Module Interface Specification for Course Buddy

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

January 18, 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 Task Module Course Module User Module Message Notification Module Friend System Module Timetable Module Pomodoro Module
Software Decision Module	PDF Extraction Module Database Module Study Plan Scheduling Module Task Priority Prediction 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
renderModelPage	*String*	*Boolean*	internetError
renderPlanPage	*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, *DBAccessCode*, *sessionToken*

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

renderHomePage(*sessionToken*):

- transition: $currentPage :=$ "Home Page"
- output: $renderSuccess := pageTitle =$ "Home Page"
- exception: internetError

renderModelPage(*sessionToken*):

- transition: $currentPage :=$ "Model Page"
- output: $renderSuccess := pageTitle =$ "Model Page"
- exception: internetError

renderPlanPage(*sessionToken*):

- transition: $currentPage :=$ Plan Page"
- output: $renderSuccess := pageTitle =$ "Plan Page"
- exception: internetError

renderUserProfilePage(*sessionToken*):

- transition: $currentPage :=$ "UserProfile Page"
- output: $renderSuccess := pageTitle =$ "UserProfile Page"
- exception: internetError

renderCoursePage(*sessionToken*):

- transition: $currentPage :=$ "Course Page"
- output: $renderSuccess := pageTitle =$ "Course Page"
- exception: internetError

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

Message Notification18, User Authentication8, Timetable11, Interface6, User17, Course10, Task9, PDF Extraction13, Pomodoro12, Study Plan Scheduling15, Friend19

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*	-	-

7.4 Semantics

7.4.1 State Variables

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

7.4.2 Environment Variables

ServerStorage, *ServerProcessor*

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: None

7.4.8 Local Functions

N/A

8 MIS of User Authentication Module

8.1 Module

Database

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*	-
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, *DBAccessCode*

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: None

getSessionToken():

- output:

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

- exception: None

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: None

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: None

8.4.5 Local Functions

N/A

9 MIS of Task Module

9.1 Module

Task

9.2 Uses

Task Priority Prediction16

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
addTask	*String*	*Boolean*	-
updateTask	*String*	*Boolean*	-
deleteTask	*String*	*Boolean*	-
getTask	*String*	*List[String]*	-

9.4 Semantics

9.4.1 State Variables

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

9.4.2 Environment Variables

DBAccessID, *DBAccessCode*

9.4.3 Assumptions

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

9.4.4 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:

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

- 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: None

deleteTask(*taskId*):

- transition: $taskList := (task \setminus task \mid task.taskId = taskId)$
- output: $out := !\ \exists\,(task \in taskList \mid task.taskId = taskId)$
- exception: None

getTask(*taskId*):

- output:

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

- exception: None

9.4.5 Local Functions

N/A

10 MIS of Course Module

10.1 Module

Course

10.2 Uses

Database14

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

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

10.4 Semantics

10.4.1 State Variables

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

10.4.2 Environment Variables

DBAccessID, *DBAccessCode*

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

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: None

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

10.4.5 Local Functions

N/A

11 MIS of Timetable

11.1 Module

TimeTable

11.2 Uses

BackEndWebService7

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
switchView	*String*	-	-
modifyTimetable	*String, String*	-	InvalidModificationException
exportToGoogleCalendar	-	*String*	ExportFailureException
setPreferredStudyTime	*Time, Time*	-	-
syncWithGoogleCalendar	*Task*	-	SyncFailureException

11.4 Semantics

11.4.1 State Variables

currentViewType: *String*
timetableData: *List[Dictionary]*
preferredStudyTime: *List[Tuple]*

11.4.2 Environment Variables

DBAccessID: *String*
DBAccessCode: *String*
GoogleCalendarAPIKey: *String*

11.4.3 Assumptions

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

11.4.4 Access Routine Semantics

switchView(*viewType*):

- transition: $currentViewType := viewType$
- output: *out* := null
- exception: None

modifyTimetable(*action*, *details*):

- transition:

  if valid(*action*, *details*) then timetableData := *details*

- output: *out* := null
- exception:

$$\text{exc} := \begin{cases} \text{InvalidModificationException}, & \text{if } \neg\text{valid}(action, details) \\ \text{null}, & \text{otherwise} \end{cases}$$

setPreferredStudyTime(*startTime*, *endTime*):

- transition:

  if $startTime < endTime$ then preferredStudyTime := $\{(startTime, endTime)\}$

- output: *out* := null
- exception: None

exportToGoogleCalendar():

- transition:

  if GoogleCalendarAPIKey is available() then send timetableData to Google Calendar

- output: *out* := export status
- exception:

$$\text{exc} := \begin{cases} \text{ExportFailureException}, & \text{if export fails} \\ \text{null}, & \text{otherwise} \end{cases}$$

syncWithGoogleCalendar(*Task*):

- transition:

  if GoogleCalendarAPIKey is available() ⇒ retrieve data from Google Calendar and update timetableData

- output: None
- exception:

$$\text{exc} := \begin{cases} \text{SyncFailureException}, & \text{if sync fails} \\ \text{null}, & \text{otherwise} \end{cases}$$

11.4.5 Local Functions

valid(*action*, *details*):

- output:

$$out := \begin{cases} \text{true}, & \text{if action and details are valid} \\ \text{false}, & \text{otherwise} \end{cases}$$

- exception: None

12 MIS of Pomodoro Timer Module

12.1 Module

PomodoroTimer

12.2 Uses

None

12.3 Syntax

12.3.1 Exported Access Programs

Name	In	Out	Exceptions
startSession	*Integer*	*Void*	*InvalidTimeException*
stopSession	-	*Void*	-
getRemainingTime	-	*Integer*	-
setWorkDuration	*Integer*	*Void*	*InvalidTimeException*
setBreakDuration	*Integer*	*Void*	*InvalidTimeException*

12.4 Semantics

12.4.1 State Variables

workDuration: *Integer*
breakDuration: *Integer*
remainingTime: *Integer*

12.4.2 Environment Variables

N/A

12.4.3 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 notifications for the end of work or break intervals are enabled and can be sent to the user.
- It is assumed that the module will be used in an environment where regular work-break cycles are beneficial for productivity.

12.4.4 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: None

getRemainingTime():

- output:

$$out := remainingTime$$

- exception: None

setWorkDuration(*newDuration*):

- transition:

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

- output:

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

- exception:

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

setBreakDuration(*newDuration*):

- transition:

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

- output:

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

- exception:

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

12.4.5 Local Functions

None

13 MIS of PDF Extraction Module

13.1 Module

PDFExtraction

13.2 Uses

BackEndWebService7

13.3 Syntax

13.3.1 Exported Access Programs

Name	In	Out	Exceptions
extractCourseInfo	*String*	*Dictionary*	*fileDoesNotExist*, *InvalidInputException*, *DataExtractionException*
getScoreDistribution	*Course*	*Dictionary*	-
getCourseDescription	*Course*	*String*	-
getInstructorInfo	*Course*	*Dictionary*	-
getTAsInfo	*Course*	*Dictionary*	-
communicateWithAPI	*String*	*String*	*fileDoesNotExist*, *APIResponsePArsingException*, *APICommunicationException*, *APIAuthenticationException*

13.4 Semantics

13.4.1 State Variables

PDFDocumentContent: *String*
ExtractedCourseInfo: *Dictionary*

13.4.2 Environment Variables

ChatGPTAPIKey: *String*
APIRateLimit: *Integer*

13.4.3 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.

13.4.4 Access Routine Semantics

extractCourseInfo(*String*):

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

getScoreDistribution(*Course*):

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

getCourseDescription(*Course*):

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

getInstructorInfo(*Course*):

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

- exception: None

getTAsInfo(*Course*):

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

communicateWithAPI(*String*):

- transition: APIResponses := APIResponses$\cup${Communicate($ChatGPTAPIKey, String$)}
- output: *out* := $\exists$*resp* $\in$ *APIResponses* | *Communicate*(*ChatGPTAPIKey*, *String*) = *resp* $\wedge$ *formatDa*
- 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* $\iff$ $\neg(\exists a \in AuthenticatedKeys \mid a = ChatGPTAPIKey)$

13.4.5 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 into c
  - 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)

14 MIS of Database Module

14.1 Module

Database

14.2 Uses

User17, Course10, Task9

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

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

14.4 Semantics

14.4.1 State Variables

files: *List[csv File]*

14.4.2 Environment Variables

DBAccessID, *DBAccessCode*

14.4.3 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

14.4.4 Access Routine Semantics

connectToDB(*addressOfDB*):

- output: $out := \exists(ip : ipAddress | ip = addressOfDB)$
- exception: None

getDataFile(*fileName*):

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

uploadDataFile(*fileName*):

- transition: $files := (files \cup fileName)$
- exception: None

deleteDataFile(*fileName*):

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

14.4.5 Local Functions

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

15 MIS of Study Plan Scheduling Module

15.1 Module

Study Plan Scheduling

15.2 Uses

None

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
getPlan	*set[Task]*	*dict[timeSlot: Task]*	-

15.4 Semantics

15.4.1 State Variables

timeSlotsMap: *dict[timeSlot: Task]*

15.4.2 Environment Variables

None

15.4.3 Assumptions

- The interval of time slot is fixed
- The number of tasks are always less than the number of time slots that the timetable have

15.4.4 Access Routine Semantics

getPlan(*tasks*):

- transition: timeSlotsMap = Null $\implies$ $\forall task : Task \in Tasks : \exists timeSlotsMap[i : timeSlot] = task$
- exception: None

15.4.5 Local Functions

createTimeSlot(interval: int): *List[timeSlot]*

16 MIS of Task Priority Prediction Module

16.1 Module

Task Priority Prediction

16.2 Uses

None

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
getPriority	*Task*	*String*	-

16.4 Semantics

16.4.1 State Variables

priorities: *Enum[String]*

16.4.2 Environment Variables

None

16.4.3 Assumptions

- Assume that the level of priorities are well defined
- Assume that task priorities are consistently evaluated based on these criteria

16.4.4 Access Routine Semantics

getPriority(*task*):

- output: priority[i] where

  For each task i, and for each field $j = 1, 2, ..., \text{length(task.fields)}$, with weights $weight[j] \in \mathbb{R}$:

  $$\sum_{j} (\text{weight}[j] \times \text{field}[j]) \implies \text{priority}[i]$$

- exception: None

16.4.5 Local Functions

None

17 MIS of User Module

17.1 Module

UserDB

17.2 Uses

Database Management System (DBMS)

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 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

17.4 Semantics

17.4.1 State Variables

userTable: Seq of UserData

17.4.2 Environment Variables

None

17.4.3 Assumptions

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

17.4.4 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

18 MIS of Message Notification Module

18.1 Module

MessageNotification

18.2 Uses

None

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
sendNotification	userID: UserID, message: String	-	UserNotFound

18.4 Semantics

18.4.1 State Variables

notificationQueue: Queue of Notification

18.4.2 Environment Variables

None

18.4.3 Assumptions

Notifications are sent in real-time and users are online to receive them.

18.4.4 Access Routine Semantics

sendNotification(userID, message):

- transition: notificationQueue := notificationQueue.enqueue({userID, message})
- exception: *exc* := UserNotFound when no user exists with userID

19 MIS of Friend System Module

19.1 Module

FriendSystem

19.2 Uses

Back-End Web Service7

19.3 Syntax

19.3.1 Exported Constants

None

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
addFriend	userID: UserID, friendID: UserID	-	UserNotFound, AlreadyFriends
removeFriend	userID: UserID, friendID: UserID	-	UserNotFound, NotFriends
sendMessage	userID: UserID, friendID: UserID, message: String	-	UserNotFound, NotFriends

19.4 Semantics

19.4.1 State Variables

friendList: Map of UserID to List of UserID

19.4.2 Environment Variables

None

19.4.3 Assumptions

Users can only send messages to users who are already their friends.

19.4.4 Access Routine Semantics

addFriend(userID, friendID):

- transition: friendList[userID] := friendList[userID] ∪ {friendID}

- exception: *exc* := UserNotFound when no user exists with friendID

removeFriend(userID, friendID):

- transition: friendList[userID] := friendList[userID] \ {friendID}
- exception: *exc* := UserNotFound when no user exists with friendID

sendMessage(userID, friendID, message):

- transition: *A message is sent to friendID from userID with content message.*
- exception: *exc* := UserNotFound when no user exists with friendID, NotFriends if friendID is not in friendList[userID]

20 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.