Module Interface Specification for SFWRENG 4G06 - Capstone Design Process

Team 17, DomainX

Awurama Nyarko
Haniye Hamidizadeh
Fei Xie
Ghena Hatoum

January 25, 2026

1 Revision History

Date	Developer(s)	Change
Nov 12, 2025	Awurama	Rev -1
Jan 19, 2026	Awurama	Rev 0 Based on feedback

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS Documentation.

This section records acronyms and abbreviations for easy reference. Additional terms specific to the Module Interface Specification (MIS) are included below.

Table 1: Symbols, Abbreviations, and Acronyms

Symbol / Acronym	Description
AC	Anticipated Change.
ADT	Abstract Data Type.
AHP	Analytic Hierarchy Process, method for pairwise comparison and ranking of libraries.
AI	Artificial Intelligence.
API	Application Programming Interface, mechanism for data retrieval (e.g., GitHub API, PyPI API).
BWM	Best–Worst Method.
CAS	Computing and Software Department (McMaster University).
CI/CD	Continuous Integration / Continuous Deployment, automated testing and deployment pipeline used in GitHub Actions.
CSV	Comma-Separated Values, used as an export format for datasets.
DAG	Directed Acyclic Graph.
DB	Database, a MySQL instance used for persistent data storage.
Domain	Research Software Domain.
Excel Sheets	Existing manual tools previously used for data collection.
Infrastructure	University-provided resources such as hosting, databases, and servers.
JSON	JavaScript Object Notation, data interchange format used by the system's APIs.
LLM	Large Language Model.
M	Module.
MG	Module Guide.
MIS	Module Interface Specification (this document).
ML	Machine Learning.
NN	Neural Network.
NNL	Neural Network Libraries.
ORM	Object Relational Mapper, Django ORM used for database interactions.
OS	Operating System.
Packages	Software Packages.
PoC	Proof of Concept, demonstration validating workflow integration.

Continued on next page

Symbol / Acronym	Description
R	Requirement.
REST API	Representational State Transfer API, communication style used between the frontend (React) and backend (Django).
Research Subteam	Student group applying the methodology and writing the research paper.
SC	Scientific Computing.
SFWRENG 4G06	Capstone Design Process.
SRS	Software Requirements Specification.
SSB	Skew-Symmetric Bilinear.
Stakeholders	All individuals involved in or affected by the project (e.g., supervisor, researchers, domain expert).
Supervisor	Faculty member overseeing the project.
Tool	The software being developed to automate data collection, visualization, and storage.
UI	User Interface, front-end component built with React.
UI/UX	User Interface and User Experience.
UC	Unlikely Change.
VnV	Verification and Validation, process of ensuring correctness and meeting stakeholder needs.
HTTP	Hypertext Transfer Protocol, application-layer protocol for transmitting web resources.
HTTPS	Hypertext Transfer Protocol Secure, encrypted version of HTTP using TLS.
CORS	Cross-Origin Resource Sharing, browser security mechanism controlling cross-domain HTTP requests.

Contents

3 Introduction

The following document details the **Module Interface Specifications (MIS)** for the **Domain Assessment Tool**, a web-based system that automates data collection, analysis, and visualization to evaluate different research domains. The system enables users to systematically compare libraries, frameworks, or technologies by applying a structured methodology that combines both quantitative metrics and qualitative insights.

The Domain Assessment Tool is structured as a client-server system with a user interface, an application logic layer, and a persistent storage layer. Components communicate through well-defined module interfaces using serialized request/response data. Technology and framework choices are treated as design decisions captured in the MG and implementation artifacts, and are not required to interpret the module interfaces specified in this MIS.

Complementary documents include the System Requirements Specification (SRS) and the Module Guide (MG). The full documentation and implementation are maintained in the project's GitHub repository at DomainX.

4 Notation

The **Module Interface Specification (MIS)** follows the notation and principles of Hoffman and Strooper (1995), with adaptations from Ghezzi et al. (2003). Mathematical conventions align with Chapter 3 of Hoffman and Strooper (1995) and notations used in **SFWRENG 4G06 – Capstone Design Process**.

4.1 General Conventions

- **Assignment:** `x := e`
- **Conditional rules:** $(c_1 \Rightarrow r_1 \mid c_2 \Rightarrow r_2 \mid \ \ldots \ \mid c_n \Rightarrow r_n)$
- **Parameter modes:** `in`, `out`
- **Optional parameter:** `[param]`
- **Exceptions:** functions list `ExceptionName` in an *Exceptions* column; raising an exception denotes a partial function on the exceptional domain.
- **Pre/postconditions:** `requires P` / `ensures Q`
- **Undefined value / error sentinel:** $\perp$
- **Outputs:** `out := e`
- **Exceptions:** `exc := ExceptionName`

4.2 Logic and Set Notation

- **Logical operators:** $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$
- **Quantifiers:** $\forall x \in S \cdot P(x)$, $\exists x \in S \cdot P(x)$
- **Sets:** $\emptyset$, $\{x \in S | P(x)\}$, union $A \cup B$, intersection $A \cap B$, difference $A \setminus B$, cardinality $|S|$
- **Sequences/lists:** $(a_1, \ldots, a_n)$; concatenation $s \| t$; length $|s|$
- **Maps/dictionaries:** $m : K \to V$; application $m(k)$; update $m[k \mapsto v]$

4.3 Primitive and Derived Types

Data Type	Notation	Description
character	`char`	Single Unicode character.
integer	$\mathbb{Z}$	Whole numbers in $(-\infty, \infty)$.
natural number	$\mathbb{N}$	Non-negative integers $(0, 1, 2, \ldots)$.
real	$\mathbb{R}$	Real numbers.
boolean	`bool`	`True` or `False`.
string	`string`	Finite sequence of characters.
tuple	`(`T_1`,...,`T_n`)`	Fixed-size, ordered collection, possibly heterogeneous.
list / sequence	`list(T)`	Variable-size sequence of T.
set	$\mathcal{P}(T)$	Finite subset of T (power-set elements).
dictionary / map	`dict(K,V)`	Finite mapping from K to V.
option / optional type	`Option(T)`	Either `Some(T)` or `None`.

Table 2: Primitive and Derived Data Types

4.4 Units and Identifiers

- **Timestamps:** use ISO 8601 (e.g., 2025-11-11T09:30:00Z).
- **Durations:** explicit units (e.g., `min`, `hrs`).
- **Identifiers:** `domainID`, `metricID`, and `packageID` are opaque strings unless otherwise specified.

These conventions are used to define interfaces, state variables, environment variables, and access routines throughout the MIS. Derived structures (lists, maps, sets, tuples) represent collections of domain records, configuration parameters, and request/response data.. Local functions are defined by their type signatures and input/output relationships.

5 Module Decomposition

The following table (Table 3) summarizes the hierarchical organization of modules for the Domain Assessment Tool. Level 1 modules correspond to high-level abstraction layers, while Level 2 modules provide specific functionality within each layer.

Table 3: Module Hierarchy

Level 1	**Level 2**
Hardware-Hiding Module	Browser Module
Behaviour-Hiding Module	Domains Page Module
	Application UI Module
	Data Edit Module
	User Authentication Module
	User Role Access Module
	User Page Module
	Automated Metrics Module
	Comparison Module
	Configuration Module
Software Decision Module	System API Gateway Module
	Ranking Algorithm Module
	Graphing Module
	File Import Module
	File Export Module
	Repository API Module
	Database Persistence Module
	Logging Module

6 Module Interface Specifications

6.1 MIS of Browser Module (M2)

6.1.1 Module

Provides the runtime container for the client application. The browser renders markup and style content, maintains the Document Object Model (DOM), collects user input (mouse, keyboard), and provides networking and storage facilities used by the client.

6.1.2 Uses

None (external, platform-provided).

6.1.3 Syntax

Exported Constants None.

Exported Access Programs None.

6.1.4 Semantics

State Variables

- **domTree**: the in-memory Document Object Model for the current page.
- **cookieStore**: key–value cookie storage scoped to the site.
- **localStorage/sessionStorage**: Web Storage used by the UI for small client-side state.
- **cache**: HTTP/resource cache managed by the browser.

State Invariant

- **domTree** is a well-formed DOM representation for the currently loaded page.
- **cookieStore**, **localStorage**, and **sessionStorage** map string keys to string values.

Environment Variables

- **screen**: the display device used to render the client interface.
- **inputDevices**: user input sources (mouse, keyboard, touch) that generate events.
- **net**: the network interface used to send and receive requests to external services.
- **clock**: a system clock used for timeouts, caching, and session expiry.

Assumptions

- A modern standards-compliant browser (Chromium/Chrome, Firefox, Safari).
- Supports HTML5, ECMAScript 2020+, Fetch API, CORS, and Web Storage.
- The system is served over HTTPS and external services accept and return serialized request/response data.

Access Routine Semantics Not applicable (no routines exported by this module in our system).

Local Functions

- *renderPage*(): Updates the Document Object Model in response to application state changes.
- *handleUserInput*(e): Captures and dispatches user input events (mouse, keyboard) to the front-end logic.
- *sendRequest*(req): Issues network requests to backend services using the browser networking stack.
- *updateStorage*(k, v): Reads from and writes to localStorage or sessionStorage as required by the UI.

6.2 MIS of Application UI Module (M3)

6.2.1 Module

Provides the client application shell (routing, layout, and common UI state) that renders pages and components of the DomainX system and connects user interactions to the rest of the system through the defined module interfaces.

6.2.2 Uses

Browser Module (M2) (6.1)
System API Gateway Module (M10) (6.11)
Graphing Module (M12) (6.13)
File Import Module (M13) (6.14)
File Export Module (M14) (6.15)

6.2.3 Syntax

Exported Constants

- **APP_TITLE** = "DomainX"
- **DEFAULT_ROUTE** = "/domains"
- **TOAST_DURATION_MS** = 4000

Exported Access Programs

Name	In	Out	Exceptions
initAppShell	none	mountedRoot	RenderError
navigateTo	routeID, [params]	renderedView	RouteNotFound
renderComponent	componentID, props	renderedFragment	RenderError
handleUserAction	actionType, payload	dispatchResult	ActionError
showNotification	message, severity	successFlag	None

6.2.4 Semantics

State Variables

- **currentRoute**: the active route (path and params).
- **uiState**: global UI model (loading flags, toasts, modal state).
- **sessionContext**: authenticated user + role snapshot for conditional rendering.

State Invariant

- **currentRoute** is a valid route identifier supported by the UI.
- **uiState** is well-formed (e.g., toast queue length ≥ 0, modal state is consistent).
- **sessionContext** is either `None` (unauthenticated) or contains a valid user identifier and role snapshot.

Environment Variables

- **screen**: display surface used to render the user interface (provided via M2).
- **inputDevices**: user input sources (mouse, keyboard, touch) that generate UI events (via M2).
- **net**: network interface used to send and receive requests to external services (via M2).
- **clock**: system clock used for timeouts and session timing (via M2).

Assumptions

- Executed within the Browser Module (M2), which provides DOM rendering, event delivery, storage, and networking facilities.
- Network I/O occurs exclusively through the System API Gateway Module (M10); backend endpoints are not embedded in the UI.
- Graphs are rendered via M12; file workflows are delegated to M13/M14.

Access Routine Semantics **initAppShell**():

- transition: initializes the UI shell, registers routing, and initializes global UI/session state.
- output: `mountedRoot`.
- exception: `RenderError` if bootstrap fails.

navigateTo(routeID, [params]):

- transition: updates `currentRoute`; triggers route component render.
- output: `renderedView`.
- exception: `RouteNotFound` if routeID unknown.

renderComponent(componentID, props):

- transition: none (pure render of a component into the view tree).
- output: `renderedFragment`.
- exception: `RenderError` if component fails to render.

handleUserAction(actionType, payload):

- transition: may update `uiState`; may delegate to M10/M12/M13/M14/M15 via callbacks.
- output: `dispatchResult` (ack/nack).
- exception: `ActionError` on invalid action/payload.

showNotification(message, severity):

- transition: enqueues a toast into `uiState` for `TOAST_DURATION_MS`.
- output: `successFlag`.
- exception: none.

Local Functions

- `guardByRole()` — checks `sessionContext` to gate routes/components.
- `bindGraphHandlers()` — wires chart events (M12) to UI callbacks.
- `confirmBeforeLeave()` — prompts on unsaved changes before navigation.

6.3 MIS of Data Edit Module (M4)

6.3.1 Module

Handles user-initiated edits to domain, package, and metric data. Provides client-side validation and coordinates save/delete operations through the System API Gateway.

6.3.2 Uses

Application UI Module (M3) (6.2)
User Authentication Module (M5) (6.4)
User Role Access Module (M6) (6.5)
System API Gateway Module (M10) (6.11)

6.3.3 Syntax

Exported Constants

- **MAX_EDIT_BATCH** = 50
- **UNDO_STACK_LIMIT** = 10

Exported Access Programs

Name	In	Out	Exceptions
addMetric	metricData	successFlag	SchemaError
editMetric	metricID, newValue	successFlag	ValidationError
deleteMetric	metricID	successFlag	NotFoundError
undoLastEdit	–	successFlag	StackEmptyError
validateEdit	candidateChange	bool	ValidationError
commitBatch	changeList	successFlag	NetworkError

6.3.4 Semantics

State Variables

- **editBuffer**: pending changes not yet committed.
- **undoStack**: most recent operations up to `UNDO_STACK_LIMIT`.

Environment Variables

- **inputDevices**: user input sources (mouse, keyboard, touch) that generate edit events.
- **screen**: the display surface used to render edit forms and validation feedback.
- **net**: network interface used to send requests to the backend via M10 (HTTPS/JSON).
- **clock**: system clock used for timeouts and request expiry (if applicable).

Assumptions

- The user is authenticated (M5) and authorized (M6) for the requested action.
- Executed within a browser-based React UI; form state and feedback are managed by M3.
- Network I/O is performed through M10 over HTTPS using JSON payloads conforming to M10 schemas.
- Persistent storage of edits is performed by M17 *via* M10 (M4 does not access M17 directly).

Access Routine Semantics **addMetric**(metricData):

- transition: validate `metricData`; stage in `editBuffer`; on commit, send create request via M10 to M17.
- output: `successFlag` = true if accepted by backend.
- exception: `SchemaError` if required fields are missing or malformed.

editMetric(metricID, newValue):

- transition: stage field updates in `editBuffer`; mark record dirty.
- output: `successFlag` = true on commit acknowledgement.
- exception: `ValidationError` if new value violates schema or business rules.

deleteMetric(metricID):

- transition: stage delete action in `editBuffer`.
- output: `successFlag` = true if deletion is confirmed by M17 (via M10).
- exception: `NotFoundError` if the metric does not exist.

undoLastEdit():

- transition: revert the most recent staged or committed change if reversible; update `undoStack`.
- output: `successFlag` = true if an action was undone.
- exception: `StackEmptyError` if there is nothing to undo.

validateEdit(candidateChange):

- transition: none (local validation only).
- output: Boolean indicating whether `candidateChange` satisfies client-side checks.
- exception: `ValidationError` if checks fail.

commitBatch(changeList):

- transition: send up to `MAX_EDIT_BATCH` staged changes via M10; clear successful items from `editBuffer`; push entries to `undoStack`.
- output: `successFlag` = true if all changes are acknowledged by M17.
- exception: `NetworkError` if API calls fail or time out.

Local Functions

- `coerceTypes()` — cast user input to expected types.
- `diffRecords()` — compute minimal patch for update requests.
- `throttleSaves()` — coalesce frequent edits before commit.

6.4 MIS of User Authentication Module (M5)

6.4.1 Module

Provides user authentication and session management services for the system. This module validates credentials, issues and invalidates authentication tokens, and tracks active user sessions through a well-defined interface exposed to other modules.

6.4.2 Uses

System API Gateway Module (M10) (6.11)
User Role Access Module (M6) (6.5)
Database Persistence Module (M17) (6.17)

6.4.3 Syntax

Exported Constants

- **TOKEN_EXPIRY_HOURS** = 24
- **MAX_LOGIN_ATTEMPTS** = 5
- **PASSWORD_MIN_LENGTH** = 8

Exported Access Programs

Name	In	Out	Exceptions
sendInvite	email, role	inviteCode	AuthError
registerUser	email, password, inviteCode	authToken	ValidationError
loginUser	email, password	authToken	AuthError
logoutUser	authToken	successFlag	SessionError
validateSession	authToken	bool	TokenError
refreshToken	oldToken	newToken	TokenExpiredError

6.4.4 Semantics

State Variables

- **activeSessions**: map of valid tokens to user IDs and expiry timestamps.
- **failedAttempts**: counter tracking consecutive failed login attempts.

State Invariant

- Every entry in **activeSessions** maps a valid authentication token to exactly one user identifier and an expiry timestamp.
- **failedAttempts** is a non-negative integer.

Environment Variables

- **net**: network interface used to receive authentication requests and return responses via M10.
- **clock**: system clock used to evaluate token expiry and session timeouts.

Assumptions

- Authentication is token-based; issued tokens uniquely identify users and have a finite validity period.
- All credential transmissions occur via HTTPS.
- The UI (M3) performs basic client-side validation before requests are sent.
- Tokens expire after `TOKEN_EXPIRY_HOURS`.

Access Routine Semantics **registerUser**(email, password, inviteCode):

- transition: validate `inviteCode`; if valid, create a new user in M17 with a hashed password and the role associated with the invite.
- output: `authToken` if registration succeeds.
- exception: `ValidationError` if email exists, password fails checks, or `inviteCode` is invalid/expired.

sendInvite(email, role):

- transition: generate an invitation code tied to `email` and `role`; store the invite and send the code to the user.
- output: `inviteCode`.
- exception: `AuthError` if the requester is not authorized to invite users.

loginUser(email, password):

- transition: validate credentials; issue JWT; add entry to `activeSessions`.
- output: `authToken`.
- exception: `AuthError` if credentials are invalid or account is locked.

logoutUser(authToken):

- transition: invalidate token; remove from `activeSessions`.

- output: `successFlag` = true if successful.
- exception: `SessionError` if session not found.

validateSession(authToken):

- transition: none.
- output: Boolean indicating if token is valid and unexpired.
- exception: `TokenError` if malformed or tampered.

refreshToken(oldToken):

- transition: invalidate old token; issue new JWT with reset expiry.
- output: `newToken`.
- exception: `TokenExpiredError` if token already expired.

Local Functions

- `hashPassword()` – securely hashes user credentials.
- `verifyPassword()` – verifies submitted credentials against stored hashes.
- `generateToken()` – issues a signed authentication token.
- `rateLimitLogin()` – blocks login after `MAX_LOGIN_ATTEMPTS`.

6.5 MIS of User Role Access Module (M6)

6.5.1 Module

Defines, stores, and enforces role-based permissions across the system. This module determines whether an authenticated user is allowed to perform a requested action and provides role/permission lookup and update operations to authorized administrators.

6.5.2 Uses

User Authentication Module (M5) (6.4)
System API Gateway Module (M10) (6.11)
Database Persistence Module (M17) (6.17)

6.5.3 Syntax

Exported Constants

- **DEFAULT_ROLE** = "Viewer"
- **ROLE_HIERARCHY** = ["Viewer", "Editor", "Admin"]
- **PERMISSIONS_CACHE_TTL** = 300 *(seconds; cache lifetime for permission lookups)*

Exported Access Programs

Name	In	Out	Exceptions
assignRole	userID, roleName	successFlag	InvalidRoleError
getUserRole	userID	roleName	NotFoundError
verifyAccess	userID, actionType	bool	PermissionError
updateRolePermissions	roleName, newPermissions	successFlag	ConfigError
listAllRoles	–	roleList	None

6.5.4 Semantics

State Variables

- **rolesTable**: mapping of role names to permission sets (read, write, delete, admin).
- **userRoleMap**: dictionary mapping user IDs to assigned roles.
- **permissionsCache**: temporary in-memory store of validated permissions for current session.

State Invariant

- Every entry in **userRoleMap** maps a valid `userID` to a role name that appears in **ROLE_HIERARCHY**.
- **rolesTable** contains an entry for each role in **ROLE_HIERARCHY**.
- **permissionsCache** only stores permissions consistent with **rolesTable** at the time they were cached.

Environment Variables

- **net**: network interface used to request and return authorization decisions via M10.
- **clock**: system clock used to evaluate cache expiration for `PERMISSIONS_CACHE_TTL`.

Assumptions

- All users are authenticated before authorization checks are performed (via M5).
- Each `actionType` requested by other modules is mapped to a permission category understood by this module.
- Role and permission data is stored persistently via M17 and accessed through M10.

Access Routine Semantics **assignRole**(userID, roleName):

- transition: updates `userRoleMap` and backend table with new role.
- output: `successFlag` = true if update is stored.
- exception: `InvalidRoleError` if `roleName` not found in `ROLE_HIERARCHY`.

getUserRole(userID):

- transition: none.
- output: the user's current role as a string.
- exception: `NotFoundError` if user not found in table.

verifyAccess(userID, actionType):

- transition: none.
- output: Boolean — true if role's permissions include the requested `actionType`.
- exception: `PermissionError` if unauthorized.

updateRolePermissions(roleName, newPermissions):

- transition: modifies `rolesTable` for the given role and syncs to backend.
- output: `successFlag` = true if successfully updated.
- exception: `ConfigError` if update fails or backend schema mismatch.

listAllRoles():

- transition: none.
- output: list of available roles and descriptions.
- exception: none.

Local Functions

- `hasPermission(userID, action)` – returns true if user's role allows the action.
- `invalidateCache()` – clears cached permissions on role update or logout.
- `syncWithBackend()` – persists role/permission updates via M10.

6.6 MIS of User Page Module (M7)

6.6.1 Module

Implements the interactive user settings page in the DomainX web interface. This module displays and updates user account details (e.g., display name, email, password, notification preferences) by issuing requests through the System API Gateway (M10).

6.6.2 Uses

User Authentication Module (M5) (6.4)
User Role Access Module (M6) (6.5)
System API Gateway Module (M10) (6.11)
Application UI Module (M3) (6.2)

6.6.3 Syntax

Exported Constants

- **MAX_DISPLAY_NAME_LEN** = 50
- **SESSION_REFRESH_INTERVAL** = 15 *(minutes)*

Exported Access Programs

Name	In	Out	Exceptions
fetchUserProfile	authToken	userProfile	AuthError
updateUserProfile	authToken, updatedData	successFlag	ValidationError
changePassword	authToken, oldPwd, newPwd	successFlag	PasswordError
toggleNotifications	authToken, preferenceFlag	successFlag	ConfigError
deleteAccount	authToken	successFlag	PermissionError

6.6.4 Semantics

State Variables

- **userProfile**: object storing current session's user info (name, email, role, preferences).
- **localCache**: stores recently loaded profile data to reduce repeated API calls.

State Invariant

- If **localCache** contains profile data, it corresponds to the same authenticated user identified by the current `authToken`.
- **userProfile** fields respect module constraints (e.g., $|\texttt{displayName}| \leq \texttt{MAX\_DISPLAY\_NAME\_LEN}$).

Environment Variables

- **net**: network interface used to send profile update requests via M10.
- **clock**: system clock used to schedule session refresh based on `SESSION_REFRESH_INTERVAL`.

Assumptions

- User is authenticated through M5 before accessing the page.
- Profile updates require valid session tokens for backend verification.
- Password strength and validation follow rules enforced by M5.

Access Routine Semantics **fetchUserProfile**(authToken):

- transition: none.
- output: returns `userProfile` containing user's data.
- exception: `AuthError` if token invalid or expired.

updateUserProfile(authToken, updatedData):

- transition: updates local and backend profile data via M10.
- output: `successFlag` = true on confirmation.
- exception: `ValidationError` if new data violates constraints.

changePassword(authToken, oldPwd, newPwd):

- transition: validates `oldPwd`; submits password update request via M10; refreshes `userProfile` on success.

- output: `successFlag` = true on success.
- exception: `PasswordError` if old password incorrect or new one too weak.

toggleNotifications(authToken, preferenceFlag):

- transition: modifies user's notification preference and syncs to backend.
- output: `successFlag` = true if preference saved.
- exception: `ConfigError` if backend update fails.

deleteAccount(authToken):

- transition: issues delete request via M10; user data removed from backend.
- output: `successFlag` = true on completion.
- exception: `PermissionError` if user lacks required authorization.

Local Functions

- `cacheProfileData()` – stores profile data locally for faster re-rendering.
- `validateEmailFormat()` – ensures valid email syntax before submission.
- `displayConfirmation()` – shows confirmation prompts for irreversible actions.

6.7 MIS of Automated Metrics Module (M8)

6.7.1 Module

Implements the logic for automated retrieval, calculation, and storage of quantitative metrics associated with repositories or packages. This module interacts with external APIs through M15 to gather data such as stars, forks, downloads, or commit frequency and stores them in the internal database for further ranking and analysis.

6.7.2 Uses

Repository API Module (M15) (6.16)
System API Gateway Module (M10) (6.11)
Ranking Algorithm Module (M11) (6.12)

6.7.3 Syntax

Exported Constants

- **UPDATE_INTERVAL_HOURS** = 24
- **MAX_RETRIES** = 3
- **SUPPORTED_METRICS** =
- **SUPPORTED_METRICS** =
  - **GitHub API:** {stars, forks, watchers, openPullRequests, closedPullRequests}
  - **Git statistics (git_stats):** {textFileCount, binaryFileCount, textLineCount, totalLinesAdded, totalLinesDeleted, totalCommits, commitsByYearLastFive, commitsByMonth}
  - **Source code analysis (scc):** {textFileCount, textLineCount, codeLineCount, commentLineCount, blankLineCount}

Exported Access Programs

Name	In	Out	Exceptions
fetchMetricsFromRepo	packageID	metricsData	APIError
validateMetrics	metricsData	bool	ValidationError
storeValidatedMetrics	metricsData	successFlag	WriteError
enqueueMetricsFetch	packageID	successFlag	QueueError
processMetricsQueue	None	successFlag	WorkerError

6.7.4 Semantics

State Variables

- **metricsQueue**: list of repositories scheduled for update.
- **lastUpdateTime**: timestamp of the last successful metric refresh.

State Invariant

- `lastUpdateTime` is either `None` or a valid ISO 8601 timestamp.
- Each entry in `metricsQueue` is a valid `packageID`.

Environment Variables

- **net**: network interface used to contact external APIs (via M15) and submit stored metric results (via M10).
- **clock**: system clock used to trigger periodic refresh every `UPDATE_INTERVAL_HOURS`.

Assumptions

- API rate limits are respected by using cached data or delay-based retries.
- Network failures are retried up to `MAX_RETRIES`.
- Fetched data conforms to a consistent JSON schema before validation.

Access Routine Semantics **fetchMetricsFromRepo**(packageID):

- transition: sends a GET request via M15 to external repository API.
- output: `metricsData` — structured dictionary of retrieved metrics.
- exception: `APIError` if API call fails or response malformed.

validateMetrics(metricsData):

- transition: none.
- output: Boolean value indicating if all required metrics are valid.
- exception: `ValidationError` if missing or inconsistent fields.

storeValidatedMetrics(metricsData):

- transition:
- transition: submits validated metric records to persistent storage via M10.
- output: `successFlag` = true if stored successfully.
- exception: `WriteError` if database transaction fails.

enqueueMetricsFetch(packageID):

- transition: add `packageID` to `metricsQueue` for deferred processing.
- output: `successFlag` = true if enqueue succeeds.
- exception: `QueueError` if the request cannot be queued.

processMetricsQueue():

- transition: dequeue pending repositories and fetch supported metrics via M15; validate and store results via M10.
- output: `successFlag` = true if all queued jobs complete.
- exception: `WorkerError` if processing fails or is interrupted.

Local Functions

- `normalizeMetricValues()` — adjusts different metrics to comparable scales before storage.
- `retryFailedRequests()` — retries failed fetch operations respecting delay intervals.
- `updateLastRunTimestamp()` — records timestamp of most recent completed update.

6.8 MIS of Domains Page Module (M9)

6.8.1 Module

Implements the logic for presenting and interacting with domain-related data. This module retrieves, organizes, and displays domains, their associated packages, metrics, and descriptions, and supports user-driven exploration and filtering through the defined system interfaces.

6.8.2 Uses

Configuration Module (M19) (6.10)
System API Gateway Module (M10) (6.11)
Graphing Module (M12) (6.13)
Ranking Algorithm Module (M11) (6.12)

6.8.3 Syntax

Exported Constants

- **DEFAULT_SORT_ORDER** = “alphabetical“
- **MAX_DISPLAYED_DOMAINS** = 50
- **DEFAULT_METRIC_SET** = [“stars“, “forks“, “downloads“]

Exported Access Programs

Name	In	Out	Exceptions
fetchAllDomains	filterOptions	domainList	DatabaseError
displayDomainDetails	domainID	renderedView	DataNotFoundError
renderDomainMetrics	domainID, metricType	chartObject	RenderError
searchDomains	searchTerm	filteredList	None

6.8.4 Semantics

State Variables

- **domainCache**: dictionary storing the most recently viewed domains for quick access.
- **activeFilters**: list of active search or metric filters applied by the user.

State Invariant

- Every entry in **domainCache** corresponds to a valid domain identifier.
- **activeFilters** only contains filter criteria supported by this module.

Environment Variables

- **screen**: display surface used to present domain lists and details.
- **inputDevices**: user input sources (mouse, keyboard, touch) used for navigation and filtering.
- **net**: network interface used to retrieve domain data and metrics via M10.

Assumptions

- Domain and metric data is available through the System API Gateway (M10).
- Users have appropriate permissions (via M6) to view domain data.
- Visualization requests are delegated to the Graphing Module (M12).

Access Routine Semantics **fetchAllDomains**(filterOptions):

- transition: none.
- output: list of domain objects matching `filterOptions`.
- exception: `DataAccessError` if domain data cannot be retrieved.

displayDomainDetails(domainID):

- transition: retrieves the selected domain's details and updates the active view.
- output: rendered domain view.
- exception: `DataNotFoundError` if the domain ID does not exist.

renderDomainMetrics(domainID, metricType):

- transition: retrieves stored metric data for the domain via M10 and delegates visualization to M12.
- output: visualization object.
- exception: `RenderError` if visualization fails.

searchDomains(searchTerm):

- transition: updates the domain list view with matching results.
- output: filtered domain list.
- exception: none.

Local Functions

- `applyFilters()` — applies active filters to retrieved domain list.
- `updateCache()` — refreshes domain cache with recently accessed data.
- `formatMetricData()` — adjusts metric values for standardized display.

6.9 MIS of Comparison Module (M16)

6.9.1 Module

Implements the logic that defines and manages comparison methods between software packages or domains. This module allows users to compare entities based on quantitative metrics (e.g., stars, forks, commits, downloads) and qualitative attributes (e.g., documentation quality, update frequency). It serves as a bridge between the user interface, ranking algorithm, and metrics data to compute and display meaningful comparisons.

6.9.2 Uses

Ranking Algorithm Module (M11) (6.12)
Graphing Module (M12) (6.13)
System API Gateway Module (M10) (6.11)

6.9.3 Syntax

Exported Constants

- **DEFAULT_METHOD** = “AHP”
- **SUPPORTED_METHODS** = [“AHP“, “BWM“, “SSB“]
- **DEFAULT_VISUALIZATION** = “barChart“

Exported Access Programs

Name	In	Out	Exceptions
selectComparisonMethod	methodName	successFlag	InvalidMethodError
computeComparison	domainSet, metricWeights	comparisonTable	ComputationError
fetchComparisonData	domainSet	metricsData	DatabaseError
visualizeComparison	comparisonTable, chartType	chartObject	RenderError

6.9.4 Semantics

State Variables

- **activeMethod**: string representing the currently selected comparison method.
- **comparisonResults**: cached dictionary holding the most recent comparison output.

State Invariant

- **activeMethod** $\in$ `SUPPORTED_METHODS`.
- If **comparisonResults** is non-empty, it was produced using **activeMethod**.

Environment Variables

- **screen**: display surface used to present comparison tables and charts.
- **inputDevices**: user input sources used to select domains/methods and interact with results.
- **net**: network interface used to request comparison data via M10.

Assumptions

- Required domain/metric data for comparisons is accessible through M10.
- The user selects a valid method from `SUPPORTED_METHODS`.
- Visualization requests are delegated to M12.

Access Routine Semantics **selectComparisonMethod**(methodName):

- transition: updates `activeMethod`.
- output: `successFlag` = true if method is supported.
- exception: `InvalidMethodError` if method not in `SUPPORTED_METHODS`.

computeComparison(domainSet, metricWeights):

- transition: requests required metric data via M10 and computes comparison results using the selected method; may delegate scoring/weighting to M11.
- output: `comparisonTable` containing ranked and weighted scores.
- exception: `ComputationError` if calculation fails or data incomplete.

fetchComparisonData(domainSet):

- transition: none.
- output: retrieves the metric data required to compare all entities in `domainSet`.
- exception: `DataAccessError` if data cannot be retrieved through M10.

visualizeComparison(comparisonTable, chartType):

- transition: renders visual representation of results via M12.
- output: `chartObject` representing the visualization of `comparisonTable`.
- exception: `RenderError` if graph generation fails.

Local Functions

- `normalizeWeights()` — ensures weights sum to 1 before calculation.
- `computeAHPMatrix()` — performs pairwise comparisons for AHP method.
- `applyBWM()` — computes ranking scores using Best-Worst Method.

6.10 MIS of Configuration Module (M19)

6.10.1 Module

Implements the logic for storing, retrieving, and updating configuration data associated with each authenticated user. This includes user preferences such as selected visualization settings, default domain filters, notification preferences, and saved comparison parameters. It interacts with M5 (User Authentication) to identify the active user and with M17 (Database Persistence Module) for persistent storage of configuration data.

6.10.2 Uses

User Authentication Module (M5) (6.4)
Database Persistence Module (M17) (6.17)
System API Gateway Module (M10) (6.11)

6.10.3 Syntax

Exported Constants

- **DEFAULT_LANGUAGE** = "en"
- **DEFAULT_THEME** = "light"
- **DEFAULT_NOTIFICATION_SETTINGS** = {email: true, inApp: true}

Exported Access Programs

Name	In	Out	Exceptions
getUserConfig	userID	configData	DataNotFoundError
updateUserConfig	userID, configData	successFlag	WriteError
resetToDefaults	userID	configData	None
fetchAllConfigs	None	configList	DatabaseError

6.10.4 Semantics

State Variables

- **userConfigs**: dictionary mapping each userID to their saved configuration data.
- **defaultSettings**: dictionary storing global default configuration values.

State Invariant

- For every **userID** in **userConfigs**, **userConfigs(userID)** conforms to the configuration schema.
- **defaultSettings** conforms to the same schema.

Environment Variables None. This module does not interact with externally supplied environment configuration values.

Assumptions

- Each userID exists and is validated through M5 before configuration access.
- Configuration data adheres to a predefined JSON schema.
- Default values are stored within this module and initialized at system startup.

Access Routine Semantics **getUserConfig**(userID):

- transition: none.
- output: returns configuration settings for the given user.
- exception: `DataNotFoundError` if user has no stored configuration.

updateUserConfig(userID, configData):

- transition: updates stored configuration for the given user in the database.
- output: `successFlag` = true if update is successful.
- exception: `WriteError` if database transaction fails.

resetToDefaults(userID):

- transition: resets all stored configuration to default values.
- output: `configData` reflecting the defaults.
- exception: none.

fetchAllConfigs():

- transition: none.
- output: list of all stored configuration records for administrative view.
- exception: `DatabaseError` if persistence retrieval fails.

Local Functions

- `validateConfigSchema()` — ensures submitted configuration matches the JSON schema.
- `mergeDefaults()` — merges user-provided configuration with stored defaults.
- `logConfigChange()` — records modification timestamps for audit purposes.

6.11 MIS of System API Gateway Module (M10)

6.11.1 Module

Implements the backend REST API layer that governs communication between the frontend and the server application, and mediates access to persisted data via M17. This module manages request handling, validation, and serialization across all feature modules. It ensures that frontend requests are authenticated, validated, and dispatched to the correct internal service routines.

6.11.2 Uses

Database Persistence Module (M17) (6.17)
User Authentication Module (M5) (6.4)
Automated Metrics Module (M8) (6.7)
Repository API Module (M15) (6.16)
Configuration Module (M19) (6.10)
User Role Access Module (M6) (6.5)

6.11.3 Syntax

Exported Constants

- **API_VERSION** = "v1"
- **DEFAULT_CONTENT_TYPE** = "application/json"
- **REQUEST_TIMEOUT_SEC** = 30
- **MAX_CONNECTIONS** = 100

Exported Access Programs

Name	In	Out	Exceptions
handleHTTPRequest	request	response	NetworkError
authenticate	request	userContext	AuthError
authorize	userContext, action	bool	PermissionError
validatePayload	endpoint, payload	bool	ValidationError
dispatchToModule	endpoint, payload, userContext	serviceOutput	RouteError
buildResponse	serviceOutput	response	SerializeError

6.11.4 Semantics

State Variables

- **routeTable**: mapping of API endpoints to handler routines.
- **middlewareChain**: ordered list of middleware applied to each request (auth, validation, logging).

State Invariant

- Every endpoint in **routeTable** maps to exactly one handler routine.
- All responses returned by **handleHTTPRequest** are JSON-serializable.

Environment Variables

- **DJANGO_SECRET_KEY**: secret used to sign cookies/tokens.
- **ALLOWED_HOSTS**: permitted hostnames for HTTP requests.
- **CORS_ALLOWED_ORIGINS**: permitted frontend origins for cross-origin requests.

Assumptions

- All endpoints conform to REST conventions (GET, POST, PUT, DELETE).
- Network connections are encrypted via HTTPS.
- Frontend tokens are validated by M5 (User Authentication Module).
- The backend is executed using standard server entry points (development runner and WSGI/ASGI deployment interfaces).

Access Routine Semantics **handleHTTPRequest**(request):

- transition: receives an incoming HTTP request; invokes `authenticate()` and `authorize()`; validates the request payload; dispatches the request to the appropriate service routine; serializes the service result into an HTTP response.
- output: `response` containing status code and JSON body.
- exception: `NetworkError` if request handling fails or the request cannot be processed.

authenticate(request):

- transition: extracts credentials (e.g., session/JWT token) from `request` and validates them via M5.

- output: `userContext` (authenticated user identity and session info).
- exception: `AuthError` if credentials are invalid, expired, or missing.

authorize(userContext, action):

- transition: checks whether `userContext` is permitted to perform `action` using M6.
- output: Boolean indicating whether access is permitted.
- exception: `PermissionError` if the user is not authorized for the requested action.

validatePayload(endpoint, payload):

- transition: validates `payload` against the expected schema for `endpoint`.
- output: Boolean indicating whether the payload is valid.
- exception: `ValidationError` if required fields are missing or malformed.

dispatchToModule(endpoint, payload, userContext):

- transition: maps `endpoint` to the appropriate handler routine and invokes the corresponding module/service using `payload` and `userContext`.
- output: `serviceOutput` returned by the destination module.
- exception: `RouteError` if `endpoint` is not mapped to any handler.

buildResponse(serviceOutput):

- transition: serializes `serviceOutput` into a JSON response body and attaches the appropriate HTTP status code.
- output: `response` containing serialized output.
- exception: `SerializeError` if response construction fails.

Local Functions

- `serializeResponse()` — converts internal service output into a transferable response format.
- `validateRequest()` — checks incoming request data against the expected schema for the target endpoint.
- `applyMiddleware()` — applies common gateway processing steps (authentication, authorization, logging).
- `logRequest()` — records request and response metadata for monitoring and auditing.

6.12 MIS of Ranking Algorithm Module (M11)

6.12.1 Module

Implements ranking algorithms that compute relative scores for software packages or domains based on weighted criteria. The module supports pairwise-comparison methods such as Analytic Hierarchy Process (AHP) and is extensible to additional ranking models (e.g., BWM, SSB). It produces normalized ranking results suitable for downstream comparison and visualization.

6.12.2 Uses

Comparison Module (M16) (6.9)
Configuration Module (M19) (6.10)

6.12.3 Syntax

Exported Constants

- **DEFAULT_METHOD** = "AHP"
- **CONSISTENCY_THRESHOLD** = 0.1
- **SUPPORTED_METHODS** = ["AHP", "BWM", "SSB"]

Exported Access Programs

Name	In	Out	Exceptions
computeRankScores	metricsTable, userWeights	rankTable	InvalidMetricError
applyAHPWeights	criteriaMatrix	weightedMatrix	MatrixError
normalizeScores	rankTable	normalizedTable	NormalizationError
getTopPackages	normalizedTable, limit	rankedList	None

6.12.4 Semantics

State Variables

- **criteriaMatrix**: matrix of pairwise criteria comparisons.
- **normalizedScores**: vector of final ranking results.

State Invariant

- **criteriaMatrix**, if defined, is square and has strictly positive entries.
- **normalizedScores**, if defined, sums to 1.

Environment Variables None. This module does not interact with externally supplied environment configuration values.

Assumptions

- Criteria weights are positive and consistent within the threshold.
- All metric inputs are validated prior to ranking.
- The consistency ratio for pairwise comparisons does not exceed `CONSISTENCY_THRESHOLD`.

Access Routine Semantics **computeRankScores**(metricsTable, userWeights):

- transition: constructs a weighted decision matrix using the selected ranking method.
- output: `rankTable` — ranked list of packages or domains.
- exception: `InvalidMetricError` if metrics missing or inconsistent.

applyAHPWeights(criteriaMatrix):

- transition: applies pairwise weights to compute priority vectors.
- output: `weightedMatrix`.
- exception: `MatrixError` if dimensions invalid or inconsistent.

normalizeScores(rankTable):

- transition: rescales scores so total weight = 1.
- output: normalized score table.
- exception: `NormalizationError` if division by 0 or invalid data.

getTopPackages(normalizedTable, limit):

- transition: none.
- output: top-n ranked packages.
- exception: none.

Local Functions

- `checkConsistency()` — calculates consistency ratio for AHP matrices.
- `aggregateWeights()` — merges user-defined and default weight sets.
- `normalizeVector()` — scales weight vectors for comparison output.

6.13 MIS of Graphing Module (M12)

6.13.1 Module

Implements visualization operations for presenting metric data and comparison results. This module generates graphical representations (e.g., bar, line, pie, or radar charts) from structured input data and produces outputs suitable for display or export.

6.13.2 Uses

System API Gateway Module (M10) (6.11)
Comparison Module (M16) (6.9)
Configuration Module (M19) (6.10)

6.13.3 Syntax

Exported Constants

- **DEFAULT_STYLE** = "standard"
- **DEFAULT_FIGSIZE** = (width, height)
- **SUPPORTED_CHARTS** = ["bar", "line", "pie", "radar"]
- **SUPPORTED_EXPORT_FORMATS** = ["png", "svg", "pdf"]

Exported Access Programs

Name	In	Out	Exceptions
generateGraph	metricData, graphType, config	graphImage	GraphError
updateGraphStyle	styleParams	successFlag	StyleError
exportGraph	graphImage, format, fileName	exportPath	ExportError
renderComparisonPlot	domainMetrics, metricType	comparisonImage	DataError

6.13.4 Semantics

State Variables

- **currentGraph**: last generated Matplotlib figure object.
- **styleConfig**: dictionary storing current theme and formatting options.

State Invariant

- If **currentGraph** is defined, it corresponds to the most recent successful graph generation request.
- **styleConfig** contains only supported visualization parameters.

Environment Variables

- **screen**: display surface used to present generated visualizations.
- **fs**: file system used to store exported graph images.
- **net**: network interface used to return visualization outputs via M10.

Assumptions

- Input datasets contain numeric or categorical values compatible with Matplotlib.
- Configuration settings comply with Matplotlib formatting constraints.
- Graph generation occurs on the backend before transfer to the frontend for rendering.

Access Routine Semantics **generateGraph**(metricData, graphType, config):

- transition: generates a visualization according to `graphType` and `config`.
- output: `graphImage` representing the rendered visualization.
- exception: `GraphError` if data invalid or plotting fails.

updateGraphStyle(styleParams):

- transition: updates `styleConfig` (color scheme, font, size).
- output: `successFlag` = true if update applied.
- exception: `StyleError` if parameters are unsupported.

exportGraph(graphImage, format, fileName):

- transition: writes `graphImage` to persistent storage in the requested format.
- output: `exportPath` of saved image.
- exception: `ExportError` if write operation fails.

renderComparisonPlot(domainMetrics, metricType):

- transition: overlays multiple data series for comparative visualization.
- output: `comparisonImage` — rendered comparative plot.
- exception: `DataError` if inconsistent metric arrays are provided.

Local Functions

- `normalizeData()` — ensures all metric values are scaled before plotting.
- `applyTheme()` — applies style configuration globally.
- `cleanTemporaryArtifacts()` — removes temporary visualization artifacts after export.

6.14 MIS of File Import Module (M13)

6.14.1 Module

Handles ingestion of user-provided data files and converts them into structured datasets for internal use. This module parses, validates, and transforms CSV or spreadsheet files into records conforming to the system's domain and metric schemas, and forwards validated data for persistence through the backend API.

6.14.2 Uses

System API Gateway Module (M10) (6.11)
Configuration Module (M19) (6.10)

6.14.3 Syntax

Exported Constants

- **SUPPORTED_FORMATS** = [“csv“, “xlsx“]
- **MAX_FILE_SIZE_MB** = 10
- **DEFAULT_ENCODING** = “utf-8“
- **REQUIRED_FIELDS** = [“DomainName“, “PackageName“, “MetricName“, “Value“]

Exported Access Programs

Name	In	Out	Exceptions
parseFile	filePath, fileType	dataFrame	ParseError
validateImportedData	dataFrame	bool	ValidationError
storeImportedData	dataFrame	successFlag	WriteError
getImportSummary	dataFrame	summaryDict	None

6.14.4 Semantics

State Variables

- **lastImportedFile**: path or identifier of the last processed file.
- **importStatus**: status flag indicating the most recent operation result.

State Invariant

- **importStatus** reflects the outcome of the most recent import attempt.
- If **lastImportedFile** is defined, it refers to the most recently processed input.

Environment Variables

- **fs**: file system interface used to read uploaded files.
- **encoding**: externally determined character encoding for text files.

Assumptions

- Uploaded files conform to the system's schema and naming conventions.
- File size and encoding are supported and within defined constraints.
- Invalid rows are filtered out and logged, not silently dropped.

Access Routine Semantics **parseFile**(filePath, fileType):

- transition: reads the input file and transforms it into a structured dataset representation.
- output: `dataFrame` containing structured data.
- exception: `ParseError` if the file cannot be opened or parsed.

validateImportedData(dataFrame):

- transition: checks for required fields and data-type consistency.
- output: Boolean indicating whether validation succeeded.
- exception: `ValidationError` if mandatory columns are missing or invalid types detected.

storeImportedData(dataFrame):

- transition: submits validated records to the backend via M10 for persistence.
- output: `successFlag` = true if the backend acknowledges storage.
- exception: `WriteError` if persistence fails or is rejected.

getImportSummary(dataFrame):

- transition: none.
- output: `summaryDict` — total rows processed, valid, invalid, and stored.
- exception: none.

Local Functions

- `cleanColumnNames()` — standardizes column names to match system schema.
- `detectDelimiter()` — determines delimiter automatically for CSV files.
- `logImportActivity()` — records metadata of each file import for traceability.

6.15 MIS of File Export Module (M14)

6.15.1 Module

Handles transformation of system data into exportable file formats such as CSV or spreadsheet files. This module gathers domain, metric, and ranking data from the backend services, converts it into standardized export records, and writes files suitable for external analysis tools while preserving schema and encoding.

6.15.2 Uses

System API Gateway Module (M10) (6.11)
Configuration Module (M19) (6.10)
Ranking Algorithm Module (M11) (6.12)

6.15.3 Syntax

Exported Constants

- **SUPPORTED_FORMATS** = [“csv“, “xlsx“]
- **DEFAULT_ENCODING** = “utf-8“
- **MAX_ROWS_PER_FILE** = 50000

Exported Access Programs

Name	In	Out	Exceptions
prepareExportData	queryParams, dataType	exportData	DataError
exportToCSV	exportData, fileName	filePath	WriteError
exportToExcel	exportData, fileName	filePath	WriteError
getExportSummary	filePath	summaryDict	None

6.15.4 Semantics

State Variables

- **exportStatus**: status flag of the latest export operation.
- **lastExportFile**: file name of the most recently generated export.

State Invariant

- If **exportStatus** indicates success, then **lastExportFile** refers to a file created in the most recent export.

Environment Variables

- **fs**: file system interface used to create and write export files.
- **encoding**: externally determined encoding used for text exports.

Assumptions

- Data requested for export is already validated and normalized.
- User export requests specify supported formats only.
- The file system path `EXPORT_DIRECTORY` is writable by the application.

Access Routine Semantics **prepareExportData**(queryParams, dataType):

- transition: requests the required records via M10 using `queryParams` and `dataType`.
- output: `exportData` containing normalized records ready for serialization.
- exception: `DataError` if retrieval fails or returns incomplete data.

exportToCSV(exportData, fileName):

- transition: serializes `exportData` into CSV and writes the file via **fs**.
- output: `filePath` of the generated CSV file.
- exception: `WriteError` if file I/O fails.

exportToExcel(exportData, fileName):

- transition: serializes `exportData` into spreadsheet format and writes the file via **fs**.
- output: `filePath` of the generated spreadsheet file.
- exception: `WriteError` if file I/O fails.

getExportSummary(filePath):

- transition: none.
- output: `summaryDict` — includes file name, record count, and format.
- exception: none.

Local Functions

- `sanitizeFileName()` — ensures export file names are safe and standardized.
- `splitLargeExports()` — divides large datasets into multiple files if row limit exceeded.
- `logExportActivity()` — records export metadata for auditing.

6.16 MIS of Repository API Module (M15)

6.16.1 Module

Provides an interface for interacting with external repository APIs such as GitHub, GitLab, and PyPI. This module retrieves repository metadata and metric data (e.g., stars, forks, issues, commits) required for domain analysis. It handles authentication, rate limiting, and response parsing before returning validated data to internal modules.

6.16.2 Uses

System API Gateway Module (M10) [for backend communication] (6.11)

6.16.3 Syntax

Exported Constants

- **REQUEST_TIMEOUT_SEC** = 30
- **MAX_RETRIES** = 3
- **RATE_LIMIT_THRESHOLD** = 1000

Exported Access Programs

Name	In	Out	Exceptions
setAuthToken	token	successFlag	AuthError
fetchRepoMetadata	packageID	repoData	APIError
fetchRepoMetrics	packageID, metricList	metricsData	APIError
handleRateLimit	responseHeaders	delaySeconds	RateLimitError

6.16.4 Semantics

State Variables

- **authToken**: authentication key for accessing external APIs.
- **lastResponseStatus**: HTTP status code of the most recent request.

State Invariant

- If **authToken** is defined, it is valid for at least one external repository service.
- **lastResponseStatus** reflects the most recent external API interaction.

Environment Variables

- **net**: network interface used to communicate with external repository services.
- **clock**: system clock used to evaluate rate limits and retry delays.

Assumptions

- External repository services expose stable interfaces for retrieving metadata and metrics.
- Rate limiting information is provided by external services and respected by this module.
- Authentication credentials for external services are available to this module.

Access Routine Semantics **setAuthToken**(token):

- transition: sets a new authentication token for subsequent API calls.
- output: `successFlag` = true if token successfully stored.
- exception: `AuthError` if token invalid or missing permissions.

fetchRepoMetadata(packageID):

- transition: requests repository metadata from an external repository service.
- output: `repoData` containing repository metadata attributes.
- exception: `APIError` if request fails or JSON malformed.

fetchRepoMetrics(packageID, metricList):

- transition: requests the specified metrics for the repository from an external service.
- output: `metricsData` containing retrieved metric values.
- exception: `APIError` if any metric endpoint fails or rate-limited.

handleRateLimit(responseHeaders):

- transition: calculates required delay based on rate-limit headers.
- output: `delaySeconds` = integer number of seconds to wait.
- exception: `RateLimitError` if headers missing or malformed.

Local Functions

- `parseResponse()` — converts external service responses into internal data structures.
- `retryRequest()` — retries failed external requests up to `MAX_RETRIES`.
- `logAPICall()` — records external API interactions for monitoring and diagnostics.

6.17 MIS of Database Persistence Module (M17)

6.17.1 Module

Provides persistent storage services for the system's domain, package, metric, and user data. This module encapsulates all interactions with the relational database, ensuring data integrity, schema consistency, and transactional reliability for all read/write operations.

6.17.2 Uses

None.

6.17.3 Syntax

Exported Constants

- **DB_TIMEOUT_SEC** = 15
- **AUTO_COMMIT** = true

Exported Access Programs

Name	In	Out	Exceptions
createEntity	entityType, entityData	entityID	WriteError
readEntity	entityType, query	entitySet	NotFoundError
updateEntity	entityType, entityID, patch	successFlag	WriteError
deleteEntity	entityType, entityID	successFlag	DeleteError
executeQuery	querySpec	queryResult	QueryError

6.17.4 Semantics

State Variables

- **connectionPool**: pool of reusable relational database connections.
- **lastTransactionStatus**: Boolean flag indicating success or failure of the last query.

State Invariant

- All committed database operations preserve referential integrity among stored entities.
- If **lastTransactionStatus** = true, the most recent operation was committed successfully.

Environment Variables

- **DB_HOST**: address of the database server.
- **DB_PORT**: port used to connect to the database server.
- **DB_USER**: database username credential.
- **DB_PASSWORD**: database password credential.
- **DB_NAME**: logical database name containing system tables.
- **DB_URL** (optional): single connection string that may replace the variables above.

Assumptions

- Database schema remains stable (UC1).
- Schema changes (if any) are version-controlled and applied through the system's database migration mechanism.
- Connection credentials are valid and not hard-coded.

Access Routine Semantics **createEntity**(entityType, entityData):

- transition: inserts a new entity record of type `entityType` using `entityData`.
- output: `entityID` identifying the created entity.
- exception: `WriteError` if the operation violates constraints or cannot be committed.

readEntity(entityType, query):

- transition: none.
- output: `entitySet` containing all entities of type `entityType` matching `query`.
- exception: `NotFoundError` if no matching entities exist.

updateEntity(entityType, entityID, patch):

- transition: applies `patch` updates to the entity identified by `entityID`.
- output: `successFlag` = true if the update is committed.
- exception: `WriteError` if the entity does not exist or the update cannot be committed.

deleteEntity(entityType, entityID):

- transition: removes the entity identified by `entityID` if permitted by integrity constraints.
- output: `successFlag` = true if deletion is committed.
- exception: `DeleteError` if deletion is blocked by constraints or cannot be committed.

executeQuery(querySpec):

- transition: executes the database operation described by `querySpec`.
- output: `queryResult` containing the resulting dataset or status.
- exception: `QueryError` if the query is invalid or execution fails.

Local Functions

- `openConnection()` — establishes or obtains a database connection for an operation.
- `commitOrRollback()` — commits successful operations or rolls back on failure.
- `logDBActivity()` — records persistence operations for monitoring and diagnostics.

6.18 MIS of Logging Module (M18)

6.18.1 Module

Provides centralized logging services for recording and retrieving system events and user actions. This module supports traceability, debugging, and auditing by accepting log entries from backend services and persisting them in a configurable log store.

6.18.2 Uses

System API Gateway Module (M10) (6.11)
Database Persistence Module (M17) (*optional; only if* `storeType = DB`) (6.17)

6.18.3 Syntax

Exported Constants

- **DEFAULT_LEVEL** = "INFO"
- **MAX_LOG_SIZE_MB** = 10
- **BACKUP_COUNT** = 5

Exported Access Programs

Name	In	Out	Exceptions
initLogger	logConfig	successFlag	ConfigError
logEvent	message, level, context	successFlag	WriteError
getRecentLogs	limit, [filter:LogFilter]	list(LogEntry)	ReadError
archiveLogs	None	successFlag	ArchiveError

Data Types

- **LogLevel** ::= {`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`}
- **LogEntry** ::= $\langle timestamp : string, level : LogLevel, message : string, context : dict(string, string)\rangle$
- **LogFilter** ::= $\langle minLevel : Option(LogLevel), containsText : Option(string)\rangle$
- **LogConfig** ::= $\langle storeType : \{\texttt{FILE}, \texttt{DB}\}, level : LogLevel, fileName : Option(string)\rangle$

6.18.4 Semantics

State Variables

- **activeLogger**: configured logger instance handling writes.
- **logLevel**: current logging verbosity threshold.
- **logBuffer**: queue of pending log entries awaiting persistence (if asynchronous logging is used).

State Invariant

- **activeLogger** is initialized before any successful call to `logEvent()` or `getRecentLogs()`.
- **logLevel** $\in$ {`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`}.
- If **logBuffer** is used, then $0 \leq |\textbf{logBuffer}|$ and all buffered entries have a well-formed timestamp, level, and message.
- Every persisted log entry contains at minimum: `timestamp`, `level`, and `message`.

Environment Variables

- **clock**: system clock used for timestamps.
- **logDir**: directory/location for log storage if file-based.

Assumptions

- File system permissions allow append and rotation of log files.
- Log directory structure already exists and is writable by the application.
- Critical events (ERROR, CRITICAL) are monitored through external observability tools if configured.

Project Configuration Note For the peer-implementation deliverable and the current system release, `logConfig.storeType = FILE`. Therefore, persistence and retrieval are file-based under `logDir`. Database persistence via M17 is supported by this MIS but is not required for the peer implementation. If `fileName = None`, the implementation shall use `app.log`.

Access Routine Semantics **initLogger**(logConfig):

- transition: initializes the logger with parameters defined in logConfig.
- output: `successFlag` = true if initialization succeeds.
- exception: `ConfigError` if logConfig invalid or missing keys.

logEvent(message, level, context):

- transition:
  - Create `entry : LogEntry` using the current `clock` for `timestamp`.
  - If `level` is below the configured threshold `logLevel`, do not persist and return `successFlag = false`.
  - If `logConfig.storeType = FILE`, append `entry` to `logDir/fileName` (default `app.log` if `fileName = None`).
  - If `logConfig.storeType = DB`, persist `entry` via the Database Persistence Module (M17).
- output: `successFlag` indicates whether the entry was persisted.
- exception: `WriteError` if the configured log store cannot be written.

getRecentLogs(limit, [filter]):

- transition: reads entries, applies `filter` if provided, then returns the most recent `limit` entries.
- output: `logsList : list(LogEntry)` containing up to `limit` most recent entries matching `filter` if provided.

- exception: `ReadError` if logs cannot be retrieved.

archiveLogs():

- transition: compresses and rotates existing log files once they exceed `MAX_LOG_SIZE_MB`.
- output: `successFlag` = true if archiving succeeds.
- exception: `ArchiveError` if file rotation fails.

Local Functions

- `formatLogEntry()` — attaches timestamp, severity, and context metadata.
- `rotateIfNeeded()` — triggers log rotation/archival when thresholds are exceeded.
- `emitAlert()` — notifies monitoring systems for severe events if configured.

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.

7 Appendix

7.1 Appendix — Module Hierarchy Diagram

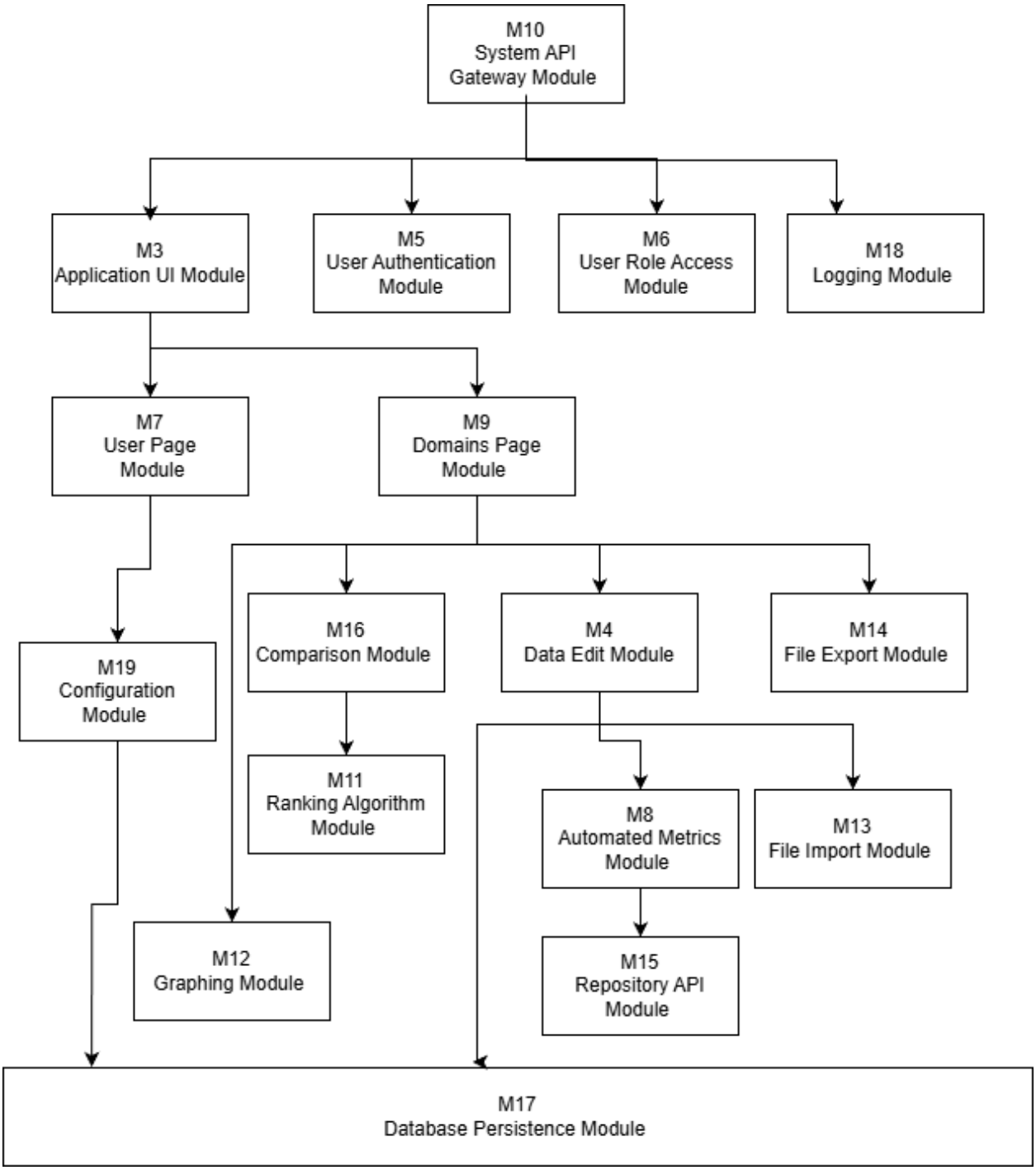

Figure 1: Module Hierarchy for DomainX

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?

   Awurama: We had a clear module hierarchy and strong alignment with the MG, which made structuring each MIS section easier. Collaboration and consistency across modules also improved clarity.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   Awurama: Managing LaTeX formatting, especially tables and long text, was time-consuming. We fixed these by using the tabularx package and standardizing our template for all modules.

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?

   Awurama: Most design choices came from client discussions with Dr. Smith and feedback from peers. For example, decisions on automation scope, ranking algorithms, and database structure came directly from those conversations.

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?

   Awurama: We refined sections of the SRS and MG for consistency, mainly updating module responsibilities and interfaces to match the finalized MIS content.

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)

   Awurama: Limited time and resources meant we couldn't build full scalability or advanced automation. With more resources, we'd expand integration testing, optimize APIs, and improve real-time analytics.

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?

   Awurama: We considered using a non-relational database and different frontend frameworks but chose Django/MySQL and React for reliability and maintainability. Alternatives offered flexibility but added complexity and reduced team familiarity.

   (LO_Explores)

Peer Implementation Reflection:

Implementing team 25's module was very manageable because the MIS clearly listed the exported constants and exported access programs which gave a strong starting point for the structure and naming. The most helpful part of the MIS was the access routine, since it clearly stated the expected conversion approach (repeated division and remainder for decimal to dozenal, and the positional evaluation for dozenal to decimals). The main challenge i think however was that the MIS did not fully define the allowed digit-symbol mapping for the two extra dozenal digits beyond 0-9, so some of the implementation choices made required assumptions to a level (e.g., choosing `X` and `E` as canonical symbols and supporting alternate inputs). Another difficulty was deciding how strict i should make normalization, especially around things like handling whitespace, casing, optional plus signs, and the leading zeros.

To improve implementability, the MIS could include explicit examples of valid and invalid dozenal strings, plus sample inputs and outputs for each access program (including edge cases like 0, negative numbers, and error strings). This process really helped to reinforce the importance of specifying specific or rather strict input validation rules and canonical formatting expectations inside the MIS so people implementing do not rely on interpretation or assumption (but i guess you always have to to some extent). Overall, implementing this module helped confirm how small missing details in a specification can affect consistency across different implementations, but a good MIS can still make implementation manageable.