

Module Interface Specification for Software Engineering

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

November 13, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Glossary

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Introduction	1
4	Notation	1
5	Module Decomposition	2
6	MIS of M??: User Authentication 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	4
7	MIS of M2: User Authorization Module	5
7.1	Module	5
7.2	Uses	5
7.3	Syntax	5
7.3.1	Exported Constants	5
7.3.2	Exported Access Programs	5
7.4	Semantics	5
7.4.1	State Variables	5
7.4.2	Environment Variables	5
7.4.3	Assumptions	5
7.4.4	Access Routine Semantics	5
7.4.5	Local Functions	6
8	MIS of M??: Form Template 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 M??: Form Submission Module	9
9.1	Module	9
9.2	Uses	9
9.3	Syntax	9
9.3.1	Exported Constants	9
9.3.2	Exported Access Programs	9
9.4	Semantics	9
9.4.1	State Variables	9
9.4.2	Environment Variables	9
9.4.3	Assumptions	9
9.4.4	Access Routine Semantics	9
9.4.5	Local Functions	10
10	MIS of M??: Event Management Module	11
10.1	Module	11
10.2	Uses	11
10.3	Syntax	11
10.3.1	Exported Constants	11
10.3.2	Exported Access Programs	11
10.4	Semantics	11
10.4.1	State Variables	11
10.4.2	Environment Variables	11
10.4.3	Assumptions	11
10.4.4	Access Routine Semantics	11
10.4.5	Local Functions	12
11	MIS of M??: Event Notification Module	13
11.1	Module	13
11.2	Uses	13
11.3	Syntax	13
11.3.1	Exported Constants	13
11.3.2	Exported Access Programs	13
11.4	Semantics	13
11.4.1	State Variables	13
11.4.2	Environment Variables	13
11.4.3	Assumptions	13

11.4.4	Access Routine Semantics	13
11.4.5	Local Functions	14
12	MIS of M??: Registration Module	15
12.1	Module	15
12.2	Uses	15
12.3	Syntax	15
12.3.1	Exported Constants	15
12.3.2	Exported Access Programs	15
12.4	Semantics	15
12.4.1	State Variables	15
12.4.2	Environment Variables	15
12.4.3	Assumptions	15
12.4.4	Access Routine Semantics	15
12.4.5	Local Functions	16
13	MIS of M??: Attendance Tracking Module	17
13.1	Module	17
13.2	Uses	17
13.3	Syntax	17
13.3.1	Exported Constants	17
13.3.2	Exported Access Programs	17
13.4	Semantics	17
13.4.1	State Variables	17
13.4.2	Environment Variables	17
13.4.3	Assumptions	17
13.4.4	Access Routine Semantics	17
13.4.5	Local Functions	18
14	MIS of M??: Report Generation Module	19
14.1	Module	19
14.2	Uses	19
14.3	Syntax	19
14.3.1	Exported Constants	19
14.3.2	Exported Access Programs	19
14.4	Semantics	19
14.4.1	State Variables	19
14.4.2	Environment Variables	19
14.4.3	Assumptions	19
14.4.4	Access Routine Semantics	19
14.4.5	Local Functions	20

15 MIS of M??: Analytics Module	21
15.1 Module	21
15.2 Uses	21
15.3 Syntax	21
15.3.1 Exported Constants	21
15.3.2 Exported Access Programs	21
15.4 Semantics	21
15.4.1 State Variables	21
15.4.2 Environment Variables	21
15.4.3 Assumptions	21
15.4.4 Access Routine Semantics	22
15.4.5 Local Functions	22
16 MIS of M??: Database Access Module	23
16.1 Module	23
16.2 Uses	23
16.3 Syntax	23
16.3.1 Exported Constants	23
16.3.2 Exported Access Programs	23
16.4 Semantics	23
16.4.1 State Variables	23
16.4.2 Environment Variables	24
16.4.3 Assumptions	24
16.4.4 Access Routine Semantics	24
16.4.5 Local Functions	25
17 MIS of M??: Audit Module	26
17.1 Module	26
17.2 Uses	26
17.3 Syntax	26
17.3.1 Exported Constants	26
17.3.2 Exported Access Programs	26
17.4 Semantics	26
17.4.1 State Variables	26
17.4.2 Environment Variables	26
17.4.3 Assumptions	26
17.4.4 Access Routine Semantics	27
17.4.5 Local Functions	27
18 Appendix	28

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 $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$.

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 $(-\infty, \infty)$
natural number	\mathbb{N}	a number without a fractional component in $[1, \infty)$
real	\mathbb{R}	any number in $(-\infty, \infty)$
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
Date and time	DateTime	A specific date and time using the Gregorian calendar and 24-hour clock
User session	Session	Stores data on a user's session and time of last validation

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
Behaviour-Hiding Modules	M??: User Authentication Module
	M??: User Authorization Module
	M??: Form Template Module
	M??: Form Submission Module
	M??: Event Management Module
	M??: Event Notification Module
	M??: Registration Module
	M??: Attendance Tracking Module
	M??: Report Generation Module
Software-Decision Modules	M??: Analytics Module
	M??: Database Access Module
	M??: Audit Module

Table 1: Module Hierarchy

6 MIS of M?: User Authentication Module

6.1 Module

Contains functionality for logging in and authenticating users.

6.2 Uses

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 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	None	None

6.4 Semantics

6.4.1 State Variables

`sessions` : `set(Session)`: Set of all active sessions

6.4.2 Environment Variables

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

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

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

- transition: creates and adds a new session to `sessions` if login is successful

- output: Cookie containing session data to be sent back to the client
- exception: None

`logout(sessionCookie: Cookie):`

Logs the specified user out.

- transition: Removes the specified session from `sessions`
- output: None
- exception: `InvalidSession` if session is not found in `sessions`

`validateSession(sessionCookie: Cookie):`

Validates and refreshes a users session.

- transition: Refreshes the timeout of the session in `sessions` if session is valid
- 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: None
- output: Boolean stating whether registration was successful
- exception: None

6.4.5 Local Functions

`refreshSessions():`

- transition: Periodically checks `sessions` and removes any session older than 3 hours
- output: None
- exception: None

7 MIS of M2: User Authorization Module

7.1 Module

Contains functionality for logging in and authenticating

7.2 Uses

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

7.4 Semantics

7.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

7.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

7.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

7.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

7.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

8 MIS of M??: Form Template Module

8.1 Module

Contains functionality for logging in and authenticating

8.2 Uses

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

8.4 Semantics

8.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

8.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

8.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

8.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

8.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

9 MIS of M??: Form Submission Module

9.1 Module

Contains functionality for logging in and authenticating

9.2 Uses

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

9.4 Semantics

9.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

9.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

9.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

9.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

9.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

10 MIS of M??: Event Management Module

10.1 Module

Contains functionality for logging in and authenticating

10.2 Uses

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

10.4 Semantics

10.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

10.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

10.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

10.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

10.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

11 MIS of M??: Event Notification Module

11.1 Module

Contains functionality for logging in and authenticating

11.2 Uses

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

11.4 Semantics

11.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

11.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

11.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

11.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

11.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

12 MIS of M??: Registration Module

12.1 Module

Contains functionality for logging in and authenticating

12.2 Uses

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

12.4 Semantics

12.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

12.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

12.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

12.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

13 MIS of M??: Attendance Tracking Module

13.1 Module

Contains functionality for logging in and authenticating

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

13.4 Semantics

13.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

13.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

13.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

13.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

13.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

14 MIS of M??: Report Generation Module

14.1 Module

Contains functionality for logging in and authenticating

14.2 Uses

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

14.4 Semantics

14.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

14.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

14.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

14.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

14.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

15 MIS of M??: Analytics Module

15.1 Module

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

15.2 Uses

Database Access Module (M??)

Registration Module (M??)

Attendance Tracking Module (M??)

Survey Response Module (M??)

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

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

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

None

15.4.3 Assumptions

- The Database Access Module (M??) 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.

15.4.4 Access Routine Semantics

`getEventStats(eventId : 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 : 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 : 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.

15.4.5 Local Functions

None

16 MIS of M??: Database Access Module

16.1 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).

16.2 Uses

None

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 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

16.4 Semantics

16.4.1 State Variables

- connectionPool : ConnectionPool
Pool of reusable connections to the PostgreSQL database.
- activeTx : set(TransactionId)
Set of identifiers for currently active transactions.

16.4.2 Environment Variables

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

16.4.3 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.

16.4.4 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.

16.4.5 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.

17 MIS of M?: Audit Module

17.1 Module

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

17.2 Uses

Database Access Module (M?)

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

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

17.4 Semantics

17.4.1 State Variables

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

17.4.2 Environment Variables

None

17.4.3 Assumptions

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

17.4.4 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.

17.4.5 Local Functions

None

18 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?
2. What pain points did you experience during this deliverable, and how did you resolve them?
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), if any, needed to be changed, and why?
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)