

Module Interface Specification for Software Engineering

Team #5, Money Making Mauraders

Zhenia Sigayev

Justin Ho

Thomas Wang

Michael Shi

Johnny Qu

November 10, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

The base set of abbreviations is defined in the Software Requirements Specification (SRS) for Software Engineering. Additional terms that appear for the first time in this MIS are summarized in Table 1.

Term	Description
JWT	JSON Web Token used for representing authenticated sessions
OCR	Optical Character Recognition service used for receipt extraction
SLA	Service Level Agreement metrics that drive <code>SaL-*</code> requirements
SSO	Single Sign-On workflow backed by the McMaster identity provider
DOM	Browser Document Object Model exposed to the UI layer
IdP	Identity Provider for authentication
Idempotency Key	Unique token that prevents duplicate submissions (used in APIs)

Table 1: Additional abbreviations used in the MIS

Contents

1 Revision History	i
2 Symbols, Abbreviations and Acronyms	ii
3 Introduction	1
4 Notation	1
4.1 Core Abstract Data Types	1
5 Module Decomposition	3
6 MIS of Hardware Hiding Module (M1)	4
6.1 Module	4
6.2 Uses	4
6.3 Syntax	4
6.3.1 Exported Constants	4
6.3.2 Exported Access Programs	4
6.4 Semantics	5
6.4.1 State Variables	5
6.4.2 Environment Variables	5
6.4.3 Assumptions	5
6.4.4 Access Routine Semantics	5
6.4.5 Local Functions	6
7 MIS of User Interface Module (M2)	6
7.1 Module	6
7.2 Uses	6
7.3 Syntax	6
7.3.1 Exported Constants	6
7.3.2 Exported Access Programs	7
7.4 Semantics	7
7.4.1 State Variables	7
7.4.2 Environment Variables	7
7.4.3 Assumptions	7
7.4.4 Access Routine Semantics	8
7.4.5 Local Functions	9
8 MIS of Request Handler Module (M3)	9
8.1 Module	9
8.2 Uses	9
8.3 Syntax	9
8.3.1 Exported Constants	9

8.3.2	Exported Access Programs	9
8.4	Semantics	9
8.4.1	State Variables	9
8.4.2	Environment Variables	10
8.4.3	Assumptions	10
8.4.4	Access Routine Semantics	10
8.4.5	Local Functions	11
9	MIS of Receipt Processing Module (M4)	11
9.1	Module	11
9.2	Uses	11
9.3	Syntax	11
9.3.1	Exported Constants	11
9.3.2	Exported Access Programs	12
9.4	Semantics	12
9.4.1	State Variables	12
9.4.2	Environment Variables	12
9.4.3	Assumptions	12
9.4.4	Access Routine Semantics	12
9.4.5	Local Functions	13
10	MIS of Notification Module (M5)	13
10.1	Module	13
10.2	Uses	13
10.3	Syntax	13
10.3.1	Exported Constants	13
10.3.2	Exported Access Programs	14
10.4	Semantics	14
10.4.1	State Variables	14
10.4.2	Environment Variables	14
10.4.3	Assumptions	14
10.4.4	Access Routine Semantics	14
10.4.5	Local Functions	15
11	MIS of Authentication Module (M6)	15
11.1	Module	15
11.2	Uses	15
11.3	Syntax	15
11.3.1	Exported Constants	15
11.3.2	Exported Access Programs	16
11.4	Semantics	16
11.4.1	State Variables	16
11.4.2	Environment Variables	16

11.4.3 Assumptions	16
11.4.4 Access Routine Semantics	16
11.4.5 Local Functions	17
12 MIS of Data Model Module (M7)	17
12.1 Module	17
12.2 Uses	17
12.3 Syntax	18
12.3.1 Exported Constants	18
12.3.2 Exported Access Programs	18
12.4 Semantics	18
12.4.1 State Variables	18
12.4.2 Environment Variables	18
12.4.3 Assumptions	18
12.4.4 Access Routine Semantics	19
12.4.5 Local Functions	20
13 MIS of Audit Logging Module (M8)	20
13.1 Module	20
13.2 Uses	20
13.3 Syntax	20
13.3.1 Exported Constants	20
13.3.2 Exported Access Programs	20
13.4 Semantics	20
13.4.1 State Variables	20
13.4.2 Environment Variables	21
13.4.3 Assumptions	21
13.4.4 Access Routine Semantics	21
13.4.5 Local Functions	22
14 Appendix	23

3 Introduction

Software Engineering (the MES Finance Tracking Platform) modernizes the reimbursement workflow for MES clubs by centralizing submissions, approvals, and record keeping. This Module Interface Specification (MIS) translates the problem narrative in the Problem Statement and Development Plan, together with the requirements captured in the SRS, into precise interfaces for every module defined in the Module Guide (MG). Each specification details the exported syntax and semantics so development teams can implement, test, and integrate their assigned module without ambiguity.

Complementary documents include the SRS for behavioural intent and the MG for architectural decomposition. The latest implementation and documentation set are hosted at <https://github.com/McMaster-Engineering-Society/MES-Finance-Tracking>.

4 Notation

The structure of the MIS for modules follows ?, with the adaptation that template modules follow ?. The mathematical notation uses the conventions from Chapter 3 of ?. The operator ‘:=‘ denotes simultaneous assignment and conditional rules follow $(c_1 \Rightarrow r_1 | \dots | c_n \Rightarrow r_n)$.

The primitive data types used by Software Engineering are listed in Table 2. Derived data types are defined in Section 4.1. Sequences are lists whose elements share a type and are denoted $\langle \rangle$. Strings are sequences of characters. Tuples are ordered collections that may combine heterogeneous types. $\mathcal{P}(X)$ denotes the power set of X .

Data Type	Notation	Description
character	char	A single symbol or digit
integer	\mathbb{Z}	Whole numbers in $(-\infty, \infty)$
natural number	\mathbb{N}	Whole numbers in $[0, \infty)$
real	\mathbb{R}	Real numbers in $(-\infty, \infty)$
boolean	bool	{true, false}

Table 2: Primitive data types

4.1 Core Abstract Data Types

Identifiers are modeled as opaque strings (UUIDs) to avoid leaking storage details:

UserID, ClubID, RequestID, ReceiptID, NotificationID, AuditID: Unique strings identifying the respective domain concept.

SessionToken, RefreshToken: Signed strings that represent authenticated sessions per ACS-*.

Timestamp: UTC instant in ISO 8601 format.

URI: Locator of a resource accessible through HTTPS or object storage.

Binary: Sequence of bytes.

Money: Tuple ($amount \in \mathbb{R}$, $currency \in \text{ISO4217}$) with precision constraints from PRE-1.

CredentialInput: Structure containing the SSO provider identifier and opaque credential payload delivered by the browser.

LineItem: ($description : \text{String}$, $quantity \in \mathbb{N}^+$, $unitCost \in \mathbb{R}_{\geq 0}$).

SecretKey/SecretValue: Strings describing infrastructure secrets.

Composite records used throughout the MIS are:

- **UserContext** = ($userId : \text{UserID}$, $role : \text{Role}$, $clubs : \mathcal{P}(\text{ClubID})$, $session : \text{SessionToken}$).
- **RequestDraft** = ($clubId$, $amount : \text{Money}$, $expenseDate : \text{Timestamp}$, $category : \text{String}$, $description : \text{String}$, $lineItems : \langle \text{LineItem} \rangle$, $receipts : \langle \text{ReceiptID} \rangle$).
- **RequestRecord** extends **RequestDraft** with ($requestId$, $status : \text{RequestStatus}$, $submittedBy : \text{UserID}$, $createdAt : \text{Timestamp}$, $updatedAt : \text{Timestamp}$, $timeline : \langle \text{StatusChange} \rangle$).
- **RequestSummary** is the projection ($requestId$, $clubId$, $status$, $amount$, $createdAt$, $updatedAt$) for list views.
- **RequestFilter** = ($clubIds : \mathcal{P}(\text{ClubID})$, $statuses : \mathcal{P}(\text{RequestStatus})$, $dateRange : \text{TimeRange}$, $searchTerm : \text{String}$).
- **RequestPatch** stores partial updates: ($fields : \text{Map}[\text{String}, \text{Any}]$, $receipts : \langle \text{ReceiptID} \rangle$).
- **ReceiptUploadMetadata** = ($filename$, $contentType$, $byteSize \leq 10 \text{ MB}$, $sourceDevice$).
- **ReceiptMetadata** = ($receiptId$, $requestId?$, $storageUri : \text{URI}$, $checksum : \text{String}$, $ocrData : \text{Map}[\text{String}, \text{String}]$, $uploadedAt : \text{Timestamp}$).
- **BudgetSnapshot** = ($clubId$, $totalAllocated : \text{Money}$, $totalSpent : \text{Money}$, $pendingAmount : \text{Money}$, $lastUpdated : \text{Timestamp}$).
- **WorkflowEvent** = ($requestId$, $previousStatus$, $newStatus$, $actor : \text{UserID}$, $timestamp : \text{Timestamp}$, $comment : \text{String}$).
- **NotificationTemplate** = ($templateId$, $channel : \text{NotificationChannel}$, $subject$, $body$, $placeholders : \mathcal{P}(\text{String})$).
- **PendingNotification** = ($notificationId$, $event : \text{WorkflowEvent}$, $targets : \langle \text{UserID} \rangle$, $template : \text{NotificationTemplate}$, $earliestSend : \text{Timestamp}$, $status : \text{NotificationStatus}$).

- DispatchReport = ($sent : \langle \text{NotificationID} \rangle$, $failed : \langle (\text{NotificationID}, \text{ErrorCode}) \rangle$).
- WorkflowConfig = ($allowedTransitions : \mathcal{P}(\text{RequestStatus} \times \text{RequestStatus})$, $budgetRules : \text{Map}[\text{String}, \text{Any}]$).
- StatusChange = ($status : \text{RequestStatus}$, $timestamp$, $actor$, $notes : \text{String}$).
- AuditEvent = ($auditId$, $actor : \text{UserID}$, $action : \text{String}$, $subjectId : \text{String}$, $metadata : \text{Map}[\text{String}, \text{String}]$, $timestamp$).
- AuditFilter = ($actors : \mathcal{P}(\text{UserID})$, $actions : \mathcal{P}(\text{String})$, $subjects : \mathcal{P}(\text{String})$, $timeRange : \text{TimeRange}$).
- AuditExport = ($content : \text{Binary}$, $format : \text{ExportFormat}$, $generatedAt : \text{Timestamp}$).
- DeploymentConfig = ($region : \text{String}$, $environment : \{\text{local}, \text{staging}, \text{prod}\}$, $featureFlags : \text{Map}[\text{String}, \text{bool}]$).
- RuntimeContext = ($processId : \text{String}$, $networkAdapters : \mathcal{P}(\text{String})$, $secrets : \mathcal{P}(\text{SecretKey})$).
- MetricRecord = ($name : \text{String}$, $value : \mathbb{R}$, $timestamp : \text{Timestamp}$, $labels : \text{Map}[\text{String}, \text{String}]$).
- TimeRange = ($start : \text{Timestamp}$, $end : \text{Timestamp}$) with $start \leq end$.

Enumerations:

- Role ::= {club_member, club_exec, mes_reviewer, mes_admin}.
- RequestStatus ::= {Draft, Submitted, UnderReview, Approved, Rejected, Reimbursed}.
- ReminderType ::= {SubmissionReminder, ReviewReminder, BudgetThreshold}.
- NotificationChannel ::= {Email, InApp} and NotificationStatus ::= {Pending, Sent, Failed}.
- ExportFormat ::= {CSV, JSON, PDF}.
- ErrorCode ::= {Unauthorized, ValidationFailed, NotFound, Conflict, StorageFailure, External}

The generic result type is $\text{Result}(T) ::= \text{Success}(value : T) \mid \text{Failure}(code : \text{ErrorCode}, message : \text{String})$. The special type **Unit** represents the empty tuple.

5 Module Decomposition

The MIS follows the module hierarchy validated in the MG (Table 3). Only the leaf modules require implementation effort, but the hardware-hiding module is documented for completeness.

Level 1	Level 2
Hardware-Hiding Module	<ul style="list-style-type: none"> • M1: Hardware Hiding Module
Behaviour-Hiding Module	<ul style="list-style-type: none"> • M2: User Interface Module • M3: Request Handler Module • M4: Receipt Processing Module • M5: Notification Module
Software Decision Module	<ul style="list-style-type: none"> • M6: Authentication Module • M7: Data Model Module • M8: Audit Logging Module

Table 3: Module hierarchy followed in the MIS

6 MIS of Hardware Hiding Module (M1)

6.1 Module

The Hardware Hiding Module encapsulates the Node.js runtime, Vercel/AWS hosting environment, managed object storage, and messaging infrastructure. Its secret is the choice of infrastructure provider (AC??, AC??) so higher-level modules can treat compute, storage, and external connectivity as abstract capabilities.

6.2 Uses

This module is provided by the operating system and cloud platform; it does not use any modules defined within this project.

6.3 Syntax

6.3.1 Exported Constants

None. Deployment details are supplied at runtime through configuration.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
provisionRuntime	depConfig: Config	Deployment-	RuntimeContext
accessSecret	key: SecretKey	Result(SecretValue)	RuntimeFault
emitTelemetry	metric: MetricRecord	Result(Unit)	RuntimeFault

6.4 Semantics

6.4.1 State Variables

- runtimeContext: RuntimeContext — represents allocated compute, networking, and filesystem resources.
- secretCache: Map(SecretKey, SecretValue) — cached secrets fetched from the platform vault.
- telemetryBuffer: {MetricRecord} — batch of metrics awaiting transmission to the provider.

6.4.2 Environment Variables

- providerApi — control plane for Vercel/AWS.
- systemClock — hardware time source used for TLS token expiry.
- networkFabric — physical network links and load balancers.

6.4.3 Assumptions

- The cloud provider guarantees HTTPS/TLS 1.2+ as mandated by INT-2.
- Secrets accessed via accessSecret are rotated and scoped by operations so that leakage cannot violate PRI-2.
- providerApi enforces quotas; higher modules are expected to use retry semantics defined in FLT-3.

6.4.4 Access Routine Semantics

provisionRuntime(*depConfig*):

- transition: runtimeContext is initialized with compute and networking handles derived from depConfig; telemetryBuffer is reset.
- output: returns the RuntimeContext describing the provisioned environment.
- exception: raises RuntimeFault if providerApi rejects the configuration.

accessSecret(*key*):

- transition: secretCache[*key*] := providerApi.fetch(*key*) if not cached.
- output: Success(secretCache[*key*]).
- exception: Failure(RuntimeFault) if providerApi denies access or the key is missing.

`emitTelemetry(metric):`

- transition: metric appended to `telemetryBuffer`; buffer is flushed asynchronously.
- output: `Success(Unit)` once provider acknowledgement is received.
- exception: `Failure(RuntimeFault)` if the telemetry endpoint is unavailable for longer than the SLA retry window.

6.4.5 Local Functions

`flushTelemetryBuffer()`: pushes `telemetryBuffer` to `providerApi` in bounded batches.

7 MIS of User Interface Module (M2)

7.1 Module

The User Interface Module renders the responsive web application (Next.js/React with NextUI and TailwindCSS) that satisfies FRQ-1–FRQ-6, APP-*, ABL-*, and EUL-1. It collects user input, orchestrates client-side validation, and invokes back-end access programs exposed by the lower modules.

7.2 Uses

M2 uses M3 for request orchestration, M4 for secure receipt streaming, and M6 for session establishment.

7.3 Syntax

7.3.1 Exported Constants

None. Styling tokens and localization strings are maintained separately.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
handleLogin	credentials: CredentialInput	Result(UserContext)	Unauthorized
renderDashboard	ctx: UserContext	ViewModel	Unauthorized
submitRequest	ctx: UserContext, draft: RequestDraft, receiptIds: <ReceiptID>	Result(RequestID)	ValidationFailed
fetchRequestFeed	ctx: UserContext, filter: RequestFilter	Result(<RequestSummary>)	Unauthorized
downloadReceipt	ctx: UserContext, requestId: RequestID, receiptId: ReceiptID	Result(ReceiptBinary)	NotFound

7.4 Semantics

7.4.1 State Variables

- sessionCache: Map(SessionToken, UserContext) — mirrors active sessions for optimistic UI updates.
- draftCache: Map(UserID, RequestDraft) — stores unsent drafts to satisfy **FLT-2**.
- viewState: Map(UserID, ViewModel) — last rendered dashboard per user for accessibility testing.

7.4.2 Environment Variables

- browserWindow — DOM exposed by the client device.
- localStorage — persistent key-value storage within the browser sandbox.
- deviceCamera — optional capture device for receipt uploads on mobile.

7.4.3 Assumptions

- Client browsers support ES2020, Service Workers, and WCAG 2.1 AA features per **ABL-***.
- Network connectivity is sufficient to maintain HTTPS sessions; degraded states are surfaced via **FLT-1**.
- CredentialInput originates from the official McMaster IdP widget (no manual password handling in the UI).

7.4.4 Access Routine Semantics

handleLogin(*credentials*):

- transition: delegates to M6 to exchange CredentialInput for SessionToken; sessionCache[session] := context on success.
- output: Success(UserContext) populated from M6 per **FRQ-6**.
- exception: Failure(Unauthorized) if M6 rejects the credentials or device clock drift invalidates the SSO assertion.

renderDashboard(*ctx*):

- transition: none (rendering is pure with respect to module state, aside from updating viewState[ctx.userId]).
- output: ViewModel containing request summaries from M3 and budget widgets from M3/M7, satisfying **FRQ-3** and **APP-***.
- exception: Unauthorized if ctx.session is not present in sessionCache.

submitRequest(*ctx, draft, receiptIds*):

- transition: validates draft locally (field constraints, duplication checks), then calls M3::createRequest; draftCache entry for ctx.userId is cleared upon success.
- output: Success(RequestID) that tracks the submission required by **FRQ-1**.
- exception: ValidationFailed when client-side validation or M3 rejects the payload; draftCache retains the last attempt to support **FLT-2**.

fetchRequestFeed(*ctx, filter*):

- transition: none; passes filter to M3 and updates viewState.
- output: Success list limited to requests the user role can access (**FRQ-6**).
- exception: Unauthorized when ctx.role lacks permission for the requested clubs.

downloadReceipt(*ctx, requestId, receiptId*):

- transition: uses M3 for authorization and M4 for the binary stream; no local state mutation.
- output: Success(ReceiptBinary) streamed to the browser to satisfy **FRQ-4**.
- exception: NotFound if the receipt does not belong to the given request or has been purged per retention rules.

7.4.5 Local Functions

validateDraft(draft): ensures required fields are present, totals match PRE-1, and attachments meet byte limits before delegating to M3.

8 MIS of Request Handler Module (M3)

8.1 Module

The Request Handler Module encapsulates the business logic for reimbursement workflows (submission, review, approval, rejection, reimbursement). It enforces the policies tied to FRQ-1-FRQ-6, SAF-*, AC??, and coordinates persistence, audit logging, and notifications.

8.2 Uses

M3 uses M4 for receipt validation, M5 for notification fan-out, M6 to verify caller context, M7 for persistent storage, and M8 for audit trails.

8.3 Syntax

8.3.1 Exported Constants

- submissionWindow := 90 days — maximum age of receipts per MES policy.
- maxDraftsPerUser := 25 — soft limit used by the rate limiter.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
createRequest	ctx: UserContext, draft: RequestDraft	Result(RequestRecord)	ValidationFailed
transitionRequest	ctx: UserContext, requestId: RequestID, newStatus: RequestStatus, notes: String	Result(RequestRecord)	authorized
getRequests	ctx: UserContext, filter: RequestFilter	Result(RequestRecord)	authorized
getBudgetPosition	ctx: UserContext, clubId: ClubID	Result(BudgetSnapshot)	authorized

8.4 Semantics

8.4.1 State Variables

- workflowRules: WorkflowConfig — allowed transitions and budget thresholds for SAF-1.

- `rateLimiter`: `Map(UserID, (N, Timestamp))` — counts submissions per rolling window.
- `requestCache`: `Map(RequestID, RequestRecord)` — short-lived cache for repeated reads (max 5 minutes).

8.4.2 Environment Variables

- `systemClock` — used for SLA timing of **SaL-1**.
- `idempotencyStore` — per-request deduplication store using Idempotency Keys.

8.4.3 Assumptions

- `UserContext` passed in has already been validated by M6 and includes RBAC claims.
- ReceiptIDs listed in `draft` were created by M4 for the same club.
- All persistence errors propagated from M7 are retried based on **FLT-3**.

8.4.4 Access Routine Semantics

`createRequest(ctx, draft)`:

- `transition`: `assertRole(ctx, {club_member, club_exec})`; ensure `draft.expenseDate` lies within `submissionWindow`; call M4 to confirm receipt associations; persist new `RequestRecord` via M7; append audit entry through M8; enqueue notifications in M5 for reviewers.
- `output`: `Success(RequestRecord)` with status **Submitted**.
- `exception`: `ValidationFailed` when invariants fail, `StorageFailure` if M7 rejects the insertion after retries.

`transitionRequest(ctx, requestId, newStatus, notes)`:

- `transition`: `assertRole` for reviewer/administrator; verify $(currentStatus, newStatus) \in workflowRules.allowedTransitions$; persist state change via M7; append `StatusChange` to timeline; emit audit event (M8) and notifications (M5).
- `output`: `Success(updated RequestRecord)`.
- `exception`: `Unauthorized` if the user lacks permission or transition invalid; `Conflict` if the record was concurrently modified.

`getRequests(ctx, filter)`:

- `transition`: none (read-only); `requestCache` warmed for repeated access.

- output: Success list filtered for clubs the caller can access (FRQ-6), supporting dashboards per FRQ-3.
- exception: Unauthorized when filter requests clubs outside ctx.clubs and ctx.role is not reviewer/admin.

`getBudgetPosition(ctx, clubId):`

- transition: none; fetches from M7 and optionally updates requestCache for aggregated fields.
- output: Success BudgetSnapshot satisfying transparency goals in the Problem Statement.
- exception: Unauthorized when ctx lacks rights for the club.

8.4.5 Local Functions

`assertRole(ctx, allowedRoles):` verifies RBAC claims and throws Unauthorized otherwise.

`emitAudit(event):` wraps M8::recordEvent with MES-specific metadata (clubId, requestId, role).

9 MIS of Receipt Processing Module (M4)

9.1 Module

The Receipt Processing Module stores, secures, and enriches digital receipts. It enforces FRQ-4, SaL-3, PRI-1, AC??, and AC?? by abstracting storage/OCR decisions from other modules.

9.2 Uses

M4 uses M1 for file/object storage and managed OCR, and M7 for persisting metadata links.

9.3 Syntax

9.3.1 Exported Constants

- `maxFileSize := 10 MB` — aligns with SaL-3.
- `allowedMimeTypes := {image/png, image/jpeg, application/pdf}`.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
archiveReceipt	ctx: UserContext, payload: ReceiptBinary, meta: ReceiptUploadMetadata	Result(ReceiptID)	ValidationFailed
extractReceiptData	receiptId: ReceiptID	Result(ReceiptMeta)	ExternalFailure
streamReceipt	ctx: UserContext, receiptId: ReceiptID	Result(ReceiptBinary)	Unauthorized

9.4 Semantics

9.4.1 State Variables

- storageBucket: URI — base location for immutable receipt blobs.
- ocrConfig: Map(String, String) — credentials and endpoint details for the OCR provider.
- checksumIndex: Map(ReceiptID, String) — SHA-256 digest for tamper detection.

9.4.2 Environment Variables

- objectStore — managed storage (e.g., S3) accessed through M1.
- ocrService — external OCR API invoked through M1 networking.
- virusScanner — service used to reject unsafe uploads.

9.4.3 Assumptions

- Payloads forwarded by M2 already enforce client-side limits, but server-side size and type checks are authoritative.
- Storage URIs remain valid for the lifetime mandated by LNG-1 unless purged per compliance.
- OCR provider SLA satisfies SaL-3; failures propagate as ExternalFailure.

9.4.4 Access Routine Semantics

archiveReceipt(*ctx, payload, meta*):

- transition: validate ctx.role is authorized for the associated club; verify meta.byteSize \leq maxFileSize and MIME type in allowedMimeTypes; scan payload; stream to object-Store via M1; persist metadata in M7; return assigned ReceiptID.
- output: Success(ReceiptID) linked to ctx.userId for traceability.

- exception: ValidationFailed if checks fail; StorageFailure when objectStore is unreachable.

`extractReceiptData(receiptId):`

- transition: fetch blob from objectStore, invoke ocrService, update ReceiptMetadata.ocrData and timeline.
- output: Success updated metadata enabling auto-fill in M2 to reduce manual entry.
- exception: ExternalFailure when ocrService cannot parse the file after retry policy.

`streamReceipt(ctx, receiptId):`

- transition: none; authorizes via M3 (request ownership) and streams binary from storageBucket.
- output: Success binary fulfilling FRQ-4.
- exception: Unauthorized if ctx lacks access or NotFound if metadata missing.

9.4.5 Local Functions

`tagReceipt(meta, ctx):` adds MES-specific tags (clubId, requestId) to object metadata for lifecycle policies.

10 MIS of Notification Module (M5)

10.1 Module

The Notification Module encapsulates all outbound communication (email, in-app alerts) triggered by workflow events, reminders, and budget thresholds. It realizes FRQ-3, IMM-2, FLT-3, and ensures extensibility per AC??.

10.2 Uses

M5 uses M1 for channel delivery (SMTP/API gateways) and M7 for persisting notification state.

10.3 Syntax

10.3.1 Exported Constants

- `maxRetries := 3` — aligns with FLT-3.
- `reminderCadence := 14 days` — default spacing for SubmissionReminder messages.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
enqueueStatusNotification	event: WorkflowEvent, targets: {UserID}	Result(NotificationID)	ExternalFailure
scheduleReminder	clubId: ClubID, reminderType: ReminderType	Result(NotificationID)	ValidationFailed
dispatchPending	currentTime: Timestamp	DispatchReport	ExternalFailure

10.4 Semantics

10.4.1 State Variables

- notificationQueue: {PendingNotification} — ordered by earliestSend.
- templateCatalog: Map(String, NotificationTemplate) — ensures consistent branding per APP-*.
- deliveryLog: Map(NotificationID, NotificationStatus) — referenced by support staff per SUP-2.

10.4.2 Environment Variables

- emailGateway — transactional email service (e.g., SendGrid).
- inAppChannel — WebSocket or SSE channel for future extensibility.
- jobScheduler — cron-like service that invokes dispatchPending.

10.4.3 Assumptions

- Target user emails exist in M7 and are verified through M6.
- Channels use TLS and are rate-limited to comply with IMM-2.
- clock skew is negligible relative to reminderCadence.

10.4.4 Access Routine Semantics

enqueueStatusNotification(*event, targets*):

- transition: select template based on *event.newStatus*; create PendingNotification entries persisted via M7; update notificationQueue.
- output: Success(NotificationID) representing the queued entry.
- exception: ExternalFailure when template rendering or persistence fails.

`scheduleReminder(clubId, reminderType):`

- transition: compute earliestSend based on reminderCadence and existing PendingNotification entries; persist new entry referencing club leadership users.
- output: Success(NotificationID).
- exception: ValidationFailed when the club has no eligible recipients or reminder already pending.

`dispatchPending(currentTime):`

- transition: for each notification with $\text{earliestSend} \leq \text{currentTime}$, attempt delivery via channel defined in template; update deliveryLog and notificationQueue; retries scheduled per maxRetries.
- output: DispatchReport enumerating successes and failures for monitoring.
- exception: ExternalFailure when channel outages prevent delivery.

10.4.5 Local Functions

`renderTemplate(template, event):` substitutes placeholders with WorkflowEvent data, ensuring financial figures follow PRE-1.

11 MIS of Authentication Module (M6)

11.1 Module

The Authentication Module manages identity verification, session issuance, and RBAC claims. It enforces ACS-1, ACS-2, INT-2, PRI-2, and isolates the decision captured by AC??.

11.2 Uses

M6 uses M1 for secure cryptographic primitives and secret storage, and M7 for persisting user profiles and role bindings.

11.3 Syntax

11.3.1 Exported Constants

- `sessionTTL := 12 hours` — maximum lifetime of an access token.
- `refreshTTL := 14 days` — lifetime for refresh tokens.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
authenticate	credentials: CredentialInput	Result(UserContext)	Unauthorized
validateSession	token: SessionToken	Result(UserContext)	Unauthorized
refreshSession	refresh: RefreshToken	Result(SessionToken)	Unauthorized
revokeSession	token: SessionToken	Result(Unit)	Unauthorized

11.4 Semantics

11.4.1 State Variables

- sessionStore: Map(SessionToken, (UserID, expiry: Timestamp)) — active sessions.
- refreshIndex: Map(RefreshToken, (UserID, expiry)) — stored hashed refresh tokens.
- jwtKeyPair: (publicKey, privateKey) — signing material for JWTs stored via M1 secrets.

11.4.2 Environment Variables

- ssoProvider — McMaster IdP endpoints.
- cryptoModule — HMAC, RSA/ECDSA primitives provided by Node.js.
- systemClock — used for TTL enforcement.

11.4.3 Assumptions

- CredentialInput arrives over HTTPS directly from the IdP component embedded in M2.
- User roles stored in M7 are consistent with MES governance.
- Clock skew between ssoProvider and the runtime does not exceed 2 minutes.

11.4.4 Access Routine Semantics

authenticate(*credentials*):

- transition: call ssoProvider to validate the assertion; read user profile from M7; issue SessionToken and RefreshToken; persist to sessionStore/refreshIndex.
- output: Success(UserContext) containing RBAC claims for FRQ-6.
- exception: Unauthorized when IdP rejects the assertion or the account lacks MES affiliation.

`validateSession(token):`

- transition: none beyond sliding-session logic (optional) to satisfy INT-2.
- output: Success(`UserContext`) if `sessionStore[token]` exists and not expired.
- exception: Unauthorized otherwise.

`refreshSession(refresh):`

- transition: ensure `refreshIndex` entry is valid; issue new `SessionToken`; rotate refresh token.
- output: Success new `SessionToken` abiding by sessionTTL.
- exception: Unauthorized when refresh token missing or expired.

`revokeSession(token):`

- transition: delete token from `sessionStore` and corresponding refresh entries; propagate revocation events to caches.
- output: Success(`Unit`), supporting forced logouts per IMM-1.
- exception: Unauthorized if token unknown (idempotent success is also acceptable).

11.4.5 Local Functions

`buildUserContext(userRecord):` constructs `UserContext` with clubs and roles from M7 while redacting PII.

12 MIS of Data Model Module (M7)

12.1 Module

The Data Model Module encapsulates all persistent storage (MongoDB collections, indexes, transactions). It implements the schemas for users, clubs, budgets, requests, receipts, and audit artifacts, satisfying PRI-1, SaL-1, CAP-*, and the anticipated change AC??.

12.2 Uses

M7 uses M1 for low-level file handles, network connectivity, encryption at rest, and backup primitives.

12.3 Syntax

12.3.1 Exported Constants

- schemaVersion := 3 — current semantic version of the database schema.
- defaultPageSize := 50 — query pagination size for collection scans.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
insertRequestRecord	ctx: UserContext, draft: RequestDraft	Result(RequestRecord)	StorageFailure
updateRequestRecord	requestId: RequestID, patch: RequestPatch	Result(RequestRecord)	StorageFailure
queryRequests	filter: RequestFilter	Result([RequestRecord])	StorageFailure
persistReceiptMetadata	ReceiptMetadata	Result(ReceiptMetadata)	StorageFailure
fetchBudgetSnapshot	clubId: ClubID	Result(BudgetSnapshot)	StorageFailure
storeAuditEvent	event: AuditEvent	Result(AuditID)	StorageFailure

12.4 Semantics

12.4.1 State Variables

- connectionPool — handles to MongoDB nodes with automatic failover.
- collectionMap — mapping of logical collections (requests, receipts, budgets, users, audits) to physical names.
- migrationHistory — list of migration scripts applied up to schemaVersion.

12.4.2 Environment Variables

- mongoCluster — managed MongoDB service (Replica Set).
- storageEncryptionKey — fetched from M1 secrets for Field Level Encryption.
- backupScheduler — service that triggers snapshots.

12.4.3 Assumptions

- MongoDB provides ACID transactions for single-document writes, satisfying **PRI-1**.
- Indexes exist on {clubId, status, createdAt} to meet **SaL-1**.
- Connectivity to mongoCluster is stable; transient faults are retried before surfacing StorageFailure.

12.4.4 Access Routine Semantics

insertRequestRecord(*ctx, draft*):

- transition: create RequestRecord with generated RequestID, status **Submitted**, metadata from ctx; insert into requests collection; update budget aggregates.
- output: Success persisted RequestRecord with createdAt timestamp.
- exception: StorageFailure if uniqueness constraints (requestId, receiptIds) would be violated.

updateRequestRecord(*requestId, patch*):

- transition: apply patch fields (atomic update); maintain updatedAt timestamp and timeline.
- output: Success updated RequestRecord.
- exception: StorageFailure if the record is missing or write concern fails.

queryRequests(*filter*):

- transition: none; executes aggregation pipeline based on filter with pagination.
- output: Success list ordered by createdAt descending.
- exception: StorageFailure if query planner fails or connection lost.

persistReceiptMetadata(*meta*):

- transition: upsert metadata ensuring receiptId uniqueness; maintain indexes on checksum for deduplication.
- output: Success stored metadata.
- exception: StorageFailure on write concern errors.

fetchBudgetSnapshot(*clubId*):

- transition: none; reads budgets collection and aggregates pending totals from requests.
- output: Success BudgetSnapshot used by M2 dashboards.
- exception: StorageFailure on read errors.

storeAuditEvent(*event*):

- transition: insert audit entry and rotate partitions per retention policy.
- output: Success(AuditID) referencing the stored event.
- exception: StorageFailure when partition write fails.

12.4.5 Local Functions

ensureIndexes(): verifies critical indexes exist before accepting traffic (invoked at application startup).

13 MIS of Audit Logging Module (M8)

13.1 Module

The Audit Logging Module records immutable evidence for user and system actions to satisfy FRQ-5, AUD-1, AUD-2, SUP-2, and regulatory expectations from MES. Its secret is the log storage format and retention policy.

13.2 Uses

M8 uses M1 for secure timestamping and immutable blob storage, and M7 for indexing audit metadata.

13.3 Syntax

13.3.1 Exported Constants

- retentionPeriod := 7 years — aligns with MES finance policy.
- exportChunkSize := 10,000 events per batch.

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
recordEvent	ctx: UserContext, event: AuditEvent	Result(AuditID)	StorageFailure
queryEvents	ctx: UserContext, filter: AuditFilter	Result(<AuditEvent>)[]	Unauthorized
exportEvents	ctx: UserContext, range: TimeRange, format: ExportFormat	Result(AuditExport\$)	StorageFailure

13.4 Semantics

13.4.1 State Variables

- retentionPolicy: Map(String, TimeRange) — determines purge schedule per action type.

- `signingKey`: `SecretKey` — used to sign audit digests.
- `exportTracker`: `Map(UserID, Timestamp)` — throttles export frequency to prevent abuse.

13.4.2 Environment Variables

- `immutableStore` — write-once storage bucket for exported archives.
- `monotonicClock` — ensures ordering for tamper detection.
- `complianceQueue` — channel notifying MES admins about nearing retention cut-offs.

13.4.3 Assumptions

- Only users with role $\in \{\text{mes_reviewer}, \text{mes_admin}\}$ may call query/export routines.
- The storage medium provides append-only guarantees to preserve integrity.
- System clock monotonicity is enforced via NTP per IMM-1.

13.4.4 Access Routine Semantics

`recordEvent(ctx, event)`:

- transition: enrich event.metadata with ctx.role and request/club references; call M7::storeAuditEvent, sign digest and optionally forward to immutableStore; schedule purge job for retentionPeriod.
- output: Success(AuditID).
- exception: StorageFailure if persistence fails (caller decides whether to retry or halt per criticality).

`queryEvents(ctx, filter)`:

- transition: none; verifies ctx.role authorization before delegating to M7 query; redacts sensitive metadata for club-level reviewers.
- output: Success list ordered descending by timestamp, supporting SUP-2.
- exception: Unauthorized if ctx.role lacks privileges.

`exportEvents(ctx, range, format)`:

- transition: ensure ctx.role is `mes_admin`; fetch events chunked by exportChunkSize; package into requested format; store in immutableStore and return download pointer.
- output: Success(AuditExport) with signed checksum.
- exception: StorageFailure when immutableStore unavailable or query fails.

13.4.5 Local Functions

`summarizeEvent(events)`: aggregates high-level statistics (counts by action) for dashboard widgets without exposing raw data.

14 Appendix

No supplemental material is provided at this time.

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:

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)