Module Interface Specification for SFWRENG 4G06 - Capstone Design Process

Team 17, DomainX

Awurama Nyarko
Haniye Hamidizadeh
Fei Xie
Ghena Hatoum

January 19, 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.

This tool is implemented using a **React** frontend and a **Django** backend, with a **MySQL** database managed through the Django Object–Relational Mapper (ORM). Data exchange between system components follows a **REST API** architecture using **JSON** payloads. These technologies were selected based on their stability and are considered **unlikely to change** throughout the project lifecycle.

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:** $\bot$

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:** $\emptyset$, $\{x \in S \mid P(x)\}$, 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	$(\text{T}_1, \ldots, \text{T}_n)$	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 API payloads. 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 React front end. The browser renders HTML/CSS/JS, maintains the page's Document Object Model (DOM), collects user input (mouse, keyboard), and handles network requests to the backend.

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.

Environment Variables   None. This module relies solely on capabilities provided by the host browser runtime and does not consume externally supplied configuration values.

Assumptions

- A modern standards-compliant browser (Chromium/Chrome, Firefox, Safari).

- Supports HTML5, ECMAScript 2020+, Fetch API, CORS, and Web Storage.

- Site is served over HTTPS; backend endpoints expose JSON over REST.

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 React-based front-end shell (routing, layout, and common UI state) that renders all pages and components of the DomainX app, and wires user interactions to the rest of the system.

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)
Repository API Module (M15) (6.16)

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.

Environment Variables None. This module does not consume externally supplied configuration values and relies solely on constants defined within the module and services provided by dependent modules.

Assumptions

- Executed within a modern standards-compliant browser providing a Document Object Model (DOM).

- Implemented using React with client-side routing in a single-page application (SPA) environment.

- 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: mounts the React root, initializes router and global providers (theme, session).

- 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)
Database Persistence Module (M17) (6.17)

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   None. This module does not consume externally supplied configuration values; it relies on services provided by M3/M10 and the host browser runtime.

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 behind M10.

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 using Django's built-in authentication framework and JWT-based tokens (DRF SimpleJWT). This module exposes a simplified interface to the rest of the system rather than reimplementing authentication logic.

6.4.2 Uses

System API Gateway Module (M10) [Django REST Framework, JSON] (6.11)
User Role Access Module (M6) (6.5)
Database Persistence Module (M17) [Django ORM] (6.17)
Application UI Module (M3) (6.2)

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, invite-Code	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.

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

Assumptions

- Authentication uses Django's `AbstractUser` and DRF SimpleJWT (JWT-based tokens).

- 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()` – wraps Django's `make_password()` to store password hashes.

- `verifyPassword()` – compares plaintext input to stored hash.

- `generateToken()` – issues signed JWT using Django secret key.

- `rateLimitLogin()` – blocks login after `MAX_LOGIN_ATTEMPTS`.

6.5 MIS of User Role Access Module (M6)

6.5.1 Module

Defines, stores, and enforces user permissions and access levels across the DomainX web application. This module ensures that users can only view or modify data permitted by their assigned roles, using the role-based access control (RBAC) mechanism built on Django's authorization framework.

6.5.2 Uses

User Authentication Module (M5) (6.4)
System API Gateway Module (M10) [Django REST Framework, JSON] (6.11)
Database Persistence Module (M17) [Django ORM] (6.17)
Application UI Module (M3) (6.2)

6.5.3 Syntax

Exported Constants

- **DEFAULT_ROLE** = "Viewer"

- **ROLE_HIERARCHY** = ["Viewer", "Editor", "Admin"]

- **PERMISSIONS_CACHE_TTL** = 300 *(seconds; cached client-side for performance)*

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.

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

Assumptions

- All users are authenticated before accessing role-based resources (via M5).

- Role hierarchy is static and unlikely to change during this version.

- Permissions are fetched from the backend via JSON and cached for `PERMISSIONS_CACHE_TTL` seconds.

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()` – pushes updates to Django ORM 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 such as display name, email, password, and notification preferences, integrating with the backend through M12.

6.6.2 Uses

User Authentication Module (M5) (6.4)
User Role Access Module (M6) (6.5)
System API Gateway Module (M10) [Django REST Framework, JSON] (6.11)
Database Persistence Module (M17) [Django ORM] (6.17)
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.

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

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 old password; updates to `newPwd` via M5 backend call.

- 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) [GitHub API, PyPI API] (6.16)
System API Gateway Module (M10) [Django REST Framework, JSON] (6.11)
Database Persistence Module (M17) [Django ORM] (6.17)
Ranking Algorithm Module (M11) (6.12)

6.7.3 Syntax

Exported Constants

- **UPDATE_INTERVAL_HOURS** = 24

- **MAX_RETRIES** = 3

- **SUPPORTED_METRICS** =

  - **GitHub API:**
    * stars
    * forks
    * watchers
    * openPullRequests
    * closedPullRequests
  - **Git statistics (git_stats):**
    * textFileCount
    * binaryFileCount
    * textLineCount

18

* 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.

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

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: inserts or updates metric entries in M17.

- 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 in M17.

- 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.

20

6.8 MIS of Domains Page Module (M9)

6.8.1 Module

Implements the logic and presentation layer for displaying all domain-related data within the DomainX platform. This includes the retrieval, organization, and rendering of domains, their associated packages, metrics, and textual descriptions for end users through a React interface connected to the Django backend.

6.8.2 Uses

Configuration Module (M19) [Django ORM, PostgreSQL] (6.10)
System API Gateway Module (M10) [Django REST Framework] (6.11)
Graphing Module (M12) [Chart.js / Plotly via React] (6.13)
Ranking Algorithm Module (M11) [for displaying domain scores] (6.12)
Automated Metrics Module (M8) (6.7)

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.

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

Assumptions

- Domain data is properly structured and indexed in the database.

- Graph rendering libraries (Chart.js or Plotly) are installed and functional.

- User has appropriate permissions via M6 (User Role Access Module) to view domain data.

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

- transition: none.

- output: list of domain objects matching `filterOptions`.

- exception: `DatabaseError` if ORM query fails.

displayDomainDetails(domainID):

- transition: loads the selected domain's details into `DomainDetails.jsx`.

- output: rendered domain view.

- exception: `DataNotFoundError` if the domain ID does not exist.

renderDomainMetrics(domainID, metricType):

- transition: retrieves the domain's metric data via M8 and renders a chart through 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) [AHP / BWM weighting methods] (6.12)
Automated Metrics Module (M8) [metrics retrieval] (6.7)
Database Persistence Module (M17) [Django ORM – metrics and package data] (6.17)
Graphing Module (M12) [Chart.js / Plotly for visual comparison] (6.13)
System API Gateway Module (M10) [Django REST Framework] (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.

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

Assumptions

- All metric data used for comparison is already validated by M8.

- User selects a valid comparison method from `SUPPORTED_METHODS`.

- Database schema for domains and metrics is consistent with UC1 (structural stability).

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: executes the selected comparison algorithm using data fetched via M11 and M17.

- output: `comparisonTable` containing ranked and weighted scores.

- exception: `ComputationError` if calculation fails or data incomplete.

fetchComparisonData(domainSet):

- transition: none.

- output: retrieves metric records for all domains in the set.

- exception: `DatabaseError` if ORM query fails.

visualizeComparison(comparisonTable, chartType):

- transition: renders visual representation of results via M12.

- output: chart object displayed in the React UI.

- 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) [Django authentication system] (6.4)
Database Persistence Module (M17) [Django ORM / PostgreSQL] (6.17)
System API Gateway Module (M10) [REST API for data retrieval] (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.

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

25

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 ORM query 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 Django server, the PostgreSQL database, and the React frontend. This module manages business logic flow, data serialization, and application state across all feature modules. It ensures that frontend requests are authenticated, validated, and routed to the correct Django views or serializers.

The backend application is initialized and run using Django's standard entry points (`manage.py` for development and WSGI/ASGI for deployment).

6.11.2 Uses

Database Persistence Module (M17) [Django ORM / PostgreSQL] (6.17)
User Authentication Module (M5) [Django Auth Middleware] (6.4)
Automated Metrics Module (M8) [API data collection integration] (6.7)
Repository API Module (M15) [external API requests] (6.16)
Configuration Module (M19) (6.10)

6.11.3 Syntax

Exported Constants

- **BASE_URL** = "https://api.domainx.ca/v1/"

- **REQUEST_TIMEOUT_SEC** = 30

- **MAX_CONNECTIONS** = 100

- **CONTENT_TYPE** = "application/json"

Exported Access Programs

Name	In	Out	Exceptions
sendRequest	endpoint, payload, method	responseData	NetworkError
authenticateRequest	sessionToken	authorizedHeader	AuthError
handleResponse	rawResponse	parsedResponse	ParseError
routeToService	endpoint, payload	serviceOutput	RouteError

6.11.4 Semantics

State Variables

- **activeSessions**: dictionary of authenticated users and tokens.

- **apiRegistry**: mapping of available service endpoints $\rightarrow$ handler functions.

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

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).

Access Routine Semantics   sendRequest(endpoint, payload, method):

- transition: sends an HTTP request to `endpoint` with `payload`.

- output: `responseData` containing parsed JSON response.

- exception: `NetworkError` if the request fails or times out.

authenticateRequest(sessionToken):

- transition: adds authorization header to outgoing requests.

- output: `authorizedHeader`.

- exception: `AuthError` if token is invalid or expired.

handleResponse(rawResponse):

- transition: parses JSON and validates response schema.

- output: structured `parsedResponse`.

- exception: `ParseError` if invalid JSON is returned.

routeToService(endpoint, payload):

- transition: maps endpoint to corresponding Django service function.

- output: processed response object.

- exception: `RouteError` if endpoint is undefined in `apiRegistry`.

Local Functions

- `serializeData()` — converts Django models to JSON format for frontend consumption.

- `deserializeRequest()` — validates incoming JSON requests against defined schemas.

- `logTransaction()` — records API request metadata for monitoring and debugging.

6.12 MIS of Ranking Algorithm Module (M11)

6.12.1 Module

Implements the Analytic Hierarchy Process (AHP) ranking algorithm that computes pairwise comparisons of software packages or domains based on weighted criteria. This module quantifies qualitative assessments and produces normalized ranking scores to determine relative performance or importance. It is implemented using the AHPy library, with extension points for future ranking models such as Best–Worst Method (BWM) or Skew–Symmetric Bilinear (SSB).

6.12.2 Uses

Database Persistence Module (M17) [metrics and domain data via Django ORM] (6.17)
Automated Metrics Module (M8) [input metric data] (6.7)
Comparison Module (M16) [for result visualization] (6.9)
Configuration Module (M19) [for user-defined weights and preferences] (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.

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 AHP consistency ratio does not exceed 0.1.

Access Routine Semantics   computeRankScores(metricsTable, userWeights):

- transition: constructs a weighted decision matrix using AHPy.

- 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 the logic and utilities for generating graphs and data visualizations from processed metric datasets. This module supports visual comparison of domains, packages, and metric trends using the Matplotlib library. It operates as a backend visualization service, receiving numerical or categorical data from other modules and outputting visual plots (bar, line, pie, or radar charts) for display or export.

6.13.2 Uses

Automated Metrics Module (M8) [provides metric values] (6.7)
Database Persistence Module (M17) [retrieves stored dataset entries] (6.17)
System API Gateway Module (M10) [routes graphing requests] (6.11)
Comparison Module (M16) [requests comparative plots] (6.9)
Configuration Module (M19) [retrieves user visualization preferences] (6.10)

6.13.3 Syntax

Exported Constants

- **DEFAULT_STYLE** = "seaborn-v0_8-darkgrid"

- **DEFAULT_FIGSIZE** = (10, 6)

- **SUPPORTED_CHARTS** = ["bar", "line", "pie", "radar"]

- **EXPORT_FORMAT** = [".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.

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

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: constructs a Matplotlib figure according to input type and configuration.

- output: `graphImage` — rendered visualization object.

- 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):

32

- transition: saves figure to specified file path in given 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.

- `cleanTempFiles()` — clears cached images 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 Excel or CSV files into standardized data frames compatible with the system's metric and domain schema. It ensures type consistency and data integrity before persisting the data to the database.

6.14.2 Uses

Database Persistence Module (M17) [for data persistence] (6.17)
System API Gateway Module (M10) [for upload routing and validation] (6.11)
Configuration Module (M19) [for file-format settings] (6.10)
Automated Metrics Module (M8) [for synchronization of imported metrics] (6.7)

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.

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

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: opens and reads the file using `pandas.read_csv()` or `pandas.read_excel()`.

- 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: inserts validated rows into the database through the Django ORM (M9).

- output: `successFlag` = true if all rows stored successfully.

- exception: `WriteError` if a database transaction fails.

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 formats such as CSV or Excel. This module converts domain, metric, and ranking data from database tables into standardized file outputs compatible with external analysis tools. It ensures that exported files preserve correct formatting, encoding, and schema alignment for interoperability.

6.15.2 Uses

Database Persistence Module (M17) [retrieves structured data for export] (6.17)
System API Gateway Module (M10) [handles export requests and delivery] (6.11)
Configuration Module (M19) [provides user-selected export settings] (6.10)
Ranking Algorithm Module (M11) [supplies ranking data for export] (6.12)

6.15.3 Syntax

Exported Constants

- **SUPPORTED_FORMATS** = [".csv", ".xlsx"]

- **DEFAULT_ENCODING** = "utf-8"

- **EXPORT_DIRECTORY** = "/exports/"

- **MAX_ROWS_PER_FILE** = 50,000

Exported Access Programs

Name	In	Out	Exceptions
prepareExportData	queryParams, dataType	dataFrame	DataError
exportToCSV	dataFrame, fileName	filePath	WriteError
exportToExcel	dataFrame, 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.

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

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: retrieves records from the database according to filters and data type.

- output: `dataFrame` with processed export data.

- exception: `DataError` if query fails or data unavailable.

exportToCSV(dataFrame, fileName):

- transition: writes data to a comma-separated file using `pandas.to_csv()`.

- output: `filePath` to generated CSV file.

- exception: `WriteError` if file I/O fails.

exportToExcel(dataFrame, fileName):

- transition: writes data to an Excel file using `pandas.ExcelWriter()`.

36

- output: `filePath` to generated Excel file.

- exception: `WriteError` if Excel writer 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

Automated Metrics Module (M8) [for metric retrieval requests] (6.7)
System API Gateway Module (M10) [for backend communication] (6.11)
Database Persistence Module (M17) [for storing fetched data] (6.17)

6.16.3 Syntax

Exported Constants

- **GITHUB_BASE_URL** = "https://api.github.com/"

- **PYPI_BASE_URL** = "https://pypi.org/pypi/"

- **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.

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

Assumptions

- API tokens are valid and refreshed automatically if expired.

- Rate limiting is handled gracefully using `Retry-After` headers.

- JSON responses from external APIs follow their documented schemas.

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: sends GET request to GitHub/PyPI API for project metadata.

- output: `repoData` — dictionary of metadata (name, owner, description, license).

- exception: `APIError` if request fails or JSON malformed.

fetchRepoMetrics(packageID, metricList):

- transition: sends API requests for specific metrics (e.g., stars, forks, downloads).

- output: `metricsData` — structured dictionary of 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

- `parseJSONResponse()` — converts JSON responses into Python dictionaries.

- `retryRequest()` — retries failed API calls up to `MAX_RETRIES`.

- `logAPICall()` — records each API interaction for debugging and monitoring.

6.17 MIS of Database Persistence Module (M17)

6.17.1 Module

Handles all data persistence and retrieval operations for the system's relational database. This module defines and manages models representing entities such as domains, packages, metrics, and users. It ensures data integrity, schema consistency, and transactional reliability using Django's ORM layer with a MySQL backend.

6.17.2 Uses

Configuration Module (M19) [for connection credentials] (6.10)

6.17.3 Syntax

Exported Constants

- **DB_NAME** = "domainx_db"

- **DB_ENGINE** = "django.db.backends.mysql"

- **DB_TIMEOUT_SEC** = 15

- **AUTO_COMMIT** = true

Exported Access Programs

Name	In	Out	Exceptions
createRecord	tableName, dataDict	successFlag	WriteError
readRecord	tableName, queryFilters	resultSet	NotFoundError
updateRecord	tableName, recordID, updatedFields	successFlag	WriteError
deleteRecord	tableName, recordID	successFlag	DeleteError
executeRawQuery	queryString	queryResult	QueryError

6.17.4 Semantics

State Variables

- **connectionPool**: pool of reusable MySQL database connections.

- **lastTransactionStatus**: Boolean flag indicating success or failure of the last query.

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

Assumptions

- Database schema remains stable (UC1).

- All ORM migrations are version-controlled through Django's migration system.

- Connection credentials are valid and not hard-coded.

Access Routine Semantics   **createRecord**(tableName, dataDict):

- transition: inserts new record into table using Django ORM's `Model.objects.create()`.

- output: `successFlag` = true if commit successful.

- exception: `WriteError` if transaction fails or constraint violated.

readRecord(tableName, queryFilters):

- transition: queries table for matching records using `filter()` or `get()`.

- output: `resultSet` = list or object containing retrieved records.

- exception: `NotFoundError` if no results found.

updateRecord(tableName, recordID, updatedFields):

- transition: updates record fields and commits changes to the database.

- output: `successFlag` = true if update succeeds.

- exception: `WriteError` if record not found or data invalid.

deleteRecord(tableName, recordID):

- transition: deletes a record using ORM's `delete()` method.

- output: `successFlag` = true if deletion succeeds.

- exception: `DeleteError` if record missing or database locked.

executeRawQuery(queryString):

- transition: executes custom SQL queries directly via ORM's `raw()` method.

- output: `queryResult` containing query output.

- exception: `QueryError` if query invalid or connection lost.

Local Functions

- `initializeConnectionPool()` — sets up reusable database connections on startup.

- `rollbackTransaction()` — rolls back failed transactions to preserve integrity.

- `logQueryActivity()` — logs all executed queries for debugging and analytics.

6.18 MIS of Logging Module (M18)

6.18.1 Module

Provides centralized logging functionality for recording and retrieving events, user activities, and system actions. It captures events from backend services, database operations, and API calls, storing them in structured log files for traceability and debugging. This module helps ensure accountability, auditability, and system reliability throughout DomainX.

6.18.2 Uses

System API Gateway Module (M10) [for capturing backend requests and responses] (6.11)
Database Persistence Module (M17) [for storing error logs and event history] (6.17)
Configuration Module (M19) [for environment-specific logging paths and levels] (6.10)

6.18.3   Syntax

Exported Constants

- **LOG_FILE_PATH** = "/var/log/domainx/app.log"

- **DEFAULT_LEVEL** = "INFO"

- **MAX_LOG_SIZE_MB** = 10

- **BACKUP_COUNT** = 5

Exported Access Programs

Name	In	Out	Exceptions
initLogger	configFilePath	successFlag	ConfigError
logEvent	message, level, context	successFlag	WriteError
getRecentLogs	limit	logsList	FileError
archiveLogs	None	successFlag	ArchiveError

6.18.4   Semantics

State Variables

- **activeLogger**: configured logger instance handling all writes.

- **logLevel**: current logging verbosity threshold (DEBUG, INFO, WARNING, ERROR).

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

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.

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

- transition: initializes the logger with parameters defined in the config file.

- output: `successFlag` = true if initialization succeeds.

- exception: `ConfigError` if file invalid or missing keys.

logEvent(message, level, context):

- transition: formats and writes a log entry to file and console.

- output: `successFlag` = true if write successful.

- exception: `WriteError` if log file inaccessible or write fails.

getRecentLogs(limit):

- transition: reads from log file up to the specified limit of lines.

- output: `logsList` = list of recent log entries.

- exception: `FileError` if log file missing or corrupted.

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

- `formatLogMessage()` — prepends timestamps and log levels to messages.

- `rotateLogs()` — performs log rotation based on file size thresholds.

- `sendCriticalAlert()` — triggers email or console alerts for severe events.

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.

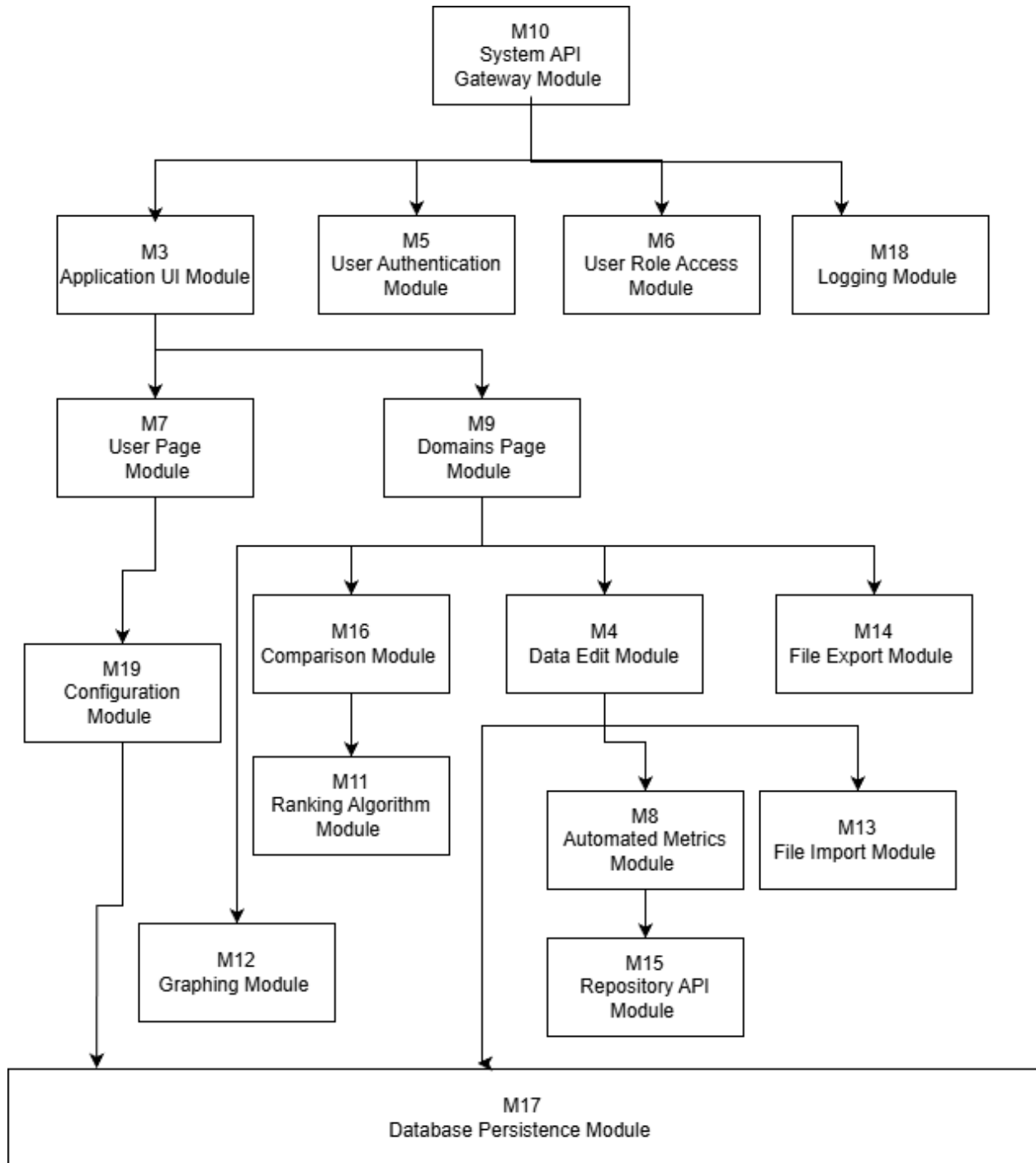# 7 Appendix

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)