Module Interface Specification for SFWRENG 4G06 - Capstone Design Process

Team 17, DomainX

Awurama Nyarko
Haniye Hamidizadeh
Fei Xie
Ghena Hatoum

November 5, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS Documentation

This section records information for easy reference. Additional terms specific to the MIS have been included below.

Table 1: Symbols, Abbreviations, and Acronyms

Symbol / Acronym	Description
AC	Anticipated Change.
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).
ADT	Abstract Data Type.
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 — export format for datasets.
DAG	Directed Acyclic Graph.
DB	Database — 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.
LLM	Large Language Model.
M	Module.
MG	Module Guide.
ML	Machine Learning.
NN	Neural Network.
NNL	Neural Network Libraries.
OS	Operating System.
Packages	Software Packages.
PoC	Proof of Concept — demonstration validating workflow integration.
R	Requirement.
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.
UC	Unlikely Change.
VnV	Verification and Validation — process of ensuring correctness and meeting stakeholder needs.

Contents

3 Introduction

The following document details the Module Interface Specifications 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.

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: https://github.com/thaafei/DomainX/tree/main.

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).

General Conventions

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

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: $\langle a_1, \ldots, a_n \rangle$; concatenation $s \parallel t$; length $|s|$
- Maps/dictionaries: $m : K \to V$; application $m(k)$; update $m[k \mapsto v]$

- Ranges/intervals: $[a, b]$, $[a, b)$ as usual
- Composition: $(g \circ f)(x) = g(f(x))$

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 $(0, 1, 2, \dots)$.
real	$\mathbb{R}$	Real numbers.
boolean	`bool`	`True` or `False`.
string	`string`	Finite sequence of characters.
tuple	`(`T_1`, ..., `T_n`)`	Fixed-size, ordered, possibly heterogeneous.
list/sequence	`list(T)`	Variable-size sequence of `T`.
set	$\mathcal{P}(T)$	Finite subset of `T` (powerset elements).
dictionary/map	`dict(K,V)`	Finite mapping from `K` to `V`.
option	`Option(T)`	Either `Some(T)` or `None` (for optional outputs).

Units and Identifiers

- Timestamps use ISO 8601. Durations use explicit units (e.g., `min`, `hrs`).
- Identifiers such as `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 described by type signatures, parameter modes, pre/postconditions, and declared exceptions.

5 Module Decomposition

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

Level 1	Level 2
Hardware-Hiding Module	Browser Module (M2)
Behaviour-Hiding Module	Domain View Module (M3) Data Edit Module (M4) User Authentication Module (M5) User Role Access Module (M6) Metrics Storage Module (M7) Package Storage Module (M8) Domain Storage Module (M9) Automated Metrics Module (M10) Localization Module (M11)
Software Decision Module	System API Gateway Module (M12) Ranking Algorithm Module (M13) Graphing Module (M14) File Import Module (M15) File Export Module (M16) Repository API Module (M17) Domain Comparison Module (M18) Database Configuration Module (M19) Configuration Constants Module (M20)

Table 2: Module Hierarchy for the Domain Assessment Tool

6 Module Interface Specifications

6.1 MIS of Domain View Module

6.1.1 Module

Provides the front-end interface for visualizing and retrieving domain-specific metrics. Allows users to browse domains, inspect metadata, and trigger analytics visualizations.

6.1.2 Uses

Browser Module (M2)
System API Gateway (M12)
Graphing Module (M14)

6.1.3 Syntax

Exported Constants

- **MAX_DISPLAY_DOMAINS** = 100
- **DEFAULT_CHART_TYPE** = "bar"

Exported Access Programs

Name	In	Out	Exceptions
loadDomains	filterOptions	domainList	NetworkError
displayDomainDetails	domainID	renderedView	DataNotFound
renderChart	metricData, chartType	chartObject	RenderError

6.1.4 Semantics

State Variables

- **selectedDomain**: current domain ID shown in view.
- **displayBuffer**: cached metrics for active domain.

Environment Variables Browser window / UI canvas environment.

Assumptions Implemented in the React-based front-end of the Domain Assessment Tool. Assumes network connectivity and valid API response format (JSON).

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

- transition: fetch domain metadata from API gateway and update displayBuffer.
- output: list of available domains.
- exception: NetworkError if API request fails.

displayDomainDetails(domainID):

- transition: updates UI to show metrics and metadata for selected domain.
- output: rendered domain view.
- exception: DataNotFound if domain ID is invalid.

renderChart(metricData, chartType):

- transition: invokes Graphing Module to create visualization.
- output: chart object for UI display.
- exception: RenderError if chart generation fails.

Local Functions

- `formatTooltip()` – constructs metric labels for hover display.
- `applyTheme()` – applies current UI theme to chart components.
- `paginateDomains()` – handles scroll-based pagination.

Table 3: Review Summary — Domain View Module

Aspect	Observation	Suggested MG Update
Uses relation	Depends on API Gateway and Graphing modules for data and rendering support.	Ensure both dependencies are declared in MG.
Exported programs	Three core UI access points align with MG services.	None.
Environment variables	Browser canvas not explicitly mentioned in MG.	Add note under Hardware-Hiding Modules.

6.2 MIS of Data Edit Module

6.2.1 Module

Handles modification of existing metric entries, allowing authorized users to add, edit, or remove data related to domains. Ensures changes are validated before being committed to persistent storage through the API Gateway.

6.2.2 Uses

Domain View Module (M3)
User Authentication Module (M5)
System API Gateway Module (M12)
Metrics Storage Module (M7)

6.2.3 Syntax

Exported Constants

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

Exported Access Programs

Name	In	Out	Exceptions
editMetric	metricID, newValue	successFlag	ValidationError
deleteMetric	metricID	successFlag	NotFoundError
addMetric	metricData	successFlag	SchemaError
undoLastEdit	None	successFlag	StackEmptyError

6.2.4 Semantics

State Variables

- **editBuffer**: stores pending changes before confirmation.
- **undoStack**: maintains history of last ten user operations.

Environment Variables User session context (active login, role, and permissions).

Assumptions Assumes an active authenticated user session and a stable API connection. Changes are only committed after local validation passes and a confirmation is received from the Metrics Storage Module.

Access Routine Semantics **editMetric**(metricID, newValue):

- transition: updates metric in `editBuffer` and requests validation.
- output: successFlag = true if edit applied successfully.
- exception: `ValidationError` if new value violates schema constraints.

deleteMetric(metricID):

- transition: marks metric as deleted and removes from local view.
- output: successFlag = true if deletion confirmed by storage.
- exception: `NotFoundError` if metric ID is invalid or already deleted.

addMetric(metricData):

- transition: sends new metric record to storage after schema validation.
- output: successFlag indicating storage acknowledgement.
- exception: `SchemaError` for structural mismatches.

undoLastEdit():

- transition: reverses most recent edit or deletion.
- output: successFlag = true if operation successfully rolled back.
- exception: `StackEmptyError` if no operations are available to undo.

Local Functions

- `validateMetricSchema()` – checks type and completeness of new metric data.
- `logEditAction()` – records user edits for audit tracking.
- `rollbackEdit()` – restores prior state from `undoStack`.

Table 4: Review Summary — Data Edit Module

Aspect	Observation	Suggested MG Update
Uses relation	Matches dependencies (User Authentication, API Gateway, Metrics Storage).	None required.
Exported programs	CRUD-style access routines consistent with MG services.	None.
Assumptions	Requires authenticated user session — clarify in MG "Services" notes.	Add authentication dependency explicitly.

6.3 MIS of User Authentication Module

6.3.1 Module

Manages user authentication processes, including login, logout, and token validation. It ensures only authorized users can access or modify domain data and enforces session timeouts.

6.3.2 Uses

Browser Module (M2)
User Role Access Module (M6)
System API Gateway Module (M12)

6.3.3 Syntax

Exported Constants

- **TOKEN_EXPIRY_MINUTES** = 30
- **MAX_LOGIN_ATTEMPTS** = 5

Exported Access Programs

Name	In	Out	Exceptions
loginUser	username, password	sessionToken	AuthError
logoutUser	sessionToken	successFlag	TokenError
validateToken	sessionToken	Boolean	TokenExpiredError
refreshSession	sessionToken	newToken	RefreshError

6.3.4 Semantics

State Variables

- **activeSessions**: dictionary mapping tokens to user IDs and expiry timestamps.
- **loginAttempts**: counter per user for throttling repeated login failures.

Environment Variables

- Secure HTTP session layer provided by the browser.
- Encrypted credential storage service.

Assumptions Assumes all credentials are stored in a secure hashed format. Implements OAuth2-style authentication through the System API Gateway. Relies on HTTPS encryption for data transport.

Access Routine Semantics **loginUser**(username, password):

- transition: verifies credentials and generates session token.
- output: valid `sessionToken`.
- exception: `AuthError` if credentials are invalid or max attempts exceeded.

logoutUser(sessionToken):

- transition: removes token from `activeSessions`.
- output: successFlag = true on successful logout.
- exception: `TokenError` if token not recognized.

validateToken(sessionToken):

- transition: none (read-only check).
- output: true if token valid and unexpired.

- exception: `TokenExpiredError` if expired or revoked.

refreshSession(sessionToken):

- transition: invalidates old token and issues a refreshed one.
- output: `newToken`.
- exception: `RefreshError` if refresh request rejected.

Local Functions

- `hashPassword()` – applies SHA-256 or bcrypt hashing to credentials.
- `generateToken()` – produces secure signed token with expiry timestamp.
- `validateCredentials()` – queries storage to verify user/password pair.

Table 5: Review Summary — User Authentication Module

Aspect	Observation	Suggested MG Update
Uses relation	Depends on User Role Access and API Gateway for session management.	Explicitly state both dependencies in MG.
Exported programs	Standard authentication routines aligned with MG "Services."	None.
Assumptions	OAuth2 and HTTPS protocols mentioned here; MG may need a short note.	Add under "Implemented By: DomainX."

6.4 MIS of User Role Access Module

6.4.1 Module

Defines and enforces role-based permissions within the Domain Assessment Tool. Determines the level of access users have to data editing, visualization, and configuration functionalities based on their assigned roles.

6.4.2 Uses

User Authentication Module (M5)
Data Edit Module (M4)
Domain View Module (M3)
System API Gateway (M12)

6.4.3 Syntax

Exported Constants

- **ROLE_HIERARCHY** = ["Viewer", "Editor", "Admin"]
- **DEFAULT_ROLE** = "Viewer"
- **ACCESS_TIMEOUT_MIN** = 60

Exported Access Programs

Name	In	Out	Exceptions
assignRole	userID, roleType	successFlag	InvalidRoleError
verifyAccess	userID, actionType	Boolean	PermissionError
getUserRole	userID	roleType	NotFoundError
updatePermissions	roleType, permissionsList	successFlag	ConfigError

6.4.4 Semantics

State Variables

- **roleMap**: dictionary mapping users to role types.
- **permissionTable**: maps role types to allowed actions.

Environment Variables User session context provided by Authentication Module.

Assumptions Assumes authentication is successful before role verification. Roles and permission configurations are stored in a secure database accessed via the API Gateway.

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

- transition: updates `roleMap` with new user role.
- output: successFlag = true upon update confirmation.
- exception: `InvalidRoleError` if roleType not in ROLE_HIERARCHY.

verifyAccess(userID, actionType):

- transition: none (read-only verification).
- output: true if user's role allows the given actionType.

- exception: `PermissionError` if insufficient privilege.

getUserRole(userID):

- transition: none.
- output: returns user's current roleType.
- exception: `NotFoundError` if userID not mapped.

updatePermissions(roleType, permissionsList):

- transition: modifies permissionTable for given roleType.
- output: successFlag = true if update persisted.
- exception: `ConfigError` if write operation fails.

Local Functions

- `checkRoleHierarchy()` – confirms whether a user's role outranks another.
- `syncPermissionCache()` – refreshes local permissionTable from server.
- `sanitizeRoleInput()` – ensures valid string input for roleType.

Table 6: Review Summary — User Role Access Module

Aspect	Observation	Suggested MG Update
Uses relation	Depends on Authentication and API Gateway; cascades into Data Edit and Domain View.	Ensure hierarchy reflected in MG.
Exported programs	Consistent with access management responsibilities in MG.	None.
Assumptions	Role configuration persistence and secure DB not detailed in MG.	Add note under "Services."

6.5 MIS of Metrics Storage Module

6.5.1 Module

Responsible for storing, updating, and retrieving metric data associated with specific domains. This module provides a uniform interface between the front-end editing tools and the underlying database, ensuring data consistency and validation before persistence.

6.5.2 Uses

System API Gateway (M12)
Package Storage Module (M8)
Automated Metrics Module (M10)

6.5.3 Syntax

Exported Constants

- **METRIC_TABLE_NAME** = "metrics"
- **MAX_QUERY_LIMIT** = 1000
- **CACHE_EXPIRY_MINUTES** = 15

Exported Access Programs

Name	In	Out	Exceptions
fetchMetrics	domainID, filters	metricSet	QueryError
storeMetric	metricData	successFlag	WriteError
updateMetric	metricID, newValue	successFlag	ValidationError
deleteMetric	metricID	successFlag	NotFoundError

6.5.4 Semantics

State Variables

- **metricCache**: temporary cache for recently accessed metric sets.
- **connectionStatus**: flag indicating live DB connection state.

Environment Variables

- Database instance (e.g., PostgreSQL, MySQL).
- Server filesystem for backup storage.

Assumptions Assumes the database schema for metrics is predefined and aligned with the data collection format. Assumes all write operations are atomic and transactional (committed via the API Gateway).

Access Routine Semantics **fetchMetrics**(domainID, filters):

- transition: queries the database for all metrics that match domainID and filters.
- output: list of metric objects.
- exception: `QueryError` if malformed query or connection lost.

storeMetric(metricData):

- transition: validates and inserts new metric record into database.
- output: successFlag = true if insertion confirmed.
- exception: `WriteError` if database write fails.

updateMetric(metricID, newValue):

- transition: updates existing metric record with new value.
- output: successFlag = true on success.
- exception: `ValidationError` if schema mismatch detected.

deleteMetric(metricID):

- transition: removes metric record from database.
- output: successFlag = true if deletion confirmed.
- exception: `NotFoundError` if record doesn't exist.

Local Functions

- `connectDB()` – establishes or reuses a database connection.
- `validateMetricFormat()` – ensures input matches schema requirements.
- `updateCache()` – refreshes cached metric entries after modification.

Table 7: Review Summary — Metrics Storage Module

Aspect	**Observation**	**Suggested MG Update**
Uses relation	Relies on API Gateway and Automated Metrics; aligns with MG dependencies.	None.
Exported programs	CRUD and query functions match MG "Services" description.	Add cache expiry constant if missing.
Assumptions	Assumes DB transactional integrity; may need to specify DB type in MG.	Add under "Implemented By."

6.6 MIS of Package Storage Module

6.6.1 Module

Manages persistent storage and retrieval of package metadata, including identifiers, versions, dependencies, and descriptions. Provides programmatic access for the Automated Metrics and Domain Comparison modules to fetch or update package-related information.

6.6.2 Uses

System API Gateway (M12)
Metrics Storage Module (M7)
Repository API Module (M17)

6.6.3 Syntax

Exported Constants

- **PACKAGE_TABLE_NAME** = "packages"
- **MAX_PACKAGES_FETCH** = 500
- **CACHE_DURATION_MIN** = 20

Exported Access Programs

Name	In	Out	Exceptions
fetchPackage	packageID	packageData	NotFoundError
storePackage	packageData	successFlag	WriteError
updatePackageVersion	packageID, newVersion	successFlag	ValidationError
deletePackage	packageID	successFlag	NotFoundError

6.6.4 Semantics

State Variables

- **packageCache**: temporary in-memory cache of recently accessed package data.
- **lastSyncTimestamp**: record of last synchronization with Repository API.

Environment Variables Database instance and external repository (e.g., GitHub, PyPI).

Assumptions Assumes unique identifiers exist for all packages and versioning follows semantic versioning (e.g., 1.0.0). Assumes repository APIs are accessible through the API Gateway and return consistent metadata schemas.

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

- transition: retrieves package metadata from DB or cache.
- output: `packageData` containing fields such as name, version, and dependencies.
- exception: `NotFoundError` if record missing or inaccessible.

storePackage(packageData):

- transition: validates metadata and inserts new package record into DB.
- output: successFlag = true upon confirmation.
- exception: `WriteError` if write operation fails.

updatePackageVersion(packageID, newVersion):

- transition: updates version field of the specified package record.
- output: successFlag = true on success.
- exception: `ValidationError` if version string invalid or outdated.

deletePackage(packageID):

- transition: removes specified package record.
- output: successFlag = true if deletion confirmed.
- exception: `NotFoundError` if ID not found.

Local Functions

- `validateVersionFormat()` – ensures version string matches semantic versioning (MAJOR.MINOR.PATCH).
- `syncWithRepository()` – periodically updates DB entries using external API data.
- `updateCache()` – refreshes cached package info after DB updates.

Table 8: Review Summary — Package Storage Module

Aspect	Observation	Suggested MG Update
Uses relation	Matches MG (depends on API Gateway, Metrics Storage, Repository API).	None.
Exported programs	CRUD and sync routines consistent with MG "Services."	Add cache or sync constants if missing.
Assumptions	Semantic versioning and repository sync not mentioned in MG.	Add under "Services" description.

6.7 MIS of Domain Storage Module

6.7.1 Module

Provides persistent storage and retrieval for *domain* metadata (e.g., id, name, description, tags, owners, status). Exposes lookup and management routines used by view, edit, and comparison workflows.

6.7.2 Uses

System API Gateway (M12)
Metrics Storage Module (M7)
Package Storage Module (M8)

6.7.3 Syntax

Exported Constants

- **DOMAIN_TABLE_NAME** = "domains"
- **MAX_DOMAIN_QUERY** = 500
- **DEFAULT_PAGE_SIZE** = 25

Exported Access Programs

Name	In	Out	Exceptions
createDomain	domainData	domainID	WriteError
getDomain	domainID	domainRecord	NotFoundError
listDomains	filters, pageSpec	domainList	QueryError
updateDomain	domainID, newData	successFlag	ValidationError
archiveDomain	domainID	successFlag	NotFoundError

6.7.4 Semantics

State Variables

- **domainCache**: recent domain records keyed by id.
- **indexState**: cached pagination cursor/offset for last query.

Environment Variables

- Database service (relational or document).
- Server filesystem (optional) for snapshot/export.

Assumptions Assumes uniqueness constraint on `name` or `(name, owner)` per MG conventions. Assumes all writes are routed through the API Gateway with transactional consistency.

Access Routine Semantics **createDomain**(domainData):

- transition: validates and inserts new record into `DOMAIN_TABLE_NAME`.
- output: `domainID` of created record.
- exception: `WriteError` on DB failure.

getDomain(domainID):

- transition: none (read-only); may hydrate `domainCache`.
- output: `domainRecord`.
- exception: `NotFoundError` if id does not exist.

listDomains(filters, pageSpec):

- transition: updates `indexState` with latest cursor/offset.
- output: `domainList` subject to `MAX_DOMAIN_QUERY`.
- exception: `QueryError` for malformed filters or connection loss.

updateDomain(domainID, newData):

- transition: applies partial/full updates; invalidates relevant cache keys.
- output: `successFlag`.
- exception: `ValidationError` if fields violate schema/uniqueness.

archiveDomain(domainID):

- transition: flips archival flag; cascades soft-hide in dependent listings.
- output: `successFlag`.
- exception: `NotFoundError` if id missing.

Local Functions

- `validateDomainSchema()` — checks required fields and name uniqueness.
- `applyFilters()` — builds query predicates from filter map.
- `invalidateCacheKeys()` — evicts stale cache entries on write.

Table 9: Review Summary — Domain Storage Module

Aspect	Observation	Suggested MG Update
Uses relation	References API Gateway, Metrics, Packages for cross-links	Ensure all three are listed under Uses in MG
Exported routines	CRUD + pagination align with storage responsibilities	Add pagination constants to MG if absent
Assumptions	Name/owner uniqueness implied, not explicit	State uniqueness constraint in MG schema notes

6.8 MIS of Automated Metrics Module

6.8.1 Module

Automates the retrieval, validation, and storage of repository metrics for each package or domain. Interacts with the Repository API Module to collect metrics, processes and validates the data, and then stores validated results in the Metrics Storage Module.

6.8.2 Uses

Repository API Module (M17)
Metrics Storage Module (M7)
System API Gateway (M12)

6.8.3 Syntax

Exported Constants

- **METRIC_FETCH_INTERVAL_HRS** = 24
- **SUPPORTED_METRICS** = [“stars“, “forks“, “issues“, “commits“]
- **VALIDATION_THRESHOLD** = 0.05

Exported Access Programs

Name	In	Out	Exceptions
fetchMetricsFromRepo	packageID	metricsData	APIError
validateMetrics	metricsData	Boolean	ValidationError
storeValidatedMetrics	metricsData	successFlag	WriteError
scheduleAutoUpdate	None	successFlag	SchedulerError

6.8.4 Semantics

State Variables

- **lastFetchTimestamp**: stores timestamp of most recent metric retrieval.
- **pendingQueue**: list of packages awaiting metric updates.

Environment Variables

- External network (for repository API requests).
- System scheduler or cron-like service (for automation intervals).

Assumptions Assumes repositories are publicly accessible or authenticated through valid tokens. Assumes metrics follow consistent response formats across repository platforms (e.g., GitHub, PyPI). Relies on stable internet connectivity for scheduled retrieval.

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

- transition: sends API request to repository, retrieves metrics, and parses response.
- output: `metricsData` in standardized schema.
- exception: `APIError` if API call fails or response malformed.

validateMetrics(metricsData):

- transition: checks each metric for valid ranges, completeness, and data type consistency.
- output: Boolean indicating validation result.
- exception: `ValidationError` if schema mismatch or missing key metrics.

storeValidatedMetrics(metricsData):

- transition: passes validated metrics to Metrics Storage Module for persistence.
- output: successFlag = true if operation succeeds.
- exception: `WriteError` if DB operation fails.

scheduleAutoUpdate():

- transition: registers background job to periodically invoke `fetchMetricsFromRepo`.
- output: successFlag = true if scheduling confirmed.
- exception: `SchedulerError` if scheduling service unavailable.

Local Functions

- `parseRepositoryResponse()` – converts raw API payloads into unified metric schema.
- `filterInvalidEntries()` – removes zero or null metrics before validation.
- `enqueuePendingPackage()` – adds package to update queue for later processing.

Table 10: Review Summary — Automated Metrics Module

Aspect	Observation	Suggested MG Update
Uses relation	Consistent with MG (Repository API, Metrics Storage, API Gateway).	None.
Exported programs	Align with MG "Services" (fetch, validate, store).	None.
Assumptions	Adds explicit automation/scheduler assumptions.	Optionally include scheduler note in MG.

6.9 MIS of Localization Module

6.9.1 Module

Provides internationalization (i18n) and localization (l10n) support for the Domain Assessment Tool. Handles translation of UI labels, formatting of numbers, dates, and times, and region-specific preferences for consistent user experience across locales.

6.9.2 Uses

Domain View Module (M3)
System API Gateway (M12)
Browser Module (M2)

6.9.3 Syntax

Exported Constants

- **SUPPORTED_LANGUAGES** = [“en“, “fr“, “es“, “de“]
- **DEFAULT_LANGUAGE** = “en“
- **DATE_FORMAT_MAP** = {“en”: “MM/DD/YYYY“, “fr”: “DD/MM/YYYY“}

Exported Access Programs

Name	In	Out	Exceptions
setLanguage	langCode	successFlag	InvalidLangError
translateLabel	labelKey, langCode	localizedText	MissingTranslationError
formatDate	dateValue, langCode	formattedString	FormatError
applyLocalization	userPrefs	successFlag	ConfigError

6.9.4 Semantics

State Variables

- **currentLanguage**: active language setting for the current session.
- **translationCache**: map of label keys to localized text strings.

Environment Variables

- Browser locale settings.
- Server-side translation repository (JSON, YAML, or DB).

Assumptions Assumes the translation files are accessible through the API Gateway and cached client-side. All translation keys are consistent across supported languages.

Access Routine Semantics **setLanguage**(langCode):

- transition: updates `currentLanguage` and reloads translation cache.
- output: successFlag = true on successful update.
- exception: `InvalidLangError` if langCode not in SUPPORTED_LANGUAGES.

translateLabel(labelKey, langCode):

- transition: none.
- output: localized text for given labelKey.
- exception: `MissingTranslationError` if labelKey missing in dictionary.

formatDate(dateValue, langCode):

- transition: converts `dateValue` to locale-specific string.
- output: formatted date string.
- exception: `FormatError` if invalid input date or missing mapping.

applyLocalization(userPrefs):

- transition: applies language, region, and timezone settings globally in UI.
- output: successFlag = true if preferences applied successfully.
- exception: `ConfigError` if userPrefs missing required fields.

Local Functions

- `loadTranslationFile()` – fetches and parses translation resource by language code.
- `detectBrowserLocale()` – determines initial language preference from browser.
- `fallbackToDefault()` – returns English label if translation missing.

Table 11: Review Summary — Localization Module

Aspect	**Observation**	**Suggested MG Update**
Uses relation	Matches MG (Browser + API Gateway).	None.
Exported programs	Language setting and translation routines align with MG description.	Add supported language constants in MG.
Assumptions	Translation storage and caching implied but not stated.	Include note about translation source format in MG.

6.10 MIS of System API Gateway Module

6.10.1 Module

Serves as the centralized communication interface between all system components and external services. Encapsulates HTTP requests, handles authentication tokens, standardizes API responses, and enforces data validation between the front-end and back-end layers.

6.10.2 Uses

User Authentication Module (M5)
Metrics Storage Module (M7)
Package Storage Module (M8)
Repository API Module (M17)

6.10.3 Syntax

Exported Constants

- **BASE_URL** = “https://api.domainassessmenttool.ca/v1/“
- **REQUEST_TIMEOUT_SEC** = 30
- **RETRY_LIMIT** = 3
- **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
retryRequest	endpoint, payload, attempts	responseData	RetryLimitError

6.10.4 Semantics

State Variables

- **authToken**: stores the current session’s authentication token.
- **requestLog**: buffer containing recent API call metadata (timestamp, endpoint, status).

Environment Variables

- Network layer and web server hosting backend APIs.
- Internet connection availability.

Assumptions Assumes stable network connectivity and that backend endpoints follow RESTful conventions. Responses are returned as JSON-formatted payloads containing both status codes and data objects. Assumes all sensitive operations require valid authentication headers.

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

- transition: constructs HTTP request, sends to target endpoint, logs metadata.
- output: parsed JSON response.
- exception: `NetworkError` on failed connection or timeout.

authenticateRequest(sessionToken):

- transition: verifies token with Authentication Module and embeds in request header.
- output: authorizedHeader containing bearer token.
- exception: `AuthError` if invalid or expired.

handleResponse(rawResponse):

- transition: none; parses `rawResponse` into standardized schema.
- output: `parsedResponse`.
- exception: `ParseError` for malformed responses.

retryRequest(endpoint, payload, attempts):

- transition: resends failed request until success or reaching retry limit.
- output: responseData.
- exception: `RetryLimitError` if retries exhausted.

Local Functions

- `buildURL()` — concatenates BASE_URL and endpoint path.
- `serializePayload()` — converts Python/JS objects into JSON strings.
- `logRequest()` — appends call details to request log for debugging.

Table 12: Review Summary — System API Gateway Module

Aspect	Observation	Suggested MG Update
Uses relation	Aligns with MG (Auth, Storage, Repository API).	None.
Exported programs	Covers core request flow: send, authenticate, handle, retry.	None.
Assumptions	Explicitly defines REST/JSON contract and auth dependency.	Add note about retry and timeout behavior.

6.11 MIS of Ranking Algorithm Module

6.11.1 Module

Implements the ranking and weighting logic that determines the relative performance or importance of each repository or package. Uses the Analytic Hierarchy Process (AHP) and weighted normalization methods to compute comparable ranking scores from both quantitative metrics and user-defined weights.

6.11.2 Uses

Metrics Storage Module (M7)
Automated Metrics Module (M10)

6.11.3 Syntax

Exported Constants

- **DEFAULT_WEIGHTS** = {“stars”: 0.3, “forks”: 0.25, “commits”: 0.25, “issues”: 0.2}
- **MAX_SCORE** = 100
- **NORMALIZATION_METHOD** = “min-max“

Exported Access Programs

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

6.11.4 Semantics

State Variables

- **lastRankingResult**: stores the most recent computed rankings for reuse or caching.
- **criteriaMatrix**: user- or system-defined weighting matrix for the AHP method.

Environment Variables

- None directly — computation occurs entirely within software using stored data.

Assumptions Assumes all required metrics (stars, forks, commits, issues) are present in the input dataset. User-provided weights must sum to 1 for valid ranking computations. Assumes AHP consistency ratio check has already been performed (if applicable).

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

- transition: applies AHP-derived weights to each repository's metric values.
- output: `rankTable` with composite ranking scores.
- exception: `InvalidMetricError` if required metrics missing.

applyAHPWeights(criteriaMatrix):

- transition: multiplies metric importance scores by pairwise comparison matrix.
- output: `weightedMatrix` with derived weights per criterion.
- exception: `MatrixError` if inconsistent or non-square matrix.

normalizeScores(rankTable):

- transition: scales raw scores between 0 and 1 (or 0–100).

- output: `normalizedTable`.
- exception: `NormalizationError` if denominator = 0 or NaN encountered.

getTopDomains(normalizedTable, limit):

- transition: sorts and filters `normalizedTable` by score descending.
- output: list of top-ranked domains.
- exception: none.

Local Functions

- `calcWeightedSum()` – aggregates weighted metric contributions.
- `checkWeightSum()` – ensures weights sum to 1 within tolerance.
- `applyConsistencyCheck()` – optional validation for AHP matrix consistency.

Table 13: Review Summary — Ranking Algorithm Module

Aspect	Observation	Suggested MG Update
Uses relation	Matches MG (uses Metrics Storage and Automated Metrics).	None.
Exported programs	Consistent with MG "ranking computation and weighting".	Clarify AHP as explicit method.
Assumptions	Expanded to specify metric completeness and normalization behavior.	Add note about AHP consistency ratio.

6.12 MIS of Graphing Module

6.12.1 Module

Provides functionality for generating visual representations of the ranking and metric data produced by the system. Supports bar charts, scatter plots, line graphs, and radar charts for comparative analysis of repositories and domains. Implements both on-screen rendering and export to file formats.

6.12.2 Uses

Ranking Algorithm Module (M13)
Metrics Storage Module (M7)
System API Gateway (M12)

6.12.3 Syntax

Exported Constants

- **SUPPORTED_PLOTS** = [“bar“, “scatter“, “line“, “radar“]
- **DEFAULT_EXPORT_FORMAT** = “.png“
- **MAX_PLOTS_PER_VIEW** = 6

Exported Access Programs

Name	In	Out	Exceptions
generatePlot	dataFrame, plotType	plotObject	PlotTypeError
displayGraph	plotObject	renderedView	RenderError
exportGraph	plotObject, format	filePath	ExportError
updateGraphData	newData	refreshedPlot	DataError

6.12.4 Semantics

State Variables

- **graphCache**: stores recent plot configurations for reuse.
- **activeTheme**: defines the color palette and layout style.

Environment Variables

- Graphics or plotting library (e.g., Matplotlib or Plotly).
- File system for exported graph images.

Assumptions Assumes normalized and ranked data is provided by Ranking Algorithm Module. Assumes required plotting library is available in the environment and can export to image formats.

Access Routine Semantics **generatePlot**(dataFrame, plotType):

- transition: creates a new plot object according to selected type.
- output: `plotObject`.
- exception: `PlotTypeError` if plotType unsupported.

displayGraph(plotObject):

- transition: renders plot to user interface or output device.
- output: rendered visual.
- exception: `RenderError` on failure.

exportGraph(plotObject, format):

- transition: serializes plot into an image file.
- output: `filePath`.
- exception: `ExportError` if write operation fails.

updateGraphData(newData):

- transition: updates existing plot with new dataset.
- output: refreshed visualization.
- exception: `DataError` if invalid input schema.

Local Functions

- `applyTheme()` — applies active color palette.
- `configureAxes()` — sets chart labels and scaling.
- `validatePlotType()` — ensures requested type is supported.

Table 14: Review Summary — Graphing Module

Aspect	**Observation**	**Suggested MG Update**
Uses relation	Matches MG — depends on Ranking Algorithm, Metrics Storage, and API Gateway.	None.
Exported programs	Aligned with MG's "Graphing Module" functions.	None.
Assumptions	Matches MG expectations regarding normalized data and export capability.	None.

Assumes required plotting library is available in the environment and can export to image formats. This module does not perform statistical normalization; it expects input already prepared by Ranking Algorithm or upstream data prep.

6.13 MIS of Data Parser Module

6.13.1 Module

Handles extraction, validation, and transformation of input data into the internal data schema used throughout the system. Supports multiple input formats including JSON (from APIs), CSV (from manual uploads), and XML (for legacy integrations). Ensures field consistency and schema conformance before data is passed to the Metrics Storage Module.

6.13.2 Uses

System API Gateway Module (M12)
Metrics Storage Module (M7)

6.13.3 Syntax

Exported Constants

- **SUPPORTED_FORMATS** = {"json", "csv", "xml"}
- **REQUIRED_FIELDS** = {"repoName", "stars", "forks", "commits", "issues"}
- **MAX_FILE_SIZE_MB** = 25

Exported Access Programs

Name	In	Out	Exceptions
parseInput	rawData, format	structuredData	UnsupportedFormatError
validateSchema	structuredData	Boolean	SchemaValidationError
transformFields	structuredData	normalizedData	TransformationError
loadToStorage	normalizedData	confirmationMsg	StorageError

6.13.4 Semantics

State Variables

- **lastParsedFile**: metadata record of most recent input file parsed.
- **validationErrors**: list of recent schema or formatting issues detected.

Environment Variables

- Local file system or temporary memory buffer for uploads.
- Web browser or API client as input origin.

Assumptions Assumes all input files are UTF-8 encoded and within the defined file size limit. Assumes the schema used across system modules remains consistent (e.g., same metric names and datatypes). Assumes no cyclic references in JSON or XML data.

Access Routine Semantics **parseInput**(rawData, format):

- transition: determines input type and parses accordingly (JSON parser, CSV reader, or XML DOM).
- output: structured data dictionary or DataFrame.
- exception: `UnsupportedFormatError` if format not recognized.

validateSchema(structuredData):

- transition: none.
- output: Boolean flag indicating whether all required fields are present and valid.
- exception: `SchemaValidationError` if missing or mismatched fields.

transformFields(structuredData):

- transition: maps field names and data types to system standard schema.
- output: normalized dataset ready for storage.
- exception: `TransformationError` for type mismatches.

loadToStorage(normalizedData):

- transition: writes normalized data to Metrics Storage Module.
- output: confirmation message or ID.
- exception: `StorageError` if database write fails.

Local Functions

- `detectFormat()` — inspects headers or MIME type to auto-detect format.
- `convertDataTypes()` — casts strings, integers, and floats to appropriate types.
- `logParseErrors()` — records issues in validationErrors list.

Table 15: Review Summary — Data Parser Module

Aspect	Observation	Suggested MG Update
Uses relation	Matches MG — depends on API Gateway and Metrics Storage.	None.
Exported programs	Reflects parsing, validation, transformation, and load.	Add explicit size limit constant if missing.
Assumptions	UTF-8 and schema consistency clarified.	Note handling of invalid rows (drop vs. fix).

6.14 MIS of Repository API Interface

6.14.1 Module

Provides a unified interface to external repository services (e.g., GitHub, PyPI) for retrieving package metadata and activity metrics. Handles authentication headers, pagination, and rate limits, and returns responses in a normalized internal schema.

6.14.2 Uses

System API Gateway (M12)
Automated Metrics Module (M10)
Package Storage Module (M8)

6.14.3 Syntax

Exported Constants

- **BASE_REPO_API_URL** = "https://api.github.com"
- **RATE_LIMIT_WINDOW_SEC** = 3600
- **MAX_RETRY** = 3
- **USER_AGENT** = "DomainAssessmentTool/1.0"

Exported Access Programs

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

6.14.4 Semantics

State Variables

- **authToken**: bearer or PAT used for authenticated API access.
- **lastRateLimitReset**: timestamp of last known rate-limit window reset.

Environment Variables

- External repository services reachable over HTTPS.

Assumptions Assumes network availability and valid credentials for private repositories when required. Assumes external APIs return JSON and follow documented pagination and rate-limit headers.

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

- transition: stores token and updates default request headers.
- output: successFlag = true upon acceptance.
- exception: `AuthError` if token format invalid.

fetchRepoMetadata(packageID):

- transition: none; issues GET to repository API for package metadata.
- output: `repoData` (name, description, owner, urls, license).
- exception: `APIError` on non-2xx or malformed response.

fetchRepoMetrics(packageID, metricList):

- transition: none; queries API endpoints for requested metrics (e.g., stars, forks, issues, commits).
- output: `metricsData` normalized to system schema.
- exception: `APIError` on failure or schema mismatch.

handleRateLimit(responseHeaders):

- transition: computes required backoff from headers and updates `lastRateLimitReset`.
- output: `delaySeconds` to wait before retrying.
- exception: `RateLimitError` if headers missing or inconsistent.

Local Functions

- `buildRepoURL()` — constructs endpoint paths from `BASE_REPO_API_URL` and resource ids.
- `parsePaginated()` — iterates Link headers to accumulate multi-page results.
- `normalizeRepoPayload()` — maps external fields to internal schema.

Table 16: Review Summary — Repository API Interface

Aspect	Observation	Suggested MG Update
Uses relation	Serves M10 and M8 via M12 transport layer	Ensure all three edges shown in MG
Exported programs	Cover auth, metadata, metrics, and rate-limit handling	Note pagination support explicitly
Assumptions	REST+JSON, HTTPS, tokens	Add PAT scope requirements in MG

6.15 MIS of File Import Module

6.15.1 Module

Processes user-provided files (e.g., CSV/Excel) into the system's internal schema so the rest of the system can use the data. Implemented with `pandas`.

6.15.2 Uses

Browser Module (M2)
System API Gateway (M12)
Metrics Storage Module (M7)

6.15.3 Syntax

Exported Constants

- **SUPPORTED_FORMATS** = {"csv", "xlsx"}
- **MAX_FILE_SIZE_MB** = 25
- **HEADER_REQUIRED** = true

Exported Access Programs

Name	In	Out	Exceptions
uploadFile	fileBlob, meta	importSessionID	UnsupportedFormatError, SizeLimitError
parseFile	importSessionID	structuredData	ParseError
validateSchema	structuredData	Boolean	SchemaValidationError
mapFields	structuredData, fieldMap	normalizedData	TransformationError
previewSample	normalizedData, n	sampleFrame	None
commitImport	normalizedData, targetSpec	successFlag	StorageError
rollbackImport	importSessionID	successFlag	None

6.15.4 Semantics

State Variables

- **currentSession**: identifier and metadata for the active import.
- **parseLog**: list of warnings/errors (row numbers, messages).
- **fieldMap**: mapping from source columns to internal schema fields.

Environment Variables

- Temporary file storage/buffer provided by the browser runtime.
- `pandas` runtime for CSV/XLSX parsing and transformation.

Assumptions Input files are CSV/XLSX within size limits and UTF-8 compatible. Target storage and validation occur via API Gateway and Metrics Storage. This module focuses on importing/processing files for downstream use (as specified in MG).

Access Routine Semantics **uploadFile**(fileBlob, meta):

- transition: create `currentSession`, persist the raw file to temp storage, basic sniffing (delimiter/encoding).
- output: `importSessionID`.
- exception: `UnsupportedFormatError` if not in `SUPPORTED_FORMATS`; `SizeLimitError` if exceeds `MAX_FILE_SIZE_MB`.

parseFile(importSessionID):

- transition: read file via `pandas` into `structuredData`; update `parseLog`.
- output: `structuredData`.
- exception: `ParseError` on unreadable or corrupted file.

validateSchema(structuredData):

- transition: none.
- output: Boolean indicating presence and types of required fields.
- exception: `SchemaValidationError` if required columns missing/mistyped.

mapFields(structuredData, fieldMap):

- transition: rename/cast fields to system schema; coerce dates/numerics; dedupe columns.
- output: `normalizedData`.
- exception: `TransformationError` for incompatible casts or ambiguous mappings.

previewSample(normalizedData, n):

- transition: none.
- output: first `n` rows for user confirmation.
- exception: none.

commitImport(normalizedData, targetSpec):

- transition: send rows in chunks through (M12) to (M7) for persistence; record result in `parseLog`.
- output: `successFlag` = true if all chunks confirmed.
- exception: `StorageError` if backend write fails.

rollbackImport(importSessionID):

- transition: discard temp files and any uncommitted buffers associated with session.
- output: `successFlag`.
- exception: none.

Local Functions

- `sniffFormat()` — detect CSV vs. XLSX; delimiter and header presence.
- `detectEncoding()` — attempt BOM/UTF-8 detection.
- `inferDTypes()` — infer/cast numeric, categorical, datetime fields.
- `chunkedUpsert()` — batch rows and send via (M12) to (M7) with retry.
- `logIssue()` — append structured warnings/errors to `parseLog`.

Table 17: Review Summary — File Import Module

Aspect	Observation	Suggested MG Update
Role in MG	Defined as importing Excel/CSV into system data; implemented by `pandas`.	None. (Matches MG text.)
Traceability	Appears under FR9 with File Export (M16).	None. (Already traced in MG.)
Uses relation	Uses Browser (upload), API Gateway (transport), Metrics Storage (persistence).	If desired, add Browser→M15 arrow in Use Hierarchy for clarity.

6.16 MIS of File Export Module

6.16.1 Module

Converts normalized data, rankings, and results within the system into exportable file formats (CSV, XLSX, or JSON). Implements data serialization, column selection, and formatting for downstream reporting and manual review.

6.16.2 Uses

System API Gateway (M12)
Metrics Storage Module (M7)
Graphing Module (M14)

6.16.3 Syntax

Exported Constants

- **SUPPORTED_EXPORTS** = {“csv”, “xlsx”, “json”}
- **DEFAULT_EXPORT_TYPE** = “csv”

- **EXPORT_TIMESTAMP_FORMAT** = “YYYY-MM-DD_HH-MM-SS“
- **MAX_EXPORT_ROWS** = 100000

Exported Access Programs

Name	In	Out	Exceptions
generateExport	dataFrame, format	fileObject	UnsupportedFormatError
selectColumns	dataFrame, columnList	trimmedData	ColumnSelectionError
applyFormatting	trimmedData, styleOpts	formattedData	FormatError
addSummarySheet	formattedData, summaryData	enrichedFile	WriteError
writeToDisk	fileObject, destination	filePath	WriteError
sendToClient	filePath, userContext	successFlag	NetworkError

6.16.4 Semantics

State Variables

- **exportSession**: current export session metadata (user, timestamp, format).
- **exportBuffer**: in-memory representation of file before saving or sending.
- **summaryData**: optional summary table to embed with export (ranking overview, top libraries, etc.).

Environment Variables

- File system access (temporary export directory).
- `pandas` or equivalent library for writing CSV/XLSX.
- Web response handler for initiating download in browser.

Assumptions Assumes data provided by the API Gateway and Metrics Storage is already validated and normalized. Assumes sufficient memory for in-memory buffering of datasets within the export limit. Export process must not modify source data.

Access Routine Semantics **generateExport**(dataFrame, format):

- transition: serializes data to in-memory file representation based on specified format.
- output: `fileObject`.
- exception: `UnsupportedFormatError` if format not in `SUPPORTED_EXPORTS`.

selectColumns(dataFrame, columnList):

- transition: filters `dataFrame` to retain only specified columns.
- output: `trimmedData`.
- exception: `ColumnSelectionError` if requested column does not exist.

applyFormatting(trimmedData, styleOpts):

- transition: applies styling rules (e.g., bold headers, column widths, numeric precision).
- output: `formattedData`.
- exception: `FormatError` if invalid style options.

addSummarySheet(formattedData, summaryData):

- transition: inserts optional summary worksheet into export file (e.g., aggregate rankings).
- output: `enrichedFile`.
- exception: `WriteError` if sheet cannot be created.

writeToDisk(fileObject, destination):

- transition: writes serialized export to disk at given destination path.
- output: `filePath`.
- exception: `WriteError` if insufficient permissions or disk failure.

sendToClient(filePath, userContext):

- transition: triggers download via browser session; updates export log.
- output: `successFlag`.
- exception: `NetworkError` if transmission fails.

Local Functions

- `timestampFileName()` — appends timestamp suffix to exported file.
- `sanitizeHeaders()` — cleans column names of illegal characters.
- `applyExcelStyles()` — formats Excel cells for readability.
- `compressExport()` — zips file for faster transmission if large.
- `validateDestination()` — ensures path exists and writable.

Table 18: Review Summary — File Export Module

Aspect	Observation	Suggested MG Update
Role in MG	Described as converting system data to exportable file types for reporting.	None. Matches MG exactly.
Traceability	Listed under FR9 alongside File Import.	None.
Uses relation	Uses API Gateway (for retrieval), Metrics Storage (for source data), and Graphing (for optional visual export).	Add note that exports can include embedded plots.

6.17 MIS of Domain Comparison Module

6.17.1 Module

Implements algorithms and parameters used for comparing multiple domains within the system. This module facilitates both quantitative and qualitative comparison metrics to determine relationships or relative performance between different research or software domains.

6.17.2 Uses

Ranking Algorithm Module (M13)
Metrics Storage Module (M7)
Graphing Module (M14)

6.17.3 Secrets

The algorithms and parameters used for comparing domains.

6.17.4 Services

Computes and visualizes comparisons across multiple domains.

6.17.5 Implemented By

DomainX

6.17.6 Type of Module

Abstract Object

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

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

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?

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

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?

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?

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)

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)