Local Functions

N/A

9 MIS of API Gateway

9.1 Module

APIGateway

9.2 Uses

- Authentication
- ResultStorage
- MediaProcessing
- Logging
- QuestionBank
- RealTimeFeedback
- ReportGeneration

9.3 Syntax

9.3.1 Exported Constants

N/A

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
accessAuthentication	-	-	-
accessResultStorage	-	-	-
accessMediaProcessing	-	-	-
accessLogging	-	-	-
accessQuestionBank	-	-	-
accessRealTimeFeedback	-	-	-
accessReportGeneration	-	-	-

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

accessAuthentication():

- transition: Controller accesses the authentication module.
- output: N/A
- exception: N/A

accessResultStorage():

- transition: Controller accesses the results storage module.
- output: N/A
- exception: N/A

accessMediaProcessing():

- transition: Controller accesses the media processing module.
- output: N/A
- exception: N/A

accessLogging():

- transition: Controller accesses the logging module.
- output: N/A
- exception: N/A

accessQuestionBank():

- transition: Controller accesses the question bank module.
- output: N/A
- exception: N/A

accessRealTimeFeedback():

- transition: Controller accesses the real time feedback module.
- output: N/A
- exception: N/A

accessReportGeneration():

- transition: Controller accesses the report generation module.
- output: N/A
- exception: N/A

9.4.5 Local Functions

N/A

10 MIS of Authentication Module

10.1 Module

AuthenticationModule

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
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

10.4 Semantics

10.4.1 State Variables

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

10.4.2 Environment Variables

N/A

10.4.3 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

10.4.4 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.

10.4.5 Local Functions

N/A

11 MIS of Result Storage Module

11.1 Module

ResultStorageModule

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

11.4 Semantics

11.4.1 State Variables

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

11.4.2 Environment Variables

N/A

11.4.3 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.

11.4.4 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.

11.4.5 Local Functions

N/A

12 MIS of Media Processing Module

12.1 Module

MediaProcessingModule

12.2 Uses

VideoProcessingModule, AudioProcessingModule

12.3 Syntax

12.3.1 Exported Constants

N/A

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
processMedia	mediaFile: str, mediaType: str, assessmentID: str	report: MediaAnalysisReport	MediaProcessingException

12.4 Semantics

12.4.1 State Variables

- processedMedia: Map(assessmentID, MediaAnalysisReport) - stores combined results from video and audio analysis.

12.4.2 Environment Variables

N/A

12.4.3 Assumptions

- mediaType specifies whether the file is video or audio (e.g., "video", "audio").
- Delegates processing to VideoProcessingModule or AudioProcessingModule based on mediaType.

12.4.4 Access Routine Semantics

processMedia():

- transition:
  - If mediaType is "video", calls processVideo from VideoProcessingModule.
  - If mediaType is "audio", calls processAudio from AudioProcessingModule.
  - Combines results into a single MediaAnalysisReport per assessment completed.
- output: Returns a MediaAnalysisReport with details from video and audio analyses.
- exception: Throws MediaProcessingException if the file cannot be processed or delegated.

12.4.5 Local Functions

N/A

13 MIS of Logging Module

13.1 Module

LoggingModule

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
logEvent	eventType: str, message: str	status: bool	LoggingException
fetchLogs	logType: str, timeRange: TimeRange	logData: List(LogEntry)	LogFetchException
clearLogs	logType: str, timeRange: TimeRange	status: bool	LogClearException

13.4 Semantics

13.4.1 State Variables

- logs: Map(logType, List(LogEntry)) - stores all logs categorized by type.

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

- Logs are categorized by their type and timestamp for easy retrieval.
- Each log entry includes metadata such as the timestamp, severity, and module of origin.

13.4.4 Types of Logs

Event Logs

Tracks application activities, such as user logins, file uploads, and media processing events. Example: "User 'parent1' logged in successfully at 14:32:00." Error Logs

Captures unexpected behaviors, exceptions, or failures in the application. Example: "VideoProcessingException: Unable to analyze video file 'session123.mp4' due to corrupted data." Audit Logs

Records critical changes and actions for accountability, such as user role changes or log clearances. Example: "Admin user 'clinician1' updated parent access rights at 16:45:00." Performance Logs

Monitors application performance metrics like response times, memory usage, and processing durations. Example: "Media processing for 'session456' completed in 3.2 seconds with 200MB memory usage." Debug Logs

Includes detailed information for troubleshooting during development or maintenance. Example: "Entering function 'processVideo' with input file 'session789.mp4'." Security Logs

Tracks security-related events such as failed logins, access violations, or token expirations. Example: "Security alert: Failed login attempt for user 'parent2' at 18:12:30."

13.4.5 Access Routine Semantics

logEvent():

- transition: Adds a new log entry to the logs map under the appropriate eventType.
- output: Returns true if the log is successfully recorded.
- exception: Throws LoggingException if the log entry cannot be added.

fetchLogs():

- transition: Retrieves all logs of the specified logType within the provided timeRange.
- output: Returns a list of LogEntry objects matching the criteria.
- exception: Throws LogFetchException if no logs are found or retrieval fails.

clearLogs():

- transition: Removes all logs of the specified logType within the provided timeRange.
- output: Returns true if logs are successfully cleared.
- exception: Throws LogClearException if logs cannot be cleared.

13.4.6 Local Functions

N/A

14 MIS of Question Bank Module

14.1 Module

QuestionBankModule

14.2 Uses

EnglishQuestionBankModule, MandarinQuestionBankModule

14.3 Syntax

14.3.1 Exported Constants

N/A

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
selectQuestionBank	language: str	questionBank: ADT	InvalidLanguageException
retrieveQuestion	language: str, questionID: str	question: JSON object	NotFoundException

14.4 Semantics

14.4.1 State Variables

- activeQuestionBanks: Map(language, ADT) - maps language to its respective question bank module.

14.4.2 Environment Variables

N/A

14.4.3 Assumptions

- Supported languages include English and Mandarin.
- Each question bank module is preloaded with language-specific questions.

14.4.4 Access Routine Semantics

selectQuestionBank():

- transition: Selects the question bank module corresponding to the input language.

- output: Returns the selected question bank module.
- exception: Throws `InvalidLanguageException` if the input language is not supported.

retrieveQuestion():

- transition: None
- output: Retrieves the question from the appropriate question bank module.
- exception: Throws `NotFoundException` if the questionID does not exist in the selected module.

14.4.5 Local Functions

N/A

15 MIS of Real-Time Feedback Module

15.1 Module

RealTimeFeedbackModule

15.2 Uses

- Media Processing Module

15.3 Syntax

15.3.1 Exported Constants

N/A

15.3.2 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

15.4 Semantics

15.4.1 State Variables

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

15.4.2 Environment Variables

N/A

15.4.3 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.

15.4.4 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.

15.4.5 Local Functions

N/A

16 MIS of Report Generation Module

16.1 Module

ReportGenerationModule

16.2 Uses

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

16.3 Syntax

16.3.1 Exported Constants

N/A

16.3.2 Exported Access Programs

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

16.4 Semantics

16.4.1 State Variables

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

16.4.2 Environment Variables

N/A

16.4.3 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.

16.4.4 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‘.

16.4.5 Local Functions

N/A

17 MIS of Video Processing Module

17.1 Module

VideoProcessingModule

17.2 Uses

N/A MediaProcessingModule

17.3 Syntax

17.3.1 Exported Constants

N/A

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
processVideo	videoFile: str, assessmentID: str	report: VideoAnalysisReport	VideoProcessingException

17.4 Semantics

17.4.1 State Variables

- processedVideos: Map(assessmentID, VideoAnalysisReport) - stores results of processed videos.

17.4.2 Environment Variables

N/A

17.4.3 Assumptions

- Video files are in a supported format (e.g., MP4, AVI).
- Video processing is done within a time threshold for real-time feedback.

17.4.4 Access Routine Semantics

processVideo():

- transition: Analyzes the video to identify any disturbances, bias, or cheating patterns.
- output: Returns a detailed VideoAnalysisReport containing flagged events and metrics.
- exception: Throws VideoProcessingException if the file cannot be processed or analyzed.

17.4.5 Local Functions

N/A

18 MIS of Audio Processing Module

18.1 Module

AudioProcessingModule

18.2 Uses

N/A MediaProcessingModule

18.3 Syntax

18.3.1 Exported Constants

N/A

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
processAudio	audioFile: str, assessmentID: str	report: AudioAnalysisReport	AudioProcessingException

18.4 Semantics

18.4.1 State Variables

- processedAudio: Map(assessmentID, AudioAnalysisReport) - stores results of processed audio.

18.4.2 Environment Variables

N/A

18.4.3 Assumptions

- Audio files are in a supported format (e.g., WAV, MP3).
- Background noise levels are detectable and quantifiable.

18.4.4 Access Routine Semantics

processAudio():

- transition: Analyzes the audio for disturbances such as background noise or interruptions.
- output: Returns a detailed AudioAnalysisReport with flagged issues.
- exception: Throws AudioProcessingException if the file cannot be processed or analyzed.

18.4.5 Local Functions

N/A

19 MIS of English Question Bank Module

19.1 Module

EnglishQuestionBankModule

19.2 Uses

MatchingQuestionBankModule, RepetitionQuestionBankModule

19.3 Syntax

19.3.1 Exported Constants

N/A

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
getQuestion	questionID: str	question: JSON object	NotFoundException
addQuestion	question: JSON object	status: bool	StorageException

19.4 Semantics

19.4.1 State Variables

- englishQuestions: Map(questionID, JSON object) - stores English questions.

19.4.2 Environment Variables

N/A

19.4.3 Assumptions

- Questions are either matching or repetition type.
- Matching and Repetition modules are used to handle the respective types.

19.4.4 Access Routine Semantics

getQuestion():

- transition: None
- output: Returns the question corresponding to the questionID.
- exception: Throws `NotFoundException` if the questionID does not exist.

addQuestion():

- transition: Adds the input question to `englishQuestions`.
- output: Returns `true` if the question is successfully added.
- exception: Throws `StorageException` if there is an issue storing the question.

19.4.5 Local Functions

N/A

20 MIS of Mandarin Question Bank Module

20.1 Module

MandarinQuestionBankModule

20.2 Uses

MatchingQuestionBankModule, RepetitionQuestionBankModule

20.3 Syntax

20.3.1 Exported Constants

N/A

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
getQuestion	questionID: str	question: JSON object	NotFoundException
addQuestion	question: JSON object	status: bool	StorageException

20.4 Semantics

20.4.1 State Variables

- mandarinQuestions: Map(questionID, JSON object) - stores Mandarin questions.

20.4.2 Environment Variables

N/A

20.4.3 Assumptions

- Questions are either matching or repetition type.
- Matching and Repetition modules are used to handle the respective types.

20.4.4 Access Routine Semantics

getQuestion():

- transition: None
- output: Returns the question corresponding to the questionID.
- exception: Throws `NotFoundException` if the questionID does not exist.

addQuestion():

- transition: Adds the input question to `englishQuestions`.
- output: Returns `true` if the question is successfully added.
- exception: Throws `StorageException` if there is an issue storing the question.

20.4.5 Local Functions

N/A

21 MIS of Matching Question Bank Module

21.1 Module

MatchingQuestionBankModule

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
storeMatchingQuestion	question: JSON object	status: bool	StorageException
retrieveMatchingQuestion	questionID: str	question: JSON object	NotFoundException

21.4 Semantics

21.4.1 State Variables

- matchingQuestions: Map(questionID, JSON object) - stores matching questions.

21.4.2 Environment Variables

N/A

21.4.3 Assumptions

- Questions have a unique ID.
- Data is stored in a JSON format for flexibility.

21.4.4 Access Routine Semantics

storeMatchingQuestion():

- transition: Adds the question to `matchingQuestions`.
- output: Returns `true` if successfully stored.
- exception: Throws `StorageException` if there is a storage error.

retrieveMatchingQuestion():

- transition: None
- output: Returns the matching question corresponding to the questionID.
- exception: Throws `NotFoundException`

21.4.5 Local Functions

N/A

22 MIS of Repetition Question Bank Module

22.1 Module

RepetitionQuestionBankModule

22.2 Uses

N/A

22.3 Syntax

22.3.1 Exported Constants

N/A

22.3.2 Exported Access Programs

Name	In	Out	Exceptions
storeRepetitionQuestion	question: JSON object	status: bool	StorageException
retrieveRepetitionQuestion	questionID: str	question: JSON object	NotFoundException

22.4 Semantics

22.4.1 State Variables

- repetitionQuestions: Map(questionID, JSON object) - stores matching questions.

22.4.2 Environment Variables

N/A

22.4.3 Assumptions

- Questions have a unique ID.
- Data is stored in a JSON format.

22.4.4 Access Routine Semantics

storeRepetitionQuestion():

- transition: Adds the question to `repetitionQuestions`.
- output: Returns `true` if successfully stored.
- exception: Throws `StorageException` if there is a storage error.

retrieveRepetitionQuestion():

- transition: None
- output: Returns the matching question corresponding to the questionID.
- exception: Throws `NotFoundException`

22.4.5 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.

23 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?

   Promish: I think as a group we were very coordinated and had the important parts, like the module hierarchy diagram, completed before our TA meeting. This allowed us to ask Chris if our design looked good and if there was any feedback he could provide.

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

   Promish: The hardest part of this deliverable is knowing that right after it, we have to build everything within three weeks. I found myself second-guessing between what would make a good design and what would make a feasible design. We overcame this by talking to our supervisor and setting up a good timeline.

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?

   Promish: Most of the design came from prior knowledge of working in co-op as a backend and frontend developer. We also had our supervisor help with the frontend UI design, as there was a mock-up of what she wanted overall or what she didn't like.

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

   Promish: Nothing needed to change as we had already discussed in-depth what we were building and had regular meetups with our supervisor. We referenced the SRS to

ensure that our design met our goals and extended goals, but we didn't have to change any previous document because of our design.

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)

   Promish: I would say a big limitation is the time aspect of making the capstone. I feel like given more time, we would be able to build it like an agile team, so that features are constantly tested with users. We are also limited in terms of getting video metadata, so the accuracy of our video analysis might suffer because of that. Finally, we are also limited by our skills; there are some aspects of the design that we couldn't do because we didn't know how to and didn't have the time to figure it out.

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

   Promish: For our backend, we thought about a monolithic architecture style because it would be easier to implement, but the trade-off is that it's not as maintainable. Our supervisor stressed that maintainability is a big priority for her, so we ended up going with a microservice architecture style. We also considered the various interactions between how the clinician and the parent UI. We previously thought about the parent making an account and the clinician would just search for the parent to add them to their client list. However, we thought it was best if the clinician added the parent to their client list and then gave them a code. This way, we would reduce troll accounts and other security gaps.