Module Interface Specification for MES-ERP

Team #26, Ethical Pals
Sufyan Motala
Rachid Khneisser
Housam Alamour
Omar Muhammad
Taaha Atif

January 17, 2025

1 Revision History

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

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

1 Revision History i

2 Symbols, Abbreviations and Acronyms ii

3 Introduction 1

4 Notation 1

5 Timeline 2
- 5.1 Development Timeline 4
- 5.2 Testing and Verification 5
- 5.3 Responsibilities 5

6 Module Decomposition 5

7 MIS of Database Module 7
- 7.1 Module 7
- 7.2 Uses 7
- 7.3 Syntax 7
  - 7.3.1 Exported Constants 7
  - 7.3.2 Exported Access Programs 7
- 7.4 Semantics 7
  - 7.4.1 State Variables 7
  - 7.4.2 Environment Variables 8
  - 7.4.3 Assumptions 8
  - 7.4.4 Access Routine Semantics 8
  - 7.4.5 Local Functions 9

8 MIS of User Authentication & Profile Management Module 10
- 8.1 Module 10
- 8.2 Uses 10
- 8.3 Syntax 10
  - 8.3.1 Exported Constants 10
  - 8.3.2 Exported Access Programs 10
- 8.4 Semantics 11
  - 8.4.1 State Variables 11
  - 8.4.2 Environment Variables 11
  - 8.4.3 Assumptions 11
  - 8.4.4 Access Routine Semantics 11
  - 8.4.5 Local Functions 12

9 MIS of Expense Submission & Tracking Module **12**
9.1 Module . . . 12
9.2 Uses . . . 12
9.3 Syntax . . . 12
9.3.1 Exported Constants . . . 12
9.3.2 Exported Access Programs . . . 13
9.4 Semantics . . . 13
9.4.1 State Variables . . . 13
9.4.2 Environment Variables . . . 13
9.4.3 Assumptions . . . 13
9.4.4 Access Routine Semantics . . . 14
9.4.5 Local Functions . . . 15

10 MIS of Approval Workflow and Review Module **15**
10.1 Module . . . 15
10.2 Uses . . . 15
10.3 Syntax . . . 15
10.3.1 Exported Constants . . . 15
10.3.2 Exported Access Programs . . . 16
10.4 Semantics . . . 16
10.4.1 State Variables . . . 16
10.4.2 Environment Variables . . . 16
10.4.3 Assumptions . . . 16
10.4.4 Access Routine Semantics . . . 16
10.4.5 Local Functions . . . 17

11 MIS of Budget and Funding Management Module **17**
11.1 Module . . . 17
11.2 Uses . . . 17
11.3 Syntax . . . 17
11.3.1 Exported Constants . . . 17
11.3.2 Exported Access Programs . . . 17
11.4 Semantics . . . 17
11.4.1 State Variables . . . 17
11.4.2 Environment Variables . . . 17
11.4.3 Assumptions . . . 18
11.4.4 Access Routine Semantics . . . 18
11.4.5 Local Functions . . . 18

12 MIS of Reporting and Analytics Module **18**
12.1 Module . . . 18
12.2 Uses . . . 18
12.3 Syntax . . . 18

12.3.1 Exported Constants . . . 18
12.3.2 Exported Access Programs . . . 19
12.4 Semantics . . . 19
12.4.1 State Variables . . . 19
12.4.2 Environment Variables . . . 19
12.4.3 Assumptions . . . 19
12.4.4 Access Routine Semantics . . . 19
12.4.5 Local Functions . . . 19

13 MIS of Graphical User Interface (GUI) Module 19
13.1 Module . . . 19
13.2 Uses . . . 20
13.3 Syntax . . . 20
13.3.1 Exported Constants . . . 20
13.3.2 Exported Access Programs . . . 20
13.4 Semantics . . . 20
13.4.1 State Variables . . . 20
13.4.2 Environment Variables . . . 20
13.4.3 Assumptions . . . 20
13.4.4 Access Routine Semantics . . . 21
13.4.5 Local Functions . . . 21

14 MIS of Policy & Compliance Management Module 21
14.1 Module . . . 21
14.2 Uses . . . 21
14.3 Syntax . . . 21
14.3.1 Exported Constants . . . 21
14.3.2 Exported Access Programs . . . 22
14.4 Semantics . . . 22
14.4.1 State Variables . . . 22
14.4.2 Environment Variables . . . 22
14.4.3 Assumptions . . . 22
14.4.4 Access Routine Semantics . . . 22
14.4.5 Local Functions . . . 23

15 MIS of Integration with Other University Systems Module 23
15.1 Module . . . 23
15.2 Uses . . . 23
15.3 Syntax . . . 23
15.3.1 Exported Constants . . . 23
15.3.2 Exported Access Programs . . . 23
15.4 Semantics . . . 23
15.4.1 State Variables . . . 23

15.4.2 Environment Variables . . . 24
15.4.3 Assumptions . . . 24
15.4.4 Access Routine Semantics . . . 24
15.4.5 Local Functions . . . 24

16 MIS of Administrator and Configuration Panel Module 24

16.1 Module . . . 24
16.2 Uses . . . 25
16.3 Syntax . . . 25
16.3.1 Exported Constants . . . 25
16.3.2 Exported Access Programs . . . 25
16.4 Semantics . . . 25
16.4.1 State Variables . . . 25
16.4.2 Environment Variables . . . 25
16.4.3 Assumptions . . . 26
16.4.4 Access Routine Semantics . . . 26
16.4.5 Local Functions . . . 26

17 Appendix 28

18 Appendix 29

3 Introduction

The following document details the Module Interface Specifications (MIS) for the McMaster Engineering Society Custom Financial Expense Reporting Platform (MES-ERP). This platform is designed to streamline financial expense management for the McMaster Engineering Society (MES), providing an efficient and user-friendly solution for submitting, approving, and tracking reimbursement requests.

The MES-ERP aims to address the unique financial management needs of the MES by integrating expense tracking, budget management, and policy compliance into a cohesive platform. The system ensures accurate and efficient handling of financial requests while maintaining compliance with organizational policies and university regulations.

Complementary documents to this MIS include the System Requirements Specification (SRS) and the Module Guide (MG), which provide additional context and design details. The complete documentation and implementation of the MES-ERP can be found at https://github.com/Housam2020/MES-ERP.

4 Notation

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 MES-ERP.

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 MES-ERP 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, MES-ERP 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 Timeline

This section outlines the timeline for the implementation of the project. The timeline includes the development of all modules, testing, and deployment phases. Tasks are divided by modules, specifying responsibilities and key milestones.

5.1 Development Timeline

Week	**Task**	**Details**
Week 1	Initial Planning	Team meeting to finalize requirements and review the SRS. Assign responsibilities for each module.
Week 2	User Authentication and Profile Management Module	Development of secure login, roles, and basic profile updates. Begin unit testing for authentication.
Week 3	Expense Submission and Tracking Module	Implement submission forms for expenses, including receipt uploads and status tracking. Start unit testing.
Week 4	Budget and Funding Management Module	Develop logic for fetching budgets, validating funds, and updating department budgets. Integrate with the database module.
Week 5	Approval Workflow and Review Module	Implement dynamic routing rules and approval workflows. Integrate notifications for pending approvals. Conduct unit testing.
Week 6	Disbursement and Payment Processing Module	Create logic for issuing payments and generating logs for auditing. Integrate with external financial systems.
Week 7	Notifications and Communication Module	Build notifications for key events, such as request approvals or denials. Include dashboard alerts for overdue actions.
Week 8	Reporting and Analytics Module	Develop functionality for generating reports and tracking usage statistics. Validate with sample data.

5.2 Testing and Verification

Testing will be conducted in multiple phases:

- Unit Testing: Conducted during the implementation of each module (Weeks 2–11).
- Integration Testing: Performed once modules are integrated (Week 12).
- System Testing: Comprehensive testing of the entire system to ensure functionality and performance (Week 12–13).
- User Acceptance Testing: Gather feedback from end-users during Week 13 to identify potential areas for improvement.

5.3 Responsibilities

The following responsibilities are assigned to team members:

- Module Implementation: Each team member is responsible for implementing the modules assigned to them during the initial planning phase.
- Documentation: All team members contribute to the MIS and ensure consistency with the SRS.
- Testing: Shared responsibility for writing and executing test cases, with module developers performing unit tests.
- Deployment: Coordinated by the team lead, with support from all team members for configuration and setup.

6 Module Decomposition

The modules in this system are divided into the following levels:

Level 1	**Level 2**
Hardware-Hiding Module	Database Module
Behavior-Hiding Module	Approval Workflow and Review Budget and Funding Management Reporting and Analytics Policy and Compliance Management Notifications and Communication Disbursement and Payment Processing
Software Decision Module	GUI Module Integration with Other University Systems Software Design Decisions Module

Table 1: Module Decomposition

7 MIS of Database Module

7.1 Module

Database

7.2 Uses

None (Hardware-Hiding Module).

7.3 Syntax

7.3.1 Exported Constants

- DATABASE_URL: The URL for the database connection.
- MAX_BATCH_SIZE: The maximum number of records to fetch in a single query.
- MAX_CONNECTIONS: The maximum number of concurrent database connections.
- DATABASE_CREDENTIALS: The username and password for database access.
- DEFAULT_TIMEOUT: The default timeout for database queries.

7.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
query	queryString (`String`), params (`Object`)	results (`Array`)	QueryExecutionException
insert	collection (`String`), document (`Object`)	documentID (`String`)	InsertionException
update	collection (`String`), filter (`Object`), updates (`Object`)	modifiedCount (`Number`)	UpdateException
delete	collection (`String`), filter (`Object`)	deletedCount (`Number`)	DeletionException
transaction	operations (`Array`)	results (`Array`)	TransactionException

7.4 Semantics

7.4.1 State Variables

- connections: Pool of active database connections.
- collections: Schema and metadata for database collections.

- queries: Record of executed queries and their results.
- inserts: Record of inserted documents and their IDs.
- updates: Record of updated documents and their modification counts.
- deletions: Record of deleted documents and their deletion counts.
- transactions: Record of transaction operations and their results.
- collections: Record of database collections and their schemas.

7.4.2 Environment Variables

- Database Server: Hosts the database and provides access to stored data.
- Database Client: Connects to the database server and executes queries.
- Configuration: Database connection and optimization settings.

7.4.3 Assumptions

- Database server is accessible and responsive.
- Database schema is predefined and consistent across collections.
- Database queries are validated before execution.
- Authentication credentials are securely stored

7.4.4 Access Routine Semantics

`query(queryString)`:

- **Transition**: Executes a database query with the provided parameters.
- **Output**: Returns query results or raises QueryExecutionException.
- **Precondition**: Query string is valid MongoDB syntax.
- **Postcondition**: Results are properly sanitized and formatted.

`insert(collection, document)`:

- **Transition**: Inserts a new document into the specified collection.
- **Output**: Returns the unique identifier of the inserted document.
- **Exceptions**: InsertionException if the document is invalid or the operation fails.

- **Precondition**: Document matches the collection schema.

`update(collection, filter, updates)`:

- **Transition**: Updates documents matching the filter criteria.
- **Output**: Returns the count of modified documents.
- **Exceptions**: UpdateException if the operation fails or the updates are invalid.
- **Precondition**: Update operation is valid for the collection schema.
- **Postcondition**: Matching documents are updated atomically.

`delete(collection, filter)`:

- **Transition**: Removes documents matching the filter criteria.
- **Output**: Returns the count of deleted documents.
- **Precondition**: Filter is non-empty to prevent accidental collection deletion.
- **Postcondition**: Matching documents are permanently removed.

`transaction(operations)`:

- **Transition**: Executes multiple operations in a single transaction.
- **Output**: Returns results of all operations or raises TransactionException.
- **Precondition**: All operations are valid and follow the collection schema.
- **Postcondition**: All operations succeed or all are rolled back.

7.4.5 Local Functions

- `validateSchema(collection, document) -> Boolean`: Verifies document against collection schema.
- `sanitizeQuery(query) -> String`: Prevents injection attacks by sanitizing query strings.
- `manageConnections() -> void`: Monitors and manages database connection pool.
- `logQuery(query, results) -> void`: Records executed queries and results.
- `logInsert(collection, documentID) -> void`: Records inserted documents.
- `logUpdate(collection, modifiedCount) -> void`: Records updated documents.

- `logDelete(collection, deletedCount) -> void`: Records deleted documents.
- `logTransaction(operations) -> void`: Records transaction operations and results.
- `logCollection(collection, schema) -> void`: Records collection schema and metadata.

8 MIS of User Authentication & Profile Management Module

8.1 Module

User Authentication & Profile Management

8.2 Uses

- Database Module
- Integration with Other University Systems Module
- Notifications and Communication Module

8.3 Syntax

8.3.1 Exported Constants

- SESSION_TIMEOUT: The duration (in minutes) before an inactive session expires.
- MAX_LOGIN_ATTEMPTS: Maximum number of failed login attempts before account lockout.
- LOCKOUT_DURATION: Duration (in minutes) of account lockout after exceeding MAX_LOGIN_ATTEMPTS.

8.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
authenticate	credentials (`Object`)	sessionToken (`String`)	AuthenticationFailedException
getProfile	userID (`String`)	profileData (`Object`)	UserNotFoundException
updateProfile	userID (`String`), updates (`Object`)	confirmation (`Boolean`)	InvalidProfileDataException
validateSession	sessionToken (`String`)	isValid (`Boolean`)	InvalidSessionException
getRolePermissions	userID (`String`)	permissions (`Object`)	RoleNotFoundException

8.4 Semantics

8.4.1 State Variables

- sessions: A mapping of active session tokens to user information and expiration times.
- profiles: A mapping of user IDs to their profile information and preferences
- loginAttempts: A record of failed login attempts per user.

8.4.2 Environment Variables

- Identity Provider: The university's SSO system for authentication.
- User Database: Stores user profiles, roles, and authentication data.

8.4.3 Assumptions

- The university's identity provider is available and functioning
- Users have unique identifiers across the system
- Profile data is validated before storage

8.4.4 Access Routine Semantics

`authenticate(credentials)`:

- **Transition**: Validates credentials against the identity provider and creates a new session.
- **Output**: Returns a session token if successful.
- **Exceptions**: AuthenticationFailedException if the credentials are invalid.

`getProfile(userID)`:

- **Output**: Returns the user's profile data.
- **Exceptions**: UserNotFoundException if the user ID does not exist.

`updateProfile(userID, updates)`:

- **Transition**: Updates the specified user's profile with new information.
- **Output**: Returns `true` if successful.
- **Exceptions**: InvalidProfileDataException if the updates are invalid.

`validateSession(sessionToken)`:

- **Output**: Returns `true` if the session is valid and not expired
- **Exceptions**: InvalidSessionException if the session token is invalid or expired.

`getRolePermissions(userID)`:

- **Output**: Returns the permissions associated with the user's role.
- **Exceptions**: RoleNotFoundException if the user's role is not found.

8.4.5 Local Functions

None.

9 MIS of Expense Submission & Tracking Module

9.1 Module

Expense Submission & Tracking

9.2 Uses

- Database Module
- Budget and Funding Management Module
- Notifications and Communication Module
- Policy & Compliance Management Module

9.3 Syntax

9.3.1 Exported Constants

- MAX_RECEIPT_SIZE: The maximum file size (in MB) for uploaded receipts.
- ALLOWED_FILE_TYPES: List of accepted file formats for receipts [”pdf”, ”jpg”, ”png”].
- EXPENSE_CATEGORIES: Predefined expense categories [”conference”, ”travel”, ”supplies”, ”materials”].

9.3.2 Exported Access Programs

Name	Input	Output	Exceptions
submitExpense	expenseDetails (`Object`), attachments (`Array`)	requestID (`String`)	InvalidExpenseException
uploadAttachment	requestID (`String`), file (`File`)	fileID (`String`)	FileUploadException
getExpenseStatus	requestID (`String`)	status (`Object`)	RequestNotFoundException
updateExpenseDetails	requestID (`String`), updates (`Object`)	confirmation (`Boolean`)	InvalidUpdateException
searchExpenses	filters (`Object`)	expenseList (`Array`)	InvalidSearchException
categorizeExpense	expenseData (`Object`)	category (`String`)	CategorizationException

9.4 Semantics

9.4.1 State Variables

- expenses: A mapping of request IDs to their complete expense details and status.
- attachments: A mapping of file IDs to their metadata and storage locations.
- categories: A record of expense categorization rules and patterns.

9.4.2 Environment Variables

- File Storage: Stores uploaded receipts and attachments.
- OCR Service: Extracts text data from uploaded receipts for processing.
- ML Model: Assists in automatic expense categorization
- Database: Stores expense records and tracking information

9.4.3 Assumptions

- Receipts are uploaded in a standard format (PDF, JPG, PNG).
- Expense categorization is based on predefined rules and ML models.
- All monetary values are in Canadian dollars (CAD).
- File storage system has sufficient capacity for attachments.
- Users have necessary permissions to submit expenses.
- Each expense request has at least one receipt attachment.

9.4.4 Access Routine Semantics

`submitExpense(expenseDetails, attachments)`:

- **Transition**: Creates a new expense request with the provided details and attachments.
- **Output**: Returns the unique request ID if successful.
- **Exceptions**: InvalidExpenseException if the expense details are incomplete or invalid.
- **Precondition**: expenseDetails contains required fields (amount, date, purpose, funding source).
- **Postcondition**: Expense request is created with ”Submitted” status.

`uploadAttachment(requestID, file)`:

- **Transition**: Validates and stores the provided file attachment.
- **Output**: Returns a unique fileID if successful.
- **Exceptions**: FileUploadException if the file size or type is invalid.
- **Precondition**: File size and type meet system requirements.
- **Postcondition**: File is stored and linked to the expense request.

`getExpenseStatus(requestID)`:

- **Output**: Returns the current status and tracking information for the specified request.
- **Exceptions**: RequestNotFoundException if the request ID is invalid.
- **Postcondition**: Status information is retrieved for the request.

`updateExpenseDetails(requestID, updates)`:

- **Transition**: Applies valid updates to the specified expense request.
- **Output**: Returns `true` if successful.
- **Exceptions**: InvalidUpdateException if the updates are invalid.
- **Precondition**: Request is in an editable state.
- **Postcondition**: Expense details are updated and audit trail is maintained.

`searchExpenses(filters)`:

- **Output**: Returns a list of expense requests matching the specified filters.

- **Exceptions**: InvalidSearchException if filter criteria are invalid.
- **Postcondition**: Filtered expense list is returned based on search criteria.

`categorizeExpense(expenseData)`:

- **Transition**: Analyzes expense details to determine the appropriate category.
- **Output**: Returns the suggested category for the expense or raises CategorizationException.
- **Postcondition**: Expense category is assigned based on ML model and predefined rules.
- **Uses**: ML model and predefined rules for categorization.
- **Updates**: Category confidence score in expense metadata.

9.4.5 Local Functions

- `validateExpenseData(data: Object) -> Boolean`: Ensures all required fields are present and valid.
- `sanitizeAttachment(file: File) -> File`: Processes uploaded files for security.
- `calculateTotalAmount(items: Array) -> Number`: Computes total expense amount including all items.

10 MIS of Approval Workflow and Review Module

10.1 Module

Approval Workflow and Review

10.2 Uses

- Budget and Funding Management Module
- Notifications and Communication Module
- Database Module

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	Input	Output	Exceptions
routeRequest	requestID (`String`), userRole (`String`)	status (`String`)	InvalidRoleException
addNote	requestID (`String`), note (`String`)	confirmation (`Boolean`)	RequestNotFoundException
updateStatus	requestID (`String`), newStatus (`String`)	confirmation (`Boolean`)	InvalidStatusException

10.4 Semantics

10.4.1 State Variables

- requests: A mapping of request IDs to their current status and approver roles.

10.4.2 Environment Variables

- Database: Stores information about requests, user roles, and approval workflows.
- Notification System: Sends alerts for pending approvals or status updates.

10.4.3 Assumptions

- User roles are pre-validated by the authentication system.
- All approvers have access to the system during their review process.

10.4.4 Access Routine Semantics

`routeRequest(requestID, userRole)`:

- Output: Returns the current status of the request or raises an InvalidRoleException if the userRole is unauthorized.

`addNote(requestID, note)`:

- Transition: Adds a note to the request specified by `requestID`.
- Output: Returns `true` if successful or raises RequestNotFoundException if the request ID does not exist.

`updateStatus(requestID, newStatus)`:

- Transition: Updates the status of the request specified by `requestID`.
- Output: Returns `true` if successful or raises InvalidStatusException if the new status is invalid.

10.4.5 Local Functions

None.

11 MIS of Budget and Funding Management Module

11.1 Module

Budget and Funding Management

11.2 Uses

- Database Module
- Notifications and Communication Module

11.3 Syntax

11.3.1 Exported Constants

- MAX_BUDGET: The maximum allowable budget per department.

11.3.2 Exported Access Programs

Name	Input	Output	Exceptions
getBudget	departmentID (`String`)	budgetDetails (`Object`)	DepartmentNotFoundException
validateFunds	requestAmount (`Float`), departmentID (`String`)	status (`Boolean`)	InsufficientFundsException
updateBudget	departmentID (`String`), amount (`Float`)	confirmation (`Boolean`)	BudgetOverflowException

11.4 Semantics

11.4.1 State Variables

- budgets: A mapping of department IDs to their current budget values.

11.4.2 Environment Variables

- Database: Stores information about department budgets and transactions.

11.4.3 Assumptions

- All transactions are logged for auditing purposes.
- Real-time budget data is synchronized with the university financial system.

11.4.4 Access Routine Semantics

`getBudget(departmentID)`:

- Output: Returns an object containing the budget details of the specified department or raises DepartmentNotFoundException.

`validateFunds(requestAmount, departmentID)`:

- Output: Returns `true` if sufficient funds are available, otherwise raises Insufficient-FundsException.

`updateBudget(departmentID, amount)`:

- Transition: Updates the budget for the specified department by the specified amount.
- Output: Returns `true` if successful or raises BudgetOverflowException if the update exceeds MAX_BUDGET.

11.4.5 Local Functions

None.

12 MIS of Reporting and Analytics Module

12.1 Module

Reporting and Analytics

12.2 Uses

- Database Module

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	Input	Output	Exceptions
generateReport	filters (`Object`)	report (`PDF/CSV`)	InvalidFilterException
getUsageStats	timeframe (`String`)	stats (`Object`)	None

12.4 Semantics

12.4.1 State Variables

- reports: Stores past generated reports for caching and quick access.

12.4.2 Environment Variables

- Database: Stores analytics data related to system usage and transactions.

12.4.3 Assumptions

- Data required for analytics is periodically updated and complete.

12.4.4 Access Routine Semantics

`generateReport(filters)`:

- Output: Returns a report in PDF or CSV format based on the filters provided, or raises InvalidFilterException if the filters are malformed.

`getUsageStats(timeframe)`:

- Output: Returns an object containing system usage statistics for the specified timeframe.

12.4.5 Local Functions

None.

13 MIS of Graphical User Interface (GUI) Module

13.1 Module

Graphical User Interface (GUI)

13.2 Uses

- Notifications and Communication Module
- Database Module
- Approval Workflow and Review Module

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
`renderDashboard`	`userID (String)`	`dashboardView (HTML/JSON)`	`UserNotFoundException`
`updateView`	`viewName (String)`	`success (Boolean)`	`ViewNotFoundException`
`handleInput`	`event (Object)`	`actionResponse (Boolean)`	`InputException`

13.4 Semantics

13.4.1 State Variables

- **activeView**: The current view being displayed to the user, identified by its name (e.g., "Dashboard", "Expense Submission").
- **userSession**: Details about the currently logged-in user, including preferences and session data.

13.4.2 Environment Variables

- **Browser Window**: Displays the GUI and captures user inputs.
- **Server API**: Fetches data to dynamically update the GUI based on user interactions.

13.4.3 Assumptions

- The browser supports modern web standards (HTML5, CSS3, JavaScript).
- Users have active and authenticated sessions before interacting with the GUI.

13.4.4 Access Routine Semantics

`renderDashboard(userID)`:

- **Output**: Renders the dashboard for the given `userID` or raises a `UserNotFoundException` if the user is not valid.

`updateView(viewName)`:

- **Output**: Updates the active view to the one specified by `viewName` and returns `true` if successful, or raises a `ViewNotFoundException`.

`handleInput(event)`:

- **Transition**: Processes user interactions, such as button clicks or form submissions, and triggers corresponding actions.
- **Output**: Returns `true` if the input is handled successfully or raises an `InputException`.

13.4.5 Local Functions

None.

14 MIS of Policy & Compliance Management Module

14.1 Module

Policy & Compliance Management

14.2 Uses

- Database Module
- Notifications and Communication Module

14.3 Syntax

14.3.1 Exported Constants

- `MAX_REIMBURSEMENT`: The maximum allowable reimbursement amount for a single request.
- `TRAVEL_APPROVAL_LIMIT`: The threshold above which travel expenses require prior approval.

14.3.2 Exported Access Programs

Name	Input	Output	Exceptions
`validateRequest`	`requestDetails (Object)`	`validationStatus (Boolean)`	`PolicyViolationException`
`getPolicyRules`	`policyType (String)`	`rules (Object)`	`PolicyNotFoundException`
`logComplianceCheck`	`requestID (String), result (Boolean)`	`confirmation (Boolean)`	`LogFailureException`

14.4 Semantics

14.4.1 State Variables

- **policies**: A mapping of policy types to their respective rules and thresholds.
- **complianceLogs**: A record of compliance checks performed on submitted requests.

14.4.2 Environment Variables

- **Policy Database**: Stores policy definitions and thresholds.
- **Audit System**: Logs compliance checks and violations for review.

14.4.3 Assumptions

- Policy rules are periodically updated to align with organizational regulations.
- Compliance checks are triggered automatically during the request submission process.

14.4.4 Access Routine Semantics

`validateRequest(requestDetails)`:

- **Output**: Returns `true` if the request complies with all applicable policies or raises a `PolicyViolationException` if a rule is violated.

`getPolicyRules(policyType)`:

- **Output**: Returns the rules for the specified `policyType` or raises a `PolicyNotFoundException` if the policy type is invalid.

`logComplianceCheck(requestID, result)`:

- **Transition**: Records the result of a compliance check for the given `requestID`.
- **Output**: Returns `true` if the log is updated successfully or raises a `LogFailureException`.

14.4.5 Local Functions

None.

15 MIS of Integration with Other University Systems Module

15.1 Module

Integration with Other University Systems

15.2 Uses

- Database Module
- External APIs (University Systems)

15.3 Syntax

15.3.1 Exported Constants

- `STUDENT_INFO_API_URL`: The endpoint for accessing the university's Student Information System (SIS).
- `FINANCE_SYSTEM_API_URL`: The endpoint for accessing the university's financial system.

15.3.2 Exported Access Programs

Name	Input	Output	Exceptions
`fetchStudentInfo`	`studentID (String)`	`studentDetails (Object)`	`StudentNotFoundException`
`syncFinancialData`	`departmentID (String)`	`syncStatus (Boolean)`	`FinanceSyncFailureException`
`verifyEnrollment`	`studentID (String)`	`enrollmentStatus (Boolean)`	`EnrollmentVerificationExceptio`

15.4 Semantics

15.4.1 State Variables

- **universityData**: A cache of information fetched from external university systems for performance optimization.

- **syncLogs**: A record of synchronization activities between the system and external APIs.

15.4.2 Environment Variables

- **University SIS**: Provides student details such as enrollment status, department, and contact information.
- **University Finance System**: Manages departmental budgets, account balances, and payment records.

15.4.3 Assumptions

- API endpoints for university systems are reliable and adhere to predefined contracts.
- Authentication credentials for accessing university systems are securely stored and updated as needed.

15.4.4 Access Routine Semantics

`fetchStudentInfo(studentID)`:

- **Output**: Returns the details of the student identified by `studentID`, or raises `StudentNotFoundExcep` if no matching record is found.

`syncFinancialData(departmentID)`:

- **Transition**: Synchronizes financial data for the given `departmentID` with the university's finance system.
- **Output**: Returns `true` if the synchronization is successful or raises `FinanceSyncFailureException` if an error occurs.

`verifyEnrollment(studentID)`:

- **Output**: Returns `true` if the student is enrolled, or raises `EnrollmentVerificationException` if verification fails.

15.4.5 Local Functions

None.

16 MIS of Administrator and Configuration Panel Module

16.1 Module

Administrator and Configuration Panel

16.2 Uses

- Database Module
- Notifications and Communication Module
- Integration with Other University Systems Module

16.3 Syntax

16.3.1 Exported Constants

- `DEFAULT_ROLE_PERMISSIONS`: A mapping of roles to their default access permissions.
- `MAX_NOTIFICATION_TEMPLATES`: The maximum number of customizable notification templates allowed.

16.3.2 Exported Access Programs

Name	**Input**	**Output**	**Exceptions**
addRole	roleName (String), permissions (List)	confirmation (Boolean)	RoleAlreadyExistsException
updateApprovalChain	chainConfig (Object)	confirmation (Boolean)	InvalidChainConfigException
editNotificationTemplate	templateID (String), newTemplate (String)	confirmation (Boolean)	TemplateNotFoundException
viewLogs	timeframe (String)	logData (Object)	None

16.4 Semantics

16.4.1 State Variables

- **roles**: A mapping of roles to their permissions and associated users.
- **notificationTemplates**: A collection of templates for system notifications.
- **auditLogs**: A collection of system logs for administrative actions.

16.4.2 Environment Variables

- **Database**: Stores configuration settings, logs, and role assignments.
- **Notification System**: Delivers updates to users based on configured templates.

16.4.3 Assumptions

- Role and permission updates propagate immediately to all system components.
- Audit logs are retained and accessible for the configured retention period.

16.4.4 Access Routine Semantics

`addRole(roleName, permissions)`:

- **Transition**: Adds a new role with the specified permissions to the system.
- **Output**: Returns `true` if the role is added successfully or raises `RoleAlreadyExistsException`.

`updateApprovalChain(chainConfig)`:

- **Transition**: Updates the approval chain configuration based on the provided settings.
- **Output**: Returns `true` if the update is successful or raises `InvalidChainConfigException`.

`editNotificationTemplate(templateID, newTemplate)`:

- **Transition**: Updates the specified notification template with the new content.
- **Output**: Returns `true` if the template is updated successfully or raises `TemplateNotFoundException`

`viewLogs(timeframe)`:

- **Output**: Returns log data for the specified timeframe.

16.4.5 Local Functions

None.

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.

17 Appendix

[Extra information if required. Currently, none. —SS]

18 Appendix

[Extra information if required. Currently, none. —SS]

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?**
   Rachid: Collaborating as a team was smooth, and we were able to divide the modules effectively, which streamlined the writing process.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   Rachid: One challenge was ensuring consistency across modules. Regular team reviews and communication helped resolve any inconsistencies.

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?**
   Rachid: Many decisions, like integrating notifications, came directly from stakeholder feedback. Others, like modular decomposition, were based on best practices and team experience.

4. **While creating the design doc, what parts of your other documents (e.g., requirements, hazard analysis, etc.), if any, needed to be changed, and why?**
   Rachid: The requirements document was updated to better align with the final design, specifically in the Reporting module to include export formats.

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)**
   Rachid: With unlimited resources, we could enhance system scalability and user interface design, making it more robust and user-friendly.

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)
Rachid: We considered alternative approaches for managing the workflow logic, such as using external libraries, but chose an in-house solution for simplicity and better control over implementation.