Module Interface Specification for Software Engineering

Team #13, ARC
Avanish Ahluwalia
Russell Davidson
Rafey Malik
Abdul Zulfiqar

January 18, 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 [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

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 **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. 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	
Behaviour-Hiding	Input Parameters Output Format Output Verification Temperature ODEs Energy Equations Control Module Specification Parameters Module
Software Decision	Sequence Data Structure ODE Solver Plotting

Table 1: Module Hierarchy

6 MIS of Settings Module

6.1 Module

Settings Module

6.2 Uses

- Database/Network Manager Module
- Error Manager Module
- Authentication Module

6.3 Syntax

6.3.1 Exported Constants

- None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
UpdateSettings	Dictionary of key-value pairs	Bool - Success or Fail	InvalidKeyException
FetchSettings	-	Dictionary of current settings	-
ResetToDefaults	-	Bool - Success or Fail	-
UpdateProfileDetails	Dictionary of profile-related key-value pairs	Bool - Success or Fail	InvalidKeyException
FetchProfileDetails	-	Dictionary of profile-related data	-
ChangePassword	Old password, new password	Bool - Success or Fail	AuthenticationException

6.4 Semantics

6.4.1 State Variables

- settings: A dictionary containing user-configurable settings and their current values.
- profileDetails: A dictionary storing user profile-related data (e.g., name, avatar, bio).

6.4.2 Environment Variables

- Device Storage: Used to persist settings and profile details.
- Encryption Mechanism: Ensures sensitive data like passwords and privacy settings are encrypted during storage and transmission.

6.4.3 Assumptions

- The device has sufficient storage for saving and retrieving settings and profile details.
- All keys provided for updates are predefined and valid.
- The user is authenticated before accessing this module.

6.4.4 Access Routine Semantics

UpdateSettings(newSettings):

- **Transition:** Updates the corresponding entries in the settings dictionary with the provided key-value pairs.
- **Output:** Returns true if all updates succeed, false otherwise.
- **Exception:** Throws InvalidKeyException if an invalid key is provided.

FetchSettings():

- **Output:** Returns the current settings dictionary.
- **Exception:** None

ResetToDefaults():

- **Transition:** Resets all settings in the dictionary to their default values.
- **Output:** Returns true if the reset succeeds, false otherwise.
- **Exception:** None

UpdateProfileDetails(newProfileDetails):

- **Transition:** Updates the corresponding entries in the profileDetails dictionary with the provided key-value pairs.
- **Output:** Returns true if all updates succeed, false otherwise.
- **Exception:** Throws InvalidKeyException if an invalid key is provided.

FetchProfileDetails():

- **Output:** Returns the current profileDetails dictionary.
- **Exception:** None

ChangePassword(oldPassword, newPassword):

- **Transition:** Validates the old password and updates the password to the new one if valid.
- **Output:** Returns true if the password is successfully changed, false otherwise.
- **Exception:** Throws AuthenticationException if the old password is incorrect or the user session is invalid.

6.4.5 Local Functions

- ValidateKey(key): Ensures the provided key is predefined and valid for the settings or profileDetails dictionaries.
- EncryptData(data): Applies encryption to sensitive data before storage or transmission.

7 MIS of Help Module

7.1 Module

Help Module

7.2 Uses

- User Interface Manager Module
- Content Storage Module

7.3 Syntax

7.3.1 Exported Constants

- None

7.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
ViewHelpTopics	-	List of help topics	-
SearchHelp	Query string	Relevant help content	InvalidQueryException
FetchFAQ	-	List of frequently asked questions	-
ContactSupport	User message, user contact info	Confirmation of message sent	MessageDeliveryException
SubmitFeedback	Feedback message	Bool - Success or Fail	FeedbackSubmissionException

7.4 Semantics

7.4.1 State Variables

- helpSections: A collection of predefined help sections, each identified by a unique ID and containing text, images, or links.
- searchIndex: An index to facilitate quick searches within help content.

7.4.2 Environment Variables

- Content Storage: Stores the help documentation (e.g., FAQs, guides, tutorials).
- Device Network: Used for fetching online help content if local content is not available.

7.4.3 Assumptions

- All help content is preloaded in the Content Storage or accessible via a network connection.
- The section identifiers used are valid and correspond to existing help sections.

7.4.4 Access Routine Semantics

FetchHelpContent(sectionID):

- **Transition:** None
- **Output:** Returns the help content associated with the given section identifier.
- **Exception:** Throws ContentNotFoundException if the sectionID is invalid or not found.

SearchHelpContent(query):

- **Transition:** None
- **Output:** Returns a list of section identifiers that are relevant to the search query.
- **Exception:** None

NavigateToHelpSection(sectionID):

- **Transition:** Updates the displayed help section in the user interface to the section corresponding to the given identifier.
- **Output:** Returns true if navigation is successful, false otherwise.
- **Exception:** Throws InvalidSectionException if the sectionID is invalid.

7.4.5 Local Functions

- ValidateSectionID(sectionID): Ensures the section identifier is valid and corresponds to an existing help section.
- PerformSearch(query): Matches the search query against the searchIndex and returns relevant section identifiers.

8 MIS of Friends Module

8.1 Module

Friends Module

8.2 Uses

- User Interface Manager Module
- Database Manager Module
- Notification Module

8.3 Syntax

8.3.1 Exported Constants

- None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
AddFriend	User ID of friend	Bool - Success or Fail	FriendRequestException
RemoveFriend	User ID of friend	Bool - Success or Fail	FriendNotFoundException
FetchFriendList	-	List of friends	-
AcceptFriendRequest	Request ID	Bool - Success or Fail	InvalidRequestException
RejectFriendRequest	Request ID	Bool - Success or Fail	InvalidRequestException
BlockUser	User ID of user to block	Bool - Success or Fail	BlockException

8.4 Semantics

8.4.1 State Variables

- friendsList: A list of user identifiers representing the user's current friends.
- pendingRequests: A list of incoming and outgoing friend requests.

8.4.2 Environment Variables

- Database: Stores the user's friends and pending requests.
- Notification System: Sends notifications for friend requests and updates.

8.4.3 Assumptions

- All user identifiers provided are valid and correspond to registered users.
- The database connection is available and operational during module use.

8.4.4 Access Routine Semantics

AddFriend(userID):

- **Transition:** Adds the specified user to the friends list if the request is valid and approved.
- **Output:** Returns true if the friend is successfully added, false otherwise.
- **Exception:** Throws UserNotFoundException if the specified userID does not exist.

RemoveFriend(userID):

- **Transition:** Removes the specified user from the friends list.
- **Output:** Returns true if the friend is successfully removed, false otherwise.
- **Exception:** Throws UserNotFoundException if the specified userID does not exist.

FetchFriendsList():

- **Output:** Returns a list of user identifiers representing the user's current friends.
- **Exception:** None

SendFriendRequest(userID):

- **Transition:** Adds a new friend request to the pendingRequests list.
- **Output:** Returns true if the request is successfully sent, false otherwise.
- **Exception:** Throws UserNotFoundException if the specified userID does not exist.

RespondToFriendRequest(requestID, response):

- **Transition:** Updates the status of the specified friend request (accept or reject).
- **Output:** Returns true if the response is successfully recorded, false otherwise.
- **Exception:** Throws RequestNotFoundException if the specified requestID does not exist.

8.4.5 Local Functions

- ValidateUserID(userID): Ensures the user identifier corresponds to a valid registered user.
- NotifyUser(userID, message): Sends a notification to the specified user.

9 MIS of Collision Hazard Detection Module

9.1 Module

Collision Hazard Detection Module

9.2 Uses

- AR Object Manager Module
- Device Sensors Module (e.g., Camera, LiDAR)
- Notification Module

9.3 Syntax

9.3.1 Exported Constants

- None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
DetectCollision	User's current position, AR object positions	Boolean (Collision Detected/Not Detected)	SensorDataUnavailableException
FetchHazardDetails	Collision ID	Hazard details (Type, Severity, Location)	HazardNotFoundException
LogCollisionEvent	Collision details (Position, Time, Severity)	Bool - Success or Fail	LoggingException
ClearHazardAlerts	-	Bool - Success or Fail	AlertClearanceException
UpdateDetectionSettings	Dictionary of detection settings	Bool - Success or Fail	InvalidSettingsException

9.4 Semantics

9.4.1 State Variables

- collisionEvents: A log of detected collisions, including timestamps, object data, and resolutions.

9.4.2 Environment Variables

- Device Sensors: Provides real-time data for collision detection (e.g., LiDAR, camera).
- Notification System: Sends alerts when collisions are detected.
- Storage System: Logs collision events for future analysis.

9.4.3 Assumptions

- Device sensors are operational and capable of providing accurate real-time data.
- The system has access to sufficient storage for logging collision events.
- All objects in the AR environment are registered and have defined collision boundaries.

9.4.4 Access Routine Semantics

DetectCollision(sensorData):

- **Transition:** None
- **Output:** Returns true if a collision is detected based on sensor data, false otherwise.
- **Exception:** Throws SensorUnavailableException if real-time data cannot be accessed.

ResolveCollision(collisionData):

- **Transition:** Attempts to resolve the collision by adjusting object positions or notifying the user.
- **Output:** Returns true if the collision is successfully resolved, false otherwise.
- **Exception:** Throws CollisionNotResolvableException if the collision cannot be resolved.

LogCollisionEvent(eventData):

- **Transition:** Records the collision event in the system log.
- **Output:** Returns true if the event is successfully logged, false otherwise.
- **Exception:** Throws LogWriteException if the log cannot be updated.

9.4.5 Local Functions

- AnalyzeSensorData(sensorData): Processes real-time data to determine if a collision is imminent.
- NotifyUserOfCollision(collisionData): Sends an alert to the user about a detected collision.

10 MIS of Tour Proximity Detection Module

10.1 Module

Tour Proximity Detection Module

10.2 Uses

- GPS Module
- Notification Module
- AR Object Manager Module

10.3 Syntax

10.3.1 Exported Constants

- None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
DetectNearbyTours	User's current location, registered tour locations	List of tours within proximity	LocationDataUnavailableException
NotifyTourProximity	User's proximity to a specific tour	Notification message	NotificationFailureException
UpdateTourProximity	Updated user location, updated tour locations	Bool - Success or Fail	UpdateFailedException
FetchProximityDetails	User ID, tour ID	Details of proximity data	DataFetchException

10.4 Semantics

10.4.1 State Variables

- proximityEvents: A log of detected proximity events, including timestamps, tour data, and notifications sent.
- currentLocation: The user's most recently reported GPS coordinates.

10.4.2 Environment Variables

- GPS System: Provides the user's real-time location.
- Notification System: Sends alerts when a user is near a tour location.
- Storage System: Logs proximity detection events for future analysis.

10.4.3 Assumptions

- The GPS system is operational and capable of providing accurate location data.
- The system has sufficient storage for logging proximity detection events.
- All tours in the system have clearly defined geographical boundaries.

10.4.4 Access Routine Semantics

DetectNearbyTour(currentLocation):

- **Transition:** None
- **Output:** Returns a list of tours within proximity to the user's current location.
- **Exception:** Throws LocationUnavailableException if GPS data cannot be accessed.

TriggerTourNotification(tourData):

- **Transition:** Sends a notification to the user about the nearby tour.
- **Output:** Returns true if the notification is successfully sent, false otherwise.
- **Exception:** Throws NotificationSendException if the notification fails to send.

LogProximityEvent(eventData):

- **Transition:** Records the proximity event in the system log.
- **Output:** Returns true if the event is successfully logged, false otherwise.
- **Exception:** Throws LogWriteException if the log cannot be updated.

10.4.5 Local Functions

- AnalyzeLocationData(locationData): Processes the current GPS location to identify nearby tours.
- NotifyUserOfProximity(tourData): Sends an alert to the user about a detected nearby tour.

11 MIS of Notifications Module

11.1 Module

Notifications Module

11.2 Uses

- User Interface Manager Module
- Database Manager Module

11.3 Syntax

11.3.1 Exported Constants

- None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
SendNotification	User ID, notification message	Bool - Success or Fail	NotificationSendException
FetchNotifications	User ID	List of notifications	DataFetchException
MarkNotificationRead	Notification ID	Bool - Success or Fail	NotificationNotFoundException
DeleteNotification	Notification ID	Bool - Success or Fail	DeleteFailedException
UpdateNotificationSettings	User ID, settings data	Bool - Success or Fail	InvalidSettingsException

11.4 Semantics

11.4.1 State Variables

- notifications: A list of notifications associated with each user, including their status (read/unread).

11.4.2 Environment Variables

- Notification System: Handles the actual delivery of notifications to users.
- Database: Stores notifications for retrieval and management.

11.4.3 Assumptions

- The notification delivery service is operational and capable of sending notifications in real-time.
- All user identifiers and notification identifiers are valid and exist in the database.

11.4.4 Access Routine Semantics

SendNotification(notificationData):

- **Transition:** Adds the notification to the database and attempts to deliver it to the specified user.
- **Output:** Returns true if the notification is successfully sent, false otherwise.
- **Exception:** Throws NotificationSendException if the delivery fails.

FetchNotifications(userID):

- **Output:** Returns a list of notifications for the specified user.
- **Exception:** Throws UserNotFoundException if the user ID is not found.

MarkNotificationAsRead(notificationID):

- **Transition:** Updates the status of the specified notification to "read" in the database.
- **Output:** Returns true if the status update is successful, false otherwise.
- **Exception:** Throws NotificationNotFoundException if the notification ID is not found.

DeleteNotification(notificationID):

- **Transition:** Removes the specified notification from the database.
- **Output:** Returns true if the notification is successfully deleted, false otherwise.
- **Exception:** Throws NotificationNotFoundException if the notification ID is not found.

11.4.5 Local Functions

- ValidateNotificationData(notificationData): Ensures the notification data is valid before sending.
- NotifyUser(notificationData): Sends the notification to the user using the delivery system.

12 MIS of Authentication Module

12.1 Module

Authentication Module

12.2 Uses

- Database Manager Module
- Encryption Module
- Notification Module (for two-factor authentication)

12.3 Syntax

12.3.1 Exported Constants

- None

12.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
SendNotification	User ID, notification message	Bool - Success or Fail	NotificationSendException
FetchNotifications	User ID	List of notifications	DataFetchException
MarkNotificationRead	Notification ID	Bool - Success or Fail	NotificationNotFoundException
DeleteNotification	Notification ID	Bool - Success or Fail	DeleteFailedException
UpdateNotificationSettings	User ID, settings data	Bool - Success or Fail	InvalidSettingsException

12.4 Semantics

12.4.1 State Variables

- users: A collection of user data, including credentials, authentication settings, and two-factor status.
- activeSessions: A list of currently authenticated user sessions.

12.4.2 Environment Variables

- Database: Stores user credentials and authentication-related data.
- Encryption System: Encrypts sensitive user information like passwords.
- Notification System: Sends two-factor authentication codes or password reset links.

12.4.3 Assumptions

- All usernames and passwords are stored securely and hashed using industry-standard encryption.
- The system has access to a functional notification system for delivering codes and reset links.
- Users provide valid inputs during registration and login attempts.

12.4.4 Access Routine Semantics

AuthenticateUser(username, password):

- **Transition:** Verifies the username and password against stored credentials.
- **Output:** Returns true if the credentials are valid, false otherwise.
- **Exception:** Throws AuthenticationFailedException if the credentials do not match.

RegisterUser(registrationData):

- **Transition:** Adds a new user to the database with the provided registration data.
- **Output:** Returns true if the registration is successful, false otherwise.
- **Exception:** Throws RegistrationFailedException if the registration fails due to invalid data or duplication.

ResetPassword(emailOrUsername):

- **Transition:** Sends a password reset link or code to the associated email.
- **Output:** Returns true if the reset link/code is sent successfully, false otherwise.
- **Exception:** Throws UserNotFoundException if the email or username is not found.

EnableTwoFactorAuth(userID):

- **Transition:** Enables two-factor authentication for the specified user.
- **Output:** Returns true if the feature is enabled successfully, false otherwise.

- **Exception:** Throws UserNotFoundException if the user ID is invalid.

ValidateTwoFactorCode(userID, code):

- **Transition:** Validates the two-factor authentication code provided by the user.
- **Output:** Returns true if the code is valid, false otherwise.
- **Exception:** Throws InvalidCodeException if the code is incorrect or expired.

12.4.5 Local Functions

- HashPassword(password): Generates a secure hash for the given password.
- GenerateTwoFactorCode(userID): Creates a time-sensitive code for two-factor authentication.
- SendNotification(userID, message): Sends a notification to the specified user with relevant authentication details.

13 MIS of Object Render Module

13.1 Module

Object Render Module

13.2 Uses

- Maps Module
- Object Placement Module

13.3 Syntax

13.3.1 Exported Constants

- **RENDER_RESOLUTION_DEFAULT**: Default resolution for rendering objects.
- **RENDER_FPS_LIMIT**: Frame-per-second limit for rendering.
- **RENDER_QUALITY_OPTIONS**: Preset quality levels (e.g., low, medium, high).

13.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
RenderObject	Object ID, Position, Orientation	Rendered Object	RenderingException
AdjustRenderSettings	Dictionary of key-value pairs	Bool - Success or Fail	InvalidSettingException
FetchRenderSettings	-	Dictionary of current render settings	-
PauseRendering	-	Bool - Success or Fail	RenderingException
ResumeRendering	-	Bool - Success or Fail	RenderingException

13.4 Semantics

13.4.1 State Variables

- **currentRenderSettings**: Stores the current rendering settings, such as resolution and FPS.
- **renderQueue**: A queue of objects to be rendered.

13.4.2 Environment Variables

- Graphics Processing Unit (GPU)
- Rendering Library (e.g., OpenGL, Vulkan, Unity Renderer)

13.4.3 Assumptions

- Objects to be rendered are correctly formatted and preprocessed.
- The rendering hardware meets the minimum requirements.

13.4.4 Access Routine Semantics

RenderObject(Object ID, Position, Orientation):

- transition: Adds the object to the render queue and renders it based on the given parameters.
- output: The rendered object appears in the virtual environment.
- exception: Throws `RenderingException` if the rendering fails due to hardware or software issues.

AdjustRenderSettings(Dictionary of key-value pairs):

- transition: Updates the `currentRenderSettings` variable with the provided values.
- output: Returns a success or failure boolean.
- exception: Throws `InvalidSettingException` if the provided settings are invalid.

FetchRenderSettings():

- transition: None.
- output: Returns the `currentRenderSettings`.
- exception: None.

PauseRendering():

- transition: Pauses the rendering process.
- output: Returns a success or failure boolean.
- exception: Throws `RenderingException` if pausing fails.

ResumeRendering():

- transition: Resumes the rendering process.
- output: Returns a success or failure boolean.
- exception: Throws `RenderingException` if resuming fails.

13.4.5 Local Functions

- **ValidateRenderSettings(settings)**: Ensures that the given render settings are within acceptable ranges.
- **OptimizeRenderQueue()**: Reorders the render queue to improve performance.

14 MIS of Touring Module

14.1 Module

Touring Module

14.2 Uses

Realm Interface Module, Maps Module, Notifications Module

14.3 Syntax

14.3.1 Exported Constants

- **DEFAULT_TOUR_RADIUS:** The default radius for proximity detection during tours.

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
StartTour	Tour ID	Bool - Success or Fail	InvalidTourIDException
PauseTour	-	Bool - Success or Fail	-
EndTour	Tour ID	Bool - Success or Fail	InvalidTourIDException
FetchTourDetails	Tour ID	Tour Object	InvalidTourIDException

14.4 Semantics

14.4.1 State Variables

- **CurrentTour:** Stores the details of the ongoing tour.

14.4.2 Environment Variables

GPS and Maps API for location tracking.

14.4.3 Assumptions

It is assumed that GPS and Maps API services are functional.

14.4.4 Access Routine Semantics

StartTour:

- transition: Initializes the tour with the given Tour ID and marks it as active.
- output: Returns success if the tour starts successfully.

- exception: Throws InvalidTourIDException if the Tour ID does not exist.

PauseTour:

- transition: Pauses the current active tour.
- output: Returns success if the tour is paused successfully.

EndTour:

- transition: Ends the current active tour and updates its status.
- output: Returns success if the tour is ended successfully.
- exception: Throws InvalidTourIDException if the Tour ID does not exist.

15 MIS of Tour List Module

15.1 Module

Tour List Module

15.2 Uses

Touring Module, Maps Module

15.3 Syntax

15.3.1 Exported Constants

- **MAX_TOUR_ENTRIES:** The maximum number of tours that can be displayed in the list.

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
FetchTourList	None	Array of Tours	-
SearchTours	Search Query	Array of Tours	-
SortTours	Sorting Criteria	Array of Tours	-

15.4 Semantics

15.4.1 State Variables

- **TourList:** Stores a list of available tours.

15.4.2 Environment Variables

Database connection to retrieve available tours.

15.4.3 Assumptions

It is assumed that the database is functional and contains valid tour data.

15.4.4 Access Routine Semantics

FetchTourList:

- transition: Retrieves all available tours from the database.
- output: Returns an array of available tours.

SearchTours:

- transition: Filters the available tours based on the search query.
- output: Returns an array of tours matching the search criteria.

SortTours:

- transition: Sorts the available tours based on the specified criteria.
- output: Returns an array of tours sorted by the given criteria.

16 MIS of Tour Management Module

16.1 Module

Tour Management Module

16.2 Uses

Realm Interface Module, Server Database Manager Module

16.3 Syntax

16.3.1 Exported Constants

- **MAX_TOUR_POINTS:** The maximum number of waypoints allowed in a single tour.

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
CreateTour	Tour Object	Bool - Success or Fail	InvalidTourDataException
UpdateTour	Tour ID, Updated Tour Data	Bool - Success or Fail	InvalidTourIDException
DeleteTour	Tour ID	Bool - Success or Fail	InvalidTourIDException

16.4 Semantics

16.4.1 State Variables

- **ManagedTours:** Stores a list of tours created and managed by the organization.

16.4.2 Environment Variables

Database connection to manage tour data.

16.4.3 Assumptions

It is assumed that the database is functional and contains valid tour data.

16.4.4 Access Routine Semantics

CreateTour:

- transition: Adds a new tour to the database.

- output: Returns success if the tour is created successfully.
- exception: Throws InvalidTourDataException if the tour data is invalid.

UpdateTour:

- transition: Updates the details of an existing tour in the database.
- output: Returns success if the tour is updated successfully.
- exception: Throws InvalidTourIDException if the Tour ID does not exist.

DeleteTour:

- transition: Removes a tour from the database.
- output: Returns success if the tour is deleted successfully.
- exception: Throws InvalidTourIDException if the Tour ID does not exist.

References

17 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?

   The modules were distributed relatively well according to what each group member seems most comfortable and experienced with.

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

   Deciding the modules initially was a bit of a challenge, but after speaking with our TA, it helped us decide which modules to split and which ones to combine.

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

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

   We did not need to change any parts of other documents.

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

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