Module Interface Specification for Software Engineering

Team #12, Streamliners
Mahad Ahmed
Abyan Jaigirdar
Prerna Prabhu
Farhan Rahman
Ali Zia

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 Documentation at SRS and MG Documentation at MG.

Contents

3 Introduction

The following document details the Module Interface Specification for the Large Event Management System (LEMS). It defines the detailed interfaces, inputs, outputs, and dependencies among the software modules that comprise the system. Building on the design decomposition outlined in the Module Guide and the functional and non-functional requirements in the Software Requirements Specification (SRS), this document establishes a precise contract for how each module communicates and interacts within the overall architecture.

The LEMS platform is a centralized event management solution developed for the McMaster Engineering Society (MES) to support the organization and execution of large-scale student events such as the Fireball Formal, Graduation Formal, and Pub Nights. The system integrates event registration, ticketing, waivers, payment processing, and check-in into a single platform accessible via web and mobile interfaces. To ensure modularity, maintainability, and reliability, the system is decomposed into independent yet cohesive modules such as the Payment Processing Module, Role-Based and Feature-Based Access Control (RBAC/FBAC) Module, and the Bus/Table/RSVP Sign-Up Module.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/4G06-Streamliners/MacSync.

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**	
character	char	
integer	$\mathbb{Z}$	a number without a fractional
natural number	$\mathbb{N}$	a number without a fractiona
real	$\mathbb{R}$	a

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
Hardware-Hiding Module
Behaviour-Hiding Module
Software Decision Module

Table 1: Module Hierarchy

6 Module Interface Specifications

This section provides the Module Interface Specifications (MIS) for each module defined in the Module Guide. Each module is documented independently, following the notation and structure described in Section 2.

6.1 MIS of Payment Processing Module (M1)

6.1.1 Module

Payment Processing Module (M1)

This module prepares and submits payment requests from the client side. It validates the payment data, attaches authentication headers, and communicates with the backend API that integrates with Stripe. The module does not handle any payment verification locally. Instead, verification and execution of charges or refunds are performed by the backend using Stripe's secure payment APIs, configured in the Payment Configuration Module (M4).

6.1.2 Uses

- API Layer (HTTP communication with backend)
- User Authorization Module (M8)
- Payment Configuration Module (M4)
- Notification Handling Module (M7)

6.2 Syntax

6.2.1 Exported Constants

N/A

6.2.2 Exported Access Programs

Name	**In**	**Out**	**Exceptio**
submitPayment	PaymentInfo	ConfirmationResponse	InvalidPaymentData, NetworkErr
validatePaymentData	PaymentInfo	Boolean	InvalidPaymentDa
requestRefund	RefundRequest	RefundResponse	InvalidRefundRequest, NetworkErr

6.3 Semantics

6.3.1 State Variables

N/A

6.3.2 Environment Variables

- Network connection to the backend API
- AuthToken from User Authorization Module (M8)

6.3.3 Assumptions

- User must be authenticated before submitting a payment or requesting a refund.
- Device must have a stable network connection.
- The backend exposes REST endpoints:
  - `POST /api/payments`: initialize Stripe PaymentIntent
  - `POST /api/payments/refund`: request a Stripe refund
- The backend manages all Stripe secret keys and secure payment operations.

6.3.4 Access Routine Semantics

submitPayment(paymentInfo):

- transition: N/A.
- output: A ConfirmationResponse containing the client secret, payment result, or any backend-supplied status (e.g., succeeded, failed).
- exception:
  - InvalidPaymentData: required fields missing or malformed
  - NetworkError: request fails or cannot reach backend

validatePaymentData(paymentInfo):

- transition: N/A
- output: Returns true if all required fields (amount, event ID, billing details) are correctly defined.
- exception: InvalidPaymentData if fields are empty or violate basic constraints.

requestRefund(refundRequest):

- transition: N/A

- output: A RefundResponse containing refund status, amount refunded, and backend-supplied outcome (e.g. pending, succeeded, failed).
- exception:
  - InvalidRefundRequest: missing payment ID, invalid amount
  - NetworkError: unable to send or receive the refund request

6.3.5 Local Functions

- submitAPIRequest(jsonBody, authToken, endpoint): sends the HTTP POST request to the backend API.
- handleAPIResponse(response): converts backend JSON into the correct response object (ConfirmationResponse or RefundResponse).

6.4 MIS of RBAC/FBAC Access Control Module (M2)

6.5 MIS of Payment Processing Module (M1)

`RBACModule`

Payment Processing Module (M1)

- User Authorization Module (M8)
- Access Control Service Module (M5)

6.6 Syntax

6.6.1 Exported Constants

- `DEFAULT_ROLE`: `str`
  Default user role when signing up for the platform.
- `ACCESS_POLICY_SCHEMA: JSON`
  Schema for the capability snapshot retrieved from backend M5.

6.6.2 Exported Access Programs

Name	In
`getUserRole`	userID: Str
`getUserCapabilities`	userID: Str
`authorizeAction`	action: Str
`syncCapabilities`	userID: Str

6.7 Semantics

6.7.1 State Variables

- `capabilitySnapshot : dict[Str, list[Str]]`
  Locally cached copy of backend-authorized capabilities for the active user.
- `userRole: Str`
  UI-level role used for UX grouping.

6.7.2 Environment Variables

- `ACCESS_CONTROL_ENDPOINT : Str`
  Backend API to fetch the authoritative permissions.

6.7.3 Assumptions

- Authorization requests are made only for defined features or actions.
- M5 always provides the source-of-truth permission set.

6.7.4 Access Routine Semantics

`getUserRole(userID: Str)`:

- transition: None
- input: A unique user identifier.
- output: Returns the role string associated with the given user.
- exception: `UserNotFound`

`getUserCapabilities(userID: Str)`:

- transition: None
- input: A unique user identifier.
- output: Returns list of available features/actions.
- exception: `UserNotFound`

`authorizeAction(userID: Str, action: Str)`:

- transition: None
- input: User ID and attempted action.
- output: Returns `True` if permitted.

- exception: `AuthorizationError`

`syncCapabilities(userID: Str):`

- transition: Fetches capability snapshot from M5.
- input: User ID
- output: `True` on success.
- exception: `FetchError`

6.7.5 Local Functions

- None.

6.8 MIS of Sign-Up Module (M3)

6.9 Module

[SignupModule —SS]

6.10 Uses

- A back-end or configuration service to store and retrieve confirmed signups.
- A front-end/UI component for users to input sign-up information.

6.11 Syntax

6.11.1 Exported Constants

- SIGNUP_TYPES: ("RSVP", "Table", "Bus")
- `MAX_PARTICIPANTS_PER_EVENT`: $\mathbb{N}$
- `DEFAULT_ROLE`: `str`
  Default user role when signing up for the platform.
- `ACCESS_POLICY_SCHEMA: JSON`
  Schema for the capability snapshot retrieved from backend M5.

6.11.2 Exported Access Programs

Name	In
submitSignup	userID: Str, eventID: Str, data: JSON
getSignupStatus	userID: Str, eventID: Str
cancelSignup	userID: Str, eventID: Str

6.12 Semantics

6.12.1 State Variables

- signupDetails: dict[(userID, eventID): JSON]

6.12.2 Environment Variables

- SIGNUP_API: Str

6.12.3 Assumptions

- User has a valid authenticated session.

6.12.4 Access Routine Semantics

submitSignup(UserID, eventID, data):

- transition: Adds new signup entry.
- output: True if successful.
- exception: SignupError

getSignupStatus(userID, eventID):

- transition: None
- output: JSON signup info.
- exception: NotFoundError

cancelSignup(userID, eventID):

- transition: Removes signup entry.
- output: True if successful.
- exception: CancellationError

6.12.5 Local Functions

validateSignupData(formData: JSON):

- transition: None
- output: True if valid.
- exception: ValidationError

checkCapacity(eventID: str):

- transition: None
- output: True if capacity remains.
- exception: CapacityExceededError

6.13 MIS of Payment Configuration Module (M4)

6.14 Module

`PaymentConfigurationModule`

This module centralizes all configuration required for secure payment operations. It exposes safe, restricted payment settings to other modules such as the Payment Processing Module (M1) while hiding sensitive Stripe configuration logic and backend API wiring.

6.15 Uses

- Payment Processing Module (M1)
- User Authorization Module (M8)
- Access Control Module (M5)
- Notification Handling Module (M7)
- Backend Stripe Integration Service

6.16 Syntax

6.16.1 Exported Constants

- STRIPE_PUBLIC_KEY : Str
- PAYMENT_API_BASE : URL
- CURRENCY_CODE : Str
- TAX_RATE : Float
- INTENT_CACHE_TTL_SEC : Nat

6.16.2 Exported Access Programs

Name	In	Out	Exceptions
getPublicStripeKey	–	Str	KeyNotAvailable
getPaymentAPIEndpoint	OperationType	URL	InvalidOperationType
getCurrencyConfig	–	CurrencyConfig	–
shouldRefreshPaymentIntent	LastUpdated: Time	Bool	–
getReceiptConfig	–	ReceiptConfig	–

6.17 Semantics

6.17.1 State Variables

- publicKey : Str
- paymentAPIMap : dict[OperationType $\rightarrow$ URL]
- currencyConfig : CurrencyConfig
- receiptConfig : ReceiptConfig
- cacheTTL : Nat

6.17.2 Environment Variables

- ENV_STRIPE_PUBLIC_KEY : Str
- ENV_PAYMENT_API_BASE : URL
- SystemTime : Time

6.17.3 Assumptions

- Stripe secret keys live only on backend.
- Backend exposes payment endpoints.

6.17.4 Access Routine Semantics

getPublicStripeKey():

- transition: none
- output: publicKey

getPaymentAPIEndpoint(opType):

- transition: none

- output: URL for operation

getCurrencyConfig():

- transition: none
- output: currencyConfig

shouldRefreshPaymentIntent(lastUpdated):

- transition: none
- output: True if TTL expired

getReceiptConfig():

- transition: none
- output: receiptConfig

6.17.5 Local Functions

- loadEnvValue
- buildEndpoint
- validatePublicKey
- computeTTL

6.18 MIS of Access Control Module (M5)

6.19 Module

`AccessControlService`

6.20 Uses

- Authorization Module (M8)
- Persistence Layer (DB)
- Caching Layer

6.21 Syntax

6.21.1 Exported Constants

- POLICY_SCHEMA : JSON
- AUTH_ERROR_CODE : Int

6.21.2 Exported Access Programs

Name	In
checkPermission	action: Str
assignRole	userID: Str, role: Str
revokeRole	userID: Str, role: Str
setPolicy	policy: JSON
getPolicy	-

6.22 Semantics

6.22.1 State Variables

- roleAssignments : dict[userID → set[role]]
- policyDocument : JSON
- policyVersion : Int

6.22.2 Environment Variables

- AUTH_DB_URI : Str
- CACHE_URI : Str

6.22.3 Assumptions

- All modules consult M5 before protected actions.
- M2 never bypasses M5.

6.22.4 Access Routine Semantics

checkPermission(action):

- transition: none
- output: True if authorized

assignRole(userID, role):

- transition: adds role
- output: True

revokeRole(userID, role):

- transition: removes role

- output: True

setPolicy(policy):

- transition: replaces policy, increments version
- output: True

getPolicy():

- output: policyDocument

6.22.5 Local Functions

- None.

6.23 MIS of Registration Rules Module (M6)

6.23.1 Module

Registration Rules Module (M6)

This module enforces all event-level registration constraints for Bus Sign-Up (M3.1), Table Sign-Up (M3.2), and RSVP Sign-Up (M3.3). It retrieves event rules such as capacity limits, table constraints, bus seating limits, registration deadlines, and RSVP policies from the backend datastore.

6.23.2 Uses

- Access Control Module (M5)
- User Authorization Module (M8)
- Backend datastore
- Sign-Up Module (M3)

6.24 Syntax

6.24.1 Exported Constants

N/A

6.24.2 Exported Access Programs

Name	In	Out	Exceptions
validateCapacity	EventInfo, UserInfo	Boolean	EventFull, InvalidEvent
validateDeadline	EventInfo, UserInfo	Boolean	DeadlinePassed, InvalidEvent
validateEligibility	EventInfo, UserInfo	Boolean	PermissionDenied
getEventRules	EventID	EventRules	InvalidEvent

6.25 Semantics

6.25.1 State Variables

None.

6.25.2 Environment Variables

- Database connection

6.25.3 Assumptions

- Database contains valid rules.

6.25.4 Access Routine Semantics

validateCapacity(eventInfo, userInfo):

- output: True if capacity available.
- exception: EventFull, InvalidEvent

validateDeadline(eventInfo, userInfo):

- output: True if deadline not passed.
- exception: DeadlinePassed, InvalidEvent

validateEligibility(eventInfo, userInfo):

- output: True if user meets criteria.
- exception: PermissionDenied

getEventRules(eventID):

- output: event rules.
- exception: InvalidEvent

6.25.5 Local Functions

- lookupRules
- isBeforeDeadline
- hasRemainingCapacity
- meetsEligibilityCriteria

6.26 MIS of Notification Handling Module (M7)

6.27 Module

NotificationModule

6.28 Uses

- Front-end notification UI

6.29 Syntax

6.29.1 Exported Constants

- NOTIFICATION_TYPES: (”Confirmation”, ”Reminder”, ”Cancellation”)

6.29.2 Exported Access Programs

Name	
SendNotification	userID: Str, message: Str, notification_type:
getNotificationHistory	userID:
subscribeUser	userID: Str, subscribed: b
UnsubscribeUser	userID: Str, subscribed: b

6.30 Semantics

6.30.1 State Variables

- userNotifPreference: dict[UserID: bool]

6.30.2 Environment Variables

- EMAIL_API: Str
- SMS_API: Str

6.30.3 Assumptions

- Users opted into notifications.

6.30.4 Access Routine Semantics

SendNotification(userID, message, notification_type):

- transition: sends notification
- output: True if sent
- exception: NotificationError

getNotificationHistory(userID):

- output: list of notifications
- exception: RetrievalError

subscribeUser(userID, subscribed):

- transition: updates preference
- output: True
- exception: SubscribingError

UnsubscribeUser(userID, subscribed):

- transition: preference false
- output: True
- exception: UnsubscribingError

6.30.5 Local Functions

- None

6.31 MIS of User Authorization Module (M8)

6.32 Module

`UserAuthorizationModule`

This module provides authentication and identity verification for all other modules in the system.

6.33 Uses

- Access Control Module (M5)
- Payment Processing Module (M1)
- Sign-Up Module (M3)
- Notification Handling Module (M7)
- External Authentication Provider

6.34 Syntax

6.34.1 Exported Constants

- TOKEN_EXPIRY_MIN : Nat
- REFRESH_TOKEN_SUPPORTED : Bool

6.34.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
login	Credentials	AuthToken	InvalidCredentials, NetworkError
logout	AuthToken	Bool	TokenError
validateToken	AuthToken	Bool	TokenExpired, TokenInvalid
getUserInfo	AuthToken	UserProfile	TokenInvalid, NetworkError
refreshToken	RefreshToken	AuthToken	TokenInvalid, NotSupported

6.35 Semantics

6.35.1 State Variables

- activeSessions : dict[token $\rightarrow$ UserID]
- tokenExpiryTimes : dict[token $\rightarrow$ Time]

6.35.2 Environment Variables

- AUTH_SERVER_URL : URL
- SystemTime : Time

6.35.3 Assumptions

- All modules calling this one must supply a valid token.
- External auth provider is available.

6.35.4 Access Routine Semantics

login(credentials):

- transition: adds session
- output: AuthToken
- exception: InvalidCredentials, NetworkError

logout(token):

- transition: removes session
- output: True
- exception: TokenError

validateToken(token):

- output: True if valid and unexpired
- exception: TokenExpired, TokenInvalid

getUserInfo(token):

- output: UserProfile
- exception: TokenInvalid, NetworkError

refreshToken(refreshToken):

- transition: replaces token
- output: new AuthToken
- exception: TokenInvalid, NotSupported

6.35.5 Local Functions

- decodeToken
- isExpired
- requestAuthServer

7 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:

7.1 Mahad Ahmed

1. What went well while writing this deliverable?

   Something that went well while writing this deliverable was my knowledge for designing system architecture from previous courses. Specifically, 3A04 where we designed system architecture similar to this before creating our software. Using that previous knowledge really helped me complete this document.

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

   I was tasked with creating the MIS for the RBAC Module. I found it challenging due to my lack of understanding of how a RBAC/FBAC system functions, so creating the functions and design for it proved to be challenging. To resolve this, I did research on RBAC/FBAC on how they are designed and their capabilities.

7.2 Prerna Prabhu

1. What went well while writing this deliverable?

   During this deliverable, I think we learned from our pervious deliverables and we were able to divide up tasks pretty evenly and quickly at the start based on the estimated amount of work each one would be, and we also looked at any dependencies between the different sections prior to assigning the work to each team member. In this case this meant figuring out the modules that we needed for our project, discussing them, and reviewing sections and work that were solely focused on explaining the modules, so that the other parts that depended on module definition could be completed on time without rush.

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

   During this deliverable, we initially experienced difficulties with what modules we wanted to define and how the would be defined. We were also confused as a team on what some of the requirements were for these modules and how they would be defined. Through discussion as a team, we were able to identify the modules that were needed and tweak any changes and improvements early on, without waiting until closer to the deadline. We were also able to ask more questions to our supervisor and TA to work through these challenges and gain more insight on what was expected from us.

7.3 Ali Zia

1. What went well while writing this deliverable?

   The division of labour went really well where tasks were assigned fairly and we continued to improve in terms of communicaiton and coordinating our efforts to achieve results and complete our tasks in time

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

   A challenge we encountered was figuring out how to group the system functionaliteis into modules without overlapping responsiblites and there were moments where team members had slightly different interpretations. However, through open communication and spending time understanding our perspectives and feedback we were able to overcome and address the hallenges

7.4 Farhan Rahman

1. What went well while writing this deliverable?

   Our team divided the work effectively and continued the same collaboration approach from earlier milestones, but this time we were more intentional about distributing effort based on each person's strengths rather than just splitting sections. This helped us produce higher-quality work and kept progress steady. We also leveraged each member's area of expertise to ensure each part was written clearly and aligned with the system's design.

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

   Because this deliverable involved defining key modules and system components, it took time to fine-tune the details and ensure accuracy that could be carried forward to implementation. Some sections required multiple iterations to make sure they were both technically correct and consistent with the overall system architecture. Coordinating these interdependent parts and aligning them with previous milestones was challenging, but careful reviews and group collaboration helped us resolve it.

7.5 Abyan Jaigirdar

1. What went well while writing this deliverable?

   Our team worked really well together during this milestone. We divided up the sections early on and made sure everyone understood their part before starting. Communication was smooth, and we checked in often to make sure everything fit together. This helped us stay organized and finish the deliverable on time without last-minute stress.

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

   At first, we had trouble deciding how to separate the system into modules and which features belonged where. Some overlap and confusion came up during drafting, but after a few team discussions and review sessions, we cleared things up and finalized a structure we were all confident in.

7.6 Team

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?

   Several of our design decisions came from directly talking to MES executives. For example, RBAC/FBAC came from stakeholder feedback emphasizing the need for permission management across the platform. Furthermore, the integration of Stripe for payments is another design decision directly influenced by our stakeholder where they emphasized the need for secure payments.

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?

   While creating the design doc, multiple areas were identified where updates would be necessary. An example is in the hazard analysis such as potential authorization issues. These changes have been taken note of and will be updated in the next revision of the document.

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)

   A limitation to our solution is our dependency on external services. For example, we are depending on Stripe to have 99.99% uptime to meet our software needs. Another dependency is using the McMaster identity platform to sign-in and authenticate users.

If we had the luxury of time and resources, we would opt to create our own payment gateway that met the security concerns of our stakeholder.

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 many design solutions for our payment gateway such as paddle or square, but we settled with Stripe with our stakeholder for its simplicty and easy integration. We also considered many types of authentication for the platform such as creating our own authentication where users sign up and register with their own email. However, we decided to use the McMaster identity platfrom since it would be the easiest to verify users and keep users to only valid McMaster students.