Module Interface Specification for Software Engineering

Team 2, SyntaxSentinals
Mohammad Mohsin Khan
Lucas Chen
Dennis Fong
Julian Cecchini
Luigi Quattrociocchi

January 17, 2025

1 Revision History

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

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

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

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from **?**, with the addition that template modules have been adapted from **?**. The mathematical notation comes from Chapter 3 of **?**. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Software Engineering.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
Behaviour-Hiding	Input Parameters User Authentication Module Code Upload Module Results Upload Module Threshold Adjustment Module Abstract ML Model Module Tokenization Module AST Module Embedded Module Report Generation Module Email Sending Module
Software Decision	Sequence Data Structure NLP Model Module Similarity Scoring Module Report Results Module Flagging Module

Table 1: Module Hierarchy

6 MIS of User Authorization Module

This module provides functionality for user account creation, user login, and access control, relying on **Auth0** as the implementation mechanism. It safeguards the application's **secrets** (credentials, tokens, etc.) and handles authentication and authorization **services**.

6.1 Module

`AuthModule`

6.2 Uses

- Auth0 library (for handling OAuth/OpenID Connect flows, token verification, etc.)
- Internal user database or identity provider (as configured in Auth0)
- Configuration for secrets management (e.g., environment variables or secure vault)

6.3 Syntax

6.3.1 Exported Constants

- `Module`: Export of the AuthModule React component.

6.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
`loginWithRedirect`	-	AuthToken	LoginError
`logout`	-	-	LogoutError
`signup`	-	AuthToken	SignupError

6.4 Semantics

6.4.1 State Variables

- `isAuthenticated`: Boolean indicating whether the user is currently logged in.

6.4.2 Environment Variables

- `AUTH0_CLIENT_ID`: The client identifier for the Auth0 application.
- `AUTH0_DOMAIN`: The domain used by Auth0 for authentication requests.

6.4.3 Assumptions

- The Auth0 services are available and correctly configured (i.e., valid Client ID, Domain, and Client Secret).
- Network connectivity is available to communicate with Auth0 endpoints.
- User credentials conform to the expected format (valid email, password policy).
- The developer using this module has handled any necessary front-end redirection or session cookies for web-based flows.

6.4.4 Access Routine Semantics

`loginWithRedirect`():

- **transition:**
  - Validates user credentials with Auth0.
  - `currentSession` is updated with returned AuthToken and user info on success.
- **output:** Returns an `AuthToken` containing user claims.
- **exception:** `LoginError` if credentials are invalid or Auth0 is unreachable.

`logout`(AuthToken):

- **transition:** Invalidates `currentSession` (or the provided token) by revoking the Auth0 session or clearing local storage.
- **output:** None.
- **exception:** `LogoutError` if the token is invalid or an Auth0 error occurs.

`signup`(userInfo):

- **transition:**
  - Redirect user to Auth0 for account creation.
  - On success, `currentSession` is redirected back to SyntaxSentienals and updated with new user's AuthToken.
- **output:** Returns `AuthToken` for the newly created user.
- **exception:** `SignupError` if account creation fails (e.g., email already in use).

6.4.5 Local Functions

No local functions are required for this module.

7 MIS of Code Upload Module

7.1 Module

`CodeUploadModule`

Secrets: The format and transport of the input data for the model.

Services: Converts the input data files into the data structure used by the NLP model module and passes it to the backend.

Implemented By: Software Engineering

Type of Module: Library Component

7.2 Uses

- File system or equivalent I/O library (for reading and writing local files)
- HTTP client or backend connector (for sending data to the backend)
- Parser or utility library for code/data formatting, if necessary

7.3 Syntax

7.3.1 Exported Constants

- `MAX_FILE_LENGTH`: The maximum allowed code lines in a single file for upload.
- `ALLOWED_FILE_TYPES`: A list of permissible file extensions (e.g., `.py, .txt, .zip`).

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
`uploadFile`	filePath : String	boolean	FileError
`validateFileFormat`	filePath : String	boolean	FormatError
`convertFileToData`	filePath : String	DataStruct	ConversionError
`sendDataToBackend`	data : DataStruct	boolean	BackendError

7.4 Semantics

7.4.1 State Variables

- `uploadedFile`: Stores the path (or reference) to the currently uploaded file.
- `parsedData`: Stores the in-memory data structure resulting from converting the file.

7.4.2 Environment Variables

- `TEMP_UPLOAD_PATH`: Directory path for temporarily storing uploaded files.
- `BACKEND_URL`: URL endpoint for sending processed data to the backend.

7.4.3 Assumptions

- The file path provided exists and points to a valid file.
- Sufficient storage space is available in `TEMP_UPLOAD_PATH`.
- The backend service is reachable under `BACKEND_URL`.
- Uploaded files comply with any project-specific format or version constraints.

7.4.4 Access Routine Semantics

`uploadFile`(*filePath*):

- **transition:**
  - Copy the file from *filePath* to `TEMP_UPLOAD_PATH`.
  - Update `uploadedFile` to reflect the new file location.
- **output:** Returns `true` on success.
- **exception:** `FileError` if file I/O fails or *filePath* is invalid.

`validateFileFormat`(*filePath*):

- **transition:** None (no internal state change).
- **output:** Returns `true` if the file meets `MAX_FILE_SIZE` and `ALLOWED_FILE_TYPES` conditions.
- **exception:** `FormatError` if the file type or size is invalid.

`convertFileToData`(*filePath*):

- **transition:**
  - Reads raw file content from the `uploadedFile`.
  - Parses and converts the content into `parsedData`.
- **output:** A `DataStruct` representing the file's contents.
- **exception:** `ConversionError` if file parsing fails or content is malformed.

`sendDataToBackend`(*data*):

- **transition:** None (communicates externally, no internal state change).
- **output:** `true` if the backend confirms successful data receipt.
- **exception:** `BackendError` if backend is unreachable or fails to accept data.

7.4.5 Local Functions

- `readLocalFile(path)`: Internal function for raw file I/O.
- `parseCodeData(rawContent)`: Transforms raw file content into a `DataStruct`.

8 MIS of Results Upload Module

8.1 Module

`ResultsUploadModule`

8.2 Uses

- File system or equivalent I/O utilities (to read and load local report files, if applicable)
- Front-end/UI framework (to display the parsed results)
- HTTP or backend connector (if the parsed results need to be sent elsewhere)

8.3 Syntax

8.3.1 Exported Constants

- `MAX_REPORT_FILE_SIZE`: Maximum allowed file size (in bytes) for a report file.
- `ALLOWED_REPORT_TYPES`: Only .zip is allowed.

8.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
`uploadResultsFile`	filePath : String	boolean	FileError
`parseResultsFile`	filePath : String	ResultsDataStruct	ParseError
`displayResults`	data : ResultsDataStruct, uiTarget : UIContainer	boolean	DisplayError

8.4 Semantics

8.4.1 State Variables

- `uploadedReportFile`: Stores the path (or reference) to the currently uploaded report file.
- `parsedReportData`: Stores the in-memory data structure resulting from parsing the report file.

8.4.2 Environment Variables

- `TEMP_REPORT_PATH`: Directory path for temporarily storing uploaded report files.

8.4.3 Assumptions

- The file path provided points to a valid file and does not exceed `MAX_REPORT_FILE_SIZE`.
- The file type is one of `ALLOWED_REPORT_TYPES`.
- The front-end/UI framework is loaded and available for rendering the report data.

8.4.4 Access Routine Semantics

`uploadResultsFile`(*filePath*):

- **transition:**
  - Copies file from *filePath* to `TEMP_REPORT_PATH` (if needed).
  - Updates `uploadedReportFile` to reflect the new file location.
- **output:** Returns `true` if upload is successful.
- **exception:** `FileError` if reading or copying the file fails.

`parseResultsFile`(*filePath*):

- **transition:**
  - Opens and reads the specified report file.
  - Creates an in-memory `ResultsDataStruct` (`parsedReportData`) from the file content.
- **output:** A `ResultsDataStruct` representing the parsed report data.
- **exception:** `ParseError` if the file format is invalid or parsing fails.

`displayResults`(*data, uiTarget*):

- **transition:** None (no change to internal state).
- **output:** Returns `true` if the report data is successfully rendered in the specified UI container.
- **exception:** `DisplayError` if rendering or updating the UI fails.

8.4.5 Local Functions

- `readLocalReportFile(filePath)`: Handles raw file I/O for reading report files.
- `parseReportContent(rawContent)`: Transforms the raw file content into a `ResultsDataStruct`.
- `renderReport(data, container)`: UI logic to display `ResultsDataStruct` in a given front-end container.

9 MIS of Threshold Adjustment Module

9.1 Module

`ThresholdAdjustmentModule`

9.2 Uses

- A back-end or configuration service (to store and retrieve the threshold settings)
- A front-end/UI component (the actual slider element the user interacts with)
- Possibly a validation or range-check module (to ensure threshold inputs are within acceptable bounds)

9.3 Syntax

9.3.1 User-Defined Data Types

- `ThresholdValue`: A numeric type (e.g., `float` in $[0, 1]$ or `int` in $[0, 100]$) that the slider can represent.
- `ThresholdRange`: A structure or pair `(minValue, maxValue)` denoting the allowable slider bounds.
- `Boolean`: A logical type that can be either `true` or `false`.
- `ExceptionType`: A generic exception category (e.g., `ThresholdError`).

9.3.2 Exported Constants

- `DEFAULT_THRESHOLD` : `ThresholdValue` (e.g., 0.75) used if no custom threshold is set.
- `THRESHOLD_RANGE` : `ThresholdRange` (e.g., `(0, 1)`) defining the slider's permissible bounds.

9.3.3 Exported Access Programs

Name	In	Out	Exceptions
`getThreshold`	-	`ThresholdValue`	`ThresholdError`
`setThreshold`	*newVal : ThresholdValue*	`Boolean`	`ThresholdError`
`validateThreshold`	*value : ThresholdValue*	`Boolean`	`ThresholdError`

9.4 Semantics

9.4.1 State Variables

- `currentThreshold` : `ThresholdValue`
  Represents the current position of the slider, reflecting the chosen plagiarism detection threshold.

9.4.2 Environment Variables

- `THRESHOLD_CONFIG_ENDPOINT` : `String`
  The network endpoint or file resource where the threshold configuration is stored/persisted.

9.4.3 Assumptions

- `currentThreshold` is always within `THRESHOLD_RANGE`.
- The user moves the slider to pick a threshold within valid bounds.
- Any saved or loaded threshold configurations adhere to the same data format as defined here.

9.4.4 Access Routine Semantics

`getThreshold`():

- **transition:** None (no change to internal state).
- **output:** Returns the current threshold (slider position), `currentThreshold`.

- **exception:** `ThresholdError` if the threshold is undefined or fails to load from persistence.

`setThreshold`(*newVal*):

- **transition:**
  - Uses `validateThreshold` to check if *newVal* falls within `THRESHOLD_RANGE`.
  - Updates `currentThreshold` to *newVal* if valid.
  - Saves the new value to the configuration endpoint or local store.
- **output:** `true` if *newVal* is successfully set; otherwise `false`.
- **exception:** `ThresholdError` if *newVal* is out of range or otherwise invalid.

`validateThreshold`(*value*):

- **transition:** None (does not change internal state).
- **output:** `true` if *value* is in `THRESHOLD_RANGE`; otherwise `false`.
- **exception:** `ThresholdError` if *value* is malformed (e.g., not a number).

9.4.5 Local Functions

- `readCurrentThreshold()` : Internal function to read the stored threshold from `THRESHOLD_CONFIG_EN`
- `writeCurrentThreshold(value : ThresholdValue)` : Internal function to persist *value* at `THRESHOLD_CONFIG_ENDPOINT`.

10 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)