Module Interface Specification for Software Engineering

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

April 4, 2025

1 Revision History

Date	Version	Notes
January 17	1.0	Initial documentation

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS and MG.

Contents

3 Introduction

The following document details the Module Interface Specifications for SyntaxSentinals.

This project seeks to create a plagiarism algorithm that relies on NLP techniques of present to account for semantics and prevent primitive cir- cumvention of plagiarism detection, such as the addition of benign lines or variable name changes. The users of our product will primarily be those con- cerned with fairness and integrity of code submissions within a competitive environment, such as professors or code competition holders.

Users are intended to use the resulting product of our project by giving it code snippets and receiving a plagiarism report in return. This report will contain a set of similarity scores for inputted code snippets, which when assessed against an outputted threshold will indicate likelihood of plagiarsm having taken place. This will benefit the users by allowing them to more accurately assess the presence of plagiarized work, creating a fairer environment for competition and rewarding coders correctly. Ultimately, the project aims to help users achieve an environment that cycles merit instead of cheating, which is believed to be a primary interest of users too.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/SyntaxSentinels/SyntaxSentinels.

4 Notation

Below is a summary of the notations used in this document:

Data Type	Notation	Description
character	char	A single symbol or digit.
integer	$\mathbb{Z}$	A whole number in the range $(-\infty, \infty)$.
natural number	$\mathbb{N}$	A whole number in the range $[1, \infty)$.
real	$\mathbb{R}$	Any number in the range $(-\infty, \infty)$.
boolean	bool	A logical value that can either be `true` or `false`.
string	str	A sequence of characters.
tuple	tuple	An ordered collection of elements, potentially of different types.
Abstract Syntax Tree	AST	Tree representing code like that described here https://docs.python.org/3/library/ast.html
file	file	An opaque handle to a file on the disk, or a collection of binary data to be written to a file

The following conventions are also used:

- **Assignment**: The operator := denotes assignment.
- **Conditional Rules**: Conditional statements follow the structure $(c_1 \Rightarrow r_1 \mid c_2 \Rightarrow r_2 \mid \ldots \mid c_n \Rightarrow r_n)$, where c_i are conditions and r_i are corresponding results.
- **Access Programs**: Functions and methods are defined with their inputs, outputs, and exceptions as described in the syntax sections of each module.

5 Module Decomposition

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

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	User Authentication Module Code Upload Module Report Results Module Flagging Module Threshold Adjustment Module
Software Decision Module	Report Generation Module Similarity Scoring Module NLP Model Module Abstract ML Model Module Tokenization Module AST Module

Table 1: Module Hierarchy

6 MIS of Code Upload Module

6.1 Module

`CodeUploadModule`

6.2 Uses

- File system or equivalent I/O library (for reading and writing local files)
- Parser or utility library for code/data formatting, if necessary

6.3 Syntax

6.3.1 Exported Constants

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

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
`uploadFile`	filePath : Str	success: bool	FileError
`validateFileFormat`	filePath : Str	valid: bool	FormatError
`convertFileToData`	filePath : Str	-	ConversionError
`getParsedData`	-	snippets: JSON	-

6.4 Semantics

6.4.1 State Variables

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

6.4.2 Environment Variables

- `TEMP_UPLOAD_PATH`: Str
  Directory path for temporarily storing uploaded files.

6.4.3 Assumptions

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

6.4.4 Access Routine Semantics

`uploadFile`(filePath: Str):

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

`validateFileFormat`(filePath: Str):

- **transition:** None (no internal state change).
- **input:** Path representing location of snippets on user computer.
- **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: Str):

- **transition:**
  - Reads raw file content from the `uploadedFile`.
  - Parses and converts the content into JSON for assignment to `parsedData`.
- **input:** Path representing location of snippets on user computer.
- **output:** None
- **exception:** `ConversionError` if file parsing fails or content is malformed.

`getParsedData`():

- **transition:** None (no internal state change).

- **input:** None
- **output:** JSON representing files uploaded by user.
- **exception:** None

6.4.5 Local Functions

`readLocalFile(path: Str)`:

- **transition:** None (no internal state change.)
- **input:** Path representing location of snippets on user computer.
- **output:** *list [String]* object holding all info from each file.
- **exception:** `FileError` if file I/O fails or *filePath* is invalid.

`parseCodeData(rawContent: list[Str])`: Transforms raw file content into a `JSON`.

- **transition:** None (no internal state change).
- **input:** *list [String]* object holding all info from each file.
- **output:** JSON representing code snippet contents.
- **exception:** None

6.5 Module

`ResultsUploadModule`

6.6 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)

6.7 Syntax

6.7.1 Exported Constants

- `MAX_REPORT_FILE_SIZE`: $\mathbb{N}$
  Maximum allowed file size (in bytes) for a report file.
- `ALLOWED_REPORT_TYPES`: $\mathbb{N}$
  Only .zip is allowed.

6.7.2 Exported Access Programs

Name	In	Out	Exceptions
`uploadResultsFile`	filePath: Str	success: bool	FileError
`parseResultsFile`	filePath: Str	report: JSON	ParseError

6.8 Semantics

6.8.1 State Variables

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

6.8.2 Environment Variables

- None

6.8.3 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.

6.8.4 Access Routine Semantics

`uploadResultsFile`(filePath: Str):

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

`parseResultsFile`(filePath: Str):

- **transition:**
  - Opens and reads the specified report file.

  - Updates `parsedReportData` with `JSON` assembled from the file content.

- **input:** Path representing location of snippets on user computer.
- **output:** A `JSON` representing the parsed report data.
- **exception:** `ParseError` if the file format is invalid or parsing fails.

6.8.5 Local Functions

`readLocalReportFile(path: Str)`:

- **transition:** None (no internal state change.)
- **input:** Path representing location of snippets on user computer.
- **output:** *list [String]* object holding all info from each file.
- **exception:** `FileError` if file I/O fails or *filePath* is invalid.

`parseReportContent(rawContent: list[Str])`: Transforms raw file content into a `JSON`.

- **transition:** None (no internal state change).
- **input:** *list [String]* object holding all info from each file.
- **output:** JSON representing code snippet contents.
- **exception:** None

7 MIS of Threshold Adjustment Module

7.1 Module

`ThresholdModule`

7.2 Uses

- A back-end or configuration service (to store and retrieve the threshold settings)
- A front-end/UI component for user manipulation to achieve their desired threshold

7.3 Syntax

7.3.1 Exported Constants

- `DEFAULT_THRESHOLD` : $\mathbb{N}$
  A real number (e.g., 0.75) used if no custom threshold is set.

- `THRESHOLD_RANGE` : $2 - tuple \in \mathbb{R}^2$
  A 2-tuple of real numbers (e.g., `(0, 1)`) defining the permissable bounds for a threshold.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
`getThreshold`	-	`value:` $\mathbb{R}$	`ThresholdError`
`setThreshold`	*newVal:* $\mathbb{R}$	`success: bool`	`ThresholdError`
`validateThreshold`	*value:* $\mathbb{R}$	`success: bool`	`ThresholdError`

7.4 Semantics

7.4.1 State Variables

- `currentThreshold` : $\mathbb{R}$
  Represents the chosen plagiarism detection threshold.

7.4.2 Environment Variables

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

7.4.3 Assumptions

- `currentThreshold` is always within `THRESHOLD_RANGE`.

- Any saved or loaded threshold configurations adhere to the same data format as defined here.

7.4.4 Access Routine Semantics

`getThreshold`():

- **transition:** None (no change to internal state).

- **input:** None

- **output:** Returns the current threshold, i.e., `currentThreshold`.
- **exception:** `ThresholdError` if the threshold is undefined or fails to load from persistence.

`setThreshold`(newVal: $\mathbb{R}$):

- **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.
- **input:** A real number representing user's new desired threshold.
- **output:** `true` if *newVal* is successfully set; otherwise `false`.
- **exception:** `ThresholdError` if *newVal* is out of range or otherwise invalid.

`validateThreshold`(val: $\mathbb{R}$):

- **transition:** None (does not change internal state).
- **input:** A real number representing user's new desired threshold.
- **output:** `true` if *value* is in `THRESHOLD_RANGE`; otherwise `false`.
- **exception:** `ThresholdError` if *value* is malformed (e.g., not a number).

7.4.5 Local Functions

- None

8 MIS of Flagging Module

8.1 Module

`FlaggingModule`

8.2 Uses

- Front-end/UI framework for displaying flagged items.

8.3 Syntax

8.3.1 Exported Constants

- None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
`getFlaggedStatus`	submissionID: Str	status: bool	-
`setFlaggedStatus`	submissionID: Str, flag: bool	-	-
`getAllFlagged`	-	flagged: list[Str]	-

8.4 Semantics

8.4.1 State Variables

- `flaggedSubmissions`: dict[Str, bool] A dictionary of flagged submissions.

8.4.2 Environment Variables

- None

8.4.3 Assumptions

- None

8.4.4 Access Routine Semantics

`getFlaggedStatus(submissionID: str)`:

- **transition:** None (no internal state change.)
- **input:** The unique `submissionID` for which the flagged status is queried
- **output:** Returns the flagged status (`true` or `false`) for the given `submissionID`
- **exception:** None

`setFlaggedStatus(submissionID: str, flag: bool)`:

- **transition:**
  - Updates the flagged status of the specified `submissionID` to the provided `flag` value
- **input:**
  - `submissionID`: The unique identifier of the submission to update
  - `flag`: A boolean value indicating the new flagged status (`true` or `false`)
- **output:** None.

- **exception:** None

`getAllFlagged()`:

- **transition:** None (no internal state change.)
- **input:** None
- **output:** Returns a list of all `submissionID`s that are currently flagged
- **exception:** None

8.4.5 Local Functions

- None

9 MIS of Report Results Module

9.1 Module

`ResultsModule`

9.2 Uses

- Frontend for rendering reports visually for users
- HTTP client or backend connector (for sending data to the backend)
- Code Upload Module to obtain currently parsed code for request

9.3 Syntax

9.3.1 Exported Access Program

Name	In	Out	Exceptions
`sendDataToBackend`	data : JSON	report: JSON	BackendError
`renderReport`	report: JSON	-	-

9.4 Semantics

9.4.1 State Variables

- `report`: JSON
  Info for visuals of plagiarism report.

9.4.2 Environment Variables

- `BACKEND_URL`: Str
  URL endpoint for sending processed data to the backend.

9.4.3 Assumptions

- The backend service is reachable under `BACKEND_URL`.

9.4.4 Access Routine Semantics

`sendDataToBackend`(data: JSON):

- **transition:** None (no internal state change.)
- **input:** JSON representing code snippet contents.
- **output:** JSON representing plagiarism analysis on code snippets.
- **exception:** `BackendError` if backend is unreachable or fails to accept data.

`renderReport(reports: ReportDataStruct)`:

- **transition:** None (no internal state change) but renders a visual representation of the report in the UI
- **input:** The report data JSON of the report to be rendered
- **output:** None
- **exception:** None

9.4.5 Local Functions

No local functions are required for this module.

10 NLP Module

10.1 Module

`NLPModule`

10.2 Uses

- Abstract ML Model Module
- Tokenization Module
- AST Module

10.3 Syntax

10.3.1 Exported Constants

- None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
`combinedPredict`	data: JSON	relations: list[dict[Str, $\mathbb{R}$]]	-

10.4 Semantics

10.4.1 State Variables

- None

10.4.2 Environment Variables

- None

10.4.3 Assumptions

- The code snippets inputted within any data received comes from one programming language.

10.4.4 Access Routine Semantics

`combinedPredict`(data: JSON):

- **transition:** None (no internal state change.)
- **input:** Data representing inputted code snippets in a JSON format.
- **output:** A variable denoted as relations of type `list[dict[Str, ℝ]]` that contains an assembly of results from each of the used modules so the results can be combined to get a more balanced perspective of relations between code snippets.
- **exception: None**

10.4.5 Local Functions

No local functions are required for this module.

11 MIS of Abstract Model Module

11.1 Module

`MLModule`

11.2 Uses

- `transformers` library

11.3 Syntax

11.3.1 Exported Constants

- None.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
`train`	data: 2-tuple[list[Str], dict[Str, $\mathbb{R}$]], supervised: bool , timeout: $\mathbb{Z}$	None	TimeOutException
`predict`	data: list[Str]	result: dict[Str, $\mathbb{R}$]	-

11.4 Semantics

11.4.1 State Variables

- weightings: list[$\mathbb{R}$]
  Weightings that affect prediction of model

11.4.2 Environment Variables

- None

11.4.3 Assumptions

- The code snippets inputted within any data received comes from one programming language.

11.4.4 Access Routine Semantics

`train(data: 2-tuple[list[Str], dict[Str, ℝ]], supervised: bool, timeout: ℝ)`:

- **transition:**
  - weightings := updated_weightings – assign new weightings from batch training to weightings
- **input:** code snippets received for training purposes coupled with predictions which may or may not be empty depending on if learning is supervised or not (*true or false*).
- **output:** None
- **exception:** `TimeOutException` if training time exceeds time limit allotted.

`predict(data: list[Str])`:

- **transition:** None (no internal state change.)
- **input:** Data representing inputted code snippets parsed into string format.
- **output:** Prediction object containing semantic relations between code snippets contained within the DataStruct data.
- **exception:** None

12 MIS of Tokenization Modlue

12.1 Module

`TokenizationModule`

12.2 Uses

- None

12.3 Syntax

12.3.1 Exported Constants

- None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
`tokenize`	source: Str	tokens: list[ℕ]	TokenizeError

12.4 Semantics

12.4.1 State Variables

- None

12.4.2 Environment Variables

- None

12.4.3 Assumptions

- None

12.4.4 Access Routine Semantics

`tokenize(source: str)`:

- **transition:**
  - None
- **input:** A single code snippet.
- **output:** Returns a list of tokens (integers) corresponding to the input source text.
- **exception:** `TokenizeError` if the source code is syntactically invalid.

12.4.5 Local Functions

`pollOneToken(`$2-tuple \in \mathbb{N}^2$`)`:

- **transition:**
  - None
- **output:** Returns a single token read from the given indices of the source string, or None if invalid.
- **exception:** None

13 MIS of Abstract Syntax Tree Module

13.1 Module

`ASTModule`

13.2 Uses

- Built in (for python) or external tree parsers

13.3 Syntax

13.3.1 Exported Constants

- None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
`parse`	rawSource: Str	tree: AST	invalidSyntaxException

13.4 Semantics

13.4.1 State Variables

- None

13.4.2 Environment Variables

- None

13.4.3 Assumptions

- The code snippet provided is syntactically correct or can be parsed with the given rules.

13.4.4 Access Routine Semantics

`parse(rawSource: str)`:

- **transition:** None (no internal state change.)
- **input:** A string representing the input source code
- **output:** Returns the root node of the AST
- **exception:** invalidSyntaxException

13.4.5 Local Functions

No local functions are required for this module.

14 Similarity Scoring Module

14.1 Module

`SimScoreModule`

14.2 Uses

- NLP module

14.3 Syntax

14.3.1 Exported Constants

- None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
`score`	data: JSON	scores: list[2-tuple[str]: $\mathbb{R}$]	-

14.4 Semantics

14.4.1 State Variables

- None

14.4.2 Environment Variables

- None

14.4.3 Assumptions

- The code snippets inputted within any data received comes from one programming language.

14.4.4 Access Routine Semantics

`score`(data: JSON):

- **transition:** None (no internal state change.)
- **input:** Data containing code snippets inputted from the user in the front end.
- **output:** list with associations between code snippet pairings and a score indiciating similarity

- **exception:** None

14.4.5 Local Functions

No local functions are required for this module.

15 Report Generation Module

15.1 Module

`RepGenModule`

15.2 Uses

- Similarity Scoring Module

15.3 Syntax

15.3.1 Exported Constants

- None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
`generate`	data: JSON	report: JSON	-

15.4 Semantics

15.4.1 State Variables

- None

15.4.2 Environment Variables

- None

15.4.3 Assumptions

- The code snippets inputted within any data received comes from one programming language.

15.4.4 Access Routine Semantics

`generate`(data: JSON):

- **transition:** None (no internal state change.)
- **input:** Data containing code snippets inputted from the user in the front end.
- **output:** JSON object wrapping visuals associated with report and similarity scorings to be received by the front end for displaying analysis to the user.
- **exception: None**

15.4.5 Local Functions

`assembleVisuals`(data: JSON):

- **transition:** None (no internal state change.)
- **input:** Data containing code snippets inputted from the user in the front end.
- **output:** JSON
- **exception:** None

15.4.6 State Variables

- None

15.4.7 Environment Variables

- `SMTP_CONFIG`: Configuration details for connecting to the SMTP server (e.g., host, port, authentication).

15.4.8 Assumptions

- A database is configured to store completed reports.

15.4.9 Access Routine Semantics

`storeReport`(report: JSON):

- **transition:**
  - Stores the report in the database.
- **input:** JSON object representing the report data.
- **output:** None
- **exception:** None

15.4.10 Local Functions

No local functions are required for this module.

References

16 Appendix

This section highlights components of the systems that are not modules themselves, but support the other modules in the document.

Appendix — Networking Considerations

- Rate limiting is planned to support the Code Upload Module
  - This ensures code uploads cannot be sent in excessive amounts, and also prevents DoS attacks.
  - Rate limiting will also ensure fair processing time for all users.
- The report generation module will be hidden behind an API layer in the back end.
  - The API will be implemented using a framework such as Django or Flask.

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?**
   The process of writing this deliverable was smooth due to the clear structure and guidelines provided. The team collaborated effectively, leveraging each member's strengths. Additionally, the availability of comprehensive documentation and resources facilitated the writing process.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
   One of the main pain points was ensuring consistency across different sections of the document. To resolve this, we conducted regular team meetings to review progress

and align on the content. Another challenge was integrating feedback from various team members, which sometimes led to conflicting requirements. We addressed this by prioritizing feedback based on its impact on the project and seeking clarification when necessary.

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?**
   None of our existing documents needed to be changed as they were correct upon review.

4. **While creating the design doc, what parts of your other documents (e.g., requirements, hazard analysis, etc.), if any, needed to be changed, and why?**
   During the creation of the design document, we identified the need to update the requirements document to reflect changes in the authentication mechanism. Additionally, the hazard analysis document was revised to include potential security risks associated with external service integrations. These changes were necessary to ensure all documents were aligned and accurately represented the current state of the project.

5. **What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better?**
   One limitation of our solution is the reliance on external services, which introduces dependencies and potential points of failure. Given unlimited resources, we could develop in-house solutions for critical services to reduce dependency risks. Additionally, we could invest in more robust testing and monitoring tools to enhance the system's reliability and performance. Expanding the team to include specialists in security and performance optimization would also contribute to a more resilient solution.

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?**
   We considered several design alternatives, including using different authentication providers and data storage solutions. For example, we evaluated Firebase Authentication as an alternative to Auth0. While Firebase offers seamless integration with other Firebase services, Auth0 was chosen for its advanced security features and flexibility.