Module Guide 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 longtable pagination and uses diagram.
Monday, November 11, 2025	1.1	Implemented traceability pagination and new uses diagram proof of concept.

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
API	Application Programming Interface
BUC	Business Use Case
CI/CD	Continuous Integration/Continuous Deployment
DAG	Directed Acyclic Graph
FRQ	Functional Requirement
M	Module
MG	Module Guide
MES	McMaster Engineering Society
OCR	Optical Character Recognition
OS	Operating System
RBAC	Role-Based Access Control
R	Requirement
SRS	Software Requirements Specification
SSO	Single Sign-On
UC	Unlikely Change
UI	User Interface
UX	User Experience
WCAG	Web Content Accessibility Guidelines

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific OCR service used for receipt processing (e.g., Amazon Textract vs. Google Vision API).

AC2: The cloud storage provider for receipt files (e.g., AWS S3 vs. Google Cloud Storage).

AC3: The notification service provider (e.g., SendGrid vs. Nodemailer vs. custom solution).

AC4: The authentication provider and protocol (e.g., NextAuth.js configuration, SSO integration).

AC5: The specific UI component library and styling framework (e.g., NextUI vs. custom components).

AC6: The database technology (e.g., MongoDB vs. PostgreSQL migration).

AC7: The reimbursement approval workflow and business rules.

AC8: External API integrations (e.g., payment processing, email services).

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The system will be a web-based application accessible through browsers.

UC2: The core technology stack will remain JavaScript/TypeScript based (React/Next.js/Node.js).

UC3: The system will be integrated with the MES monorepo and development practices.

UC4: The system will comply with McMaster University policies and serve MES clubs.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

Level 1	**Level 2**
Hardware-Hiding Module	• M1: Hardware Hiding Module
Behaviour-Hiding Module	• M2: User Interface Module
	• M3: Request Handler Module
	• M4: Receipt Processing Module
	• M5: Notification Module
Software Decision Module	• M6: Authentication Module
	• M7: Data Model Module
	• M8: Audit Logging Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

The primary design decision is to structure the application as a client-server web application using the Next.js full-stack framework. This allows for a clear separation between the front-end UI (Behaviour-Hiding) and the back-end business logic and data management (Software Decision), while the Hardware-Hiding module is provided by the underlying Node.js runtime and cloud platform (Vercel/AWS). This architecture directly supports key non-functional requirements like security (PRI, IMM), performance (SaL), and maintainability (MNT).

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the

module is provided by the operating system or by standard programming language libraries. *MES Finance Platform* means the module will be implemented by this project.

Only the leaf modules in the hierarchy have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware (e.g., server infrastructure, cloud platform APIs).

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware (e.g., file storage, network) and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS (Node.js runtime, Vercel/AWS cloud platform)

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: -

7.2.1 User Interface Module (M2)

Secrets: The specific UI components, layouts, and client-side navigation logic.

Services: Provides the visual interface for users to interact with the system. Renders forms for submitting reimbursement requests, dashboards for viewing request status, and admin panels for reviewing requests. Handles client-side validation and user input. Implements responsive design and accessibility requirements (WCAG).

Implemented By: MES Finance Platform (Next.js/React, NextUI, TailwindCSS)

Type of Module: Abstract Object

7.2.2 Request Handler Module (M3)

Secrets: The workflow and business logic for processing reimbursement requests (submission, review, approval, rejection).

Services: Provides API endpoints for creating, reading, updating, and deleting reimbursement requests. Enforces business rules (e.g., only club executives can submit, only admins can approve). Manages the state transitions of a request (e.g., from 'submitted' to 'approved'). Handles budget allocation checks and financial validations.

Implemented By: MES Finance Platform (Next.js API Routes, Node.js)

Type of Module: Abstract Object

7.2.3 Receipt Processing Module (M4)

Secrets: The algorithms and external services used for processing receipt images (e.g., OCR service integration, file storage management).

Services: Handles the upload, storage (e.g., to AWS S3), and processing of receipt images. Extracts relevant data (amount, date, vendor) using OCR technology. Validates file formats and sizes. Provides secure access to stored receipts.

Implemented By: MES Finance Platform (Node.js, AWS Textract/S3)

Type of Module: Abstract Object

7.2.4 Notification Module (M5)

Secrets: The specific services and templates used for external communication (e.g., email, in-app alerts).

Services: Sends notifications to users based on system events (e.g., request submitted, request reviewed, payment issued). Abstracts the underlying communication channel. Manages notification templates and delivery status.

Implemented By: MES Finance Platform (Node.js, SendGrid or similar)

Type of Module: Abstract Object

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: -

7.3.1 Authentication Module (M6)

Secrets: The protocol and algorithms for verifying user identity and managing sessions.

Services: Provides services for user authentication (login/logout) and authorization (RBAC). Determines if a user has the required role (club member, executive, MES admin) to perform an action. Manages user sessions and secure token handling.

Implemented By: MES Finance Platform (NextAuth.js)

Type of Module: Abstract Object

7.3.2 Data Model Module (M7)

Secrets: The structure and schema of the persistent data and the database management system itself.

Services: Provides a consistent interface for all data persistence and retrieval operations. Manages connections to the database (MongoDB). Defines and enforces the data schema for Users, Clubs, Reimbursement Requests, Budgets, etc. Ensures data integrity and relationships.

Implemented By: MES Finance Platform (MongoDB, Mongoose ODM)

Type of Module: Abstract Data Type

7.3.3 Audit Logging Module (M8)

Secrets: The format and storage mechanism for the audit trail.

Services: Records immutable logs of significant system actions (e.g., request submission, approval, status change) along with user ID and timestamp, as required by the SRS. Provides querying capabilities for audit review and compliance reporting.

Implemented By: MES Finance Platform (Node.js, MongoDB)

Type of Module: Abstract Object

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Table 2: Trace Between Requirements and Modules

Req.	**Modules**
FRQ-1: Submit expense claims	M2, M3, M6, M7
FRQ-2: Review/approve/reject requests	M2, M3, M6, M7
FRQ-3: Track claim status	M2, M3, M7
FRQ-4: Store digital receipts	M3, M4, M7
FRQ-5: Maintain audit trail	M3, M7, M8
FRQ-6: Access control for submissions	M2, M3, M6, M7
APP-1: MES branding	M2
APP-2: Clean, modern layout	M2
APP-3: Responsive design	M2
APP-4: Accessible contrast ratios	M2
EUL-1: First-time user tutorial	M2
ABL-1: WCAG compliance	M2
ABL-2: Light/dark mode	M2
ABL-3: Semantic HTML	M2
ABL-4: Image alt text	M2
ABL-5: Keyboard navigation	M2
SaL-1: API response times	M1, M3, M7
SaL-2: Page load times	M1, M2
SaL-3: File upload times	M1, M4
SAF-1: No unauthorized reimbursements	M3, M6
SAF-2: Monetary data validation	M3, M7
PRE-1: Financial precision	M3, M7
FLT-1: Graceful error handling	M1, M2, M3
FLT-2: Incremental data saving	M2, M7
FLT-3: API retry logic	M3
CAP-1: Concurrent users	M1, M3, M7
CAP-2: Database capacity	M1, M7

Continued on next page

Req.	Modules
EXT-1: Horizontal scaling	M1, M3, M7
EXT-2: Feature extensibility	M2, M3, M7
LNG-1: 5-year operational lifespan	M1, M2, M3, M7
LNG-2: Technology upgrades	All Modules
ACS-1: Authentication required	M6
ACS-2: Role-based access control	M6, M3
INT-1: Form validation	M2, M3
INT-2: Security protections	M3, M6, M7
PRI-1: Secure data storage	M7
PRI-2: Data encryption	M1, M7
AUD-1: Action logging	M3, M8
AUD-2: Log retention	M7, M8
IMM-1: Vulnerability protection	M3, M6
IMM-2: Rate limiting	M3
MNT-1: Code formatting	All Modules
MNT-2: Dependency updates	All Modules
SUP-1: Developer documentation	All Modules
SUP-2: Monitoring access	M1, M8
ADP-1: Policy adaptability	M3
ADP-2: UI extensibility	M2

Table 3: Trace Between Anticipated Changes and Modules

AC	Modules
AC1	M4
AC2	M4
AC3	M5
AC4	M6
AC5	M2
AC6	M7
AC7	M3
AC8	M3, M5

9 Use Hierarchy Between Modules

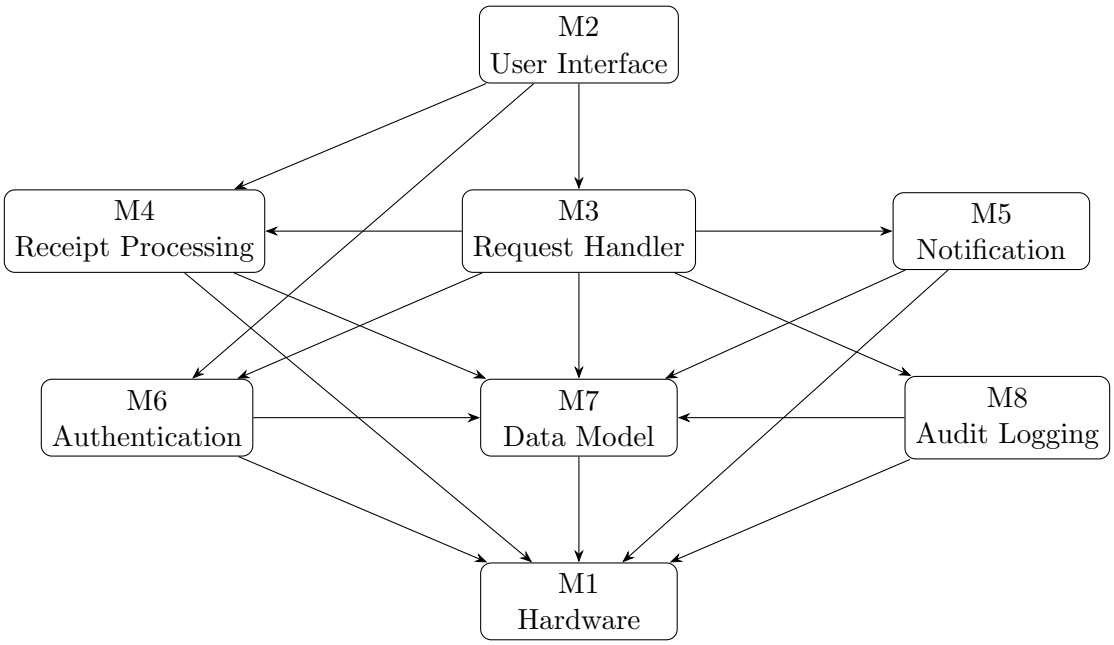

Figure 1: Uses relationships between modules.

M2 uses $\{M3, M4, M6\}$: Screens and flows render request data provided by M3 and push user actions back through its APIs, which keeps `FRQ-1`–`FRQ-3` behaviour identical across clients. Upload widgets stream binaries to M4 so that the receipt constraints in `FRQ-4` and `SaL-3` are enforced once instead of being re-implemented in the browser. Role-aware rendering and secure navigation rely on session hooks from M6, satisfying `FRQ-6`, `APP-3`, and the accessibility targets in `ABL`.

M3 uses $\{M4, M5, M6, M7, M8\}$: Request lifecycle logic calls M6 first to confirm the actor's role (`FRQ-6`, `SAF-1`), persists changes through M7 to preserve transactional integrity and the latency/capacity bounds in `SaL-1` and `CAP-1`, and records each transition via M8 (`FRQ-5`, `AUD-1`, `AUD-2`). When payloads contain receipts, M3 streams to M4 to reuse validation and OCR logic mandated by `FRQ-4`. Workflow outcomes emit events to M5 so stakeholders receive acknowledgements and escalations without embedding channel logic in the handler.

M4 uses $\{M1, M7\}$: The receipt pipeline calls storage, OCR, and queueing facilities supplied by M1 (e.g., AWS S3/Textract) to meet `SaL-3` while remaining replaceable. Metadata for every file, checksum, and extraction result is persisted through M7 so that receipts remain linked to their originating requests (`FRQ-4`, `PRI-1`).

M5 uses $\{M1, M7\}$: Notification templates are rendered with request, user, and club data fetched via M7, ensuring messages reflect the authoritative state described in `FRQ-2`

and FRQ-3. Delivery relies on the email/SMS gateways abstracted by M1, allowing the provider in AC3 to change independently while still meeting IMM-2 and FLT-3 retry expectations.

M6 uses $\{M1, M7\}$: Authentication depends on cryptographic primitives, secrets storage, and SSO adapters from M1 while persisting user profiles, role grants, and refresh tokens through M7. This encapsulation satisfies FRQ-6, PRI-2, and INT-2; higher layers only interact with opaque tokens or guard functions.

M7 uses $M1$: The data model sits directly atop the managed database and backup facilities exposed by M1. By isolating queries, schema migrations, and replication hooks here, the system respects AC6 while insulating FRQ-1–FRQ-5, SaL-1, CAP-2, and LNG-2 from upstream change.

M8 uses $\{M1, M7\}$: Audit events are appended to the durable stores defined by M7 and flushed to immutable storage supplied by M1 (e.g., append-only buckets or managed log services). This architecture keeps FRQ-5, AUD-1, AUD-2, and SUP-2 compliant without burdening request handlers with vendor-specific retention logic.

10 User Interfaces

The user interface will be designed as a responsive web application using Next.js/React with NextUI components and TailwindCSS for styling. Key interface components include:

- **Login Portal**: Role-based access with McMaster SSO integration
- **Club Member Dashboard**: View budget, submit requests, track status
- **Reimbursement Submission Form**: Upload receipts, enter expense details
- **Admin Review Panel**: Approve/reject requests, manage clubs/users
- **Audit Trail Viewer**: Access logs and financial reports

The interface will follow MES branding guidelines and WCAG 2.1 AA accessibility standards.

11 Design of Communication Protocols

The system will use the following communication protocols:

- **RESTful APIs**: For frontend-backend communication using JSON format
- **HTTPS/TLS 1.2+**: For all network communications to ensure security
- **WebSockets**: Optional for real-time notifications (future enhancement)
- **SMTP/Email API**: For notification delivery to users

12 Timeline

The development timeline is structured as follows:

- **November 2025**: Complete POC
- **March 2026**: Basic components completed
- **April 2026**: Testing and integation completed

Team responsibilities:

- **Zhenia Sigayev**: Authentication Module (M6), Audit Logging (M8)
- **Justin Ho**: User Interface Module (M2), Styling and Accessibility
- **Thomas Wang**: Request Handler Module (M3), Business Logic
- **Michael Shi**: Data Model Module (M7), Database Design
- **Johnny Qu**: Receipt Processing (M4), Notification Module (M5)
- **All**: Integration testing, documentation, and deployment

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.