Module Interface Specification for McMaster Engineering Society Custom Financial Expense Reporting Platform

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

April 4, 2025

1 Revision History

Date	Version	Notes
Jan 17 2025	1.0	All Sections
Mar 24 2025	2.0	Updated Authentication, see commit: 4cef88d
Apr 04 2025	3.0	Updated Semantics and Clarity, see commit: 664a7ef
Apr 04 2025	4.0	SRS Traceability, Local Function documentation, improved clarity, see commit: fc3f256

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at

Contents

3 Introduction

The following document details the Module Interface Specifications for the MES Finance Platform. The document specifies how each module interfaces with other parts of the program. Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/ausbennett/mes-finance-platform.

4 Notation

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. 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 McMaster Engineering Society Custom Financial Expense Reporting Platform.

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 McMaster Engineering Society Custom Financial Expense Reporting Platform 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, McMaster Engineering Society Custom Financial Expense Reporting Platform 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	Account Management Module Requests Module Notification Module User Dashboard Module ~~Authentication Module~~ Email Module Account Management Controller Module Requests Controller Module
Software Decision	Clubs Database Users Database Requests Database Graphical User Interface

Table 1: Module Hierarchy

6 MIS of Account Management API

6.1 Module

Account Management API

6.2 Uses

Account Management Controller

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
auth	• token: String (JWT)	• tokenStatus: Enum["valid", "expired", "invalid"] • userRole: Enum["user", "admin", "guest"]	InvalidTokenException
loginUser	• email: String (format: RFC 5322)	• status: Enum["email sent", "error"]	EmailNotFoundException
registerUser	• name: String (3-50 chars) • email: String (format: RFC 5322) • role: Enum["user", "admin"] • password: String (min 8 chars)	• userId: String (UUIDv4) • status: Enum["success", "error"] • message: String	• DatabaseException • ValidationException
getAllUsers	• adminToken: String (JWT)	• users: Array[{ – id: String (UUIDv4) – name: String – email: String }]	AuthorizationException

getUser	• userId: String (UUIDv4)	• user: {
		– id: String
		– name: String
		– email: String
		– role: String
		}
			UserNotFoundException

editProfile	• userId: String (UUIDv4)
	• updates: {
	– name: String (optional)
	– profilePic: URL (optional)
	}
		• status: Enum["updated", "failed"]	DatabaseException

	• clubId: String (UUIDv4) • updates: { – clubName:	• status: Enum["updated"	

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

MongoDB connection (via Mongoose)

6.4.3 Assumptions

- Valid and authenticated tokens are provided for admin and user-specific actions.
- All inputs are sanitized before being processed.

6.4.4 Access Routine Semantics

auth(token: String):

- transition: Validates the provided token and grants access.
- output: returns a JSON object with detailed information about the result.
- exception: InvalidTokenException if token is malformed or expired.

loginUser(email: String):

- transition: Sends a confirmation link to the provided email.
- output: returns a JSON object with detailed information about the result.
- exception: EmailNotFoundException if email does not exist in the system.

registerUser(userDetails: JSON):

- transition: Adds a new user record to the database.
- output: returns a JSON object with detailed information about the result.
- exception: DatabaseException if there is an issue saving to MongoDB.

getAllUsers(adminToken: String):

- input: admin auth token
- output: returns a JSON object with detailed information about the result and array (users).
- exception: DatabaseException if there is an issue communicating to MongoDB.

getUser(userID: String):

- input: userID of user
- output: returns a JSON object with detailed information about the result.
- exception: DatabaseException if there is an issue communicating to MongoDB.

editUser(userID: String, updates: JSON):

- input: userID, and a JSON object containing updates to user information.
- output: returns a JSON object with detailed information about the result.
- exception: DatabaseException if there is an issue communicating to MongoDB.

editClub(clubID: String, updates: JSON):

- input: clubID, and a JSON object containing updates to club information.
- output: returns a JSON object with detailed information about the result.
- exception: DatabaseException if there is an issue communicating to MongoDB.

6.4.5 Local Functions

None

7 MIS of Requests Module

7.1 Module

Requests

7.2 Uses

Requests Controller, Plaid Service API

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
submitReimbursement	requestData: JSON	Boolean	ValidationException
submitPayment	paymentData: JSON	Boolean	PaymentProcessingExceptio
processLedger	ledgerData: JSON	Boolean	ReconciliationException

7.4 Semantics

7.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

7.4.2 State Variables

None

7.4.3 Environment Variables

- Plaid Service API for payment and ledger reconciliation

7.4.4 Assumptions

- All inputs are validated prior to processing. - Plaid Service API is available and operational.

7.4.5 Access Routine Semantics

submitReimbursement(requestData: JSON):

- transition: Stores the reimbursement request and initiates processing via the Requests Controller.
- output: Returns true if the request is successfully submitted.
- exception: ValidationException if the input data is invalid.

7.4.6 Local Functions

None

8 MIS of Notification Module

8.1 Module

Notification Module

8.2 Uses

Requests Module

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
notifyUser	email: String	String	EmailNotFoundException

8.4 Semantics

8.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

8.4.2 State Variables

User Details (email and notification status)

8.4.3 Environment Variables

None

8.4.4 Assumptions

None

8.4.5 Access Routine Semantics

notifyUser(email: String):

- transition: Queries Requests module for user info including email and request status.
- output: Returns an email body to be given to emailer API.
- exception: EmailNotFoundException if the user has no valid email to be returned.

8.4.6 Local Functions

- Validation functions for email.
- Functions to compose email body.

9 MIS of User Dashboard Module

9.1 Module

User Dashboard

9.2 Uses

Requests Module, Account Management API, ~~Authentication Module~~

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
viewDashboard	userId: String	JSON	AuthorizationException
viewRequests	userId: String	Array (Requests)	AuthorizationException
editProfile	userId: String, updates: JSON	Boolean	UpdateException

9.4 Semantics

9.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

9.4.2 State Variables

None

9.4.3 Environment Variables

- Connections to other modules for data abstraction.

9.4.4 Assumptions

- The user is authenticated and authorized before accessing the dashboard.

9.4.5 Access Routine Semantics

viewDashboard(userId: String):

- transition:
  - Validate userId format (UUIDv4 regex: `^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-4[0-9a-fA-F]{3}-`
  - Sanitize inputs using OWASP ZAP standards
- output: Returns user's dashboard data with XSS-protected strings
- exception: AuthorizationException if validation fails

9.4.6 Local Functions

None

10 MIS of Authentication Module

Authentication Module was deemed out of scope. It has been decided that we will instead integrate with the existing authentication service.

10.1 Module

Authentication

10.2 Uses

Emailer API, JWT tokens

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
sendConfirmation	email: String	Boolean	EmailException
verifyToken	token: String	Boolean	InvalidTokenException
authenticateUser	credentials: JSON	Boolean	AuthenticationException

10.4 Semantics

10.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

10.4.2 State Variables

- Active JWT tokens.

10.4.3 Environment Variables

- Email service for sending confirmation links.

10.4.4 Assumptions

- Email service API maintains 99.9% uptime (per provider SLA)
- All notifications adhere to RFC 5322 email standards
- Email body templates are pre-approved by MES stakeholders
- Network latency between modules remains below 200ms

10.4.5 Access Routine Semantics

sendConfirmation(email: String):

- transition: Sends a confirmation email with a token link.
- output: Returns true if the email is successfully sent.
- exception: EmailException if the email service fails.

10.4.6 Local Functions

None

11 MIS of Emailer API

11.1 Module

Emailer API

11.2 Uses

Account Management Module, Notification Module

11.3 Syntax

11.3.1 Exported Constants

Email sending address (An automated, do-not-reply email adress)

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
sendEmail	body: JSON, address: String	Boolean	InvalidEmailException

11.4 Semantics

11.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

11.4.2 State Variables

None

11.4.3 Environment Variables

Connection to donotreply automated email service

11.4.4 Assumptions

- An external API will be used. Specifics TBD

11.4.5 Access Routine Semantics

sendEmail(body: JSON):

- transition: Sends an email with body to the address specified.
- output: Returns a success or failure message depending on if the email was successfully sent.
- exception: InvalidEmailException if the address is invalid or the body is unsendable.

11.4.6 Local Functions

- `validateEmail(address: String): Boolean` - Checks RFC 5322 compliance using regex - Returns true if valid, false otherwise
- `sanitizeBody(body: JSON): String` - Removes HTML/CSS/JS tags from email content - Escapes special characters using OWASP guidelines

12 MIS of Account Management Controller

12.1 Module

Account Management Controller

12.2 Uses

Mongoose Schema, MongoDB

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
createUser	userDetails: JSON	JSON	DatabaseException
findUser	userId: String	JSON	UserNotFoundException
updateUser	userId: String, updates: JSON	JSON	DatabaseException
deleteUser	userId: String	JSON	AuthorizationException

12.4 Semantics

12.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

12.4.2 State Variables

MongoDB User Schema (defines fields like email, password, roles, etc.)

12.4.3 Environment Variables

MongoDB connection via Mongoose (database connection client)

12.4.4 Assumptions

- Mongoose (database connection client) is properly configured and connected to MongoDB.
- User schema validations are performed automatically during operations.

12.4.5 Access Routine Semantics

createUser(userDetails: JSON):

- transition: Saves a new user record to MongoDB.
- output: Returns a JSON object with detailed information about the result.
- exception: DatabaseException if saving fails due to validation or connection issues.

findUser(userId: String):

- transition: Queries the MongoDB collection for the specified user.
- output: Returns user data in JSON format.
- exception: UserNotFoundException if the user ID does not exist.

updateUser(userId: String, updates: JSON):

- transition: Updates the MongoDB collection for the specified user information.
- output: Returns a JSON object with detailed information about the result.
- exception: UserNotFoundException if the user ID does not exist.

deleteUser(userId: String):

- transition: Removes the specified user from the MongoDB collection.
- output: Returns a JSON object with detailed information about the result.
- exception: UserNotFoundException if the user ID does not exist.

12.4.6 Local Functions

- Validation functions for email and password.

13 MIS of Requests Controller Module

13.1 Module

Requests Controller

13.2 Uses

Database (via ORM)

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
storeReimbursement	requestData: JSON	Boolean	DatabaseException
storePayment	paymentData: JSON	Boolean	DatabaseException
reconcileLedger	ledgerData: JSON	Boolean	DatabaseException

13.4 Semantics

13.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

13.4.2 State Variables

- MongoDB collections for requests and ledgers

13.4.3 Environment Variables

- Database connection (via Mongoose ORM)

13.4.4 Assumptions

- Database schema is correctly defined and applied. - Database connection is persistent.

13.4.5 Access Routine Semantics

storeReimbursement(requestData: JSON):

- transition: Saves the reimbursement request in the database.
- output: Returns true if the operation is successful.
- exception: DatabaseException if the request cannot be stored.

13.4.6 Local Functions

None

14 MIS of Clubs Database

14.1 Module

Clubs Database

14.2 Uses

Mongoose Schema, MongoDB

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
addClub	clubDetails: JSON	JSON	DatabaseException
getClub	clubId: String	JSON	ClubNotFoundException
updateClub	clubId: String, updates: JSON	JSON	DatabaseException
deleteClub	clubId: String	JSON	AuthorizationException

14.4 Semantics

14.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

14.4.2 State Variables

MongoDB Club Schema

14.4.3 Environment Variables

MongoDB connection via Mongoose (database connection client)

14.4.4 Assumptions

- Mongoose (database connection client) is properly configured and connected to MongoDB.
- Club schema validations are performed automatically during operations.

14.4.5 Access Routine Semantics

addClub(clubDetails: JSON):

- transition: Adds a new club record to MongoDB.
- output: Creates club object and returns as JSON
- exception: DatabaseException if saving fails due to validation or connection issues.

getClub(clubId: String):

- transition: Fetches club data from the database.
- output: Returns club data in JSON format.
- exception: ClubNotFoundException if the club ID does not exist.

updateClub(clubId: String, updates: JSON):

- transition: Updates the MongoDB collection for the specified club information.
- output: Returns a JSON object with detailed information about the result.
- exception: ClubNotFoundException if the club ID does not exist.

deleteClub(clubId: String):

- transition: Removes the specified club from the MongoDB collection.
- output: Returns a JSON object with detailed information about the result.
- exception: ClubNotFoundException if the club ID does not exist.

14.4.6 Local Functions

None

15 MIS of User Database

15.1 Module

User Database

15.2 Uses

Mongoose Schema, MongoDB

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
addUser	userDetails: JSON	JSON	DatabaseException
getUser	userId: String	JSON	UserNotFoundException
updateUser	userId: String, updates: JSON	JSON	DatabaseException
deleteUser	userId: String	JSON	AuthorizationException

15.4 Semantics

15.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

15.4.2 State Variables

MongoDB User Schema

15.4.3 Environment Variables

MongoDB connection via Mongoose (database connection client)

15.4.4 Assumptions

- Mongoose (database connection client) is properly configured and connected to MongoDB.
- User schema validations are performed automatically during operations.

15.4.5 Access Routine Semantics

addUser(userDetails: JSON):

- transition: Adds a new user record to MongoDB.
- output: Creates user object and returns as JSON
- exception: DatabaseException if saving fails due to validation or connection issues.

getUser(userId: String):

- transition: Fetches user data from the database.
- output: Returns user data in JSON format.
- exception: UserNotFoundException if the user ID does not exist.

updateUser(userId: String, updates: JSON):

- transition: Updates the MongoDB collection for the specified user information.
- output: Returns a JSON object with detailed information about the result.
- exception: UserNotFoundException if the user ID does not exist.

deleteUser(userId: String):

- transition: Removes the specified user from the MongoDB collection.
- output: Returns a JSON object with detailed information about the result.
- exception: UserNotFoundException if the user ID does not exist.

15.4.6 Local Functions

None

16 MIS of Requests Database

16.1 Module

Requests Database

16.2 Uses

Mongoose Schema, MongoDB

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
addRequest	requestDetails: JSON	JSON	DatabaseException
getRequest	requestId: String	JSON	RequestNotFoundException
updateRequest	requestId: String, updates: JSON	JSON	DatabaseException
deleteRequest	requestId: String	JSON	AuthorizationException

16.4 Semantics

16.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

16.4.2 State Variables

MongoDB Request Schema

16.4.3 Environment Variables

MongoDB connection via Mongoose (database connection client)

16.4.4 Assumptions

- Mongoose (database connection client) is properly configured and connected to MongoDB.
- Request schema validations are performed automatically during operations.

16.4.5 Access Routine Semantics

addRequest(requestDetails: JSON):

- transition: Adds a new request record to MongoDB.
- output: Creates request object and returns as JSON
- exception: DatabaseException if saving fails due to validation or connection issues.

getRequest(requestId: String):

- transition: Fetches request data from the database.
- output: Returns request data in JSON format.
- exception: RequestNotFoundException if the request ID does not exist.

updateRequest(requestId: String, updates: JSON):

- transition: Updates the MongoDB collection for the specified request information.
- output: Returns a JSON object with detailed information about the result.
- exception: RequestNotFoundException if the request ID does not exist.

deleteRequest(requestId: String):

- transition: Removes the specified request from the MongoDB collection.
- output: Returns a JSON object with detailed information about the result.
- exception: RequestNotFoundException if the request ID does not exist.

16.4.6 Local Functions

None

17 MIS of Graphical User Interface

17.1 Module

Graphical User Interface

17.2 Uses

React.js, API Endpoints

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
renderDashboard	userToken: String	JS	AuthorizationException
handleInput	inputData: JSON	Boolean	InputValidationException

17.4 Semantics

17.4.1 SRS Traceability

- Linked to SRS Section 9.1.1 (Functional Requirements for Reimbursement)
- Maps to SRS Section 15.3 (Privacy Requirements)

17.4.2 State Variables

None

17.4.3 Environment Variables

Web browser, network connection

17.4.4 Assumptions

- User authentication is performed before accessing GUI components
- Input data is validated at the client side

17.4.5 Access Routine Semantics

renderDashboard(userToken: String):

- transition: Renders user dashboard based on provided token.
- output: Returns a rendered GUI to the user
- exception: AuthorizationException if the token is invalid.

17.4.6 Local Functions

Fetch API data

18 Appendix

Appendix — Reflection

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?
   One thing that went well during this deliverable was the team meeting where we discussed the design of the project in detail. It allowed us all to have a clear understanding of the modules we would be focusing on, which then made it easy to divide responsibilities since everyone was on the same page. Since we were all able to visualize how each module interacted with others, it made writing the document much easier.

2. What pain points did you experience during this deliverable, and how did you resolve them?
   One of the pain points we came across during this deliverable was aligning the sections with the evolving requirements. Coming back from winter break, there were still some requirements and specifications that needed to be cleared up. This is what drove us to schedule multiple team meetings to help us map-out a high-level design of the project which then helped us complete this deliverable.

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?
   Some key design decisions, such as the focus on functionality and ensuring all request systems work seamlessly, were driven by client feedback. They emphasized the importance of a reliable and efficient platform over an aesthetically pleasing front-end. Other decisions like our choice of technology stack, were based on the MES' existing technologies to ensure compatability.

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?

While creating this document, no other previous deliverables had to be changed. Before starting, we had closed all existing issues such as TA feedback and peer review. Having already gone through each document, we ensured that they were up-to-date and contained all the necessary information.

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)
   One limitation of our current solution is that it does not account for certain scalability challenges that could arise in the future, such as handling a high volume of simultaneous requests or users. With unlimited resources, we would implement a more robust infrastructure with auto-scaling capabilities and an optimized database that would be able to handle more traffic. Another big thing we would change with more resources is that we would invest in user experience design to ensure the system is intuitive and easy to use, even as more features are added.

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)
   One alternative design considered was using a monolithic approach, where all modules were tightly integrated into a single unit. This would have simplified some aspects of the system, particularly in terms of testing and deployment. However, the downside would be a lack of flexibility and scalability, as adding new features or making changes to one part of the system could potentially impact others. We opted for a modular design instead, as it offers better flexibility, scalability, and maintainability in the long term, allowing us to develop and deploy individual components independently.