Module Interface Specification for Software Engineering

Team #5, Money Making Mauraders
Zhenia Sigayev
Justin Ho
Thomas Wang
Michael Shi
Johnny Qu

January 14, 2026

1 Revision History

Date	Version	Notes
Wednesday, January 14, 2026	1.2	Added table grid/wrapping fixes, map notation, and removed template text.
Tuesday, November 12, 2025	1.1	Added feedback from design doc marking; clarified tables and notation.
Monday, November 10, 2025	1.0	Initial Document

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/zheniasigayev/MES-Finance-Tracking/blob/main/docs/SRS/SRS.pdf.

Contents

3 Introduction

The following document details the Module Interface Specifications (MIS) for the MES Finance Tracking Platform, a web-based system that streamlines the submission, approval, and monitoring of reimbursement requests for McMaster Engineering Society clubs. By centralizing digital expense forms, supporting documentation, and reviewer workflows, the platform shortens processing times and improves transparency for students and finance administrators alike.

This document specifies the MIS for each software module in the MES Finance Tracking Platform, outlining how components interact to deliver the functionality described in the accompanying design artifacts.

Complementary documents include the System Requirement Specifications, Module Guide, and complete documentation and implementation can be found at https://github.com/zheniasigayev/MES-Finance-Tracking.

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 (-∞, ∞)

Derived data types used in this document:

- **Sequence**: ordered list of elements of the same type (notation: sequence T).
- **String**: sequence of characters.
- **Tuple**: ordered list of values that may be of different types (notation: $(t_1, t_2, \dots)$).
- **Map**: written as map $A \to B$; a total association from keys of type A to values of type B (absent keys are errors unless stated otherwise).

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	• M1: Hardware Hiding Module
Behaviour-Hiding	Input Parameters Output Format Output Verification Temperature ODEs Energy Equations Control Module Specification Parameters Module
Software Decision	Sequence Data Structure ODE Solver Plotting

Table 1: Module Hierarchy

6 MIS of Hardware Hiding Module (M1)

6.1 Module

Hardware Hiding Module (M1)

6.2 Uses

None.

6.3 Syntax

6.3.1 Exported Constants

- **RuntimeVersion**: string — Declares the LTS Node.js runtime version made available by the hosting platform (Vercel/AWS) so that the Behaviour-Hiding and Software Decision modules can align their server-side features and package targets.
- **DefaultRegion**: string — Identifies the primary cloud deployment region used for the platform's compute, database, and object storage endpoints to satisfy latency expectations from the SRS operational environment.
- **MaxConcurrentRequests**: $\mathbb{N}$ — Upper bound on simultaneous backend invocations guaranteed by the provider, chosen to satisfy the SRS capacity requirement of supporting at least 500 concurrent users.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
getRuntimeContext	-	RuntimeContext	ConfigNotFound
provisionPersistence	PersistenceTarget, PersistencePolicy	PersistenceHandle	ProvisioningError
openFileChannel	StorageRequest	StorageHandle	StorageError
resolveSecret	SecretKey	SecretValue	SecretNotFound

6.4 Semantics

6.4.1 State Variables

- **activeHandles**: sequence of PersistenceHandle — Tracks open database or object storage connections issued through the module.
- **cachedSecrets**: map SecretKey $\rightarrow$ SecretValue — Maintains decrypted secrets for the lifetime of a runtime invocation to minimize calls to the provider's secret manager.

6.4.2 Environment Variables

- **cloudPlatform**: descriptor of the underlying infrastructure (e.g., Vercel Edge Functions or AWS Lambda + S3) that supplies compute, networking, and storage abstractions.
- **processEnv**: map string → string exposing environment variables injected by the hosting provider, including connection strings and API keys referenced in the Development Plan.
- **filesystem**: ephemeral file system mount allocated per invocation for staging uploads before they are persisted to long-term storage.
- **networkInterface**: outbound HTTPS client managed by the runtime for communicating with third-party services such as SendGrid or future payment APIs.

6.4.3 Assumptions

- The cloud platform guarantees availability targets and auto-scaling properties outlined in the SRS environment and capacity requirements.
- Required environment variables (database URIs, storage bucket identifiers, API credentials) are injected securely by the deployment pipeline defined in the Development Plan.
- Serverless function instances share no persistent disk state; any durable data must be written through `provisionPersistence` or `openFileChannel`.

6.4.4 Access Routine Semantics

getRuntimeContext( )

- transition: None.
- output: Returns a RuntimeContext record containing **RuntimeVersion**, **DefaultRegion**, exposed environment variables, and runtime limits (memory, execution time) advertised by **cloudPlatform**.
- exception: ConfigNotFound is raised if mandatory runtime metadata is missing from **processEnv**.

provisionPersistence(target, policy)

- transition: Adds a PersistenceHandle corresponding to the requested *target* (document store, object storage, or cache) to **activeHandles**. The handle encapsulates connection pooling or signed URLs as required by the target.

- output: Returns the newly created PersistenceHandle configured according to *policy* (e.g., read/write role, retention period).
- exception: ProvisioningError if the target infrastructure is unreachable or policy constraints cannot be satisfied by the provider.

openFileChannel(request)

- transition: Streams the binary payload described by *request* from **filesystem** to an object storage location derived from **cloudPlatform**. Updates **activeHandles** with the resulting StorageHandle for lifecycle management.
- output: Returns a StorageHandle containing the canonical URI and checksum for the persisted object so that the Receipt Processing module can reference it.
- exception: StorageError when the payload exceeds provider-imposed limits or when the storage backend reports an error.

resolveSecret(key)

- transition: If *key* is not present in **cachedSecrets**, retrieves the secret value from the provider-managed vault and caches it for the remaining invocation lifespan.
- output: Returns the SecretValue associated with *key* so downstream modules can authenticate with external services (e.g., MongoDB, SendGrid).
- exception: SecretNotFound when the requested key is absent from the vault or access is denied.

6.4.5 Local Functions

None.

6.4.6 Considerations

- This module virtualizes physical infrastructure so higher-level modules can operate independent of Vercel/AWS specific APIs, satisfying the information-hiding intent of the Module Guide.
- Capacity thresholds published through **MaxConcurrentRequests** ensure the Request Handler and Notification modules can plan throttling logic that respects SRS capacity and scalability constraints.
- Secrets resolved via `resolveSecret` enable future integration with the MES monorepo while keeping credential management centralized, as described in the Development Plan.

7 MIS of User Interface Module (M2)

7.1 Module

User Interface Module (M2)

7.2 Uses

- Request Handler Module (M3) for fetching and mutating reimbursement request data through stable APIs.
- Receipt Processing Module (M4) for streaming uploaded receipt binaries to the backend and surfacing validation errors to the user.
- Authentication Module (M6) for deriving the current user's role, session state, and effective permissions to drive role-aware rendering.

7.3 Syntax

7.3.1 Exported Constants

- **MAX_VISIBLE_REQUESTS_PER_PAGE**: N — Default number of reimbursement requests shown in paginated dashboards (e.g., 20), chosen to balance readability and page load times.
- **AUTOSAVE_INTERVAL_MS**: N — Interval in milliseconds between automatic draft saves on long forms, supporting incremental data saving and fault tolerance.
- **TOAST_DURATION_MS**: N — Duration in milliseconds that transient notification banners remain visible after operations (submit, approve, error).
- **SUPPORTED_LOCALES**: sequence of String — List of locale identifiers (e.g., {`"en-CA"`}) used for formatting dates, currency, and numbers.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderLanding Page	sessionToken	ViewState	SessionInvalid
renderDashboard	userId, sessionToken	ViewState	SessionInvalid, AccessDenied
renderRequest Form	requestId (optional), sessionToken	ViewState	SessionInvalid, RequestNotFound, AccessDenied
handleForm Submit	formData, sessionToken	NavigationOutcome	SessionInvalid, ValidationError, BackendFailure
toggleTheme	preferredTheme, sessionToken	UISettings	SessionInvalid
showTutorial	sessionToken	TutorialState	SessionInvalid

7.4 Semantics

7.4.1 State Variables

- **currentView**: ViewState — Encodes the currently active screen, filters, and any transient UI state (e.g., open modals, active tab).
- **draftBuffers**: map RequestId → FormDraft — Holds unsaved and autosaved form contents for reimbursement requests while the user is editing.
- **uiSettings**: UISettings — Captures per-user display preferences such as theme (light/dark), density, and accessibility options (reduced motion).

7.4.2 Environment Variables

- **browserWindow**: Represents the client-side execution environment (viewport size, input devices, network connectivity).
- **routingLayer**: Client-side router used to navigate between pages and synchronize URL state with **currentView**.
- **requestAPI**: Interface to Request Handler Module (M3) used to fetch and mutate reimbursement request data.
- **receiptAPI**: Interface to Receipt Processing Module (M4) used to upload files and retrieve receipt-related errors.
- **authClient**: Interface to Authentication Module (M6) providing current user identity, roles, and session validity.

7.4.3 Assumptions

- The user's identity and role have been established by Authentication Module (M6) before privileged UI actions are attempted.
- Request Handler Module (M3) and Receipt Processing Module (M4) expose stable HTTP or RPC endpoints with documented error codes.
- The browser environment supports standard web APIs used by the framework (Next.js/React), or suitable polyfills are applied.
- Network connectivity may be intermittent; autosave and optimistic updates must handle transient failures gracefully.
- Accessibility requirements (keyboard navigation, contrast ratios, ARIA attributes) are enforced at design time and verified via tooling.

7.4.4 Access Routine Semantics

renderLandingPage(sessionToken)

- transition: Validates *sessionToken* via **authClient**. Initializes **currentView** to the appropriate landing surface (login screen for anonymous users, dashboard redirect for authenticated users).
- output: Returns a ViewState representing the landing page, including any contextual alerts (e.g., maintenance banners).
- exception: Raises SessionInvalid if *sessionToken* is expired, malformed, or rejected by **authClient**.

renderDashboard(userId, sessionToken)

- transition: Confirms session validity and that *userId* matches the authenticated identity or has delegated access. Fetches summarized request data via **requestAPI** and populates **currentView** with paginated tables, filters, and quick actions.
- output: Returns a ViewState containing the dashboard layout, including current filters, selected club, and summarized metrics (e.g., pending, approved, rejected counts).
- exception: Raises SessionInvalid on authentication failure, or AccessDenied if the user lacks permission to view the specified dashboard scope.

renderRequestForm(requestId, sessionToken)

- transition: If *requestId* is provided, retrieves the existing reimbursement request via **requestAPI** and seeds **draftBuffers**[requestId] with persisted values. Otherwise, initializes a new draft entry. Applies client-side validation rules and populates dynamic form fields (e.g., club list, budget lines).

- output: Returns a ViewState representing the active reimbursement form, including any validation messages and previously autosaved draft content.
- exception: Raises SessionInvalid if authentication fails, RequestNotFound if *requestId* does not exist or is inaccessible, or AccessDenied if the user cannot edit the targeted request.

handleFormSubmit(formData, sessionToken)

- transition: Performs client-side validations (required fields, numeric ranges, file constraints) and displays inline errors if any rule fails. On success, calls **requestAPI** and optionally **receiptAPI** to persist the request and any attached receipts. Clears the corresponding entry in **draftBuffers** upon confirmed backend success and updates **currentView** to reflect the new status.
- output: Returns a NavigationOutcome indicating whether to remain on the form (with errors), navigate back to the dashboard, or redirect to a confirmation view.
- exception: Raises SessionInvalid if the session check fails, ValidationError if client-side validation detects issues, or BackendFailure if the call to downstream modules fails after validation passes.

toggleTheme(preferredTheme, sessionToken)

- transition: Validates *preferredTheme* against supported values (e.g., light, dark, system). Updates **uiSettings** and persists the preference via a lightweight call to **requestAPI** or a dedicated settings endpoint.
- output: Returns updated UISettings describing the active theme and related appearance options.
- exception: Raises SessionInvalid if the user's session cannot be confirmed.

showTutorial(sessionToken)

- transition: Determines if the first-time user tutorial should be displayed based on profile flags and prior completions. Updates **currentView** to include tutorial overlays and progressive hints on the dashboard or form pages.
- output: Returns a TutorialState describing which steps are active and whether completion has been recorded.
- exception: Raises SessionInvalid if the session is not valid.

7.4.5 Local Functions

None.

8 MIS of Receipt Processing Module (M4)

8.1 Module

ReceiptProcessing

8.2 Uses

- Authentication Module (M6) for verifying that a caller has sufficient permissions to interact with a stored receipt.
- Data Model Module (M7) for persisting receipt metadata, OCR extraction results, and associating receipts with reimbursement requests.
- Audit Logging Module (M8) for recording immutable traces of upload, access, extraction, and deletion actions.
- External storage and OCR services (e.g., AWS S3 and Amazon Textract) identified in the Module Guide and SRS for hosted file storage and automated data extraction.

8.3 Syntax

8.3.1 Exported Constants

- **ACCEPTED_FILE_TYPES**: Set of MIME types {`image/png`, `image/jpeg`, `application/pdf`} defining the allowable receipt formats described in the SRS.
- **MAX_RECEIPT_FILE_SIZE**: Maximum upload size of 10 MB, ensuring conformance with the non-functional upload latency target.
- **SIGNED_URL_TTL**: Duration (in minutes) for which a generated secure access link to a stored receipt remains valid.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
uploadReceipt	ReceiptFile, ReceiptMetadata, UserID	ReceiptRecord	InvalidFileFormat, FileTooLarge, UnauthorizedAccess, StorageFailure
extractReceiptData	ReceiptID	ExtractedReceiptData	ReceiptNotFound, OCRFailure, StorageFailure
getReceiptLink	ReceiptID, UserID	URL	ReceiptNotFound, UnauthorizedAccess, StorageFailure
removeReceipt	ReceiptID, UserID	–	ReceiptNotFound, UnauthorizedAccess, ImmutableRequestState, StorageFailure

8.4 Semantics

8.4.1 State Variables

- receiptIndex: map ReceiptID → ReceiptRecord capturing storage location, uploader, request association, timestamps, and validation flags.
- extractionCache: map ReceiptID → ExtractedReceiptData persisted for reuse across module calls.
- storageCredentials: Handle with scoped credentials for interacting with the managed receipt storage service.

8.4.2 Environment Variables

- storageService: External object storage endpoint (e.g., AWS S3 bucket) for binary receipt files.

- ocrService: External OCR provider (e.g., Amazon Textract) capable of parsing receipt images into structured data.
- auditChannel: Interface exposed by the Audit Logging Module (M8) for recording receipt lifecycle events.

8.4.3 Assumptions

- The caller has already been authenticated and provides a UserID that maps to an existing account and role via the Authentication Module.
- The reimbursement request referenced within ReceiptMetadata exists and is mutable according to business rules managed by the Request Handler Module (M3).
- Credentials for storageService and ocrService are valid and provisioned prior to module invocation.
- Network connectivity to external services is available when upload, extraction, or deletion operations are attempted.

8.4.4 Access Routine Semantics

uploadReceipt(receiptFile, metadata, uploaderId):

- transition: Validate that receiptFile MIME type is in ACCEPTED_FILE_TYPES and its size does not exceed MAX_RECEIPT_FILE_SIZE. Verify that uploaderId is authorized to add receipts for the target reimbursement request. Persist receiptFile to storageService, create or update the associated ReceiptRecord in receiptIndex with a new ReceiptID, storage pointer, metadata, and timestamp, and emit an audit log entry.
- output: Return the created ReceiptRecord, including the ReceiptID assigned to the stored file.
- exception: Raise InvalidFileFormat if MIME validation fails, FileTooLarge if size constraints are exceeded, UnauthorizedAccess if uploaderId lacks the required role, or StorageFailure if persistence to storageService fails.

extractReceiptData(receiptId):

- transition: If extractionCache already contains receiptId, reuse the cached result; otherwise, retrieve the receipt binary from storageService and invoke ocrService to parse the receipt. Persist the structured response in extractionCache and associate it with the corresponding ReceiptRecord for downstream processing and validation.
- output: Return the ExtractedReceiptData containing amounts, vendor, dates, and other parsed fields for the receipt.

- exception: Raise ReceiptNotFound if receiptId is absent from receiptIndex, OCRFailure if ocrService cannot produce a result, or StorageFailure if the receipt binary cannot be retrieved.

getReceiptLink(receiptId, requesterId):

- transition: Confirm that requesterId is permitted to view the receipt according to reimbursement request visibility rules. Optionally record the access attempt through auditChannel.
- output: Return a time-bound, pre-signed URL (valid for SIGNED_URL_TTL) that allows the requester to download the stored receipt from storageService.
- exception: Raise ReceiptNotFound if receiptId is unknown, UnauthorizedAccess if requesterId lacks privileges, or StorageFailure if URL generation fails.

removeReceipt(receiptId, requesterId):

- transition: Verify that requesterId has permission to remove the receipt and that the linked reimbursement request is still editable. Delete the receipt binary from storageService, remove the entry from receiptIndex and extractionCache, and log the removal.
- output: None.
- exception: Raise ReceiptNotFound if the ReceiptID is missing, UnauthorizedAccess if requesterId lacks rights, ImmutableRequestState if the reimbursement request is locked (e.g., already approved), or StorageFailure if deletion from storageService fails.

8.4.5 Local Functions

- isAllowedFormat(fileMime): Boolean helper returning true when fileMime is contained in ACCEPTED_FILE_TYPES.
- isAuthorized(actorId, requestId, action): Queries the Authentication Module for the actor's roles and the Request Handler Module for the reimbursement state to ensure an operation is permitted.
- writeAuditEntry(event): Delegates to auditChannel to persist a structured log for compliance traceability.

9 MIS of Notification Module (M5)

9.1 Module

NotificationModule

9.2 Uses

- Receipt Processing Module (M4) sending a notifcation to the user upon successful upload and processing of a receipt.
- Authentication Module (M6) for verifying user contact details and preferences before dispatching notifications.
- Data Model Module (M7) for logging notification history and statuses after the messages are sent.

9.3 Syntax

9.3.1 Exported Constants

- notificationTypes: Set of supported notification categories, including receiptUploaded, requestApproved, requestRejected, defining the scenarios in which users can be notified.
- clubId: Unique identifier for the McMaster Engineering Society club associated with the user receiving the notification.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
notifyUser	userId, notificationType, email	Email Content	InvalidEmailFormat, UnauthorizedAccess, NotificationFailure

9.4 Semantics

9.4.1 State Variables

- reimbursementProcessIndex: map RequestProcessStep → NotificationType indicating when each notification should be sent.
- notificationIndex: map NotificationID → NotificationRecord capturing recipient, notification type, notification message, timestamp, and delivery status.

9.4.2 Environment Variables

- None

9.4.3 Assumptions

- There will be 2 types of notifcations. One will notify the user within the application UI that their receipt has been successfully uploaded and processed. The second will notify both the user and the club executive team when a reimbursement request has been approved or rejected via email.

9.4.4 Access Routine Semantics

notifyUser(userId, notificationType, notificationContent):

- transition: Validate that userId is authorized to receive notifications and that notificationType is supported. Construct the notification message using notificationContent and dispatch it to the current user and the club executive team. Log the notification attempt in notificationIndex with status.
- output: Return the full notification content that was sent to the user.
- exception: NotificationFailure if the email service reports an error.

notifyClubExecutive(clubId, notificationType, notificationContent):

- transition: Retrieve the contact details of the club executive team associated with clubId. Construct the notification message using notificationContent and dispatch it to the club executive team. Log the notification attempt in notificationIndex with status.
- output: Return the full notification content that was sent to the club executive team.
- exception: NotificationFailure if the email service reports an error.

10 MIS of Authentication Module (M6)

10.1 Module

Authentication Module

10.2 Uses

- Hardware Hiding Module (M1) for retrieving environment secrets, hashing parameters, and SSO adapters required to validate identities securely.
- Data Model Module (M7) for persisting user profiles, membership links, refresh tokens, and role assignments that underpin authorization checks.

10.3 Syntax

10.3.1 Exported Constants

- **ACCESS_TOKEN_TTL_MINUTES**: $\mathbb{N}$ — Length of time a signed access token remains valid before re-authentication is required.
- **REFRESH_TOKEN_TTL_DAYS**: $\mathbb{N}$ — Maximum lifetime of a refresh token stored in the Data Model module.
- **MAX_FAILED_ATTEMPTS**: $\mathbb{N}$ — Number of consecutive failed login attempts permitted before the account enters a lockout period as required by ACS-1.
- **SUPPORTED_ROLES**: Set of strings $\{\texttt{member}, \texttt{executive}, \texttt{financeAdmin}\}$ describing valid RBAC roles referenced across the MIS.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
authenticateUser	Credentials, ClientContext	AuthResult	InvalidCredentials, IdentityProviderError, AccountLocked
issueSession	UserID, DeviceFingerprint	SessionEnvelope	UnauthorizedAccess, SecretRotationInProgress
verifyAuthorization	SessionToken, Action, ResourceID	AccessDecision	InvalidToken, UnauthorizedAccess
revokeSession	SessionToken	-	InvalidToken

10.4 Semantics

10.4.1 State Variables

- **sessionStore**: map SessionToken → SessionRecord capturing issuedAt, expiresAt, userId, device metadata, and whether the session has been revoked.
- **refreshTokenIndex**: map RefreshToken → RefreshRecord linking long-lived tokens to userId and rotation counters.
- **failedAttemptCounter**: map UserID → $\mathbb{N}$ tracking consecutive unsuccessful authentication attempts to enforce **MAX_FAILED_ATTEMPTS**.
- **rbacPolicy**: map Role → set(ActionPattern) representing the RBAC rules derived from **SUPPORTED_ROLES**.

10.4.2 Environment Variables

- **identityProvider**: External identity platform (e.g., Microsoft Entra ID via NextAuth.js) that validates credentials and SSO assertions.
- **secretManager**: Handle to encrypted signing keys and hashing parameters supplied through the Hardware Hiding Module.
- **clock**: Trusted time source shared across the deployment for expiry evaluation.
- **auditChannel**: External logging sink (via M8) where authentication outcomes are dispatched for compliance review.

10.4.3 Assumptions

- The identityProvider exposes OAuth/OpenID Connect endpoints compatible with the SSO configuration outlined in the Module Guide.
- All communications with external providers occur over TLS and secrets retrieved through **secretManager** remain valid for the duration of the invocation unless signaled by rotation events.
- User records stored through the Data Model module contain normalized identifiers and active role assignments so RBAC checks operate deterministically.
- Audit logging is asynchronous, allowing authentication flows to proceed even if auditChannel experiences transient latency.

10.4.4 Access Routine Semantics

authenticateUser(credentials, context):

- transition: Submit credentials to identityProvider. On success, reset the caller's entry in **failedAttemptCounter**; on failure, increment the counter and lock the account when **MAX_FAILED_ATTEMPTS** is reached. Emit a summary event to **auditChannel**.
- output: Return an AuthResult containing userId, issued claims (roles, permissions), and whether multi-factor verification is required.
- exception: InvalidCredentials when the identityProvider rejects the credentials, IdentityProviderError on remote service failures, AccountLocked when the caller exceeded **MAX_FAILED_ATTEMPTS**.

issueSession(userId, fingerprint):

- transition: Retrieve signing secrets from **secretManager**, derive the user's roles from **rbacPolicy** and the Data Model module, and mint access/refresh tokens scoped to **ACCESS_TOKEN_TTL_MINUTES** and **REFRESH_TOKEN_TTL_DAYS**. Persist the resulting SessionRecord in **sessionStore** and update **refreshTokenIndex**.
- output: Return a SessionEnvelope with encoded SessionToken, RefreshToken, expiration timestamps, and derived claims.
- exception: UnauthorizedAccess when the user does not have an active account, SecretRotationInProgress when signing keys are being rolled and issuing a new session would violate key consistency.

verifyAuthorization(sessionToken, action, resourceId):

- transition: Validate sessionToken signature using **secretManager**, confirm it exists in **sessionStore**, and evaluate whether the embedded role grants permission for the requested action/resource pair using **rbacPolicy**. Refresh rolling expiration if a sliding window policy is configured.
- output: Return an AccessDecision (`allow` or `deny`) plus contextual information (derived role, reason).
- exception: InvalidToken when the token signature/expiry fails validation, UnauthorizedAccess when RBAC rules do not permit the action.

revokeSession(sessionToken):

- transition: Mark the corresponding SessionRecord inside **sessionStore** as revoked, delete associated refresh tokens from **refreshTokenIndex**, and notify **auditChannel**.
- output: None.
- exception: InvalidToken if the supplied token cannot be found or has already expired.

10.4.5 Local Functions

- **loadUserProfile(userId)**: Queries the Data Model module for the latest user metadata and roles.
- **deriveClaims(profile, context)**: Normalizes the user's club memberships, privileges, and session context (device, location) into the claims stored within tokens.
- **recordAudit(event)**: Normalizes authentication outcomes and writes them to **auditChannel**.

10.4.6 Considerations

- RBAC definitions in **rbacPolicy** must remain synchronized with Request Handler use cases to ensure FRQ-6 and ACS-2 are met.
- Session issuance intentionally depends on Hardware Hiding for signing secrets so cryptographic agility (algorithm or key changes) is captured within a single module boundary.
- This module exposes only opaque tokens; upstream callers never manipulate passwords or session internals directly, preserving the information-hiding intent of the Module Guide.

11 MIS of Data Model Module (M7)

11.1 Module

Data Model Module

11.2 Uses

- None, this is a foundational module that other modules depend on for data persistence.

11.3 Syntax

11.3.1 Exported Constants

- databaseURL: string that represents the database connection URL
- maxBatchSize: value representing the maximum number of records that can be processed in a single batch operation
- maxConnections: value representing the maximum number of concurrent connections to the database
- defaultTimeout: value representing the default timeout duration for database operations

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
query	queryString, parameters	queryResult	InvalidQuery, DatabaseConnectionError, TimeoutError
insert	tableName, receiptImage/document: Object	insertResult, recordId	InvalidRecordData, DatabaseConnectionError, TimeoutError
update	tableName, recordId, updatedFields	updateResult, recordId	RecordNotFound, InvalidUpdateData, DatabaseConnectionError, TimeoutError
delete	tableName, recordId	deleteResult	RecordNotFound, DatabaseConnectionError, TimeoutError

11.4 Semantics

11.4.1 State Variables

- databaseState: Represents the current state of the database and its connection status

11.4.2 Environment Variables

- databaseServer: represents the external database system holding the data
- fileStorageService: represents the external file storage service for storing receipt images/documents

11.4.3 Assumptions

- The databaseServer is accessible and running
- The fileStorageService is accessible and running
- The input parameters for each access program are valid and conform to the expected formats
- The module has the necessary permissions to perform CRUD operations on the database and file storage service
- The database schema is predefined and known to the module
- Authentication and authorization are handled by other modules before accessing this module
- Network connectivity to the databaseServer and fileStorageService is stable during operations

11.4.4 Access Routine Semantics

query(queryString, tableName, params):

- transition: Verify databaseState is connected. Execute the queryString with the provided parameters against the databaseServer.
- output: Return the queryResult containing the results of the executed query.
- exception: Raise InvalidQuery if the queryString is malformed, DatabaseConnectionError if unable to connect to the database, or TimeoutError if the operation exceeds defaultTimeout.

insert(tableName, document):

- transition: Verify databaseState is connected. Insert the provided document into the specified tableName in the databaseServer.
- output: Return the insertResult indicating success and the recordId of the newly created record.

- exception: Raise InvalidRecordData if the document is malformed, DatabaseConnectionError if unable to connect to the database, or TimeoutError if the operation exceeds defaultTimeout.

update(tableName, recordId, updatedFields):

- transition: Verify databaseState is connected. Update the record with recordId in tableName using the provided updatedFields.
- output: Return the updateResult indicating success and the recordId of the updated record.
- exception: Raise RecordNotFound if the recordId does not exist, InvalidUpdateData if updatedFields are malformed, DatabaseConnectionError if unable to connect to the database, or TimeoutError if the operation exceeds defaultTimeout.

delete(tableName, recordId):

- transition: Verify databaseState is connected. Delete the record with recordId from tableName.
- output: Return the deleteResult indicating success.
- exception: Raise RecordNotFound if the recordId does not exist, DatabaseConnectionError if unable to connect to the database, or TimeoutError if the operation exceeds defaultTimeout.

12 MIS of Audit Logging Module (M8)

12.1 Module

Audit Logging Module (M8)

12.2 Uses

- Hardware Hiding Module (M1) for access to durable, append-only storage and time sources.
- Data Model Module (M7) for persisting audit event metadata alongside core domain entities (users, requests, clubs).

12.3 Syntax

12.3.1 Exported Constants

- **AUDIT_RETENTION_YEARS**: N — Minimum duration that audit records must be retained to satisfy policy and compliance requirements.
- **MAX_EVENT_BATCH_SIZE**: N — Maximum number of audit events processed in a single batch when streaming to long-term storage.
- **PII_REDACTION_FIELDS**: sequence of String — List of field names whose values must be masked or omitted in stored audit payloads.
- **DEFAULT_TIME_WINDOW_DAYS**: N — Default time window for audit queries when no explicit range is supplied.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
logAction	actionType, actorId, resourceId, metadata	AuditEventId	StorageFailure, RedactionError
getEventsByRequest	requestId, timeWindow	seq of AuditEvent	RequestNotFound, StorageFailure
getEventsByUser	userId, timeWindow	seq of AuditEvent	UserNotFound, StorageFailure
searchEvents	filterCriteria	seq of AuditEvent	StorageFailure
purgeExpiredEvents		N	StorageFailure

12.4 Semantics

12.4.1 State Variables

- **eventStream**: append-only sequence of AuditEvent — Canonical log of all recorded audit actions in arrival order.
- **requestIndex**: map RequestId → sequence of AuditEventId — Secondary index for efficiently resolving events associated with a given reimbursement request.
- **userIndex**: map UserId → sequence of AuditEventId — Secondary index for resolving events initiated by or affecting a particular user.
- **retentionPolicy**: record of retentionYears: N, archiveTarget: StorageHandle — Encodes how and where aged events are archived or purged.

12.4.2 Environment Variables

- **clock**: Time source used to stamp events with an immutable timestamp at creation.
- **auditStore**: Durable storage backend (e.g., append-only bucket or managed log service) provisioned through Hardware Hiding Module (M1).
- **dataModel**: Interface to Data Model Module (M7) for resolving user, club, and request identifiers and enforcing referential integrity.

12.4.3 Assumptions

- All callers provide stable identifiers for *actorId* and *resourceId* that are resolvable through Data Model Module (M7).
- Clock skew is within acceptable bounds for compliance reporting and does not require cross-node reconciliation inside this module.
- The underlying auditStore honors append-only semantics for committed events and enforces configured retention guarantees.
- Personally identifiable information (PII) is either not logged or is redacted according to **PII_REDACTION_FIELDS** before persistence.
- High-volume producers batch log requests where appropriate to respect latency and throughput constraints.

12.4.4 Access Routine Semantics

logAction(actionType, actorId, resourceId, metadata)

- transition: Constructs an AuditEvent from the inputs and current time from **clock**. Applies redaction rules to *metadata* based on **PII_REDACTION_FIELDS**. Appends the event to **eventStream** and persists it to **auditStore**. Updates **requestIndex** and **userIndex** if *resourceId* or *actorId* refers to known entities.
- output: Returns the AuditEventId assigned to the newly persisted event.
- exception: Raises RedactionError if required masking cannot be applied safely, or StorageFailure if append or index updates fail.

getEventsByRequest(requestId, timeWindow)

- transition: Resolves *requestId* via **dataModel** to confirm the request exists. Determines the effective time window, defaulting to **DEFAULT_TIME_WINDOW_DAYS** if not specified. Uses **requestIndex** (when populated) to locate relevant event identifiers, then fetches the corresponding events from **auditStore**, filtering them to the specified timeWindow.
- output: Returns a sequence of AuditEvent records ordered by timestamp, each describing a state transition or action associated with the given request.
- exception: Raises RequestNotFound if *requestId* is unknown, or StorageFailure if retrieval from **auditStore** fails.

getEventsByUser(userId, timeWindow)

- transition: Resolves *userId* via **dataModel**. Uses **userIndex** to obtain relevant AuditEventIds and retrieves the full events from **auditStore**, applying the given timeWindow.
- output: Returns a sequence of AuditEvent records for which the specified user acted as the principal (actorId) or is the subject of the action.
- exception: Raises UserNotFound if *userId* does not exist, or StorageFailure if the underlying store returns an error.

searchEvents(filterCriteria)

- transition: Interprets filterCriteria (e.g., actionType, status change, clubId, date range) and performs a query against **auditStore** and indexes to identify matching events. May combine results from **requestIndex** and **userIndex** when filters reference both user and request.
- output: Returns a sequence of AuditEvent records matching the filterCriteria, suitable for compliance review or incident investigation.

- exception: Raises StorageFailure if the query cannot be executed against the backing store.

purgeExpiredEvents()

- transition: Scans **eventStream** (or an archival manifest in **auditStore**) for events older than **AUDIT_RETENTION_YEARS**. Removes or archives them according to **retentionPolicy**, updating any auxiliary indexes (**requestIndex**, **userIndex**) to remove references to deleted events.
- output: Returns the number N of events that were purged or archived in this invocation.
- exception: Raises StorageFailure if the deletion or archival operations against **auditStore** fail.

12.4.5 Local Functions

None.

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.

13 Appendix

This appendix is intentionally left blank for this deliverable. Future updates may add supplementary figures or calculations.

1. What went well while writing this deliverable?

   (a) Justin: We had good communication in terms of defining modules and expanding on each others work. This allows all the work to seamlessly connect to create a proper design document

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

   (a) Justin: One pain point was handling the workload and understanding how to properly write about the modules. Clarification with the TA and discussion through calls helped us clear up any misunderstandings

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?

   (a) Justin: Most of our design decisions came from prexisting client infrastructure. We wanted to make sure our design fit well with what the client already had in place. Any new features we chose based on popular design decisions seen in industry (through our personal experiences)

4. 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?

   (a) Justin: None of our other documents needed to be changed while creating the design doc. Our design doc was able to be created based on the information we had in our other documents

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)

   (a) Justin: Our current solution is limited by the external services we are using. Given unlimited resources, we could create our own storage and OCR services that are more tailored to our specific use case. This would allow for better performance and potentially lower costs in the long run.

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)

(a) Justin: Most of our other design decisions revolved around which external services to use. My main focus was about which database (PostgreSQL vs MongoDB). We chose MongoDB because it was the prexisting database the client was using and we believed it would be better to work off their infrastructure integrate our own, even though PostgreSQL may have been a better fit for our data model.