Module Interface Specification for Software Engineering

Team 4, EvENGage
Virochaan Ravichandran Gowri
Omar Al-Asfar
Rayyan Suhail
Ibrahim Quraishi
Mohammad Mahdi Mahboob

January 28, 2026

1 Revision History

Date	Version	Notes
November 13, 2025	1.0	Revision -1
January 21, 2026	1.1	Revision 0: Added formal state and transition specifications.
January 21, 2026	1.1	Revision 0: Added all reflections and grammar fixes, removed all scaffolding.

2 Symbols, Abbreviations and Acronyms

Glossary

admin A privileged system user affiliated with McMaster Engineering Society (MES) who can access survey, event, and attendee records; track finances; and update event statuses. plural.

attendee An individual who attends an MES event, and uses the system to register, receive event updates, provide feedback, or fill out survey responses. plural.

Acronyms

CFES Canadian Federation of Engineering Students.

MES McMaster Engineering Society.

Contents

1 Revision History i

2 Symbols, Abbreviations and Acronyms ii

3 Introduction 1

4 Notation 1

5 Module Decomposition 2

6 MIS of M1: Main System Module 3
- 6.1 Module . . . 3
- 6.2 Uses . . . 3
- 6.3 Syntax . . . 3
  - 6.3.1 Exported Constants . . . 3
  - 6.3.2 Exported Access Programs . . . 3
- 6.4 Semantics . . . 3
  - 6.4.1 State Variables . . . 3
  - 6.4.2 Environment Variables . . . 3
  - 6.4.3 Assumptions . . . 3
  - 6.4.4 Access Routine Semantics . . . 3
  - 6.4.5 Local Functions . . . 3

7 MIS of M2: User Authentication Module 4
- 7.1 Module . . . 4
- 7.2 Uses . . . 4
- 7.3 Syntax . . . 4
  - 7.3.1 Exported Constants . . . 4
  - 7.3.2 Exported Access Programs . . . 4
- 7.4 Semantics . . . 4
  - 7.4.1 State Variables . . . 4
  - 7.4.2 Environment Variables . . . 4
  - 7.4.3 Assumptions . . . 4
  - 7.4.4 Access Routine Semantics . . . 5
  - 7.4.5 Local Functions . . . 5

8 MIS of M3: User Authorization Module 7
- 8.1 Module . . . 7
- 8.2 Uses . . . 7
- 8.3 Syntax . . . 7
  - 8.3.1 Exported Constants . . . 7
  - 8.3.2 Exported Access Programs . . . 7

8.4 Semantics . . . 7
8.4.1 State Variables . . . 7
8.4.2 Environment Variables . . . 7
8.4.3 Assumptions . . . 7
8.4.4 Access Routine Semantics . . . 7
8.4.5 Local Functions . . . 8

9 MIS of M4: Form Template Module **9**
9.1 Module . . . 9
9.2 Uses . . . 9
9.3 Syntax . . . 10
9.3.1 Exported Constants . . . 10
9.3.2 Exported Access Programs . . . 10
9.4 Semantics . . . 10
9.4.1 State Variables . . . 10
9.4.2 Environment Variables . . . 11
9.4.3 Assumptions . . . 11
9.4.4 Access Routine Semantics . . . 11
9.4.5 Local Functions . . . 13

10 MIS of M5: Form Submission Module **14**
10.1 Module . . . 14
10.2 Uses . . . 14
10.3 Syntax . . . 14
10.3.1 Exported Constants . . . 14
10.3.2 Exported Access Programs . . . 14
10.4 Semantics . . . 14
10.4.1 State Variables . . . 14
10.4.2 State Invariants . . . 14
10.4.3 Environment Variables . . . 15
10.4.4 Assumptions . . . 15
10.4.5 Access Routine Semantics . . . 15
10.4.6 Local Functions . . . 16

11 MIS of M6: Event Management Module **17**
11.1 Module . . . 17
11.2 Uses . . . 17
11.3 Syntax . . . 17
11.3.1 Exported Constants . . . 17
11.3.2 Exported Access Programs . . . 17
11.4 Semantics . . . 17
11.4.1 State Variables . . . 17
11.4.2 Environment Variables . . . 17

11.4.3 Assumptions . . . 17
11.4.4 Access Routine Semantics . . . 17
11.4.5 Local Functions . . . 18

12 MIS of M7: Event Notification Module **19**
12.1 Module . . . 19
12.2 Uses . . . 19
12.3 Syntax . . . 19
12.3.1 Exported Constants . . . 19
12.3.2 Exported Access Programs . . . 19
12.4 Semantics . . . 19
12.4.1 State Variables . . . 19
12.4.2 Environment Variables . . . 19
12.4.3 Assumptions . . . 19
12.4.4 Access Routine Semantics . . . 19
12.4.5 Local Functions . . . 20

13 MIS of M8: Registration Module **21**
13.1 Module . . . 21
13.2 Uses . . . 21
13.3 Syntax . . . 21
13.3.1 Exported Constants . . . 21
13.3.2 Exported Access Programs . . . 21
13.4 Semantics . . . 21
13.4.1 State Variables . . . 21
13.4.2 Environment Variables . . . 21
13.4.3 Assumptions . . . 22
13.4.4 Access Routine Semantics . . . 22
13.4.5 Local Functions . . . 23

14 MIS of M9: Attendance Tracking Module **24**
14.1 Module . . . 24
14.2 Uses . . . 24
14.3 Syntax . . . 24
14.3.1 Exported Constants . . . 24
14.3.2 Exported Access Programs . . . 24
14.4 Semantics . . . 24
14.4.1 State Variables . . . 24
14.4.2 Environment Variables . . . 24
14.4.3 Assumptions . . . 24
14.4.4 Access Routine Semantics . . . 24
14.4.5 Local Functions . . . 25

15 MIS of M10: Report Generation Module **26**
15.1 Module . . . 26
15.2 Uses . . . 26
15.3 Syntax . . . 26
15.3.1 Exported Constants . . . 26
15.3.2 Exported Access Programs . . . 26
15.4 Semantics . . . 26
15.4.1 State Variables . . . 26
15.4.2 Environment Variables . . . 26
15.4.3 Assumptions . . . 26
15.4.4 State Invariants . . . 27
15.4.5 Access Routine Semantics . . . 27
15.4.6 Local Functions . . . 27

16 MIS of M11: Analytics Module **28**
16.1 Module . . . 28
16.2 Uses . . . 28
16.3 Syntax . . . 28
16.3.1 Exported Constants . . . 28
16.3.2 Exported Access Programs . . . 28
16.4 Semantics . . . 28
16.4.1 State Variables . . . 28
16.4.2 Environment Variables . . . 28
16.4.3 Assumptions . . . 28
16.4.4 Formal Definitions . . . 29
16.4.5 Access Routine Semantics . . . 29
16.4.6 Local Functions . . . 30

17 MIS of M12: Database Access Module **31**
17.1 Module . . . 31
17.2 Uses . . . 31
17.3 Syntax . . . 31
17.3.1 Exported Constants . . . 31
17.3.2 Exported Access Programs . . . 31
17.4 Semantics . . . 31
17.4.1 State Variables . . . 31
17.4.2 Environment Variables . . . 32
17.4.3 Assumptions . . . 32
17.4.4 Access Routine Semantics . . . 32
17.4.5 Local Functions . . . 33

18 MIS of M13: Audit Module **34**
18.1 Module . . . 34
18.2 Uses . . . 34
18.3 Syntax . . . 34
18.3.1 Exported Constants . . . 34
18.3.2 Exported Access Programs . . . 34
18.4 Semantics . . . 34
18.4.1 State Variables . . . 34
18.4.2 Environment Variables . . . 34
18.4.3 Assumptions . . . 34
18.4.4 Access Routine Semantics . . . 35
18.4.5 Local Functions . . . 35

3 Introduction

The following document details the Module Interface Specifications for EvENGage. EvENGage is a custom event and survey management system being designed for the MES to simplify and centralize the process of hosting events, conferences, and surveys.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/VirochaanRG/MES-Event-Management-System/.

4 Notation

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 (-∞, ∞)
string	string	An ordered list of characters of any length
Cookie	Cookie	A file stored on the client device storing user data
List of type T	list(T)	A dynamically sized list of elements of type T
Set of type T	set(T)	A dynamically sized set of elements of type T
Map of type K to V	map(K, V)	A collection mapping keys of type K to values of type V. Shorthand notation for set(tuple(K, V)).
Tuple of types T1, T2, ...	tuple(T1, T2, ...)	A finite ordered collection of elements of the specified types
Date and time	DateTime	A specific date and time using the Gregorian calendar and 24-hour clock

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 Module	No Modules
Behaviour-Hiding Modules	M1: Main System Module M2: User Authentication Module M3: User Authorization Module M4: Form Template Module M5: Form Submission Module M6: Event Management Module M7: Event Notification Module M8: Registration Module M9: Attendance Tracking Module M10: Report Generation Module
Software-Decision Modules	M11: Analytics Module M12: Database Access Module M13: Audit Module

Table 1: Module Hierarchy

6 MIS of M1: Main System Module

6.1 Module

Acts as the entry point for all other modules

6.2 Uses

Acts as the entry point for M2, M4, M6, M10.

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

None

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

None

6.4.3 Assumptions

This module can be invoked externally by a user who has access to the backend server.

6.4.4 Access Routine Semantics

N/A

6.4.5 Local Functions

None

7 MIS of M2: User Authentication Module

7.1 Module

Contains functionality for authenticating users and registering new users. **Note: Parts of this module was defined using discrete math.**

7.2 Uses

None

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
`attemptLogin`	`username : string` `password : string`	`sessionCookie : Cookie`	None
`logout`	`sessionCookie : Cookie`	None	`InvalidSession`
`validateSession`	`sessionCookie : Cookie`	`sessionIsValid : bool`	None
`registerUser`	`username : string` `password : string`	`registrationSuccessful : bool`	None

7.4 Semantics

7.4.1 State Variables

`sessions : set(Cookie)`: Set of all active sessions
`credentials : map(string, string)`: Mapping of all registered usernames to their passwords

7.4.2 Environment Variables

`currentTime : DateTime`: Stores the current date and time

7.4.3 Assumptions

Passwords are encrypted before storage.

7.4.4 Access Routine Semantics

`attemptLogin(username : string, password : string)`:
Attempts a login given a username and password.

- transition: If `tuple(username, password)` $\in$ `credentials`, then `sessions := sessions` $\cup$ `set(generateCookie(username))`
- output: Cookie containing session data to be sent back to the client
- exception: None

`logout(sessionCookie: Cookie)`:
Logs the specified user out.

- transition: `sessions := sessions - set(sessionCookie)`
- output: None
- exception: InvalidSession if session is not found in `sessions`

`validateSession(sessionCookie: Cookie)`:
Validates and refreshes a users session.

- transition: If `sessionCookie` $\in$ `sessions`, then `sessionCookie.lastRefresh := currentTime`
- output: Boolean indicating if sessionCookie exists in `sessions`
- exception: None

`registerUser(username : string, password : string)`:
Registers a new user to the database of users.

- transition: `credentials := credentials` $\cup$ `set(tuple(username, password))`
- output: Boolean stating whether registration was successful
- exception: None

7.4.5 Local Functions

`generateCookie(username : stiring)`:
Generates a cookie for a user.

- transition: None
- output: Cookie containing `username` and `lastRefresh := currentTime`
- exception: None

`refreshSessions()`:
Checks all sessions and invalidates old sessions. Scheduled to run periodically.

- transition: $\forall$ `s` $\in$ `sessions | s.lastRefresh - currentTime > 3hrs, sessions := sessions - s`
- output: None
- exception: None

8 MIS of M3: User Authorization Module

8.1 Module

Contains functionality for role based access management

8.2 Uses

Uses M2 for authentication before checking authorizations.

8.3 Syntax

8.3.1 Exported Constants

`validRoles : set(Role)`: The set of all valid roles that can be assigned to users.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
addRole	userId : ℕ role : Role	None	InvalidRole InvalidUser
removeRole	userId : ℕ role : Role	None	InvalidRole InvalidUser
hasPermission	userId : ℕ role : Role	hasPermission : bool	InvalidRole InvalidUser
getUserRoles	userId : ℕ	roles : set(Role)	InvalidUser

8.4 Semantics

8.4.1 State Variables

`roles : map(ℕ, set(T))`: Mapping of user ID to the set of roles the user is assigned.

8.4.2 Environment Variables

None

8.4.3 Assumptions

At least one user who has permission to assign and revoke roles already exists in the system.

8.4.4 Access Routine Semantics

`addRole(userId : ℕ, role : Role)`:
Assigns a role to a user.

- transition: Inserts the specified role into the set attached to the user ID in `roles`
- output: None
- exception: InvalidRole if the role does not exist, InvalidUser if the user does not exist

`removeRole(userId : ℕ, role : Role)`:
Revokes a role from a user.

- transition: Removes the specified role from the set attached to the user ID in `roles`
- output: None
- exception: InvalidRole if the role does not exist, InvalidUser if the user does not exist

`hasPermission(userId : ℕ, role : Role)`:
Checks if a user has a certain permission.

- transition: None
- output: A boolean value which is true if the set attached to the user ID in `roles` contains the specified role
- exception: InvalidRole if the role does not exist, InvalidUser if the user does not exist

`getUserRoles(userId : ℕ)`:
Gets all roles assigned to a given user.

- transition: None
- output: The set of roles attached to the user ID in `roles`
- exception: InvalidUser if the user does not exist

8.4.5 Local Functions

None

9 MIS of M4: Form Template Module

9.1 Module

Contains functionality for creating and linking form modules.

9.2 Uses

Uses M3 to check for form managment permissions.

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
createModule	moduleTitle : string questions : list(Question)	module : FormModule	None
deleteModule	moduleId : $\mathbb{N}$	None	InvalidModule
editModule	moduleId : $\mathbb{N}$ newModuleName : string newQuestions : list(Question)	None	InvalidModule
copyModule	moduleId : $\mathbb{N}$	copiedModule : FormModule	InvalidModule
getModules	None	modules : set(FormModule)	None
linkQuestion	question : Question linkedModule: list(string)	None	InvalidModule IllegalArgument
createForm	formName : string startingModuleId : $\mathbb{N}$ formSettings : map(string, string)	Form	None
deleteForm	formId : $\mathbb{N}$	None	InvalidForm
editForm	newFormName : string startingModuleId : $\mathbb{N}$ newFormSettings : map(string, string)	None	InvalidForm
releaseForm	formId : $\mathbb{N}$, deadline : DateTime	None	InvalidForm
hideForm	formName : string, startingModuleId : $\mathbb{N}$	None	InvalidForm

9.4 Semantics

9.4.1 State Variables

modules : set(FormModule): Set of all created form modules.
createdForms : set(Form): Set of all created forms.

9.4.2 Environment Variables

`currentTime : DateTime`: The current date and time

9.4.3 Assumptions

The calculated current date and time is within 5 seconds of the real date and time

9.4.4 Access Routine Semantics

`createModule(moduleTitle : string, questions : list(Question))`:
Creates a new module given a title and a list of questions.

- transition: Inserts the created module into `modules`
- output: The created FormModule
- exception: None

`deleteModule(moduleId : ℕ)`:
Deletes a module.

- transition: Removes the created module from `modules`
- output: None
- exception: InvalidModule if module does not exist

`editModule(moduleId : ℕ, newModuleName : string, newQuestions : list(Question))`:
Edits the module, assigning a new name and question list.

- transition: Removes the specified module, then inserts the modified one into `modules`
- output: None
- exception: InvalidModule if module does not exist

`copyModule(moduleId : ℕ)`:
Copies an existing module.

- transition: Creates a copy of the given module with a different module ID, then inserts it into `modules`
- output: The copied FormModule
- exception: InvalidModule if module does not exist

`getModules()`:
Retrieves the set of all modules.

- transition: None
- output: A copy of `modules`
- exception: None

`linkQuestion(question : Question, linkedModuleIds: list(ℕ))`:
Links the answers of a question to another module for branching forms.

- transition: Modifies the module in `modules` containing the specified Question
- output: None
- exception: InvalidModule if module does not exist, IllegalArgument if the size of linkedModuleIds does not match the number of possible answers in the question.

`createForm(formTitle : string, startingModuleId: string, formSettings : map(string, string))`:
Creates a form starting from a specified module with certain settings (e.g. multiple submissions, editable submissions)

- transition: Inserts the created form into `createdForms`
- output: The created Form
- exception: None

`deleteForm(formId : ℕ)`:
Deletes the specified form.

- transition: Removes the specified form from `createdForms`
- output: None
- exception: InvalidForm if form does not exist

`editForm(formTitle : string, startingModuleId: string, formSettings : map(string, string))`:
Edits the specified form.

- transition: Removes the specified form from `createdForms` and inserts the modified form
- output: None
- exception: InvalidForm if form does not exist

`releaseForm(formId: string, deadline : DateTime)`:
Releases a form for submissions with the specified deadline.

- transition: Modifies the visibility of the specified form in `createdForms`
- output: None
- exception: InvalidForm if form does not exist

`hideForm(formId: string)`:
Closes submissions for a form.

- transition: Modifies the visibility of the specified form in `createdForms`
- output: None
- exception: InvalidForm if form does not exist

9.4.5 Local Functions

`expireForms()`:
Periodically checks `createdForms` for expired deadlines and closes them.

- transition: Modifies the visibility of the forms in `createdForms`
- output: None
- exception: None

10 MIS of M5: Form Submission Module

10.1 Module

Contains functionality for submitting reponses to forms.

10.2 Uses

Uses M4 to retreive fillable forms.

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
`getFillableForms`	`userId : ℕ`	`fillableForms : set(Form)`	InvalidUser

`respondToForm`	`userId : ℕ`
	`formId : ℕ`
	`formResponse : map(Question, string)`
		`responseId : ℕ`	`InvalidUser`
			`InvalidForm`
			`IllegalArgument`

`getResponses`	`userId : ℕ` `formId : ℕ`	`responseIds : set(ℕ)`	`InvalidUser` `InvalidForm`

`editResponse`	`userId : ℕ`
	`responseId : ℕ`
	`formResponse : map(Question, string)`
		None	`InvalidUser`
			`InvalidForm`
			`IllegalArgument`
			`UnsupportedOperation`

10.4 Semantics

10.4.1 State Variables

`Responses` : set(tuple(responseId : ℕ, userId : ℕ, formId : ℕ, answers : map(Question, string)))

10.4.2 State Invariants

- $\forall r_1, r_2 \in Responses \mid r_1.responseId = r_2.responseId \Rightarrow r_1 = r_2$
- $\forall r \in Responses \mid r.userId \in \mathbb{N} \land r.formId \in \mathbb{N}$
- $\forall r \in Responses \mid dom(r.answers) \subseteq Questions(r.formId)$

10.4.3 Environment Variables

None

10.4.4 Assumptions

None

10.4.5 Access Routine Semantics

`getFillableForms(userId : ℕ)`:
Retreives all forms fillable by the user.

- transition: None
- output: The set of all forms available for the user to fill
- exception: InvalidUser if user does not exist

`respondToForm(userId : ℕ, formId: string, formResponse: map(Question, string))`:
Records a user's response to a form.

- transition: Inserts the created response ID into `userResponses` and `formResponses`
- output: None
- exception: InvalidUser if user does not exist, InvalidForm if the form does not exist, IllegalArgument if the responses do not match the expected format given by the form (e.g. unidentified question)

`getResponses(userId : ℕ, formId: string)`:
Retrieves all of a user's responses for a given forms.

- transition: None
- output: The set of all responses for the user in `userResponses` for a particular form in `formResponses`
- exception: InvalidUser if user does not exist, InvalidForm if the form does not exist

`editResponse(userId : ℕ, formId: string, formResponse: map(Question, string))`:
Modifies a user's response to a form.

- transition: Edits the reponse for the given form (if it allows editing) in `userResponses` and `formResponses`
- output: None

- exception: InvalidUser if user does not exist, InvalidForm if the form does not exist, IllegalArgument if the responses do not match the expected format given by the form, UnsupportedOperation if the form does not allow editing responses

10.4.6 Local Functions

None

11 MIS of M6: Event Management Module

11.1 Module

Contains functionality for creating and managing events.

11.2 Uses

Uses M3 to check permissions for managing events, and M7 for sending event related notifications.

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
createEvent	eventDetails : EventData	eventId : ℕ	IllegalArgument
editEvent	eventId : ℕ eventDetails : ℕ	None	InvalidEvent IllegalArgument
deleteEvent	eventId : ℕ	None	InvalidEvent
getAllEvents	None	eventIds: set(ℕ)	None
getEventDetails	eventId : ℕ	eventData : EventData	InvalidEvent

11.4 Semantics

11.4.1 State Variables

events : map(ℕ, EventData): Mapping of all event IDs to their details

11.4.2 Environment Variables

currentTime : DateTime: Current date and time

11.4.3 Assumptions

The calculated current date and time is within 5 seconds of the actual date and time

11.4.4 Access Routine Semantics

createEvent(eventDetails : EventData):
Creates a new event.

- transition: Inserts an event into events.

- output: ID of the created event
- exception: None

`editEvent(eventId : string, eventDetails : EventData)`:
Edits an existing event.

- transition: Removes the event details in `events` with the new details for the event ID
- output: None
- exception: InvalidEvent if event does not exist

`deleteEvent(eventId : string)`:
Deletes/cancels an event.

- transition: Removes the event from `events`
- output: None
- exception: InvalidEvent if event does not exist

`getAllEvents()`:
Retrieves all events.

- transition: None
- output: A copy of all keys in `events`
- exception: None

`getEventDetails(eventId : string)`:
Retrieves the details of an event.

- transition: None
- output: The event details for the given event ID in `events`
- exception: InvalidEvent if event does not exist

11.4.5 Local Functions

None

12 MIS of M7: Event Notification Module

12.1 Module

Contains functionality for notifying users abouts upcoming events.

12.2 Uses

None

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
sendNotification	userIds : list(ℕ) message : string	None	InvalidUser
scheduleNotification	userIds : list(ℕ) message : string datetime: DateTime	None	InvalidEvent

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

sendNotification(userIds : list(ℕ), message : string, datetime: DateTime):
Sends a notification to the list of users specified.

- transition: None

- output: None
- exception: None

`scheduleNotification(userIds : list(ℕ), message : string, datetime : DateTime)`:
Schedules a notification to send to the list of users specified.

- transition: None
- output: None
- exception: None

12.4.5 Local Functions

`checkScheduleNotifications()`:
Periodically checks and sends out scheduled notifications at their deadline.

- transition: None
- output: None
- exception: None

13 MIS of M8: Registration Module

13.1 Module

Contains functionality for registering users for events.

13.2 Uses

- Event Management Module (M7)

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
registerUser	eventId : ℕ userId : ℕ	confirmation : string	InvalidUser InvalidEvent
delistUser	eventId : ℕ userId : ℕ	removed : bool	InvalidUser InvalidEvent
getRegistrationCode	confirmation : string	code : QRCode	InvalidConfirmation
getRegistrationCode	userId : ℕ eventId : ℕ	code : QRCode	InvalidUser InvalidEvent InvalidRegistration
getConfirmationInfo	confirmation : string	userId : ℕ eventId : ℕ	InvalidConfirmation

13.4 Semantics

13.4.1 State Variables

- eventAttendees : map(eventId : ℕ, users : set(ℕ)): A set of users who have registered for each event.
- confirmations : map(confirmation : string, registration : tuple(userId : ℕ, eventId : ℕ)): A mapping of all confirmation strings into (userId, eventId) tuples.

13.4.2 Environment Variables

None.

13.4.3 Assumptions

There is no payment handling done by this module. Any events requiring payments will have that processed separately before a user can register.

13.4.4 Access Routine Semantics

`registerUser(eventId, userId)`:

- transition: Update the `eventAttendees` sets with the new registered user. Update the `confirmations` with the generated confirmation code.
- output: Provide the generated registration code.
- exception: Returns `InvalidUser` if the given `userId` does not exist; returns `InvalidEvent` if the given `eventId` does not exist.

`delistUser(eventId, userId)`:

- transition: Update the `eventAttendees` sets with removal of the specified user. Update the `confirmations` with the removed confirmation code.
- output: Boolean confirming that there was a change in the `eventAttendees` after the user was removed from the registration of the specified event. If the user was not registered to begin with, there is no harm to the system, but `false` is returned.
- exception: Returns `InvalidUser` if the given `userId` does not exist; returns `InvalidEvent` if the given `eventId` does not exist.

`getRegistrationCode(confirmation)` (1):

- transition: None.
- output: A generated QRCode based on the given confirmation code.
- exception: Returns `InvalidConformation` if a non-existent confirmation is provided.

`getRegistrationCode(userId, eventId)` (2):

- exception: Returns `InvalidUser` if the given `userId` does not exist; returns `InvalidEvent` if the given `eventId` does not exist; returns `InvalidRegistration` if the given user did not register for the event.

`getConfirmationInfo(confirmation)`:

- transition: None.
- output: A `(userId, eventId)` tuple to specify a registration instance.
- exception: Returns `InvalidConformation` if a non-existent confirmation is provided.

13.4.5 Local Functions

Internal functions may handle confirmation string and QR Code generation. The only specification is that the same confirmation string is generated for a given `(userId, eventId)` tuple, and the same QR Code is generated for a single registration instance (i.e. from a given confirmation string).

14 MIS of M9: Attendance Tracking Module

14.1 Module

Contains functionality for tracking live event attendance.

14.2 Uses

- Event Management Module (M7)
- Registration Module (M8)

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
markAttendance	code : QRCode	confirm : bool	InvalidQRCode
markAttendance	eventId : ℕ userId : ℕ	confirm : bool	InvalidUser InvalidEvent
getAttendance	eventId : ℕ	attendance : set(ℕ)	InvalidEvent

14.4 Semantics

14.4.1 State Variables

- attendance : map(eventId : ℕ, users : set(ℕ)): A set of users who have attended each event.

14.4.2 Environment Variables

None.

14.4.3 Assumptions

Attendance only counts the first time a user enters the event. During the event, once a user is registered, physical admins can monitor re-entry of attendees.

14.4.4 Access Routine Semantics

getRegistrationCode(code) (1):

- transition: Update the attendance set for the event with the attending user if the user can attend the event.

- output: Boolean confirming the user has been tracked if they are allowed to attend the event.
- exception: Returns `InvalidQRCode` if the QR Code provided does not work, potentially implying the user cannot register for the event.

`getRegistrationCode(userId, eventId)` (2):

- exception: Returns `InvalidUser` if the given `userId` does not exist; returns `InvalidEvent` if the given `eventId` does not exist.

14.4.5 Local Functions

Internal functions may handle validating QR codes for a given event registration. There must be a way for the module to derive the corresponding registration information from the code.

15 MIS of M10: Report Generation Module

15.1 Module

Contains functionality for generating data visualizations and exporting data to specified formats.

15.2 Uses

- Main System Module (M1)
- Form Submission Module (M5)

15.3 Syntax

15.3.1 Exported Constants

- `Graphs : set(GraphType)`: The set of all valid graphs which can be rendered.
- `Formats : set(FileType)`: The set of all valid file formats which can be generated.

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
plotData	data : list($\mathbb{R}$) graph : GraphType	plot : Graph	InvalidData InvalidGraphType IncompatibleData
exportData	data : list($\mathbb{R}$) format : FileType	file : File	InvalidData InvalidFileType

15.4 Semantics

15.4.1 State Variables

None.

15.4.2 Environment Variables

None.

15.4.3 Assumptions

When $\mathbb{R}$ is specified for data input, it means that the data can be mapped in one way or another onto $\mathbb{R}$.

15.4.4 State Invariants

- $Graphs \subseteq GraphType$
- $Formats \subseteq FileType$

15.4.5 Access Routine Semantics

`plotData(data, graph)`:

- transition: None.
- output:
$$plot = Render(graph, data)$$
- exception: Returns `InvalidData` if the data is corrupt; returns `InvalidGraphType` if the given `GraphType` does not exist; returns `IncompatibleData` if the given data cannot be represented with the given graph type.

`exportData(data, format)`:

- transition: None.
- output: File of the data in the specified file format.
- exception: Returns `InvalidData` if the data is corrupt; returns `InvalidFileType` if the given `FileType` does not exist.

15.4.6 Local Functions

None.

16 MIS of M11: Analytics Module

16.1 Module

Contains functionality for computing summary statistics and trends for events, registrations, attendance, and surveys.

16.2 Uses

Database Access Module (M12)
Registration Module (M8)
Attendance Tracking Module (M9)
Form Submission Module (M5)

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
`getEventStats`	`eventId : ℕ`	`stats : EventStats`	`EventNotFound`
`getSurveyStats`	`surveyId : ℕ`	`summary : SurveySummary`	`SurveyNotFound`
`getRegistrationTrends`	`eventId : ℕ`	`trend : list(RegistrationPoint)`	`EventNotFound`

16.4 Semantics

16.4.1 State Variables

None

16.4.2 Environment Variables

None

16.4.3 Assumptions

- The Database Access Module (M11) correctly stores registrations, attendance records, and survey responses.
- The given `eventId` and `surveyId` refer to events and surveys that were, if they exist, created by other modules.

16.4.4 Formal Definitions

Let $Reg(e)$ be the set of registrations for event e and $Att(e)$ be the set of attendance records for event e.

$$Reg(e) = \{u \mid (u, e) \in Registrations\}$$
$$Att(e) = \{u \mid (u, e) \in Attendance\}$$

Define attendance rate:

$$Rate(e) = \begin{cases} 0, & |Reg(e)| = 0 \\ \frac{|Att(e)|}{|Reg(e)|}, & \text{otherwise} \end{cases}$$

16.4.5 Access Routine Semantics

`getEventStats(eventId :` $\mathbb{N}$`)`:

- transition: None (read-only). Queries the database for all registrations and attendance records associated with `eventId` and computes totals such as number of registrations, check-ins, and attendance rate.
- output: `stats` containing the computed summary values.
- exception:
  - `EventNotFound` if `eventId` does not correspond to any event.

`getSurveyStats(surveyId :` $\mathbb{N}$`)`:

- transition: None (read-only). Queries survey responses associated with `surveyId` and aggregates them per question (for example, counts per option for multiple-choice questions).
- output: `summary` containing aggregated statistics for each question.
- exception:
  - `SurveyNotFound` if `surveyId` does not correspond to any survey.

`getRegistrationTrends(eventId :` $\mathbb{N}$`)`:

- transition: None (read-only). Retrieves timestamps of registrations for `eventId` and computes cumulative registration counts over time.
- output: `trend` as a list of `RegistrationPoint` records, each containing a timestamp and cumulative registration count.
- exception:
  - `EventNotFound` if `eventId` does not correspond to any event.

16.4.6 Local Functions

None

17 MIS of M12: Database Access Module

17.1 Module

Provides a generic interface for reading from and writing to the application's PostgreSQL database. Responsible for connection management, transactions, and basic CRUD operations used by higher-level modules (e.g., event management, registration, analytics).

17.2 Uses

None

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
fetchRow	table : string, id : $\mathbb{N}$	row : Record	RecordNotFound, DatabaseError
fetchRows	table : string, filter : FilterExpr	rows : list(Record)	DatabaseError
insertRow	table : string, data : Record	id : $\mathbb{N}$	DatabaseError
updateRow	table : string, id : $\mathbb{N}$, data : Record	-	RecordNotFound, DatabaseError
deleteRow	table : string, id : $\mathbb{N}$	-	RecordNotFound, DatabaseError

17.4 Semantics

17.4.1 State Variables

- Tables : map(string, set(Record))
  Abstract model of database tables as sets of records.
- activeTx : set(TransactionId)
  Set of identifiers for currently active transactions.

17.4.2 Environment Variables

- dbServer : PostgreSQLInstance
  Running PostgreSQL database server hosting the application schema.

17.4.3 Assumptions

- The database schema has been created and migrated before any access program is invoked.
- `connectionPool` is initialized during system startup.
- `Record` and `FilterExpr` are abstract data structures whose concrete representation is handled by the ORM / query layer.

17.4.4 Access Routine Semantics

fetchRow(table : string, id : N):

- transition: None.
- output: Returns the row in `table` whose primary key equals `id`.
- exception:
  - RecordNotFound if no matching row exists.
  - DatabaseError if a low-level database error occurs.

fetchRows(table : string, filter : FilterExpr):

- transition: None.
- output: Returns all rows in `table` that satisfy `filter`.
- exception: DatabaseError if a low-level database error occurs.

insertRow(table : string, data : Record):

- transition: Inserts a new row into `table` populated with the fields in `data`.
- output: Returns the primary key `id` assigned to the new row.
- exception: DatabaseError if the insert fails (e.g., constraint violation, connectivity issues).

updateRow(table : string, id : N, data : Record):

- transition: Updates the existing row in `table` with primary key `id` using the fields in `data`.

- output: None.
- exception:
  - RecordNotFound if no matching row exists.
  - DatabaseError if the update fails.

deleteRow(table : string, id : N):

- transition: Removes the row in `table` whose primary key equals `id`.
- output: None.
- exception:
  - RecordNotFound if no matching row exists.
  - DatabaseError if the delete fails.

17.4.5 Local Functions

- acquireConnection() : Connection
  Obtains a connection from `connectionPool`, opening a new one if required.
- releaseConnection(c : Connection)
  Returns `c` to `connectionPool` or closes it on error.
- mapRowToRecord(raw : Row) : Record
  Maps a raw database row to the abstract `Record` structure.

18 MIS of M13: Audit Module

18.1 Module

Contains functionality for recording administrative and sensitive system actions in an append-only audit log for traceability and accountability.

18.2 Uses

Database Access Module (M12)

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
recordEvent	entry : AuditEntry	-	DatabaseError
getAuditLog	filter : AuditFilter	entries : list(AuditEntry)	DatabaseError

18.4 Semantics

18.4.1 State Variables

- auditLog : set(AuditEntry)
  Abstract representation of all recorded audit entries.

18.4.2 Environment Variables

None

18.4.3 Assumptions

- Audit entries produced by other modules accurately describe the action taken.
- The underlying database schema includes an audit log table.

18.4.4 Access Routine Semantics

`recordEvent(entry : AuditEntry)`:

- transition: Adds `entry` to `auditLog` and persists it through the Database Access Module.
- output: None
- exception:
  - `DatabaseError` if the entry could not be written.

`getAuditLog(filter : AuditFilter)`:

- transition: None (read-only)
- output: Returns all `AuditEntry` values matching the given filter, such as by user, action type, or date range.
- exception:
  - `DatabaseError` if retrieval fails.

18.4.5 Local Functions

None

Appendix — Reflection

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:

Reflection After Module Implementation

(After you have implemented another team's module, which means this isn't filled in until after the original deadline). What did you learn by implementing another team's module? Were all the details you needed in the documentation, or did you need to make assumptions, or ask the other team questions? If your team also had another team implement one of your modules, what was this experience like? Are there things in your documentation you could have changed to make the process go more smoothly for when an "outsider" completes some of the implementation?

I think through implementing another team's module it showed the difficulties of understanding a foreign codebase and the time it takes to make sure that your module implementation works within another team's structure. I think it shows the importance of having proper documentation on the usage of your application. It would make the process a lot quicker to get started Also reading through the other team's MIS and MG there were some things that they did involving the structure of their modules that we could potentially use within our design when we update it for Revision 1.

In general their MIS provided the required information but there were some sections that were ambiguous so it was hard to ascertain how much detail was required when implementing. Specifically when we look at the GameStateData datatype we don't know what details or fields were required so a barebones implementation was created rather than one which would have all the fields that they might want. I think also since they may not be implementing the exact version of the game Catan, it is hard figure out what ruleset that they might be going with so I had to make an assumption on what constituted a move, the scoirng and victory conditions.

I think the experience was generally seamless in having a different team implementing our

module. I believe that the module we chose was not difficult in its implementation and its specification in the MIS was detailed enough that the implementation could be done. Also we had decent documentation on how to run our platform so the other team could start developing without any help. I think if it was possible to help improve their implementation it would need some more documentation on how this module interacts with the other modules and how it works specificially with the UI as their design choices may not be the same that we wanted.

Group Reflection

1. 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?

   Some of our design decision in relation to the seperation of the modules into the event related modules and form related modules was decided through consultation with our clients. This was an important distinction for us to make because there were some design solutions where we could have had these components together. We found through dicussion that seperating these two functions into seperate module (or group of modules) would probably be the best for future maintainability, testing these modules and actually implementing them. We also had discussions relating to the report generation and data analytics module. We found that seperating these modules would be good for expandability in the future if we wanted to perform further analysis or in a different form.

   Our other design sources primarily came from our past experiences working with these types of web applications and how we wanted to develop it to make it a seamless development experience. For instance, creating a seperate data access layer module was decison we made since some team members learned from past projects that it would be redundant to try and access the database directly from the other modules and a unified interface would be better. For some other decisions we decided to look into best practices when designing web applications to ensure that our structure and implementation was good.

2. 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 found that in general that our other documents didn't really need to be modified when creating this documentation. This was because we had to essentielly utilize the requirements to inform our design decisions and the seperation of modules to implement our requirements. This meant that we built off our design from these documents so we didn't feel had any changes that needed to be made currently for the past documents in relation to our design.

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

The primary limitations of our solution pertain to practicality and realism due to our constraints as a student team with limited time and resources. Due to hardware limitations, we are not be able to fully test the scalability of our system under strenuous conditions, which could impact performance in real-world scenarios, including include registration capacity and concurrent use. With additional resources, we could perform comprhensive testing, fine-tuning the ability of the system to handle large input.

In addition, the system is lacking in terms of security, particularly in the area of user authentication and data encryption. We are limited to using straightforward security mechanisms due to the minimal timeframe. While the information of our system is not entirely critical, being limited to simple user information which is generally public, it would still be an asset and open up future possibilities regarding the level of data we are entrusted with.

While we designed our system to be modular and extensible, these improvements would take time and resources. Given unlimited resources, we would ultimately be able to optimize our design, making it more robust and reliable. There are various supplementary features we could implement as well, such as AI based analytics and reports, increasingly flexible notification systems, and live event monitoring.

4. 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)

   We considered various different design approaches, particularly in terms of how to structure the modules and their interactions. One option was integrating the database access logic directly into each module, rather than having a separate database module. While this would simplify the architecture and module implementation, it would likely lead to redundancy and complicate future modifications. We ultimately decided that a dedicated database access module would be cleaner and more maintainable.

   Likewise, we explored the possibility of implementing the authentication and authorization logic directly into the system modules. In a similar manner, we decided to separate these responsiblities into their own modules to prioritize clarity, modularity, and consistency.

   Another alternative choice involved a registration model that accounted for guests, allowing users to register multiple instances for a given event. This would increase flexibility and usability, better supporting real-world expectations. However, it would introduce complexity in registration, attendance tracking, analytics, and data management. Given our time constraints and the current scope, we decided to proceed with a simpler single registration design for now.

Virochaan Ravichandran Gowri Reflection

1. **What went well while writing this deliverable?**
   Overall we had a good understanding of what we wanted to do when designing and

creating our platform. We had a general idea of what the structure of the system was going to be and we also knew what kind of modules that we wanted to actually deliver. We had some more discussions to finetune our modules to ensure that they weren't too large or encompassing too much functionality but since we had a good understanding of it in the first place it was quite easy for us to develop a effective architecture. We also managed to have a good timeline as we had a plan for what we wanted develop and in what order already in place and we discussed before hand what sections each person would try and do.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   The main painpoint was understanding the different terms in the MIS and how to properly fill them for each module. We had the added difficulty of having multiple UI related components so it added an added difficulty to properly ensure we completed the sections. We had a small challenge in the Module Guide in ensuring that the structure and hierarchy of the modules made sense. This was because we had the possibility of cyclic dependencies so we resolved this by decomposing some of the modules further.

Omar Al-Asfar Reflection

1. **What went well while writing this deliverable?**
   Ultimately, our team worked well together to break down the system into feasible modules. We always had a solid idea of the general stucture of our system, so the design process was quite straightforward. All the major functionalities, and their interactions with one another were discussed in detail beforehand, which allowed us to focus on finer details, such as efficiency and user experience.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   The main challenge invovled applying our design ideas into the report format. Although we had a clear understanding of the system structure, it was nontheless challenging to express it in terms of the MIS semantics, especially for more abstract modules. Moreover, it was also difficult maintaining a consistent level of detail, ensuring the specification was clear but also not a full implementation. This was resolved by regularly referencing the Module Guide to ensure that we remained aligned with expectations.

Rayyan Suhail Reflection

1. **What went well while writing this deliverable?**
   What went well was having a clear overall system structure early on, which made it easier to break the platform into well-defined modules. Aligning the MIS closely with the Module Guide helped keep responsibilities clear and avoid overlap between modules. Once the structure was set, translating the design into consistent specifications was relatively straightforward and efficient.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   One of the main challenges was ensuring consistency between modules, especially with data types and interfaces across different specifications. This was resolved through multiple review passes and cross-checking modules against the rubric and Module Guide, which helped identify and correct mismatches early.

Ibrahim Quraishi Reflection

1. **What went well while writing this deliverable?**
   Once we specified the detailed design for the first module, it was very easy to define other modules based on it. This includes the formatting of the module, as well as the Uses section for linking modules together. We were also able to work as a group to ensure there were no discrepancies between modules. I feel that each module has a clear purpose and well-defined functionality, and the responsibilities are split up well between modules.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   While writing the specification of each module after they had been defined was simple, deciding how to split the system up into modules beforehand was quite difficult. This is due to the fact that our system is distributed, including sub-systems such as the backend server, web clients, admin clients, and mobile clients. This made it difficult to decide which module would be a part of which system and how many modules each system needs. We were able to solve this by more clearly defining what the role of each subsystem is and what they can be expected to be responsible for.

Mohammad Mahdi Mahboob Reflection

1. **What went well while writing this deliverable?**
   I was able to successfully clarify the feedback given by the TA during Rev -1 for this module, which confirmed that my usage of state variables was correct (in the context of our database).

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   I was first confused about the feedback given by the TA and did now know how to reconcile the TA feedback that QR code generation should be part of state. After discussing with the TA, the confusion was clarified and the issue was resolved. In addition to writing this formal document, there were some issues I faced while implementing the modules as outlined in this document, mainly with the Registration Module. These were resolved by reconfiguring the database to match closer to our representation of state variables through reading documentation for the technologies we implemented.