Module Interface Specification for Software Engineering

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

November 12, 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	
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 Receipt Processing Module (M4)

6.1 Module

ReceiptProcessing

6.2 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.

6.3 Syntax

6.3.1 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.

6.3.2 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

6.4 Semantics

6.4.1 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.

6.4.2 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.

6.4.3 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.

6.4.4 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.

6.4.5 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.

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.

7 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

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

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

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

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

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)