Module Interface Specification for Software Engineering

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

January 21, 2026

1 Revision History

Date	Version	Notes
2025-11-13	-1.0	Initial draft of MIS doc.

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

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by 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	`Str`	sequence of `char`
tuple	$\langle x_1, \ldots, x_k \rangle$	Tuple (heterogeneous)
boolean	`Bool`	boolean value `True` or `False`
integer	`Nat`	non-negative integer $(0, \infty)$ when counts/TTL are permitted to be 0
timestamp	`Time`	timestamp on system clock; used for TTL or expiry checks
set	`set`[T]	Finite unordered set of T (i.e. `set[Str]`)
dictionary	`dict`[K → V]	Finite map from K to V (i.e. `dict[OperationType` → `URL]`)
list	`list`[T]	Finite ordered list of T (i.e. `list[Str]`)
object	`JSON`	JSON object as a (key → value) mapping with JSON-serializable values
	`URL`	Absolute/relative URI string for backend endpoints
	`opType`	Operation discriminator used as a map key (i.e. `'create'`, `'refund'`)
identifier	`SRS-FR#`	Functional Requirement in the *SRS*
identifier	`SRS-NFR#`	Non-Functional Requirement in the *SRS*
identifier	`SSR-#`	Safety/Security Requirement from the *Hazard Analysis*
identifier	`M#`	Module identifier used throughout the MIS (e.g. *M1 - Payment Processing Module*)
identifier	`M#.#`	Submodule identifier when applicable (e.g. *M3.2 - Table Sign-Up Submodule*)

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	
Behaviour-Hiding Module	Payment Processing Module
	RBAC/FBAC Access Control Module
	Sign-Up Module
	Bus Sign-Up Module
	Table Sign-Up Module
	RSVP Sign-Up Module
	User Authorization Module
Software Decision Module	Payment Configuration Module
	Access Control Module
	Registration Rules Module
	Notification Handling Module

Table 1: Module Hierarchy

6 MIS of Payment Processing Module (M1)

6.1 Module

`PaymentProcessingModule`

6.2 Uses

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

6.3 Syntax

6.3.1 Exported Constants

N/A

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
`submitPayment`	PaymentInfo	ConfirmationResponse	InvalidPaymentData, NetworkError
`validatePaymentData`	PaymentInfo	Boolean	InvalidPaymentData
`requestRefund`	RefundRequest	RefundResponse	InvalidRefundRequest, NetworkError

6.4 Semantics

6.4.1 State Variables

- **PendingPayments**: Mapping of Stripe PaymentIntent IDs to pending transaction details.

  $$\{\text{paymentIntentID} \rightarrow (\text{userID},\ \text{eventID},\ \text{amount},\ \text{status=''pending''})\}$$

- **CompletedPayments**: List of finalized, successful Stripe payments.

  {(paymentID, paymentIntentID, userID, eventID, amount, status=”succeeded”, timestamp)}

- **RefundRequests**: Queue of refund operations.

  $$\{(\text{refundID},\ \text{paymentIntentID},\ \text{reason},\ \text{status})\}$$

6.4.2 Environment Variables

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

6.4.3 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.

6.4.4 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

6.4.5 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).

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

7.1 Module

`RBACModule`

7.2 Uses

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

7.3 Syntax

7.3.1 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.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
`getUserRole`	userID: Str	role: Str	UserNotFound
`getUserCapabilities`	userID: Str	capabilities: list[Str]	UserNotFound
`authorizeAction`	action: Str	authorized: bool	AuthorizationError
`syncCapabilities`	userID: Str	success: bool	FetchError

7.4 Semantics

7.4.1 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.

7.4.2 Environment Variables

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

7.4.3 Assumptions

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

7.4.4 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` If UUID cannot be identified.

`getUserCapabilities(userID: Str)`:

- transition: None
- input: A unique user identifier.
- output: Returns list of available features/actions.
- exception: `UserNotFound` If UUID cannot be identified.

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

- transition: None
- input: User ID and attempted action.
- output: Returns `True` if the action is permitted by user's role/capabilities.

- exception: `AuthorizationError` if access is denied.

`syncCapabilities(userID: Str)`:

- transition: Fetches the capability snapshot from M5 and updates internal policy schema.
- input: User ID
- output: Returns `True` on success.
- exception: `FetchError` if backend cannot be reached.

7.4.5 Local Functions

- None.

8 MIS of Sign-Up Module (M3)

8.1 Module

`SignUpModule`

8.2 Uses

- Registration Rules (M6)
- Payment Configuration (M4)
- Access Control (M5)

8.3 Syntax

8.3.1 Exported Constants

- `SIGNUP_TYPES`: (”RSVP”, ”Table”, ”Bus”)
  - Supported types of signup modules.
- `MAX_PARTICIPANTS_PER_EVENT`: $\mathbb{N}$
  - Maximum number of participants allowed per event.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
`submitSignup`	userID: Str, eventID: Str, data: JSON	success: bool	SignupError
`getSignupStatus`	userID: Str, eventID: Str	status: JSON	NotFoundError
`cancelSignup`	userID: Str, eventID: Str	success: bool	CancellationError

8.4 Semantics

8.4.1 State Variables

- signupDetails: dict[(userID, eventID): JSON]
  - Stores all signup information and metadata.

8.4.2 Environment Variables

- SIGNUP_API: Str
  - API endpoint to database for sending and storing signup data to backend.

8.4.3 Assumptions

- A user has a valid session where they are authenticated before accessing the signup module.

8.4.4 Access Routine Semantics

submitSignup(UserID, eventID, data):

- transition: Adds new entry to signupDetails for given userID and eventID.
- output: Returns true if signup is successful.
- exception: SignupError if validation fails or event is full.

getSignupStatus(userID, eventID):

- transition: None.
- output: Returns JSON information with users signup details for the given event.
- exception: NotFoundError if no record exists.

cancelSignup(userID, eventID):

- transition: Removes signup for given userID and eventID.
- output: Returns true on successful deletion.
- exception: CancellationError if no record is found for deletion

8.4.5 Local Functions

validateSignupData(formData: JSON):

- transition: None.
- output: Returns true if all required fields are complete.
- exception: ValidationError if required fields are missing or have invalid data format.s

checkCapacity(eventID: str):

- transition: None.
- output: Returns true if the current signup added to total signups is ¡ MAX_PARTICIPANTS.
- exception: CapacityExceededError if event has reached full capacity.

9 MIS of Payment Configuration Module (M4)

9.1 Module

`PaymentConfigurationModule`

9.2 Uses

- Payment Processing Module (M1): Uses exported configuration values to construct valid payment requests and interact with backend payment endpoints.
- User Authorization Module (M8): Provides authentication tokens required when modules call backend payment APIs.
- Access Control Module (M5): Ensures that only authorized user roles may access payment configuration or initiate restricted payment operations.
- Notification Handling Module (M7): Uses configured receipt and invoice formatting rules when sending payment confirmations.
- Backend Stripe Integration Service: Provides the actual PaymentIntent creation, confirmation, and refund endpoints (e.g., `/api/payments`, `/api/payments/refund`).

9.3 Syntax

9.3.1 Exported Constants

- `STRIPE_PUBLIC_KEY : Str`
  The publishable Stripe key used by the frontend; never contains secret keys.

- `PAYMENT_API_BASE : URL`
  Base backend endpoint for payment operations.

- `CURRENCY_CODE : Str`
  Currency used for all events (`CAD`).

- `TAX_RATE : Float`
  The global tax percentage applied to event payments.

- `INTENT_CACHE_TTL_SEC : Nat`
  TTL for caching Stripe PaymentIntent client secrets.

9.3.2 Exported Access Programs

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

9.4 Semantics

9.4.1 State Variables

- `publicKey : Str`

- `paymentAPIMap : dict[OperationType → URL]`

- `currencyConfig : CurrencyConfig`

- `receiptConfig : ReceiptConfig`

- `cacheTTL : Nat`

9.4.2 Environment Variables

- `ENV_STRIPE_PUBLIC_KEY : Str` - Loaded from build-time environment.
- `ENV_PAYMENT_API_BASE : URL` - Backend service base URL.
- `SystemTime :  Time` - Used for TTL and expiration checks.

9.4.3 Assumptions

- Stripe secret keys are stored only on the backend and never exposed to the client.
- Backend exposes the routes `/api/payments`, `/api/payments/confirm`, and `/api/payments/refund`.
- All callers of this module have been authenticated by M8 and authorized by M5.
- Configuration is loaded once per application session.

9.4.4 Access Routine Semantics

`getPublicStripeKey()`

- transition: none
- output: `publicKey`
- exception: `KeyNotAvailable` if key is missing or invalid.

`getPaymentAPIEndpoint(opType)`

- transition: none
- output: returns URL mapped to the given operation type.
- exception: `InvalidOperationType` if no mapping is defined.

`getCurrencyConfig()`

- transition: none
- output: returns currency and tax configuration.
- exception: none.

`shouldRefreshPaymentIntent(lastUpdated)`

- transition: none
- output: `true` if the PaymentIntent cache TTL has expired.

- exception: none.

`getReceiptConfig()`

- transition: none
- output: returns canonical receipt formatting settings for notifications.
- exception: none.

9.4.5 Local Functions

- `loadEnvValue(key)`: safely retrieves build-time environment variables.
- `buildEndpoint(base, suffix)`: constructs consistent backend payment URLs.
- `validatePublicKey(str)`: checks prefix/type of Stripe public key.
- `computeTTL(t)`: helper for determining TTL expiration.

10 MIS of Access Control Module (M5)

10.1 Module

`AccessControlService`

10.2 Uses

- Authorization Module (M8)
- Persistence Layer (DB): stores roles, features, capabilities, and user assignments.
- Caching Layer: speeds up permission lookups.

10.3 Syntax

10.3.1 Exported Constants

- `POLICY_SCHEMA : JSON`
  Defines valid structure for permission rules.
- `AUTH_ERROR_CODE : Int`
  Code returned when an API call is forbidden.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
`checkPermission`	action: Str	authorized: bool	AuthorizationError
`assignRole`	userID: Str, role: Str	success: bool	RoleError
`revokeRole`	userID: Str, role: Str	success: bool	RoleError
`setPolicy`	policy: JSON	success: bool	PolicyError
`getPolicy`	-	policy: JSON	-

10.4 Semantics

10.4.1 State Variables

- `roleAssignments : dict[userID → set[role]]`
  Maps each user role to its permitted features.

- `policyDocument : JSON`
  Full mapping of roles → features → capabilities

- `policyVersion : Int`
  Incremented upon policy update.

10.4.2 Environment Variables

- `AUTH_DB_URI : Str`
  Path to policy store.

- `CACHE_URI : Str`
  Cache location for capabilities.

10.4.3 Assumptions

- All callers (Payment, Forms, Check-In, etc.) consult this module before performing protected actions.

- M2 calls this module to enforce its authority, it does not make up its own rules.

10.4.4 Access Routine Semantics

`checkPermission(action: Str)`:

- transition: None

- input: the attempted action.

- output: Returns True if the user has the required capability to perform the requested action.

- exception: `AuthorizationError` if the user is not permitted to perform the action.

`assignRole(userID: Str, role: Str)`:

- transition: Adds the specified role to the user's stored role assignments.
- input: A unique user identifier and a valid role string.
- output: Returns True on successful role assignment.
- exception: `RoleError` if the role does not exist or cannot be assigned.

`revokeRole(userID: Str, role: Str)`:

- transition: Removes the specified role from the user's role assignments.
- input: A unique user identifier and a valid role string.
- output: Returns True on successful role removal.
- exception: `RoleError` if the specified role is not currently assigned to the user.

`setPolicy(policy: JSON)`:

- transition: Validates and replaces the policy document with the new policy, increments the internal policy version counter.
- input: A JSON object containing an updated set of roles, features, capabilities, and guard rules.
- output: Returns `True` on success.
- exception: `PolicyError` if the policy does not match the defined schema or cannot be persisted.

`getPolicy()`:

- transition: None.
- input: None.
- output: Returns the current policy document in JSON format.
- exception: None.

10.4.5 Local Functions

- None.

11 MIS of Registration Rules Module (M6)

11.1 Module

`RegistrationRulesModule`

11.2 Uses

- Access Control Module (M5): for checking user roles and eligibility
- User Authorization Module (M8): for identity data required in rule checks
- Backend datastore (via data access layer): for retrieving event rules and counts
- Sign-Up Module (M3): requests rule validation during sign-up flows

11.3 Syntax

11.3.1 Exported Constants

N/A

11.3.2 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

11.4 Semantics

11.4.1 State Variables

None.

11.4.2 Environment Variables

- Database connection (event metadata, capacity counts, deadlines)

11.4.3 Assumptions

- The database contains valid and up-to-date event policies.
- EventInfo passed in from M3 contains a valid event ID.
- UserInfo contains a valid authenticated user ID.

11.4.4 Access Routine Semantics

validateCapacity(eventInfo, userInfo):

- transition: N/A
- output: Returns true if event capacity has not been reached.
- exception: EventFull if capacity is exhausted, InvalidEvent if event does not exist.

validateDeadline(eventInfo, userInfo):

- transition: N/A
- output: Returns true if registration is still open.
- exception: DeadlinePassed if the cutoff time has passed, InvalidEvent otherwise.

validateEligibility(eventInfo, userInfo):

- transition: N/A
- output: Returns true if the user meets role-based or event-specific criteria.
- exception: PermissionDenied if the user is not allowed to register.

getEventRules(eventID):

- transition: N/A
- output: Returns the complete rule set for the given event (capacity, deadlines, policies).
- exception: InvalidEvent if the event does not exist.

11.4.5 Local Functions

- lookupRules(eventID): fetches the rule set for an event from the datastore.
- isBeforeDeadline(eventInfo): compares current time to stored deadline.
- hasRemainingCapacity(eventInfo): queries current participant count.
- meetsEligibilityCriteria(eventInfo, userInfo): checks role access and custom restrictions.

12 MIS of Notification Handling Module (M7)

12.1 Module

`NotificationModule`

12.2 Uses

- User Authorization (M8)
- Registration Rules (M6)
- Payment Configuration (M4)

12.3 Syntax

12.3.1 Exported Constants

- `NOTIFICATION_TYPES`:
  - (”Confirmation”, ”Reminder”, ”Cancellation”)
  - Supported types of notifications.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
`SendNotification`	userID: Str, message: Str, notif_type: Str	success: bool	NotificationError
`getNotificationHist`	userID: Str	Notifications: List[Str]	RetrievalError
`subscribeUser`	userID: Str, subscribed: bool	success: bool	SubscribingError
`UnsubscribeUser`	userID: Str, subscribed: bool	success: bool	UnsubscribingError

12.4 Semantics

12.4.1 State Variables

- userNotifPreference: dict[UserID: bool]
  - Contains users notification preference.

12.4.2 Environment Variables

- `EMAIL_API`: Str
  - API endpoint for email notification delivery.
- `SMS_API`: Str
  - API endpoint for SMS notification delivery.

12.4.3 Assumptions

- Users have opted into receiving notifications.
- Third party email and SMS APIs are functional.

12.4.4 Access Routine Semantics

SendNotification(userID, message, notification_type):

- transition: send notification via Email and SMS API.
- output: Returns true if successfully sent.
- exception: NotificationError if API fails and message cannot be sent.

getNotificationHist(userID):

- transition: None.
- output: Returns list of previously sent out notifications.
- exception: RetrievalError if no history exists.

subscribeUser(userID, subscribed):

- transition: Adds user to the userNotifPreference list and sets value to true.
- output: Returns true upon successful subscription.
- exception: SubscribingError if invalid subscription.

UnsubscribeUser(userID, subscribed):

- transition: Updates userNotifPreference of specified user to false.
- output: Returns true on successful unsubscribing.
- exception: UnsubscribingError if user is not subscribed.

12.4.5 Local Functions

- None

13 MIS of User Authorization Module (M8)

13.1 Module

`UserAuthorizationModule`

13.2 Uses

- Access Control Module (M5): Uses identity and token validation functions to determine a user's permissions.
- Payment Processing Module (M1): Requires a verified user identity before initiating payment or refund operations.
- Sign-Up Module (M3): Uses authenticated user identifiers when creating bus/table/RSVP sign-up records.
- Notification Handling Module (M7): Uses authenticated user information to route notifications to the correct recipient.
- External Authentication Provider (e.g., OAuth/SSO): Provides the underlying authentication mechanism for secure login.

13.3 Syntax

13.3.1 Exported Constants

- `TOKEN_EXPIRY_MIN : Nat`
  The number of minutes before an issued access token expires.
- `REFRESH_TOKEN_SUPPORTED : Bool`
  Indicates whether refresh tokens are enabled in the current deployment configuration.

13.3.2 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

13.4 Semantics

13.4.1 State Variables

- `activeSessions : dict[token → UserID]`
  Stores active session mappings for all authenticated users.

- `tokenExpiryTimes : dict[token → Time]`
  Tracks expiration timestamps for each active token.

13.4.2 Environment Variables

- `AUTH_SERVER_URL : URL`
  External authentication provider endpoint (OAuth/SSO).

- `SystemTime : Time`
  Used to validate token expiration.

13.4.3 Assumptions

- All modules calling this one must supply a token obtained from `login` or `refreshToken`.

- The external authentication provider (OAuth/SSO) is available and reliable.

- Tokens are cryptographically signed by the authentication provider and cannot be tampered with by the client.

- Session expiry rules follow the SRS and Hazard Analysis requirements (e.g., 15-minute admin timeout).

13.4.4 Access Routine Semantics

`login(credentials)`

- transition: Adds new entry to `activeSessions` and `tokenExpiryTimes`.

- output: Returns a newly issued `AuthToken`.
- exception: `InvalidCredentials` if credentials rejected or `NetworkError` if authentication provider is unreachable.

`logout(token)`

- transition: Removes `token` from all session maps.
- output: `true` on success.
- exception: `TokenError` if token is not active.

`validateToken(token)`

- transition: none.
- output: `true` if token is valid and unexpired.
- exception: `TokenInvalid` or `TokenExpired`.

`getUserInfo(token)`

- transition: none.
- output: Returns the authenticated user's profile (ID, email, name, role).
- exception: `TokenInvalid` or `NetworkError`.

`refreshToken(refreshToken)`

- transition: Replaces expired/expiring token with new token, if supported.
- output: Returns newly issued `AuthToken`.
- exception: `TokenInvalid` if the supplied refresh token is invalid, `NotSupported` if refresh tokens are disabled.

13.4.5 Local Functions

- `decodeToken(token)`: Verifies token signature and extracts identity data.
- `isExpired(token)`: Checks whether current time exceeds stored expiry.
- `requestAuthServer(endpoint, payload)`: Handles secure communication with the external authentication provider.

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

14 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:

14.1 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.

14.2 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.

14.3 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

14.4 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.

14.5 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.

14.6 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.