Module Guide for Software Engineering

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

March 26, 2025

1 Revision History

Date	Version	Notes
January 7, 2025	1.0	Initial document

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
FR	Functional Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	SyntaxSentinals Code Plagiarism Detector
UC	Unlikely Change
UI	User Interface
NLP	Natural Language Processing
ML	Machine Learning
AST	Abstract Syntax Tree
SSO	Single Sign On
MFA	Multi-Factor Authentication
SDK	Software Development Kit
HTTPS	Hypertext Transfer Protocol Secure
API	Application Programming Interface
JSON	JavaScript Object Notation

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.

- Each data structure is implemented in only one module.

- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.

- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.

- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are modifications that are likely to occur during the development or maintenance of the system. These changes are identified based on the project's goals, stakeholder feedback, and potential future requirements. By isolating these changes within specific modules, we ensure that the system remains flexible and maintainable.

The following are the anticipated changes for the SyntaxSentinals Code Plagiarism Detector:

AC1: Changes in the input format of code snippets, such as supporting additional programming languages (e.g., C++, JavaScript) or new file formats.

AC2: Upgrades to the NLP model to improve semantic understanding, such as incorporating newer machine learning techniques or larger training datasets.

AC3: Changes in the user interface, such as adding new features (e.g., online learning, language-agnostic support) or improving usability (e.g., better navigation, accessibility features).

AC4: Adjustments to the similarity threshold for plagiarism detection, allowing users to customize sensitivity levels based on their specific needs.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Switching from an NLP-based approach to a non-NLP-based approach.

UC2: Storing user data beyond the immediate task (violating the zero data retention policy).

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

User Interface Modules

M1: User Authentication Module

M2: Code Upload Module

M3: Threshold Adjustment Module

M4: Flagging Module

M5: Report Results Module

Backend Modules

M6: NLP Model Module

> **M6.1:** Abstract ML Model Module
>
> **M6.2:** Tokenization Module
>
> **M6.3:** AST Module

M7: Similarity Scoring Module

M8: Report Generation Module

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS (Khan et al., 2024). In this stage, the system is decomposed into modules. This section details the connection between functional requirements and modules to establish functionality of the design and essentiallity of the modules. The traceability section below will propose coverage of non-functional requirements by the modules. A mapping between functional requirements in the FR section of the SRS (Khan et al., 2024) and corresponding modules that provide coverage for them are also listed in Table 2. Note: mathematical notation used in the FRs ignores table of abbreviations and acronyms and derives from the SRS section cited above.

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

Req.	Modules
FR-1	M2
FR-2	M6 M7 M3
FR-3	M5
FR-4	M4
FR-5	M6 M7
FR-6	M8
FR-7	M1
FR-8	M1
FR-9	M?? M5

Table 2: Trace Between Requirements and Modules

FR-1 requires all code file inputs given by a user, $u \in U$, to be uploaded into the system, forming S. This necessitates a spot in the UI for users to insert code files and get them into circulation of the system for eventual processing by the model in the back end. These responsibilities will all be delegated to the code upload module (M2).

FR-2 requires all snippets to have a pairwise similarity score, $\mathrm{Sim}(s_i, s_j)$, and for these scores to be sent against a threshold, $t \in T$, to establish plagiarism status. To gain these

similarity scores, there must be semantic relation established between pairs (such as a vector) to calculate scores from. The NLP module M6 will be tasked with building the semantic relationship and the Similarity Scoring module M7 will handle the proceeding score calculation. To translate similarity scores into plagiarism status, thresholds will need to be obtained from user preferences and subsequently compared with scores. The Threshold module (M3) will take over these responsibilities necessary for obtaining plagarism status.

FR-3 requires a guide document. This will be provided by the results module (M5) in the frontend which will encompass both displaying analysis results to the user as well as info on how to interpret them.

FR-4 requires allowing the user to flag code snippets for their own tracking purposes to create $P \subset S$ as well as flagging pairs that are suspected for plagarism. This gives scope for a flagging module (M4) which will provide the ability to mark snippets after analysis for the user to keep special track of and potentially filter with.

FR-5 requires the collective output of similarity scores, $\text{Sim}(s_i, s_j)$, to exist regardless of threshold filtering. This should arise as an output of the scoring module (M7) which will interpret results from the NLP module (M6).

FR-6 requires generation of reports, R, using similarity scores, $\text{Sim}(s_i, s_j)$, and thresholds that already exists. This necessitates a module which can take a set of scores and thresholds that have been obtained from other modules, and make a summary report. This will be covered by the Report Generation module (M8).

FR-7 requires account creation to add a member, u, to the set of users, U, necessitating the existence of the user authentication module (M1).

FR-8 requires authenticating an account of a user, u. This falls into the scope of the user authentication module (M1).

FR-9 requires the ability to visualize a completed job via the UI. This will be handled by the results module (M5) which will take the report generated by the backend and display it to the user.

7   Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing

software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Software Engineering* means the module will be implemented by the Software Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1   Hardware Hiding Modules

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2   Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1   User Authentication Module (M1)

Secrets: The authentication and authorization mechanisms.

Services: Handles user account creation, login, and access control.

Implemented By: Auth0

Type of Module: Library

7.2.2   Code Upload Module (M2)

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

Services: Sets up pathway to receive the input data files and tranforms it into the data structure suitable for backend.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.3 Threshold Adjustment Module (M3)

Secrets: The logic for adjusting plagiarism detection thresholds.

Services: Allows users to customize the similarity threshold for plagiarism detection according to their needs.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.4 Flagging Module (M4)

Secrets: The logic for managing and storing the state of user-flagged submissions.

Services: Allows users to flag specific code snippets or results for further review.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.2.5 Report Results Module (M5)

Secrets: The logic for requesting and displaying plagiarism report from the backend.

Services: Gives spot to initiate plagarism analysis and, upon completion, visualizes results to allow user to inspect the findings of the analysis.

Implemented By: Software Engineering

Type of Module: Abstract Object

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 NLP Model Module (M6)

Secrets: The NLP-based plagiarism detection algorithms and functions.

Services: Processes code snippets and generates semantic representations (such as a vector) for similarity scoring.

Implemented By: Software Engineering

Type of Module: Library

7.3.2 Abstract ML Model Module (M6.1)

Secrets: The architecture and configuration of machine learning models.

Services: Provides a high-level interface for tuning and predicting using various pre-trained machine learning models, agnostic of the specific model to be used.

Implemented By: Hugging Face

Type of Module: Abstract Object

7.3.3 Tokenization Module (M6.2)

Secrets: The tokenization algorithms used to break code into smaller, meaningful components for analysis.

Services: Converts raw code snippets into tokens (e.g., keywords, operators) that can be processed by other modules.

Implemented By: Software Engineering

Type of Module: Library Component

7.3.4 AST Module (M6.3)

Secrets: The structure and traversal methods of the Abstract Syntax Tree (AST) used to represent code.

Services: Transforms source code into an Abstract Syntax Tree for semantic analysis and comparison.

Implemented By: Software Engineering

Type of Module: Library

7.3.5 Similarity Scoring Module (M7)

Secrets: The algorithm for calculating similarity scores.

Services: Compares semantic representations of code snippets and generates similarity scores.

Implemented By: Software Engineering

Type of Module: Library

7.3.6 Report Generation Module (M8)

Secrets: The logic for aggregating results of the NLP model and similarity scores.

Services: Produces a report with both direct and aggregated data from plagarism analysis.

Implemented By: Software Engineering

Type of Module: Abstract Object

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Traceability table between modules and the funcitonal requirements can be found at M2.

Req.	Modules
UH-L1	M5
PR-SL1	M6 M7
PR-SL2	M5
PR-PA1	M6 M7
PR-PA2	M6 M7
PR-RFT1	M2
PR-C1	M6 M7
PR-L2	M6
OE-IAS1	M8 M6
OE-IAS3	M1
MS-M1	M6
MS-A2	M6
MS-A3	M6
SR-A1	M1
SR-P1	M2 M8
SR-P2	M2
CR-L1	M2 M8
CR-L2	M2 M8
CR-SC4	M6
CR-SC5	M1 M2 M8

Table 3: Trace Between Relevant Non-Functional Requirements and Modules

AC	Modules
AC1	M2 M6
AC2	M6
AC3	M5 M1
AC4	M3

Table 4: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.
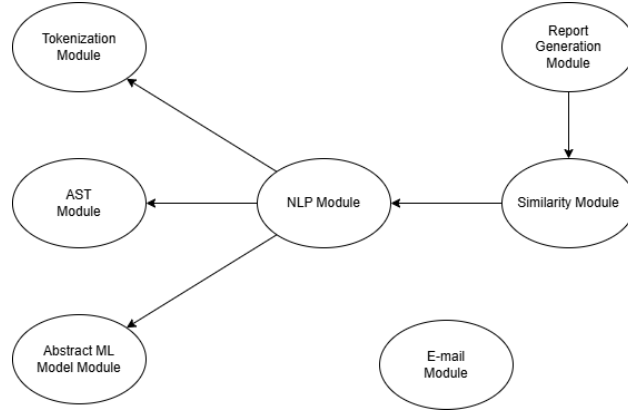
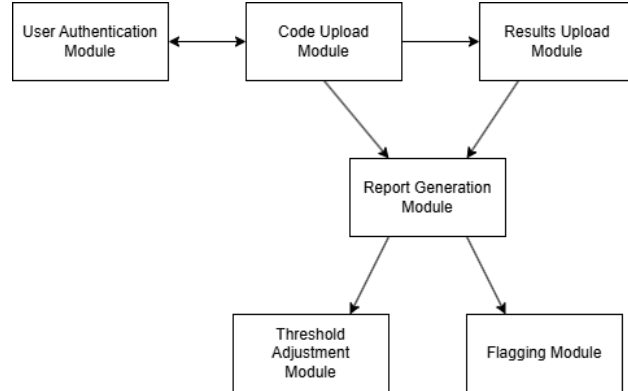Figure 1: Use hierarchy among back end modules

Figure 2: Use hierarchy among front end modules

11

10 User Interfaces

The below are mockups of the user interface for the software and are subject to change based on user feedback and design decisions.
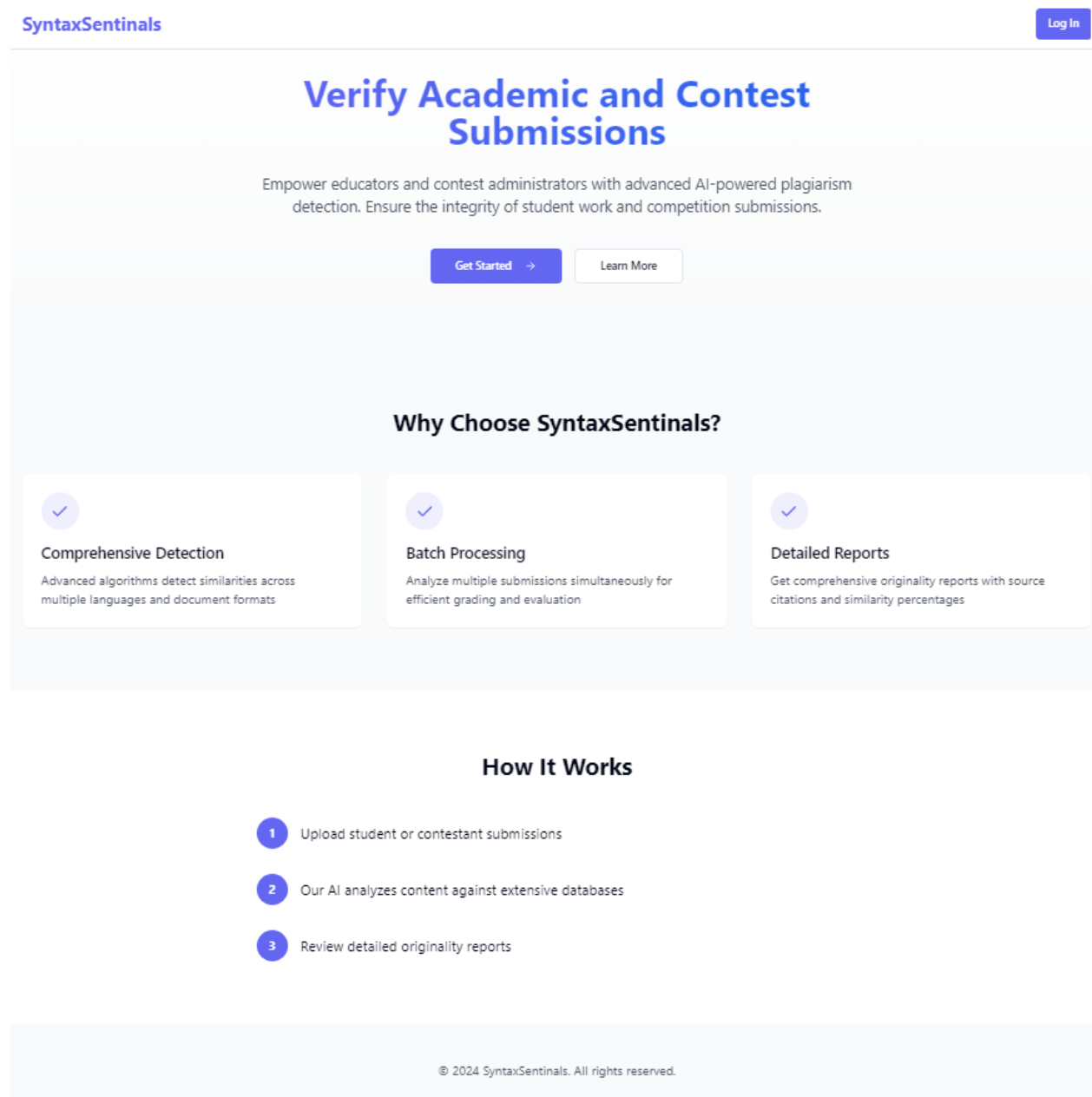

Figure 3: Landing Page

Analyze a dataset

Upload a dataset to analyze for potential code plagiarism.

Upload a file or drag and drop

ZIP files only, up to 50MB

Analysis name

Enter a name for this analysis

Analyze Dataset

View Results

Check the analysis results of your previous submissions by uploading the results zip file emailed to you.

Upload a dataset to get started.

Figure 4: Home Page

Source Code Plagiarism Detection Report

Analysis results for submitted code files
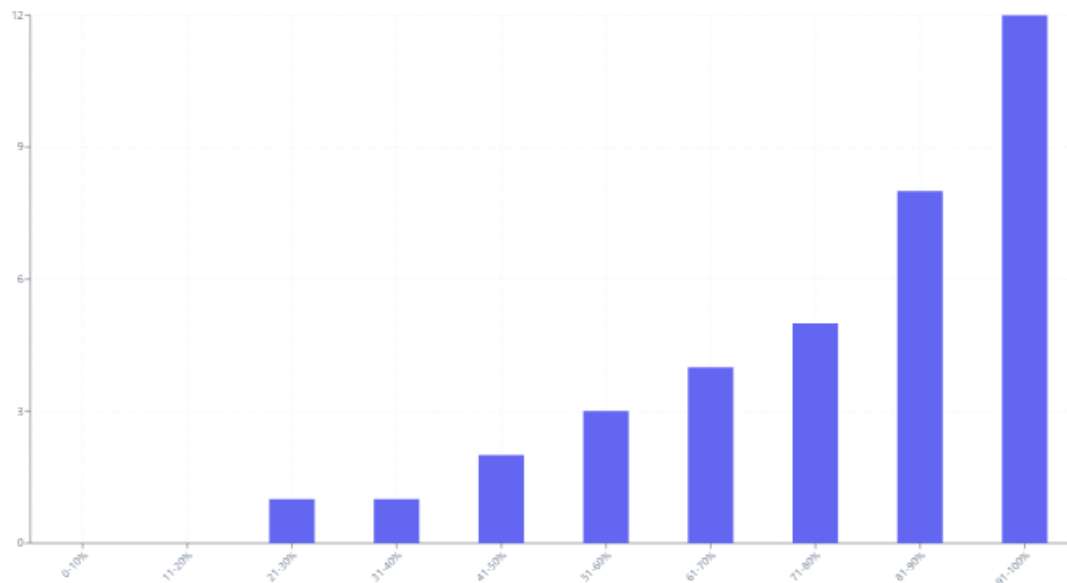
Highest Similarity	Average Similarity	Total Submissions
100%	**91%**	**5**
	Median: 79%	

Similarity Distribution

Similar Submissions

Highlights the most suspicious individual submissions, useful for exams

File 1	File 2	Similarity
01022431.py	01141513.py	100%
02085216.py	02447653.py	100%
02951809.py	03234141.py	95%
04416985.py	08165126.py	87%
08316564.py	08909969.py	75%

Figure 5: Results Page

Figure 6: settings Page

11 Design of Communication Protocols

This section outlines the external services that will be used by the software and their respective purposes.

11.1 External Services Overview

The system will integrate with Auth0 to enhance functionality and provide a seamless user experience.

11.2 Authentication Service: Auth0

Auth0 will be used for user authentication and authorization. It provides secure login, single sign-on (SSO), and multi-factor authentication (MFA) capabilities. Auth0 will handle user credentials and ensure that only authorized users can access the system.

11.3 Integration and Security

Auth0 will be integrated using its Javascript SDK. Secure communication channels (e.g., HTTPS) will be used to protect data in transit. Additionally, proper authentication and authorization mechanisms will be implemented to ensure that only authorized components can interact with Auth0.

12 Timeline

12.1 Module Implementation Timeline (Starting Jan 17)

12.1.1 User Interface (UI) Modules

- **Jan 17 - Jan 20: User Authentication Module (1)**

  - Develop UI for user registration and login.
  - Integrate with backend authentication services.
  - **Responsibility:** Lucas.

- **Jan 21 - Jan 25: Code Upload Module (2)**

  - Implement the file upload interface.
  - Validate and display upload status to the user.
  - **Responsibility:** Mohsin.

- **Jan 26 - Feb 4: Threshold Adjustment Module (3)**

  - Develop an interface for adjusting similarity thresholds.
  - Provide sliders or input boxes for customization.
  - **Responsibility:** Mohsin.

- **Feb 5 - Feb 8: Flagging Module (4)**

  - Add functionality for users to flag suspicious results.
  - Ensure flagged items are visually distinct in the UI.
  - **Responsibility:** Mohsin.

- **Feb 9 - Feb 14: Report Results Module ([5])**

    - Implement a results display interface with options for sorting and filtering.
    - Add the ability to download reports directly from the UI.
    - **Responsibility:** Lucas and Mohsin.

12.1.2 Backend Modules

- **Jan 17 - Jan 23: NLP Model Module ([6])**

    - Finalize and integrate the trained plagiarism detection model.
    - **Responsibility:** Dennis, Luigi, and Julian.

- **Jan 17 - Jan 29: Similarity Scoring Module ([7])**

    - Process NLP model outputs for similarity scores.
    - Optimize backend logic for performance.
    - **Responsibility:** Dennis, Luigi, and Julian.

- **Jan 30 - Feb 8: Report Generation Module ([8])**

    - Generate reports based on processed similarity data.
    - Ensure compatibility with frontend requirements.
    - **Responsibility:** Entire team.

References

Mohammad Mohsin Khan, Lucas Chen, Dennis Fong, Julian Cecchini, and Luigi Quattrociocchi. System requirements specification. https://github.com/SyntaxSentinels/SyntaxSentinels/blob/main/docs/SRS-Volere/SRS.pdf, November 2024.

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.