Module Interface Specification for Software Engineering

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

January 18, 2025

1 Revision History

Date	Version	Notes
January 17	1.0	Initial documentation

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS

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.

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 Results Upload Module Report Results Module Email Sending 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 User Authentication 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	ReportDataStruct	ParseError

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

- None

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 `ReportDataStruct` (`parsedReportData`) from the file content.
- **output:** A `ReportDataStruct` representing the parsed report data.
- **exception:** `ParseError` if the file format is invalid or parsing 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 `ReportDataStruct`.

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 MIS of Flagging Module

10.1 Module

`FlagModule`

10.2 Uses

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

10.3 Syntax

10.3.1 Exported Constants

- None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
`getFlaggedStatus`	submissionID: string	boolean	-
`setFlaggedStatus`	submissionID: string, flag: boolean	-	-
`getAllFlagged`	-	list[string]	-

10.4 Semantics

10.4.1 State Variables

- `flaggedSubmissions`: A list of flagged submissions.

10.4.2 Environment Variables

- None

10.4.3 Assumptions

- None

10.4.4 Access Routine Semantics

`getFlaggedStatus(submissionID: string)`:

- **transition:** None
- **input:** The unique `submissionID` for which the flagged status is queried
- **output:** Returns the flagged status (`true` or `false`) for the given `submissionID`

`setFlaggedStatus(submissionID: string, flag: boolean)`:

- **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.

`getAllFlagged()`:

- **transition:** None
- **input:** None
- **output:** Returns a list of all `submissionID`s that are currently flagged

10.4.5 Local Functions

No local functions are required for this module.

11 MIS of Report Results Module

11.1 Module

`ResultsModule`

11.2 Uses

- Frontend for rendering reports visually for users
- `ReportDataStruct` from Results Upload Module

11.3 Syntax

11.3.1 Exported Access Program

Name	In	Out	Exceptions
`renderReport`	report: ReportDataStruct	-	-

11.4 Semantics

11.4.1 State Variables

- `reports`: A list of reports to be displayed

11.4.2 Environment Variables

- None

11.4.3 Assumptions

- None

11.4.4 Access Routine Semantics

`renderReport(reports: ReportDataStruct)`:

- **input:** The report data (`ReportDataStruct`) of the report to be rendered
- **transition:** None
- **output:** Renders a visual representation of the report in the browser

11.4.5 Local Functions

No local functions are required for this module.

11.5 NLP Module

`NLPModule`

11.6 Uses

- Abstract ML Model Module
- Tokenization Module
- AST Module

11.7 Syntax

11.7.1 Exported Constants

- None

11.7.2 Exported Access Programs

Name	In	Out	Exceptions
`combinedPredict`	data: DataStruct	relations: combinedPrediction	-

11.8 Semantics

11.8.1 State Variables

- None

11.8.2 Environment Variables

- None

11.8.3 Assumptions

- The input code within DataStruct is in one programming language.

11.8.4 Access Routine Semantics

`combinedPredict`(data: DataStruct):

- **transition: None**
- **output: relations: combinedPrediction** an assembly of results from each of the used modules is combined to get a more balanced perspective
- **exception: None**

11.8.5 Local Functions

No local functions are required for this module.

12 MIS of Abstract Model Module

12.1 Module

`AbModelModule`

12.2 Uses

- `transformers` library

12.3 Syntax

12.3.1 Exported Constants

- None.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
`train`	data: DataStruct, timeout: integer	None	TimeOutException
`predict`	data: DataStruct	result: Prediction	-

12.4 Semantics

12.4.1 State Variables

- weightings

12.4.2 Environment Variables

- None

12.4.3 Assumptions

- The input code within DataStruct is in one programming language.

12.4.4 Access Routine Semantics

`train(data: DataStruct, timeout: integer)`:

- **transition:** weightings := updated_weightings – assign new weightings from batch training to weightings
- **output:** None
- **exception:** `TimeOutException` if training time exceeds time limit allotted.

`predict(data: DataStruct)`:

- **transition:** None
- **output:** Prediction object containing semantic relations between code snippets contained within the DataStruct data.
- **exception:** None

13 MIS of Tokenization Modlue

13.1 Module

`TokModule`

13.2 Uses

- None

13.3 Syntax

13.3.1 Exported Constants

- None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
`tokenize`	DataStruct	list[Token]	TokenizeError

13.4 Semantics

13.4.1 State Variables

- None

13.4.2 Environment Variables

- None

13.4.3 Assumptions

- The input code within DataStruct is in one programming language.

13.4.4 Access Routine Semantics

`tokenize(source: string)`:

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

13.4.5 Local Functions

`pollOneToken()`:

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

14 MIS of AST Module

14.1 Module

`ASTModule`

14.2 Uses

- Built in (for python) or external tokenizer and tree parsers

14.3 Syntax

14.3.1 Exported Constants

- None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
`parse`	rawSource: string	ASTNode	-

14.4 Semantics

14.4.1 Assumptions

- The expression provided to `parse` is syntactically correct or can be parsed with the given rules.

14.4.2 Access Routine Semantics

`parse(rawSource)`:

- **input:** A string representing the input source code
- **transition:** None
- **output:** Returns the root node of the AST, or None if the input was invalid

14.4.3 Local Functions

No local functions are required for this module.

14.5 Similarity Scoring Module

`SimScoreModule`

14.6 Uses

- NLP module

14.7 Syntax

14.7.1 Exported Constants

- None

14.7.2 Exported Access Programs

Name	In	Out	Exceptions
`score`	data: DataStruct	scores: Map[2-tuple[str]: list[Real]]	-

14.8 Semantics

14.8.1 State Variables

- None

14.8.2 Environment Variables

- None

14.8.3 Assumptions

- The input code within DataStruct is in one programming language.

14.8.4 Access Routine Semantics

`score`(data: DataStruct):

- **transition: None**
- **output: Map[2-tuple[str]: list[Real]]** # denote this more mathematically?
- **exception: None**

14.8.5 Local Functions

No local functions are required for this module.

14.9 Report Generation Module

`RepGenModule`

14.10 Uses

- Similarity Scoring Module

14.11 Syntax

14.11.1 Exported Constants

- None

14.11.2 Exported Access Programs

Name	In	Out	Exceptions
`generate`	data: DataStruct	report: Report-DataStruct	-

14.12 Semantics

14.12.1 State Variables

- None

14.12.2 Environment Variables

- None

14.12.3 Assumptions

- The input code within DataStruct is in one programming language.

14.12.4 Access Routine Semantics

`generate`(data: DataStruct):

- **transition: None**
- **output: ReportDataStruct** object wrapping visuals associated with report and similarity scorings to be received by the front end
- **exception: None**

14.12.5 Local Functions

`assembleVisuals`(data: DataStruct):

- **transition: None**
- **output:JSON**
- **exception: None**

15 MIS of Email Sending Module

15.1 Module

`EmailModule`

15.2 Uses

- SMTP server or email-sending service (such as SendGrid or Amazon SES)

15.3 Syntax

15.3.1 Exported Constants

- None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
`sendEmail`	recipient: string, subject: string, body: string, attachments: list[file]	boolean	-

15.4 Semantics

15.4.1 State Variables

- None

15.4.2 Environment Variables

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

15.4.3 Assumptions

- A SMTP server or email-sending service is available and configured correctly

15.4.4 Access Routine Semantics

```
sendEmail(recipient: string, subject: string,
body: string, attachments: list[file]):
```

- **input:**
  - `recipient`: Email address of the primary recipient
  - `subject`: Subject of the email
  - `body`: The main body content of the email (plain text or HTML)
  - `attachments`: A list of file objects to be attached to the email
- **transition:** None
- **output:** Returns `true` if the email is sent successfully; otherwise, throws `EmailSendException`.

15.4.5 Local Functions

No local functions are required for this module.

16 Appendix

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.