Module Interface Specification for Course Buddy

Team #5, Overwatch League
Jingyao, Qin
Qianni, Wang
Qiang, Gao
Chenwei, Song
Shuting, Shi

April 5, 2024

1 Revision History

Date	Version	Notes
2021/1/17	Version 0	Initial draft of the document

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/wangq131/4G06CapstoneProjectT5/blob/main/docs/SRS/SRS.pdf

Contents

3 Introduction

This document outlines the Module Interface Specifications (MIS) for the "Course Buddy" application, an innovative tool designed to streamline the study process for students and educators. By delineating the interactions between the software's modules, this MIS serves as a fundamental component in the development and maintenance of the application, ensuring each module's functionality aligns with the overall system architecture.

The System Requirement Specifications (SRS) and Module Guide are complementary documents that, alongside this MIS, provide a comprehensive understanding of "Course Buddy's" requirements and design. The entire documentation set, including the source code and its most current implementation, is hosted for public access at our GitHub repository: https://github.com/wangq131/4G06CapstoneProjectT5.

In this document, interface specifications are described functionally, with a focus on the inputs, outputs, and data types necessary for module interoperability. This approach provides a clear and direct understanding of module functionalities, preparing the way for detailed implementation strategies, including data structures and algorithmic solutions.

4 Notation

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 Course Buddy.

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 Course Buddy 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, Course Buddy 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 Module	Interface Module
Behaviour-Hiding Module	Back End Web Service Module User Authentication Module App Grid Module Module Task Module Course Module User Module Pomodoro Module Forum Module Feedback Module
Software Decision Module	PDF Extraction Module cGPA Calculation Module Database Module

Table 1: Module Hierarchy

6 MIS of Interface Module

6.1 Module

Interface

6.2 Uses

Back-End Web Service7

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderAuthPage	-	*Boolean*	internetError
renderHomePage	*String*	*Boolean*	internetError
renderCourseDetailPage	*String*	*Boolean*	internetError
renderFeedbackPage	*String*	*Boolean*	internetError
renderForumPage	*String*	*Boolean*	internetError
renderForumTopicPage	*String*	*Boolean*	internetError
renderSearchForumPage	*String*	*Boolean*	internetError
renderPomodoroPage	*String*	*Boolean*	internetError
renderTasksPage	*String*	*Boolean*	internetError
renderUserProfilePage	*String*	*Boolean*	internetError
renderCoursePage	*String*	*Boolean*	internetError

6.4 Semantics

6.4.1 State Variables

userName: *String*
currentPage: *String*
pageTitle: *String*
renderSuccess: *Boolean*

6.4.2 Environment Variables

DBAccessID: *String*
DBAccessCode: *String*

sessionToken: *String*

6.4.3 Assumptions

- The database server is assumed to be available 24/7 with minimal downtime for maintenance
- The volume of the data stored in the database will not exceed the capacity of the database

6.4.4 Access Routine Semantics

renderAuthPage():

- transition: $currentPage :=$ "Authentication Page"
- output: $renderSuccess := pageTitle =$ "Authentication Page"
- exception: internetError if the connection failed.

renderHomePage(*sessionToken*):

- transition: $currentPage :=$ "Home Page"
- output: $renderSuccess := pageTitle =$ "Home Page"
- exception: internetError if the connection failed.

renderCourseDetailPage(*sessionToken*):

- transition: $currentPage :=$ "CourseDetail"
- output: $renderSuccess := pageTitle =$ "CourseDetail"
- exception: internetError if the connection failed.

renderFeedbackPage(*sessionToken*):

- transition: $currentPage :=$ Feedback Page"
- output: $renderSuccess := pageTitle =$ "Feedback Page"
- exception: internetError if the connection failed.

renderForumPage(*sessionToken*):

- transition: $currentPage :=$ Forum Page"
- output: $renderSuccess := pageTitle =$ "Forum Page"

- exception: internetError if the connection failed.

renderForumTopicPage(*sessionToken*):

- transition: $currentPage :=$ ForumTopic Page"
- output: $renderSuccess := pageTitle =$ "ForumTopic Page"
- exception: internetError if the connection failed.

renderSearchForumPage(*sessionToken*):

- transition: $currentPage :=$ SearchForum Page"
- output: $renderSuccess := pageTitle =$ "SearchForum Page"
- exception: internetError if the connection failed.

renderPomodoroPage(*sessionToken*):

- transition: $currentPage :=$ Pomodoro Page"
- output: $renderSuccess := pageTitle =$ "Pomodoro Page"
- exception: internetError if the connection failed.

renderTasksPage(*sessionToken*):

- transition: $currentPage :=$ Tasks Page"
- output: $renderSuccess := pageTitle =$ "Tasks Page"
- exception: internetError if the connection failed.

renderUserProfilePage(*sessionToken*):

- transition: $currentPage :=$ "UserProfile Page"
- output: $renderSuccess := pageTitle =$ "UserProfile Page"
- exception: internetError if the connection failed.

renderCoursePage(*sessionToken*):

- transition: $currentPage :=$ "Course Page"
- output: $renderSuccess := pageTitle =$ "Course Page"
- exception: internetError if the connection failed.

6.4.5 Local Functions

createBackup: *List[csv File]*
createBackup $\equiv$ files

7 MIS of Back-End Web Service Module

7.1 Module

Back-End Web Service

7.2 Uses

User Authentication8, Interface6, App Grid9, User12, Course11, Task10, PDF Extraction16, Feedback15, Pomodoro13, Forum14, cGPA Calculation17,

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
handleRequest	*RequestData*	*ResponseData*	requestError
processData	*Data*	*ProcessedData*	processingError
sendResponse	*ResponseData*	-	responseError
handleException	*ExceptionData*	-	exceptionHandlerError

7.4 Semantics

7.4.1 State Variables

requestQueue: *Queue[RequestData]*
responseData: *ResponseData*

7.4.2 Environment Variables

ServerStorage: *String*
ServerProcessor: *String*

7.4.3 Secrets

Internal Logic and data processing methods

7.4.4 Services

Offers web services for front-end modules, handling requests, responses, and exceptions

7.4.5 Implemented By

Server-side Languages and Principles

7.4.6 Assumptions

- The server is always running and capable of handling multiple simultaneous requests.
- There is a standardized format for requests and responses between the front-end and back-end.

7.4.7 Access Routine Semantics

handleRequest(*requestData*):

- transition:

  Add *requestData* to *requestQueue*

- output:

  The response generated from processing the request

- exception: requestError

processData(*data*):

- transition:

  Process *data* using internal logic

- output:

  ProcessedData

- exception: processingError

sendResponse(*responseData*):

- transition:

  Send *responseData* to the requesting module

- exception: responseError

handleException(*exceptionData*):

- transition:

  Handle *exceptionData* according to server protocols

- exception: exceptionHandlerError

7.4.8 Local Functions

N/A

8 MIS of User Authentication Module

8.1 Module

User Authentication

8.2 Uses

Back-End Web Service7

8.3 Syntax

8.3.1 Exported Constants

sessionToken: *String*

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
verifyAuth	*String*	*Boolean*	AuthError
getSessionToken	-	*String*	internetError
newUser	*String*	*Boolean*	internetError
resetPassword	*String*	*Boolean*	internetError

8.4 Semantics

8.4.1 State Variables

userName: *String*
authSuccess: *Boolean*
sessionToken: *String*
userList: *List[User]*

8.4.2 Environment Variables

DBAccessID: *String*
DBAccessCode: *String*

8.4.3 Assumptions

- The database server is assumed to be available 24/7 with minimal downtime for maintenance
- The volume of the data stored in the database will not exceed the capacity of the database

8.4.4 Access Routine Semantics

verifyAuth(*userName*, *password*):

- transition:

$$\begin{aligned} authSuccess := \exists\, (&user \in users \wedge \\ &user.username = userName \wedge \\ &user.password = password) \end{aligned}$$

- output:

$$\begin{aligned} authSuccess := \exists\, (&user \in users \wedge \\ &user.username = userName \wedge \\ &user.password = password) \end{aligned}$$

- exception: AuthError

getSessionToken():

- output:

$$\begin{aligned} out := &sessionToken \mid \\ &authSuccess = true \\ &or \\ &\text{"No token generated"} \mid \\ &authSuccess = false \end{aligned}$$

- exception: internetError if the connection failed.

newUser(*userName*, *password*):

- transition:

$$userList := userList \cup \{(userName, password)\}$$

- output:

$$out := \begin{cases} true, & \text{if the user is successfully added;} \\ false, & \text{otherwise.} \end{cases}$$

- exception: internetError if the connection failed.

resetPassword(*userName*, *password*):

- transition:

  $\forall user \in userList$, $(user.userName = userName \Rightarrow user.password = newPassword)$

- output:

$$out := \begin{cases} true, & \text{if } \exists user \in userList \text{ with } user.userName = userName; \\ false, & \text{otherwise.} \end{cases}$$

- exception: internetError if the connection failed.

8.4.5 Local Functions

N/A

9 MIS of App Grid Module

9.1 Module

App Grid

9.2 Uses

BackEndWebService7

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
getOrder	-	*List[Icon]*	internetError
updateOorder	*List[Icon]*	*Boolean*	InvalidOrderException

9.4 Semantics

9.4.1 State Variables

iconOrder: *List[Icon]*

9.4.2 Environment Variables

DBAccessID: *String*
DBAccessCode: *String*

9.4.3 Assumptions

- The volume of data stored in the database will not exceed the capacity of the database.
- The user is logged in and has the necessary permissions to modify the app grid.
- The application state is saved after each modification to preserve the icon order.

getOrder():

- output: *out* := iconOrder

- exception: internetError if the connection failed.

updateOrder(*newOrder*):

- transition: iconOrder := *newOrder* after validating the new order.
- output: None
- exception: exc := InvalidOrderException if *newOrder* is not a valid permutation of the iconOrder.

9.4.4 Local Functions

validateOrder(*List[Icon]*):

- output:

$$out := \begin{cases} \text{true}, & \text{if the input list is a valid permutation of iconOrder} \\ \text{false}, & \text{otherwise} \end{cases}$$

- exception: None

10 MIS of Task Module

10.1 Module

Task

10.2 Uses

Pomodoro13, Database18

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
addTask	*String*	*Boolean*	internetError if the connection failed.
updateTask	*String*	*Boolean*	internetError
deleteTask	*String*	*Boolean*	internetError
getTask	*String*	*List[String]*	displayError
switchView	*String*	-	-

10.4 Semantics

10.4.1 State Variables

taskId: *String*
taskType: *String*
courseCode: *String*
taskWeight: *Double*
deadline: *String*
taskList: *List(Task)*

10.4.2 Environment Variables

DBAccessID: *String*
DBAccessCode: *String*

10.4.3 Assumptions

- The volume of the course data stored in the database will not exceed the capacity of the database

10.4.4 Access Routine Semantics

addTask(*taskId*):

- transition: $taskList := (taskList \cup (task \in taskList \mid task.taskid = taskId))$
- output: $out := \exists\,(task \in taskList \mid task.taskId = taskId)$

updateTask(*taskId*,*taskType*,*courseCode*,*taskWeight*,*deadline*):

- transition:

$$\begin{aligned} task \in taskList \mid \\ \quad & task.taskId = taskId \wedge \\ & task.taskType = taskType \wedge \\ & task.courseCode = courseCode \wedge \\ & task.taskWeight = taskWeight \wedge \\ & task.deadline = deadline \end{aligned}$$

- output:

$$\begin{aligned} out := \exists\,(task \in taskList \mid \\ & task.taskId = taskId\,; \\ & task.taskType = taskType\,; \\ & task.courseCode = courseCode\,; \\ & task.taskWeight = taskWeight\,; \\ & task.deadline = deadline) \end{aligned}$$

- exception: internetError if the connection failed.

deleteTask(*taskId*):

- transition: $taskList := (task \setminus task \mid task.taskId = taskId)$
- output: $out := !\ \exists\,(task \in taskList \mid task.taskId = taskId)$
- exception: internetError if the connection failed.

getTask(*taskId*):

- output:

$$out := \{task.taskType, task.courseCode, task.taskWeight, task.deadline \| \\ task \in taskList \wedge task.taskId = taskId\}$$

- exception: internetError if the connection failed.

switchView(*viewType*):

- transition: $currentViewType := viewType$
- output: $out :=$ null
- exception: displayError

10.4.5 Local Functions

N/A

11 MIS of Course Module

11.1 Module

Course

11.2 Uses

Database18

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
addCourse	*String*	*Boolean*	courseAlreadyExist
updateCourse	*String*	*Boolean*	uploadError
deleteCourse	*String*	*Boolean*	courseDoesNotExist
getCourse	*String*	*List[String]*	courseDoesNotExist

11.4 Semantics

11.4.1 State Variables

courseCode: *String*
courseName: *String*
courseInstructor: *String*
emailList: *List[String]*
courseList: *List(Course)*

11.4.2 Environment Variables

DBAccessID: *String*
DBAccessCode: *String*

11.4.3 Assumptions

- The volume of the course data stored in the database will not exceed the capacity of the database

11.4.4 Access Routine Semantics

addCourse(*courseCode*):

- transition: $courseList := (courseList \cup (course \in courseList \mid course.courseCode = courseCode))$
- output: $out := \exists\,(course \in courseList \mid course.courseCode = courseCode)$
- exception: courseAlreadyExist

updateCourse(*courseCode*,*courseName*,*courseInstructor*,*emailList*):

- transition:

$$\begin{aligned} course \in courseList \mid & \\ & course.courseCode = courseCode\,; \\ & course.courseName = courseName\,; \\ & course.courseInstructor = courseInstructor\,; \\ & course.emailList = emailList \end{aligned}$$

- output:

$$\begin{aligned} out := \exists\,(course \in courseList \mid & \\ & course.courseCode = courseCode \wedge \\ & course.courseName = courseName \wedge \\ & course.courseInstructor = courseInstructor \wedge \\ & course.emailList = emailList) \end{aligned}$$

- exception: uploadError

deleteCourse(*courseCode*):

- transition: $courseList := (course \setminus course \mid course.courseCode = courseCode)$
- output: $out := !\,\exists\,(course \in courseList \mid course.courseCode = courseCode)$
- exception: courseDoesNotExist

getCourse(*courseCode*):

- $$\begin{aligned} out := \{&course.courseName, course.courseInstructor, course.emailList \mid \\ &course \in courseList \wedge course.courseCode = courseCode\} \end{aligned}$$
- exception: courseDoesNotExist

11.4.5 Local Functions

N/A

12 MIS of User Module

12.1 Module

UserDB

12.2 Uses

Database Management System (DBMS)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
createUser	userData: UserData	userID: UserID	UserAlreadyExists
queryUser	userID: UserID	userData: UserData	UserNotFound
updateUser	userID: UserID, userData: UserData	-	UserNotFound
deleteUser	userID: UserID	-	UserNotFound

12.4 Semantics

12.4.1 State Variables

userTable: Seq of UserData

12.4.2 Environment Variables

None

12.4.3 Assumptions

User creation is atomic and unique identifiers are generated for each user.

12.4.4 Access Routine Semantics

createUser(userData):

- transition: userTable := userTable
- output: *out* := a unique userID

- exception: *exc* := UserAlreadyExists when there exists some user in userTable with userData.userID

13 MIS of Pomodoro Timer Module

13.1 Module

Task10

13.2 Uses

None

13.3 Syntax

13.3.1 Exported Access Programs

Name	In	Out	Exceptions
startSession	$\mathbb{Z}$	*Void*	*InvalidTimeException*
stopSession	-	*Void*	voidSession-
getRemainingTime	-	$\mathbb{Z}$	voidSession
setWorkDuration	$\mathbb{Z}$	*Void*	*InvalidTimeException*
setBreakDuration	$\mathbb{Z}$	*Void*	*InvalidTimeException*

13.4 Semantics

13.4.1 State Variables

workDuration: $\mathbb{Z}$
breakDuration: $\mathbb{Z}$
remainingTime: $\mathbb{Z}$

13.4.2 Environment Variables

N/A

13.4.3 Assumptions

- It is assumed that the user enters valid time intervals for work and break durations.
- It is assumed that the system clock is accurate and synchronized with real time.
- It is assumed that the module will be used in an environment where regular work-break cycles are beneficial for productivity.

13.4.4 Access Routine Semantics

startSession(*duration*):

- transition:

$$workDuration := duration, \quad remainingTime := duration$$

- output:

$$out := \text{null} \quad \text{if} \quad duration > 0$$

- exception:

$$exc := \begin{cases} InvalidTimeException, & \text{if } duration \leq 0 \\ \text{null}, & \text{otherwise} \end{cases}$$

stopSession():

- transition:

$$remainingTime := 0$$

- output:

$$out := \text{null}$$

- exception: voidSession

getRemainingTime():

- output:

$$out := remainingTime$$

- exception: voidSession

setWorkDuration(*newDuration*):

- transition:

$$workDuration := \begin{cases} newDuration, & \text{if } newDuration > 0 \\ workDuration, & \text{otherwise} \end{cases}$$

- output:

$$out := \text{null}$$

- exception:

$$exc := \begin{cases} InvalidTimeException, & \text{if } newDuration \leq 0 \\ \text{voidSession}, & \text{otherwise} \end{cases}$$

setBreakDuration(*newDuration*):

- transition:

$$breakDuration := \begin{cases} newDuration, & \text{if } newDuration > 0 \\ breakDuration, & \text{otherwise} \end{cases}$$

- output:

$$out := \text{null}$$

- exception:

$$exc := \begin{cases} InvalidTimeException, & \text{if } newDuration \leq 0 \\ \text{voidSession}, & \text{otherwise} \end{cases}$$

13.4.5 Local Functions

None

14 MIS of Forum Module

14.1 Module

Forum

14.2 Uses

DatabaseModule18

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
addTopic	*String, String, User*	*TopicID*	InvalidInputException
addComment	*TopicID, String, User*	*CommentID*	InvalidInputException, TopicNot-FoundException
search	*String*	*List[Topic]*	queryError
reorderTopics	*Ordering*	-	InvalidOrderException

14.4 Semantics

14.4.1 State Variables

topicList: *List[Topic]*
commentList: *List[Comment]*

14.4.2 Environment Variables

forumDisplay: *Display Area on User Interface*

14.4.3 Assumptions

- The database is available and operational.
- The user is authenticated and has the necessary permissions to add topics and comments.

14.4.4 Access Routine Semantics

addTopic(*title*, *content*, *user*):

- transition: topicList := topicList $\cup$ {(*title*, *content*, *user*, new TopicID)}
- output: *out* := new TopicID
- exception: exc := InvalidInputException if title or content are invalid (empty or too long)

addComment(*topicId*, *content*, *user*):

- transition:

  if ($\exists$topic $\in$ topicList | topic.id = *topicId*) then commentList := commentList$\cup${(*topicId*, *content*, *us*

- output: *out* := new CommentID
- exception:

$$\text{exc} := \begin{cases} \text{InvalidInputException}, & \text{if content is invalid} \\ \text{TopicNotFoundException}, & \text{if topicId does not exist in topicList} \end{cases}$$

  search(*query*):

  - transition: None
  - output: *out* := a list of topics that match the query
  - exception: queryError

  reorderTopics(*newOrder*):

  - transition: topicList := reordered topicList based on newOrder
  - output: None
  - exception: exc := InvalidOrderException if newOrder is not a valid ordering of topics

14.4.5 Local Functions

None

15 MIS of Feedback Module

15.1 Module

Feedback

15.2 Uses

BackEndWebService7

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
getFeedback	-	*List[Feedback]*	internetError
submitFeedback	*String, String, String*	*String*	FeedbackSubmissionException

15.4 Semantics

15.4.1 State Variables

feedbackList: *List[Feedback]*

15.4.2 Environment Variables

feedbackForm: *HTML Form*

15.4.3 Assumptions

- The user is logged in and has a valid session when submitting feedback.
- Feedback is stored in a persistent database.
- The user has the necessary permissions to view and submit feedback.

15.4.4 Access Routine Semantics

getFeedback():

- output: *out* := feedbackList
- exception: internetError if the connection failed.

submitFeedback(*username, feedbackType, content*):

- transition: Adds a new feedback entry to the feedbackList and updates the database using DatabaseAccess.
- output: *out* := "Feedback submitted successfully"
- exception: exc := FeedbackSubmissionException if the feedback cannot be submitted.

15.5 Considerations

- Feedback should be moderated to avoid the submission of inappropriate content.
- Rate limiting should be applied to prevent spamming of feedback submissions.
- Personal data within feedback should be handled according to privacy regulations.

16 MIS of PDF Extraction Module

16.1 Module

PDFExtraction

16.2 Uses

BackEndWebService7

16.3 Syntax

16.3.1 Exported Access Programs

Name	In	Out	Exceptions
extractCourseInfo	*String*	*Dictionary*	*fileDoesNotExist*, *InvalidInputException*, *DataExtractionException*
getScoreDistribution	*Course*	*Dictionary*	ScoreDistributionNotFound
getCourseDescription	*Course*	*String*	courseDescriptionNotFound
getInstructorInfo	*Course*	*Dictionary*	instructorInfoNotFound
getTAsInfo	*Course*	*Dictionary*	TAInfoNotFound
communicateWithAPI	*String*	*String*	*fileDoesNotExist*, *APIResponseParsingException*, *APICommunicationException*, *APIAuthenticationException*

16.4 Semantics

16.4.1 State Variables

PDFDocumentContent: *String*
ExtractedCourseInfo: *Dictionary*

16.4.2 Environment Variables

ChatGPTAPIKey: *String*
APIRateLimit: *Integer*

16.4.3 Assumptions

- It is assumed that the format and content of the course syllabus will be consistent.
- It is assumed that the PDF document contains key information about the course and this information is in text form.
- It is assumed that the *ChatGPT* API is available and accessible for information extraction related to natural language processing.
- It is assumed that the module will have access to the Internet at runtime to communicate with and obtain information from the *ChatGPT* API.

16.4.4 Access Routine Semantics

extractCourseInfo(*String*):

- transition: PDFDocumentContent := ExtractContent(*String*)
- output: $out := \exists ci \in CourseInfos \mid Extract(PDFDocumentContent, String) = ci \land formatData($
- exception:
  - $exc := fileDoesNotExist \iff \neg(\exists f \in Files \mid f = String)$
  - $exc := InvalidInputException \iff \neg(validateInput(String))$
  - $exc := DataExtractionException \iff \neg(\exists d \in Data \mid Extract(PDFDocumentContent, St$ $d)$

getScoreDistribution(*Course*):

- transition: None
- output: $out := \exists sd \in ScoreDistributions \mid sd$ corresponds to $Course \land formatData(sd)$
- exception: ScoreDistributionNotFound

getCourseDescription(*Course*):

- transition: None
- output: $out := \exists desc \in Descriptions \mid desc$ corresponds to $Course \land formatData(desc)$
- exception: courseDescriptionNotFound

getInstructorInfo(*Course*):

- transition: None
- output: $out := \exists info \in InstructorInfos \mid info$ corresponds to $Course \wedge formatData(info)$
- exception: instructorInfoNotFound

getTAsInfo(*Course*):

- transition: None
- output: $out := \exists info \in TAsInfos \mid info$ corresponds to $Course \wedge formatData(info)$
- exception: TAInfoNotFound

communicateWithAPI(*String*):

- transition: APIResponses := APIResponses$\cup\{$Communicate$(ChatGPTAPIKey, String)\}$
- output: $out := \exists resp \in APIResponses \mid Communicate(ChatGPTAPIKey, String) = resp \wedge form$
- exception:
  - $exc :=$ *fileDoesNotExist* $\iff \neg(\exists f \in Files \mid f = String)$
  - $exc :=$ *APIResponseParsingException* $\iff \neg(\exists r \in ParsableResponses \mid r = resp)$
  - $exc :=$ *APICommunicationException* $\iff \neg(\exists c \in CommunicableResponses \mid c = resp)$
  - $exc :=$ *APIAuthenticationException* $\iff \neg(\exists a \in AuthenticatedKeys \mid a = ChatGPTAPIKey)$

16.4.5 Local Functions

validateInput(*String*):

- transition: None
- output: isValid := $(String \neq \varnothing) \wedge (String \in$ ValidInputs$)$
- exception: None

formatData(*String*):

- transition: FormattedData := UpdateFormattedData(*String*)
- output:
  - formattedData := {ExtractedData | ExtractedData is derived from *String* and formatted i
  - This includes formatting *String* into structures like:
    - Course information as a *Dictionary*
    - Score distribution as a *Dictionary*
    - Course description as a *String*

        - Instructor information as a *Dictionary*
        - TA information as a *Dictionary*
- exception:
    * $exc :=$ DataFormatException $\iff \neg$(*String* can be correctly parsed and formatted)

17 MIS of CGPA Calculation

17.1 Module

CGPACalculation

17.2 Uses

Back-End Web Service7

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
getCGPA	*File*	$\mathbb{R}$	PDFParseException, CGPACalculationException

17.4 Semantics

17.4.1 State Variables

None

17.4.2 Environment Variables

transcriptFile: *PDF File*

17.4.3 Assumptions

- The transcript PDF is formatted in McMaster transcript form.
- The user has permission to upload and process the transcript.

17.4.4 Access Routine Semantics

getCGPA(*transcript*):

- transition:

  The transcript PDF is parsed, and the relevant course grades and credits are extracted.

  CGPA is calculated using the extracted data.

- output: *out* := calculated CGPA
- exception:

$$\text{exc} := \begin{cases} \text{PDFParseException}, & \text{if the transcript PDF cannot be parsed} \\ \text{CGPACalculationException}, & \text{if the CGPA cannot be calculated from the extracted} \end{cases}$$

17.5 Considerations

- The system should ensure the privacy and security of the uploaded transcripts.
- The cGPA calculation must adhere to the McMaster University's grading scheme.

18 MIS of Database Module

18.1 Module

Database

18.2 Uses

User12, Course11, Task10

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
connectToDB	*String*	*Boolean*	internetError
getDataFile	*String*	*csv File*	fileDoesNotExist
uploadDataFile	*String*	-	internetError
deleteDataFile	*String*	-	fileDoesNotExist

18.4 Semantics

18.4.1 State Variables

files: *List[csv File]*

18.4.2 Environment Variables

DBAccessID: *String*
DBAccessCode: *String*

18.4.3 Assumptions

- The database server is assumed to be available 24/7 with minimal downtime for maintenance
- The volume of the data stored in the database will not exceed the capacity of the database

18.4.4 Access Routine Semantics

connectToDB(*addressOfDB*):

- output: $out := \exists(ip : ipAddress | ip = addressOfDB)$
- exception: internetError if the connection failed.

getDataFile(*fileName*):

- output: $out := file : (file : csvFile | file = fileName \wedge file \in files)$
- exception: fileDoesNotExist

uploadDataFile(*fileName*):

- transition: $files := (files \cup fileName)$
- exception: internetError if the connection failed.

deleteDataFile(*fileName*):

- transition: $files := (files \setminus fileName)$
- exception: fileDoesNotExist

18.4.5 Local Functions

createBackup: *List[csv File]*
createBackup $\equiv$ files

19 Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.
   - UI/UX usability validation tools such as *UserTesting*, *Lookback.io*. to better evaluate our product is user-friendly in a couple of perspectives: effective, learnable, and user-friendly.
   - Dynamic Testing Tools such as *Behave*, which is a tool that allows users to write the test cases in human languages to test for python-system framework.
   - AI Model Validation Frameworks such as *Snitch AI* and *scikit-learn* which can help our trained morel enhance quality and troubleshoot quickly.
   - Static Code Analysis Tools such as *SonarQube* to ensure the code quality which also can be integrated with *CI/CD* for continuous development
   - Enhance continuous delivery/deployment by exploring the *Actions* features in *GitHub* Pro to build custom workflow pipeline.
2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

Knowledge or Skills	**Approaches**	**Assigned Team Member**	**Reason**
UI/UX Usability validation	Use *ChatGPT*, *Google*, watch online tutorials, or ask supervisor for help	Shuting, Shi	Working on the initial UI design, familiar with the key features and the components of website. Therefore, can detect the usability requirements of our target user groups and easy to make modifications accordingly
Dynamic Testing Tools	Use *ChatGPT*, *Google*, watch online tutorials	Qiang, Gao	Have the related experience in the previous co-op work terms, implemented similar functionality in previous project. Strong interest in the dynamic testing section.
AI Model Validation Framework	Use *ChatGPT*, *Google*, watch online tutorials	Qianni, Wang	Experience with many ML projects where these libraries are being used in AI programs and previous co-op work terms. Working on the model training, data-sets selection and integration, familiar with the model algorithm, easy to do modifications if encounters specific model bias.
Static Code Analysis Tools	Use *ChatGPT*, *Google*, watch online tutorials	Chenwei, Song	Experience in enhancing clean code in previous co-op work terms. Strong interest in the code analysis section.
GitHub Action Feature	Use *ChatGPT*, *Google*, and watch online tutorials	Jingyao, Qin	Strong interest in GitHub features, have related experience in the previous coop term, quick to hand on this technique.

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.