Module Interface Specification for Software Engineering

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

November 13, 2025

1 Revision History

Date	Version	Notes
Monday, November 10, 2025	1.0	Initial Document

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

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

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

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from 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 (-∞, ∞)

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	• 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 Receipt Processing Module (M4)

7.1 Module

ReceiptProcessing

7.2 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.

7.3 Syntax

7.3.1 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.

7.3.2 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

7.4 Semantics

7.4.1 State Variables

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

7.4.2 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.

7.4.3 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.

7.4.4 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.

7.4.5 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.

8 MIS of Notification Module (M5)

8.1 Module

NotificationModule

8.2 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.

8.3 Syntax

8.3.1 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.

8.3.2 Exported Access Programs

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

8.4 Semantics

8.4.1 State Variables

- reimbursementProcessIndex: Mapping the current request process/step to the corresponding notificationType to determine when notifications should be sent.
- notificationIndex: Mapping of NotificationID to NotificationRecord capturing recipient, notification type, notification message, timestamp, and delivery status.

8.4.2 Environment Variables

- None

8.4.3 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.

8.4.4 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.

9 MIS of Data Model Module (M7)

9.1 Module

Data Model Module

9.2 Uses

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

9.3 Syntax

9.3.1 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

9.3.2 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

9.4 Semantics

9.4.1 State Variables

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

9.4.2 Environment Variables

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

9.4.3 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

9.4.4 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.

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.

10 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

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

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it 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)