Module Guide for Software Engineering

Team #12, Streamliners
Mahad Ahmed
Abyan Jaigirdar
Prerna Prabhu
Farhan Rahman
Ali Zia

November 13, 2025

1 Revision History

Date	Version	Notes
2025-11-13	-1.0	Initial draft of MG doc.

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 Specification
Software Engineering	Explanation of program name
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 (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: Future updates to the **Payment Processing Module** (M1) may require migration to different payment providers (e.g., Stripe, PayPal, Square) or McMaster's internal e-commerce API.

AC2: New event types (e.g., workshops, volunteer shifts) may require additional registration flows beyond bus, table, and RSVP sign-ups. **Affected Module:** Sign-Up Module (M3).

AC3: Notification delivery methods or content templates may expand to include SMS or in-app alerts in addition to email and push notifications. **Affected Module:** Notification Handling Module (M7).

AC4: Notification delivery methods or content templates may expand to include SMS or in-app alerts in addition to email and push notifications. **Affected Module:** Notification Handling Module (M7).

AC5: Event policies such as capacity limits, sign-up deadlines, or priority access rules may change based on MES regulations. **Affected Module:** Registration Rules (M6).

AC6: Admin dashboard metrics and analytics may expand with new reporting filters or visualizations. **Affected Module:** Admin Dashboard and Analytics (M5).

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's technology stack (Next.js/React frontend, NestJS backend, PostgreSQL database) is a stable decision and unlikely to change during or after development. Altering this foundation would require complete re-architecture.

UC2: Hosting on DigitalOcean and database management using PostgreSQL containers are fixed infrastructure decisions.

UC3: Compliance with PIPEDA and McMaster University privacy policies is a permanent requirement that will not change. Any change would necessitate legal review and cross-module rewriting.

UC4: Security protocols such as TLS 1.3, AES-256 encryption, audit logging, and fail-safe defaults are core safety standards defined in the Hazard Analysis and are not expected to change.

UC5: The role hierarchy (Attendee, Organizer, Executive) and three-tier architecture (frontend, backend, database) are foundational and not expected to evolve.

UC6: The system scope will remain focused on MES event management. Expansion to unrelated domains (e.g., general club management or social networking) is not anticipated.

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.

User Interface Modules

M1: Payment Processing Module Handles the logic and communication between the frontend and backend services for processing event payments and verifying transactions.

M2: RBAC/FBAC Access Control Module Manages user permissions and determines which features are accessible to each user role within the mobile and web applications. This module relies on the User Authorization Module for user authentication and identity verification.

M3: Sign-Up Module Provides functionality for participants to register for different event activities. It contains shared logic for registration flows, along with specific submodules for each type of sign-up.

- **M3.1: Bus Sign-Up Module** Handles registration for bus transportation, including capacity validation and seat allocation.
- **M3.2: Table Sign-Up Module** Manages registration and table assignments for group events.
- **M3.3: RSVP Sign-Up Module** Handles RSVP submissions and confirmation tracking for events.

Backend Modules

M4: Payment Configuration Module Stores and manages payment parameters such as currency, and API integration details for supported payment methods.

M5: Access Control Module Implements backend logic for role-based and feature-based permissions, ensuring consistent authorization across the system.

M6: Registration Rules Module Defines and enforces sign-up policies, including event deadlines, capacity limits, and participant prioritization criteria.

M7: Notification Handling Module Manages the delivery of notifications to users across different parts of the system. It coordinates sending confirmations, updates, and reminders related to payments and registrations. This module ensures that users receive timely and relevant information through email or in-app alerts.

M8: User Authorization Module Handles user authentication, login, and session management. It verifies user credentials, and provides identity data to other modules such as RBAC/FBAC and Payment Processing. This module serves as the foundation for all user-based operations within the system.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Payment Processing Module RBAC/FBAC Access Control Module Sign-Up Module Bus Sign-Up Module Table Sign-Up Module RSVP Sign-Up Module User Authorization Module
Software Decision Module	Payment Configuration Module Access Control Module Registration Rules Module Notification Handling Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the Large Event Management System is intended to satisfy the functional requirements (F101–F116) and the associated non-functional requirements identified in the Software Requirements Specification (SRS). This section explains how these requirements guided the modular decomposition of the system into the components presented in Section 5. The connection between requirements and modules is listed in Table 2.

Requirements related to **event registration and ticketing** (F101–F104) are satisfied through the *Sign-Up Module M3* and its submodules (*M3.1–M3.3*). These modules handle all user registration flows including ticket purchasing, bus and table sign-ups, RSVPs, and refund management. They interact with the *Registration Rules Module M6* to enforce deadlines, seat capacities, and prioritization policies, ensuring that registration logic remains consistent and adaptable to different event configurations.

Payment-related requirements (F105–F107) are fulfilled by the *Payment Processing Module M1* and the *Payment Configuration Module (M4)*. Together they manage secure payment gateway integration, transaction verification, and storage of configuration data such as currency types, merchant keys, and payment methods. This modular separation supports both extensibility—allowing future integration with new payment providers—and maintainability by isolating external API dependencies.

Sign-up and allocation requirements (F108–F110), including bus registration, table assignments, and RSVP limits, are handled by specific submodules within the *Sign-Up Module M3*. Each submodule provides a tailored workflow for its respective function, while common logic is centralized through the *Sign-Up Adapter M3.1*. This satisfies the SRS need for flexible and reusable registration components that can be applied across multiple event types.

Access-control requirements (F111–F113) motivated the inclusion of both the *RBAC/FBAC Access Control Module M2* and the backend *Access Control Module M5*. These modules collectively enforce role-based and feature-based access restrictions, ensuring that users—such as administrators, volunteers, or attendees—only interact with features appropriate to their permissions. This design choice provides fine-grained authorization while remaining scalable for future event roles.

Security, privacy, and integrity requirements (F114–F116) are addressed through the *User Authorization Module M8* in conjunction with other backend modules (*M4–M7*). The authorization module manages user authentication and secure session handling, while supporting encryption and data-protection mechanisms across network communications. These features guarantee that sensitive user and payment data comply with privacy regulations and institutional security standards.

Finally, cross-cutting requirements for **notifications, confirmations, and user commu-**

nication are met by the *Notification Handling Module M7*. This module delivers confirmations, reminders, and updates via email and in-app alerts, ensuring that both attendees and administrators receive timely feedback on their actions within the system.

By associating each major requirement group in the SRS with a dedicated set of modules, the overall architecture remains flexible, maintainable, and extensible. This structured approach ensures that any future modifications—such as supporting new payment methods, adjusting registration logic, or introducing new user roles—can be achieved by updating a limited subset of modules. The detailed requirement–module correspondence is provided in Table 2 in Section 8.

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. *Software Engineering* means the module will be implemented by the Software Engineering 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

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 Payment Processing Module (M1)

Secrets: Payment gateway integration, logic for the transaction workflows.

Services: Allows for client to fulfill payment requests, executes charges/refunds, and reports all transactions to logs.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.2 RBAC/FBAC Access Control Module (M2)

Secrets: The mapping of roles to features that allow capabilities in the UI.

Services: Determines what views/actions are allowed for a signed in user.

Implemented By: Software Engineering

Type of Module: Library Component

7.2.3 Sign-Up Module (M3)

Secrets: Common sign up state machine (registered, confirmed, waitlisted).

Services: Shared logic for creating and managing signups, delegates concrete services to submodules.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.4 Bus Sign-up Module (M3.1)

Secrets: Capacity model, seat allocation logic and, waitlisting logic.

Services: Allows users to reserve bus seat while enforcing capacity.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.5 Table Sign-up Module (M3.2)

Secrets: Table size, number of tables, and grouping constraints.

Services: Allows event organizers to create/manage table groups.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.6 RSVP Sign-up Module (M3.3)

Secrets: Event capacity, RSVP deadlines.

Services: Allows users to RSVP for events. enforces constraints/limits and reports user status.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.7 User Authorization Module (M8)

Secrets: Identity verification, session tokens.

Services: Authenticates the user, issues and validates session tokens, provides identity context to M2.

Implemented By: Software Engineering

Type of Module: Library

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 Payment Configuration Module (M4)

Secrets: Supported providers, external/internal services, pricing schemas.

Services: Stores and provides payment parameters to M1.

Implemented By: Software Engineering

Type of Module: Library Component

7.3.2 Access Control Module (M5)

Secrets: Logic to evaluate permissions and stores all policies.

Services: Evaluates actions being performed by the user and grants them the correct views/resources.

Implemented By: Software Engineering

Type of Module: Library

7.3.3 Registration Rules Module (M6)

Secrets: Global event constraints (capacity, deadlines) holds all the default event details/policies.

Services: Validates sign up operations across all sign up modules. Provides M3 with core information about events.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.3.4 Notification Handling Module (M7)

Secrets: Integrations (emails/push), notification templating, retry policies.

Services: Sends confirmations, reminders and status updates triggered by M1/M3.

Implemented By: Software Engineering

Type of Module: Library Component.

8 Traceability Matrix

This section presents two traceability matrices: the first shows how each functional requirement from the SRS (F101–F116) is satisfied by one or more modules; the second links the anticipated changes identified in Section 4 to the modules that would be affected.

Req. ID	Modules Implementing Requirement
F101–F104	M3 (Sign-Up Module), M3.1-M3.3 (Sign-Up Submodules), M6 (Registration Rules Module)
F105–F107	M1 (Payment Processing Module), MM4 (Payment Configuration Module)
F108–F110	M3 (Sign-Up Module), M3.1-M3.3 (Sign-Up Submodules), M6 (Registration Rules Module)
F111–F113	M2 (RBAC/FBAC Access Control Module), M5 (Access Control Module)
F114–F116	M8 (User Authorization Module), M4-M7 (Backend Modules)

Table 2: Trace Between Requirements and Modules

Anticipated Change (AC)	Modules Affected
AC1	M1 (Payment Processing Module), M4 (Payment Configuration Module)
AC2	M3 (Sign-Up Module), M6 (Registration Rules Module)
AC4	M7 (Notification Handling Module)
AC5	M6 (Registration Rules Module), M3 (Sign-Up Submodules)
AC6	M5 (Access Control Module), M7 (Notification Handling Module)

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) 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.

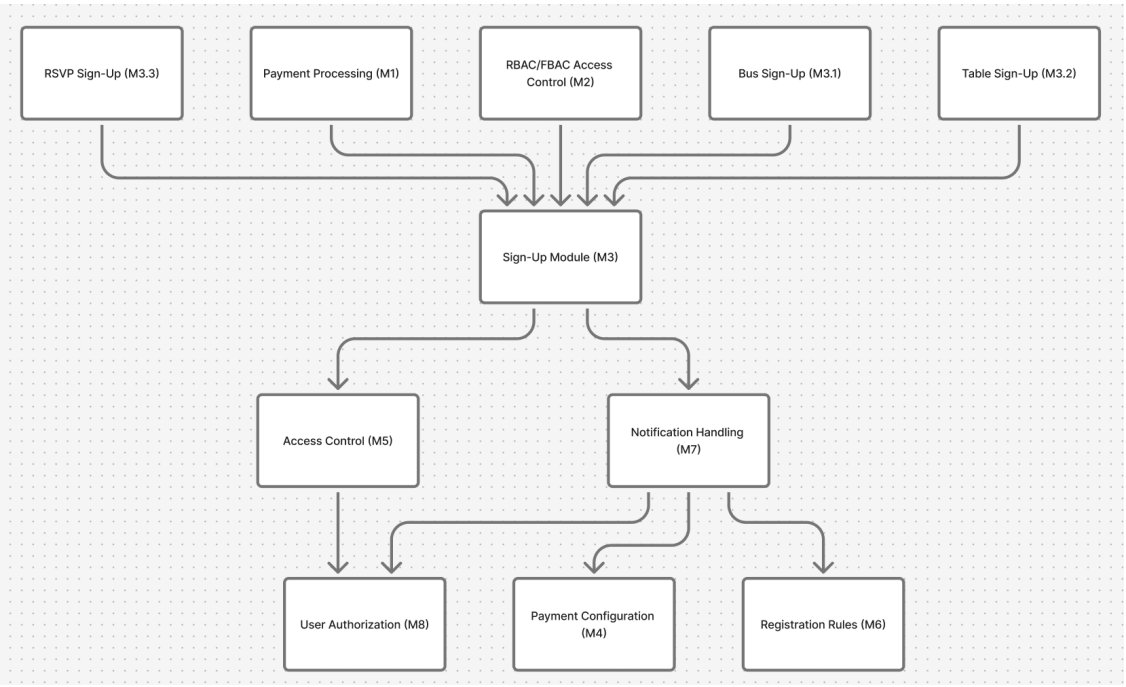

Figure 1: Use hierarchy among modules

10 User Interfaces

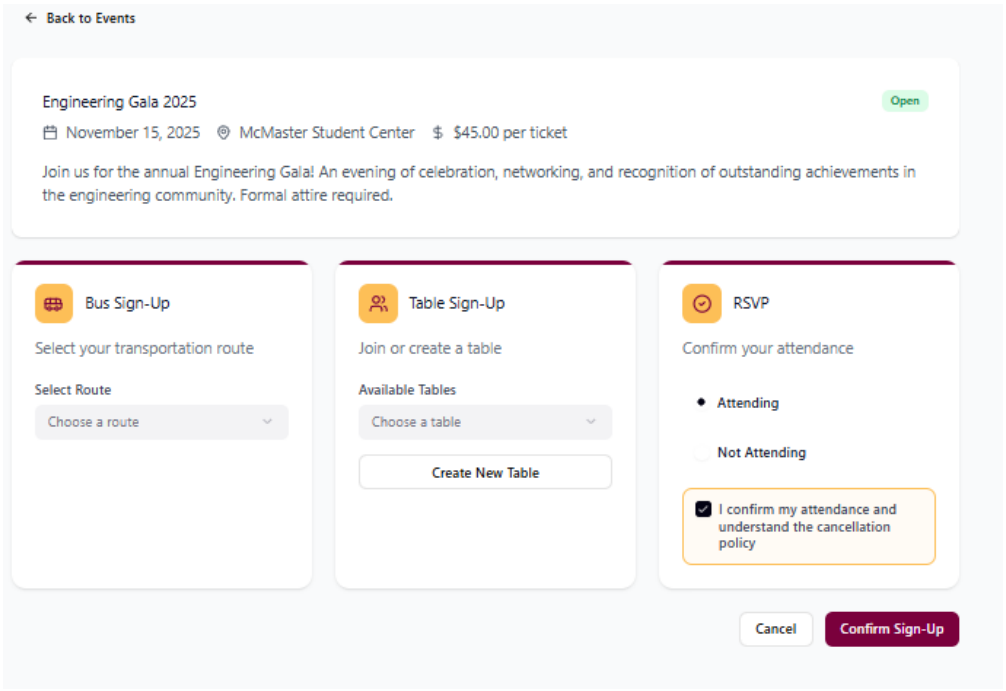

Figure 2: Signup Page

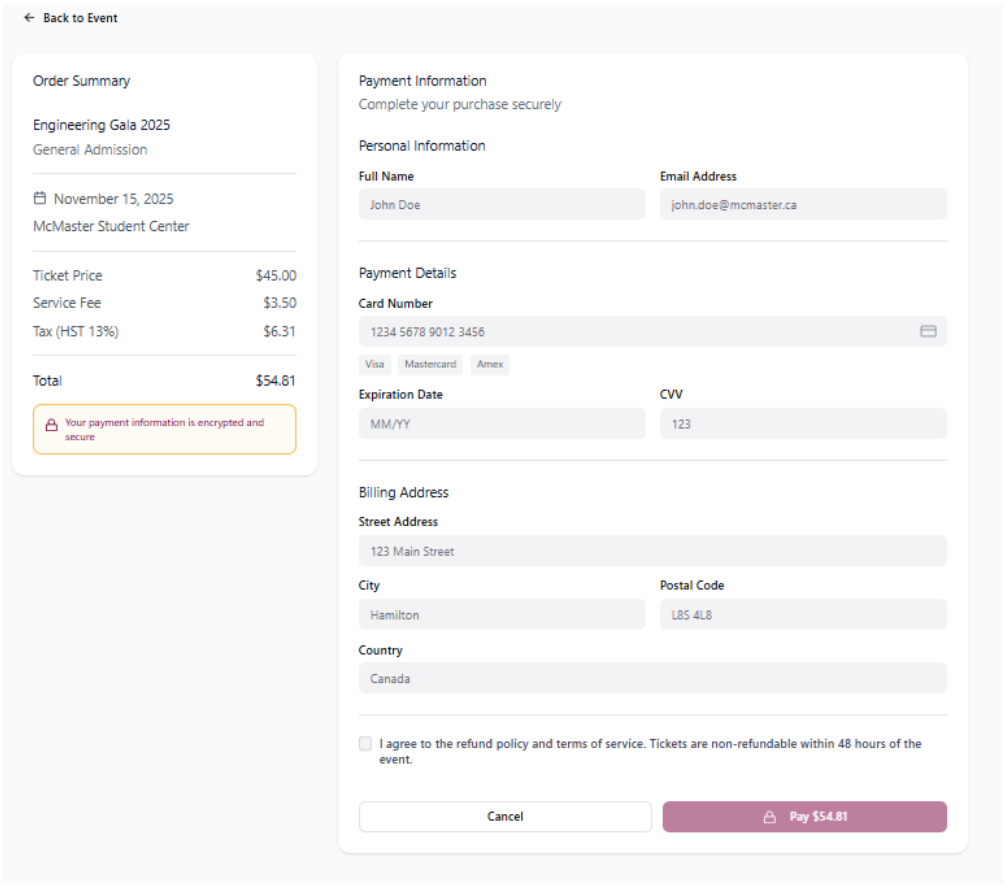

Figure 3: Payment Page

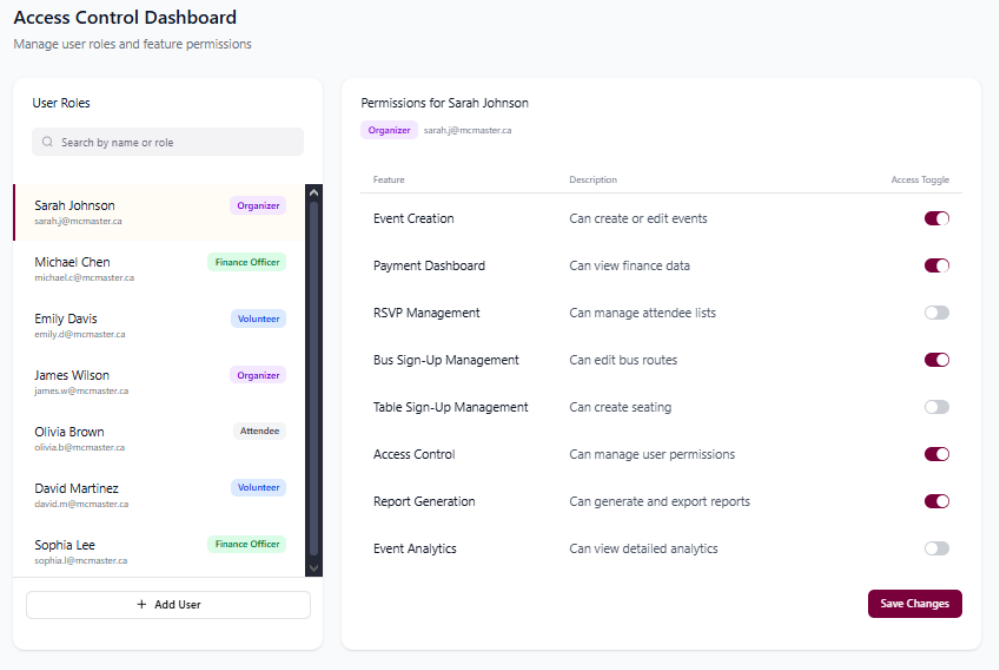

Figure 4: Role-based/Feautre-based Access Control

11 Design of Communication Protocols

This section outlines the external services that will be used by the MacSync platform ensuring appropriate integration between the application and third-party systems.

11.1 External Services Overview

The MacSync platform will interface with 2 key external APIs to support event management functions:

- **Stripe API** for secure online payments.
- **McMaster MacID SSO** to authenticate and authorise users

11.2 Payment Service: Stripe

Stripe will be used as the primary gateway for handling all necessary transcations for registering for events. All communication with Stripe occurs over secure HTTPS via Stripes RESTful API. All transaction records will be recorded, logged and stored within our database to maintain past records. This enables a secure way to integrate payments within our application without having to worry about the implications of building a custom payment stack.

11.3 Authentication and Authorization: McMaster MacID SSO

The MacSync platform will integrate with McMaster's identity platform to authenticate users and support M2. After logging in via MacID their profile will be linked to key information such as unique identifier (MacID), email address, given name, and affiliation (student, employee). By using the university SSO, we ensure that only valid McMaster students are registered on the platform and reduces authentication overhead.

12 Timeline

A breakdown of the team's task delegation and schedule can be found in the GitHub repository under the Issues and Projects tabs. The delegation assumes even workload distribution as outlined in the Development Plan.

All documentation deliverables are to be completed prior to the Final Documentation deadline in April 2026. All feature-related issues (functional and non-functional) are to be completed prior to the Final Demonstration and Capstone Expo.

The following outlines the implementation timeline:

- **Phase 1: Proof-of-Concept Demo (due November 2025)**
  - **M4: Payment Processing Module**
    - Payment workflow prototype and API integration testing (Abyan)
    - Sandbox transactions with Stripe (Farhan)
  - **M2: RBAC/FBAC Access Control Module**
    - Role and permission model research (Mahad)
    - Prototype admin/attendee login flows (Mahad)
  - **M3: Sign-Up Module**
    - RSVP and table sign-up prototype (Ali, Prerna)
    - Early UI mockups and capacity validation logic (Ali, Prerna)
- **Phase 2: Revision 0 Implementation (due February 2026)**
  - **M1: Payment Processing Module**
    - Finalize secure transaction pipeline and refund logic (Abyan)
    - Implement audit trail for financial operations (Farhan)
  - **M2: RBAC/FBAC Access Control Module**
    - Implement role templates and privilege hierarchy (Mahad)
    - Integrate authentication middleware (Mahad, Farhan)

  - **M3: Sign-Up Module**
    * Full bus/table/RSVP integration (Abyan, Prerna)
    * Database schema and validation rules (Prerna)
  - **M5: Admin Dashboard and Analytics Module**
    * Dashboard UI and analytics metrics (Ali, Abyan)
    * Implement sales/capacity charts (Ali)
  - **M6: Registration Rules Module**
    * Develop configuration logic for deadlines and waitlists (Prerna)
    * Link with sign-up validation rules (Mahad)
  - **M7: Notification Handling Module**
    * Email and push notification service setup (Farhan)
    * Implement confirmation and reminder templates (Abyan)

- **Phase 2.5: System Integration with MES Teams (January 2026)**
  - Conduct cross-team integration with the other MES Capstone groups to merge the Payments, Access Control, and Event Creation subsystems.
  - Establish shared database schema and API endpoints for unified platform operation.
  - Perform integration testing of inter-module communication (auth tokens, event sync, and ticket validation).
  - Align with MES technical leads to confirm production hosting configuration and shared environment variables.
  - Responsibilities:
    * Farhan - Cross-team liaison and CI/CD pipeline alignment
    * Mahad - Backend API merging and shared database migration scripts
    * Ali - Access token compatibility and role enforcement validation
    * Abyan, Prerna - Front-end integration testing and UI regression checks

- **Phase 3: Full System Testing and Stabilization (due March 2026)**
  - Integrate all modules and connect frontend/backend pipelines (all members)
  - Conduct automated unit and integration tests via GitHub Actions (Mahad lead)
  - Execute usability and security validation per V&V Plan (Prerna, Ali)

- **Phase 4: Final Demo and Capstone Expo (due April 2026)**
  - System polishing, documentation updates, and final feature refinement (all members)

– Final presentation video, poster, and Rev 0 submission (Streamliners Team)

All testing and CI/CD pipelines will be continuously maintained during each phase to ensure progress toward a stable and verified Rev 0 release.

References

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

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

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.