Module Guide for McMaster Engineering Society Custom Financial Expense Reporting Platform

Team #12, Reimbursement Rangers
Adam Podolak
Evan Sturmey
Christian Petricca
Austin Bennett
Jacob Kish

January 15, 2025

1 Revision History

Date	**Version**	**Notes**
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements
McMaster Engineering Society Custom Financial Expense Reporting Platform	Explanation of program
UC	Unlikely Change

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 (?). We advocate a decomposition based on the principle of information hiding (?). 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 ?, 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 (?). 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: Changes to the database schema to meet evolving data storage needs.

AC2: Expansion of the notification system to include push notifications and SMS.

AC3: Adjustments to the leger views for new regulatory compliance requirements impacting data handling.

AC4: Enhancements to the user interface for improved accessibility and user experience.

AC5: Integration with additional third-party APIs for payment processing.

AC6: Security enhacements to the login process. (i.e Integrated 2FA McMaster Login)

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: Major changes to the core system architecture, such as switching from cloud to on-premises infrastructure.

UC2: Transitioning from the existing programming language stack to a completely different stack.

UC3: Removal of essential core modules such as authentication or role-based access control.

UC4: Fundamental changes in the communication protocols, such as moving from REST to an entirely different protocol.

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.

M1: Hardware-Hiding Module

M2: Account Management Module

M3: Requests Module

M4: Notification Module

M5: User Dashboard Module

M6: Authentication Module

M7: Email Module

M8: Account Management Controller Module

M9: Requests Controller Module

M10: Clubs Database

M11: Users Database

M12: Requests Database

M13: Graphical User Interface

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.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. 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

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Account Management Module
	Requests Module
	Notification Module
	User Dashboard Module
	Authentication Module
	Email Module
	Account Management Controller Module
	Requests Controller Module
Software Decision Module	Clubs Database
	Users Database
	Requests Database
	Graphical User Interface

Table 1: Module Hierarchy

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. *McMaster Engineering Society Custom Financial Expense Reporting Platform* means the module will be implemented by the McMaster Engineering Society Custom Financial Expense Reporting Platform software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

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

Implemented By: OS

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 Account Management Module (M2)

Secrets: User account details, including personal information, access levels, and account statuses.

Services: Provides functionality for creating, updating, and deleting user accounts. It manages user roles and permissions, ensuring that appropriate access is granted to each user based on their role within the system. This module ensures secure storage of account information and integrates with the authentication module for login and identity verification purposes.

Implemented By: Account Management Controller

Type of Module: Abstract Data Type

7.2.2 Requests Module (M3)

Secrets: Internal data structure of requests, including status, history, and associated user information.

Services: Manages the creation, tracking, and approval of reimbursement requests. This module handles the submission of new requests by users, updates the status of existing requests, and logs all request-related activities for auditing purposes. It communicates with the notification module to inform users about the progress of their requests.

Implemented By: Requests Controller

Type of Module: Abstract Data Type

7.2.3 Notification Module (M4)

Secrets: Notification content, delivery methods, and scheduling information.

Services: Sends notifications to users regarding updates to their reimbursement requests, account changes, and other system events. It supports email. The module ensures that notifications are timely.

Implemented By: Emailer API

Type of Module: Abstract Data Type

7.2.4 User Dashboard Module (M5)

Secrets: The organization and structure of displayed data, including summaries and real-time updates.

Services: Provides a user-friendly interface displaying the current status of reimbursement requests, notifications, and account information. It updates dynamically based on user interactions and backend data changes. The module interacts with the requests, and account management modules to present comprehensive information.

Implemented By: MRM

Type of Module: Abstract Data Type

7.2.5 Authentication Module (M6)

Secrets: Validation processes, and session management details.

Services: Handles user login, logout, and session management. It ensures secure authentication using encrypted credentials. The module integrates with the account management module to verify user identities and enforce access control.

Implemented By: MRM

Type of Module: Abstract Data Type

7.2.6 Emailer API (M7)

Secrets: Email templates, SMTP server credentials.

Services: Provides an interface for sending emails to users. It handles notification delivery, and verification communications, and ensures successful delivery through integration with external SMTP services. The module is used by the notification module to send email-based alerts to users.

Implemented By: TBD

Type of Module: Abstract Data Type

7.2.7 Account Management Controller (M8)

Secrets: The flow of control for account operations and error handling mechanisms.

Services: Acts as a controller for the account management module. It receives requests from the user dashboard and other modules, processes them, and invokes appropriate functions within the account management module. It ensures consistency and error handling in account-related operations.

Implemented By: MRM

Type of Module: Abstract Data Type

7.2.8 Requests Controller (M9)

Secrets: The flow of control for request operations and error handling mechanisms.

Services: Acts as a controller for the requests module. It handles user inputs for creating and managing requests, processes these inputs, and invokes the necessary functions within the requests module. The controller ensures that all requests are valid and consistent with the system's requirements.

Implemented By: MRM

Type of Module: Abstract Data Type

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 Clubs Database (M10)

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:

Type of Module:

7.3.2 Users Database (M11)

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:

Type of Module:

7.3.3 Requests Database (M12)

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:

Type of Module:

7.3.4 Graphical User Interface (M13)

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:

Type of Module:

8 Traceability Matrix

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

Req.	**Modules**
R1	M1, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC??	M1
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

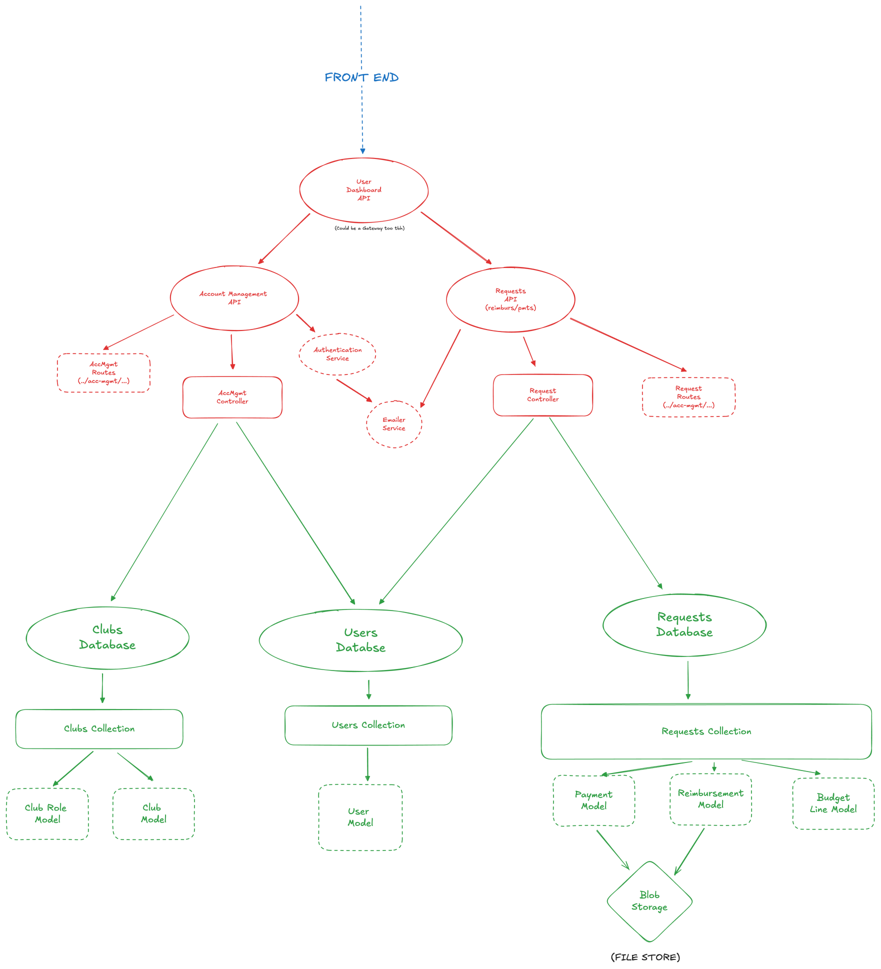

Figure 1: Use hierarchy among modules

10 User Interfaces

11 Design of Communication Protocols

12 Timeline

Timeline will be found on GitHub Projects here!