Verification and Validation Report: Software Engineering

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

April 6, 2025

1 Revision History

Date	Version	Notes
March 10th, 2025	1.0	First iteration of rough draft complete
April 4th, 2025	2.0	Final version of report complete

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
SRS	Software Requirements Specification
MOSS	Measure of Software Similarity
UI	User Interface
JSON	JavaScript Object Notation
NPM	Node Package Manager
VnV	Verification and Validation
API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Deployment

Contents

List of Tables

3 Functional Requirements Evaluation

3.0.1 Plagiarism Analysis Input Upload Tests

Applies to all FRs involving uploading code snippets to the system. This refers to test-FR-1 from the VnV Plan.

1. **test-FR-1**

   Expected result: The script should run and successfully upload the files, without encountering any errors due to the upload functionality

   Actual Result: No errors from uplading

   Evaluation: Pass. Uploading 300 files python files in zip format produced no errrors.

3.0.2 Plagiarism Analysis Result Tests

Applies to all FRs involving immediate output of plagiarism analysis. This refer to test-FR-2 from the VnV Plan.

1. **test-FR-2**

   Expected result: The analysis should not throw any errors while the script runs, up until the analysis provides a response

   Actual result: No errors were observed during script execution

   Evaluation: Pass. Processed 300 python files and produced similarity scores for each pair.

3.0.3 Guide Documentation Generation Tests

Applies to all FRs involving guide documentation generation for user. This refers to test-FR-3 from the VnV plan.

1. **test-FR-3**

   Expected result: Generation of documentation successful

   Actual result: Instructive prompts are shown on upload page to provide inherent guidance to user and a link to additional extra user documentation (hosted on github) is provided.

   Evaluation: Pass

3.0.4 Report Documentation Generation Tests

Applies to all FRs involving analysis report documentation generation for user. This refers to test-FR-4 from the VnV plan.

1. **test-FR-4**

   Expected result: Generation of report is successful, with no errors encountered

   Actual result: Report was successfully generated, no errors were found

   Evaluation: Pass. 300 python files given to the system receive a corresponding report of all pairwise similarity scores and a threshold score.

3.0.5 Account Creation Tests

Applies to all FRs involving creating an account within the system. This refers to test-FR-5 and test-FR-6 from the VnV plan

1. **test-FR-5**

   Expected result: An account should be successfully created, while the system avids any failure modes.

   Actual result: Account successfully created, and no failure modes were exhibited

   Evaluation: Pass

2. **test-FR-6**

   Expected result: Account creation should fail and the system will await further input from the user

   Actual result: Account creation fails, and the system does not proceed

   Evaluation: Pass

3.0.6 Account Login Tests

Applies to all FRs involving logging into account within the system. This refers to test-FR-7 and test-FR-8 in the VnV plan

1. **test-FR-7**

   Expected result: Logging into system with correct existing credentials should succeed, and no errors should arise from logging in

   Actual result: Logging in with existing credentials succeeds with no errors

   Evaluation: Pass

2. **test-FR-8**

   Expected result: Logging in with invalid credentials should fail

   Actual result: Logging in with invalid credentials fails

   Evaluation: Pass

3.0.7 Result Visualization Tests

Applies to all FRs involving visualizing plagiarism analysis results. This refers to test-FR-9 from the VnV plan.

1. **test-FR-10**

   Expected result: A visualization is successfully created after uploading a .zip file, with no errors observed during the process.

   Actual result: A visualization was successfully created with no errors

   Evaluation: Pass. 300 python files in a .zip given to the system receive a corresponding visualization of pair-by-pair breakdowns of where similarity was found along with a histogram that aggregates similarity scores.

4 Nonfunctional Requirements Evaluation

Note: for this section, users involved with testing are expected to have typical abilities with technology (such as being able to navigate through a touchscreen phone and basic websites). No particular expertise in software or computers is expected.

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-LF-1	LF-AR1, LF-AR2, LF-AR3, LF-SR1, LF-SR2, LF-SR3, LF-SR4	Developer reviews UI code and interface for consistency with the SRS elements specified in VnV plan such as uniform colour palette.	UI should follow defined layout, typography, alignment, color palette, tooltips, responsiveness, and font.	Developer verified uniform color palette, tooltips, responsive design, and font consistency. All requirements were met.	Pass. Monochromatic blue theme was utilized for color palette. Aspects were checked at screen sizes/resolutions of 1920 x 1080 pixels (Full HD) and 1366 x 768 (HD).

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-LF-2	CR-SC3	Developer utilizes accessibility tools to assess sufficient readability and alt-text for images	All images should have alt text, and text should meet contrast and sizing standards.	Developer verified all images have alt text and all text meets contrast and sizing requirements using Chrome DevTools.	Pass
test-UH-1	UH-E1, UH-E3, UH-L1, UH-UP1, UH-UP2, OE-P1	User goes through the onboarding process and provides feedback on usability.	User finds onboarding process intuitive with no major issues; usability rating should be 4/5 or greater.	5 users tested the onboarding process. All rated ease of use and clarity at 4/5 or greater. Usability Test Results Table.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-UH-2	UH-E2	User navigates through key functions of code submission and report upload	User should complete each task with minimal clicks required.	5 users tested navigation to code submission and report upload, reporting 2-3 clicks each from upload to select to run. Navigation Test Results Table.	Pass
test-UH-3	UH-PI2	Tester verifies that all pages and prompts are in English (US)	No foreign languages should be present on any page or prompt.	Tester verified that all pages and prompts were in English (US), no foreign languages found.	Pass
test-V-1	MS-M1, OR-R2	Developer reviews GitHub repository for historical versions and changelogs	All previous versions should be stored with performance reports and changelogs.	Developer confirmed that historical versions with performance reports and changelogs are accessible on GitHub.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-PS-1	SR-P1	Tester performs a data retention audit to identify insecurely stored user-data	No user-generated data should be found within databases, filesystem, or logs, to be stored insecurely or not be ready for deletion.	Tester found no instances of insecurely stored user-generated data during testing, all data was securely stored. Logs were clean.	Pass
test-PS-2	SR-P2	Tester verifies that backend endpoints use HTTPS encryption	All endpoints should use HTTPS with proper security headers.	Tester confirmed all backend endpoints used HTTPS with proper security headers as shown with REST API utilized.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-PS-3	CR-L1, CR-L2, CR-L3, CR-S2, CR-S5	Legal audit of the system against regulations like PIPEDA, FIPPA, etc.	System should adhere to legal standards without any breaches.	Third-party tester verified the system's compliance with PIPEDA, FIPPA, and other regulations. No breaches found. This is inherent by only utilizing publicly available repos and data	Pass
test-DS-1	MS-M3	Tester inspects the GitHub repository for bug tracking.	Bugs should be listed as issues in the repository.	Tester confirmed bugs are listed in the 'issues' section on GitHub. Chores and fixes have been dealt with as issues	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-DS-2	MS-S1, CR-SC1	Team inspects code to ensure adherence to software development standards	Code should follow SOLID principles, and be documented with appropriate tests.	Team inspected code, confirming adherence to SOLID principles, documentation, and test presence. Code for backend application follows direct interface hiding rules and currently complies. Our frontend works on a component based development path and therefore adheres too.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-D-1	UH-L2, OE-P2	Developer reviews user documentation for clarity, accuracy, and helpfulness	Documentation should be clear, non-technical, and helpful to users.	Help documentation is still to be developed per extra but current README files exist to specify steps for application use in clear and basic language for users.	Pass with exception to extra
test-PR-1	PR-SL1, PR-C1	System processes 500 code snippets for plagiarism analysis within 4 hours	Processing should complete in under 4 hours.	System completed processing within 4 hours, performance met expectations.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-PR-2	PR-RFT1	Tester submits a corrupted file and verifies system response	System should reject the corrupted input with an error message without crashing.	System rejected the corrupted input with an error message. Valid input processed without issue. i.e., non-python or tar package file creates error which frontend appropriately communicates.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-PR-3	PR-PA1, PR-PA2	Tester submits 500 code snippets for plagiarism analysis and compares results with ground truth.	System should show over 90% accuracy and less than 5% false positives.	Recently found method to test this through self-labelled datasets as it was not immediately possible from python800 datasets (no concept of similarity or ground truths) Have spent time labelling hundreds of code in manual fashion to assess this and have not had time to test it.	Future Assessment

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-PR-4	PR-SE1	Developer adds a new feature, and all existing features are tested for regression.	All previous features should work as expected after the new feature is added.	Regression tests confirmed that all features worked as expected after the new feature was added but not server side yet	Fail
test-OE-1	OE-IAS1	User connects system to a cloud service (e.g., AWS or Azure) and verifies successful connection.	System should successfully connect to cloud service and display a confirmation message.	Cloud has yet to be connected. It is a later stage development process	Future Assessment
test-OE-2	OE-IAS2	User deploys the system on a free hosting service and verifies system access after cloning from github.	System should be accessible via a public URL and function as expected.	System has not been deployed to AWS or other free hosting services yet. Later stage development.	Future Assessment

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-OE-3	OE-IAS3	User authenticates via an external service (e.g., Google, GitHub) and gains access to the UI.	System should authenticate user and redirect to the home page.	User successfully authenticated via Google through Auth Zero and was redirected to the home page.	Pass
test-M-1	MS-M2	Model release is accompanied by a report of metrics such as accuracy, precision, etc.	A report should be generated with relevant metrics for the model release.	Current model release has metrics on microsoft althought future releases will have a new report attached and that is part of later stage development	Future Assessment
test-M-2	MS-S3	User posts a request or issue on GitHub, verifying visibility to others	The request or issue should be visible to other users.	Request posted on GitHub was visible to other users as expected.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-M-3	MS-A2	Model code follows template, and components inherit interfaces/classes.	Model components should follow the predefined template and interface inheritance.	Code was inspected and confirmed that all components inherited from interfaces/classes as required. Model app structure follows SOLID principles and model itself is masked in pytorch.	Pass
test-M-4	MS-A1	Training script receives two formats of training data and completes an epoch.	Training script should successfully complete an epoch with both data formats.	Training script completed successfully with both data formats (java and python files) and have corresponding checkpoint files to show.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-M-5	MS-A3	Each model layer/component is tested with an appropriate input vector and verifies transformation.	Each layer/component should modify the input vector according to its specific architecture.	Model layers were tested, and all correctly transformed input vectors according to design. Pytorch and scipy layers used are atomic in nature and can be mixed as desired	Pass
test-S-1	SR-A1	User enters valid credentials and accesses the non-login screen UI (i.e., UI for code uploading, etc.).	User should be authenticated and granted access to the non-login screen UI.	User entered valid credentials and successfully accessed the non-login screen UI.	Pass
test-S-2	SR-A1	User submits code to API with valid auth token.	API should allow code upload and return a success response.	API accepted valid auth token and processed code upload successfully.	Pass

ID	Reference Requirements	User Action	Expected Result	Actual Result	Result
test-S-3	SR-A1	User attempts to access the API with invalid or missing auth token.	API should deny access and return an unauthorized response.	API denied access when provided with invalid or missing auth token.	Pass

5 Comparison to Existing Implementation

5.1 Overview of Existing Solutions

5.1.1 MOSS (Measure of Software Similarity)

MOSS is currently the standard tool for detecting code plagiarism in academic settings. Developed at Stanford University, it works by:

- Tokenizing code into a sequence of symbols
- Using document fingerprinting to detect similar code segments
- Generating a similarity score based on matching sequences
- Providing a web interface to view overlapping code segments

5.1.2 Other Notable Solutions

- **JPlag**: Uses a token-based approach similar to MOSS, but with different tokenization strategies
- **PLAGGIE**: Java-specific plagiarism detection tool used in academic environments
- **CodeMatch**: Commercial solution that uses both tokenization and metric-based methods

5.2 Our Implementation vs. Existing Solutions

5.2.1 Current Approach

Our implementation leverages CodeBERT, a pre-trained model for programming language understanding, to perform code pair comparison. Key aspects include:

- Utilization of transformer-based neural networks to understand code semantics
- Direct comparison of code pairs to determine similarity
- A modern, user-friendly interface for result visualization and analysis

5.2.2 Functional Comparison

Feature	**MOSS**	**Our CodeBERT Implementation**
Detection Method	Token-based fingerprinting	Neural representation and semantic understanding
Language Support	Multiple languages	Multiple languages (supported by CodeBERT)
Semantic Understanding	Limited (syntax-focused)	Enhanced (captures semantic meaning)
Resistance to Obfuscation	Low-moderate	Potentially higher
Speed	Fast for large submissions	Currently slower for large-scale comparisons
Visualization	Basic web interface	Modern, interactive UI

Table 2: Comparison between MOSS and our CodeBERT implementation

5.3 Advantages of Our Approach

1. **Semantic Understanding**: Unlike MOSS's purely syntactic approach, CodeBERT can potentially understand the meaning behind code, making it more difficult to fool with non-functional additions.

2. **Context Awareness**: Our implementation considers the context in which code appears, potentially reducing false positives from common solutions to standard problems.

3. **Modern Interface**: Our user interface provides a more intuitive experience for instructors reviewing potential plagiarism cases.

5.4 Current Limitations and Trade-offs

1. **Computational Requirements**: Neural models like CodeBERT require more computational resources than traditional approaches like MOSS.

2. **Development Maturity**: MOSS has been refined over decades, while our solution is still in the early stages of development.

3. **Validation**: Our approach needs more extensive testing across different programming languages and assignments to prove its effectiveness.

4. **Explainability**: Neural network decisions can be less transparent than token-matching approaches, making it potentially harder to explain why certain code pairs were flagged.

5.5 Design Decision Justification

1. **Choice of CodeBERT**: We selected CodeBERT over other models because it was specifically pre-trained on code from multiple programming languages, making it well-suited for understanding code semantics across different languages used in academic settings.

2. **Focus on Pair Comparison**: While MOSS compares each submission against all others, our current approach focuses on direct pair comparison, allowing for more detailed analysis of potentially plagiarized code.

3. **UI Priority**: We invested in a high-quality UI early in development to facilitate easier testing and validation of our detection results.

5.6 Future Improvements

1. **Hybrid Approach**: Combining neural understanding with traditional token-based methods to leverage strengths of both approaches.
2. **Performance Optimization**: Improving the computational efficiency to handle large class submissions more effectively.
3. **Tunable Sensitivity**: Implementing adjustable thresholds for different assignment types and programming languages.

6 Unit Testing

6.1 AnalyzeFiles Component

6.1.1 Renders the component

- **Description**: Ensures the `AnalyzeFiles` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Check for the presence of specific text.
- **Expected result**: The component renders with the text "Analyze a Dataset" and "Upload a dataset to analyze for potential code plagiarism. Use a ZIP file containing your project files."

6.1.2 Displays an alert when no file is uploaded

- **Description**: Ensures an alert is displayed when no file is uploaded.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:

1. Render the `AnalyzeFiles` component.
2. Click the "Analyze Dataset" button.

- **Expected result**: An alert with the message "Please upload a dataset file first." is displayed.

6.1.3 Displays an alert when no analysis name is entered

- **Description**: Ensures an alert is displayed when no analysis name is entered.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Upload a valid ZIP file.
  3. Click the "Analyze Dataset" button.
- **Expected result**: An alert with the message "Please enter an analysis name." is displayed.

6.1.4 Calls uploadFiles and displays success message when form is submitted

- **Description**: Ensures the `uploadFiles` function is called and a success message is displayed when the form is submitted.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Upload a valid ZIP file.
  3. Enter an analysis name.
  4. Click the "Analyze Dataset" button.

- **Expected result**: The `uploadFiles` function is called with the file and analysis name, and a success message "Analysis started successfully" is displayed.

6.1.5 Handles file drop event

- **Description**: Ensures the file drop event is handled correctly when a file is dropped on the component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Simulate a file drop event with a valid ZIP file.
- **Expected result**: The file is uploaded successfully.

6.1.6 Handles empty job list

- **Description**: Ensures the component handles an empty job list correctly by showing "No Data".
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Set the job list to an empty array.
  3. Check for the presence of "No Data" text.
- **Expected result**: The text "No Data" is displayed when the job list is empty.

6.1.7 Displays an error message if fetching jobs fails

- **Description**: Ensures an error message is displayed if fetching jobs fails.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `AnalyzeFiles` component.
  2. Mock the API call to return an error.
- **Expected result**: An error message "Failed to load job history" is displayed.

6.2 Index (Landing Page) Component

6.2.1 Renders the component

- **Description**: Ensures the `Index` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `Index` component.
  2. Check for the presence of specific text.
- **Expected result**: The component renders with the text "Verify Academic and Contest Submissions" and "Empower educators and contest administrators with advanced AI-powered plagiarism detection. Ensure the integrity of student work and competition submissions."

6.2.2 Displays loading spinner when loading

- **Description**: Ensures the loading spinner is displayed when the component is in the loading state.
- **Type**: Automatic, Functional
- **Initial state**: `isLoading` is true
- **Test case steps**:
  1. Mock the `useAuth0` hook to return `isLoading` as true.
  2. Render the `Index` component.
- **Expected result**: The text "Preparing your experience..." is displayed.

6.2.3 Calls loginWithRedirect when Get Started button is clicked

- **Description**: Ensures the `loginWithRedirect` function is called when the "Get Started" button is clicked.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `Index` component.
  2. Click the "Get Started" button.
- **Expected result**: The `loginWithRedirect` function is called.

6.2.4 Navigates to /home when authenticated

- **Description**: Ensures the user is redirected to the `Home` page when authenticated.
- **Type**: Automatic, Functional
- **Initial state**: None

- **Test case steps**:
    1. Mock the `useAuth0` hook to return `isAuthenticated` as true.
    2. Render the `Index` component.
- **Expected result**: The user is redirected to the `Home` page.

6.2.5 Displays features section

- **Description**: Ensures the features section is displayed on the `Index` component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
    1. Render the `Index` component.
- **Expected result**: The features section is displayed with the text "Why Choose SyntaxSentinels?" along with the features.

6.2.6 Displays how it works section

- **Description**: Ensures the "How It Works" section is displayed on the `Index` component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
    1. Render the `Index` component.
- **Expected result**: The "How It Works" section is displayed along with the steps.

6.2.7 Displays footer

- **Description**: Ensures the footer is displayed on the `Index` component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `Index` component.
- **Expected result**: The footer is displayed with the text "SyntaxSentinels" and the links.

6.3 Upload File Box Component

6.3.1 Renders the component

- **Description**: Ensures the `UploadFileBox` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `UploadFileBox` component.
- **Expected result**: The component renders with the text "Upload a ZIP file" and the file input.

6.3.2 Uploads a valid file

- **Description**: Ensures a valid file is uploaded using the `UploadFileBox` component.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:

1. Render the `UploadFileBox` component.
2. Upload a valid ZIP file.

- **Expected result**: The file is uploaded successfully.

6.3.3 Removes a file

- **Description**: Ensures a file is removed using the `UploadFileBox` component.
- **Type**: Automatic, Functional
- **Initial state**: A file is uploaded
- **Test case steps**:
  1. Render the `UploadFileBox` component.
  2. Click the "Remove" button.
- **Expected result**: The file is removed.

6.3.4 Clears the file list when clearFiles is true

- **Description**: Ensures the file list is cleared when the `clearFiles` prop is true.
- **Type**: Automatic, Functional
- **Initial state**: A file is uploaded
- **Test case steps**:
  1. Render the `UploadFileBox` component with the `clearFiles` prop set to true.
- **Expected result**: The file list is cleared.

6.4 NavBar Component

6.4.1 Renders the component

- **Description**: Ensures the `NavBar` component renders correctly.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the `NavBar` component.
- **Expected result**: The component renders with the logo and navigation links.

6.4.2 Displays the login button when not authenticated

- **Description**: Ensures the login button is displayed when the user is not authenticated.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Mock the `useAuth0` hook to return `isAuthenticated` as false.
  2. Render the `NavBar` component.
- **Expected result**: The login button is displayed.

6.4.3 Displays the logout button when authenticated

- **Description**: Ensures the logout button is displayed when the user is authenticated.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:

1. Mock the `useAuth0` hook to return `isAuthenticated` as true.
2. Render the `NavBar` component.

- **Expected result**: The logout button is displayed.

6.4.4 Calls loginWithRedirect when Log In button is clicked

- **Description**: Ensures the `loginWithRedirect` function is called when the "Log In" button is clicked.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
-
  1. Mock the `useAuth0` hook to return `isAuthenticated` as false.
  2. Render the `NavBar` component.
  3. Click the "Log In" button.
- **Expected result**: The `loginWithRedirect` function is called.

6.5 JobsTable Component

6.5.1 Renders the component with correct columns and data

- **Description**: Ensures the `JobsTable` component renders correctly with the correct columns and data.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
-
  1. Render the `JobsTable` component with sample data.
  2. Check for the presence of specific columns and data and expected text such as "Status", "Created At", etc.
  3. Check for the presence of specific data in the table rows such as "In Progress", "Completed", etc.

- **Expected result**: The component renders with the correct columns and data.

6.5.2 Displays a loading spinner when loading

- **Description**: Ensures the loading spinner is displayed when the component is in the loading state.
- **Type**: Automatic, Functional
- **Initial state**: `isLoading` is true
- **Test case steps**:
  1. Render the component with the loading prop set to True.
  2. Render the `JobsTable` component.
- **Expected result**: The loading spinner is displayed.

6.5.3 Does not display loading spinner when not loading

- **Description**: Ensures the loading spinner is not displayed when the component is not in the loading state.
- **Type**: Automatic, Functional
- **Initial state**: `isLoading` is false
- **Test case steps**:
  1. Render the component with the loading prop set to False.
  2. Render the `JobsTable` component.
- **Expected result**: The loading spinner is not displayed.

6.5.4 Disables the "View Results" button for non-completed jobs

- **Description**: Ensures the "View Results" button is disabled for non-completed jobs.
- **Type**: Automatic, Functional

- **Initial state**: None
- **Test case steps**:
    1. Render the component with a job that is not completed.
    2. Check the "View Results" button state.
- **Expected result**: The "View Results" button is disabled.

6.5.5 navigates to the results page when 'View Results' is clicked for a completed job

- **Description**: Ensures the user is navigated to the results page when the "View Results" button is clicked for a completed job.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
    1. Render the component with a job that is completed.
    2. Click the "View Results" button.
- **Expected result**: The user is navigated to the results page.

6.5.6 Formats dates correctly

- **Description**: Ensures the dates are formatted correctly in the table.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
    1. Render the component with sample data containing dates.
    2. Check the date format in the table rows.
- **Expected result**: The dates are formatted correctly (e.g., "MM/DD/YYYY").

6.5.7 renders 'Unnamed Analysis' if analysisName is missing

- **Description**: Ensures "Unnamed Analysis" is displayed if the analysis name is missing.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the component with a job that has no analysis name.
  2. Check the analysis name in the table row.
- **Expected result**: The text "Unnamed Analysis" is displayed.

6.5.8 renders the correct tag color and icon based on job status

- **Description**: Ensures the correct tag color and icon are displayed based on the job status.
- **Type**: Automatic, Functional
- **Initial state**: None
- **Test case steps**:
  1. Render the component with a job that has a specific status (e.g., "In Progress", "Completed").
  2. Check the tag color and icon in the table row.
- **Expected result**: The correct tag color and icon are displayed based on the job status.

7 Changes Due to Testing

During the testing process, several issues were identified and addressed, leading to improvements in both functionality and reliability:

- **Upload File Handling**

  - An issue was detected in `UploadBox.tsx`, where files were not properly validated before being uploaded. Additional validation logic was added to prevent incorrect file types from being processed.
  - A missing error message for unsupported file types was implemented to improve user feedback.

- **Error Handling in File Processing**
  - In `UploadResults.tsx`, tests revealed that errors during ZIP file processing were not handled gracefully. Additional error handling was introduced to catch and display meaningful error messages when invalid or corrupted ZIP files were uploaded.

- **Form Validation Enhancements**
  - The `AnalyzeFiles.tsx` component initially allowed form submission without an analysis name, causing backend errors. A required field check was added to prevent submissions without a name.

- **Test Coverage Improvements**
  - Some uncovered lines in `button.tsx` and `UploadBox.tsx` were identified through test coverage reports. Additional unit tests were written to cover these missing scenarios, increasing overall test coverage.
  - Server side code was not tested with a testing framework. This will be implemented in the future.

- **Cloud**
  - Cloud deployment is a future feature that will be implemented once the model is certified per priorities

8 Automated Testing

The tests mentioned in the `Unit Testing` section are automated using the Jest testing framework. The tests are run using the command `npm test`.

To ensure continuous integration and automated testing, we have set up a GitHub Actions workflow. The workflow is triggered on every push and pull request to the repository. It runs the tests on an Ubuntu environment with Node.js version 18.x.

8.1 Testing Workflow Configuration

The following is the configuration of the GitHub Actions workflow used for automated testing:

```
name: Run Frontend Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-20.04

    strategy:
      matrix:
        node-version: [18.x]

    steps:
    - name: Checkout repository
      uses: actions/checkout@v4

    - name: Set up Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${{ matrix.node-version }}

    - name: Install dependencies
      run: npm install
      working-directory: ./src/frontend

    - name: Run tests
      run: npm test
      working-directory: ./src/frontend
```

8.2 Testing Workflow Steps

- **Checkout repository**: This step uses the `actions/checkout@v4` action to check out the repository code.
- **Set up Node.js**: This step uses the `actions/setup-node@v4` action to set up Node.js with the specified version (18.x).
- **Install dependencies**: This step runs `npm install` to install the necessary dependencies for the frontend.
- **Run tests**: This step runs `npm test` to execute the Jest tests for the frontend.

By using this automated testing workflow, we ensure that our code is continuously tested and validated, providing immediate feedback on any issues that may arise from changes in the codebase.

8.3 Test Coverage Workflow Configuration

In addition to running unit tests, we have configured a separate GitHub Actions workflow to generate a test coverage report using Jest. This helps us monitor the extent to which our code is covered by automated tests.

The following is the configuration for the test coverage workflow:

```
name: Run Frontend Unit Tests

on: [push, pull_request]

jobs:
coverage:
runs-on: ubuntu-20.04

steps:
  - name: Checkout repository
    uses: actions/checkout@v3

  - name: Run Jest Coverage Report
    uses: ArtiomTr/jest-coverage-report-action@v2
    with:
```

```
working-directory: ./src/frontend
github-token: ${{ secrets.GITHUB_TOKEN }}
```

8.4 Test Coverage Workflow Steps

- **Checkout repository**: This step uses the `actions/checkout@v3` action to check out the repository code.
- **Run Jest Coverage Report**: This step uses the `ArtiomTr/jest-coverage-report-action@v2` action to generate a coverage report for the Jest tests in the frontend directory.

By incorporating test coverage reporting into our workflow, we can ensure that our automated tests provide sufficient coverage of the codebase, identifying untested areas and improving software reliability.

9 Trace to Requirements

Test ID	Requirements
test-FR-1	FR-1
test-FR-2	FR-2, FR-4, FR-5
test-FR-3	FR-3
test-FR-4	FR-6
test-FR-5	FR-7
test-FR-6	FR-7
test-FR-7	FR-8
test-FR-8	FR-8
test-FR-9	FR-9
test-LF-1	LF-AR1, LF-AR2, LF-AR3, LF-SR1, LF-SR2, LF-SR3, LF-SR4
test-LF-2	CR-SC3
test-UH-1	UH-E1, UH-E3, UH-L1, UH-UP1, UH-UP2, OE-P1

Test ID	Requirements
test-UH-2	UH-E2
test-UH-3	UH-PI2
test-V-1	MS-M1, OR-R2
test-PS-1	SR-P1
test-PS-2	SR-P2
test-PS-3	CR-L1, CR-L2, CR-L3, CR-S2, CR-S5
test-DS-1	MS-M3
test-DS-2	MS-S1, CR-SC1
test-D-1	UH-L2, OE-P2
test-PR-1	PR-SL1, PR-C1
test-PR-2	PR-PA1, PR-PA2
test-PR-3	PR-SE1
test-OE-1	OE-IAS1
test-OE-2	OE-IAS2
test-OE-3	OE-IAS3
test-M-1	MS-M2
test-M-2	MS-S3
test-M-3	MS-A2
test-M-4	MS-A1
test-M-5	MS-A3
test-S-1	SR-A1
test-S-2	SR-A1
test-S-3	SR-A1

Table 3: Traceability Matrix

10 Trace to Modules

Note: M7 encapsulates all its submodules.

Test ID	Requirements
test-FR-1	M2
test-FR-2	M4, M5, M6, M7, M8, M9
test-FR-3	M2
test-FR-4	M6, M9
test-FR-5	M1
test-FR-6	M1
test-FR-7	M1
test-FR-8	M1
test-FR-9	M3
test-LF-1	All User Interface Modules
test-LF-2	All User Interface Modules
test-UH-1	All User Interface Modules
test-UH-2	M2, M6
test-UH-3	M2, M6
test-V-1	M7, M8
test-PS-1	All Modules
test-PS-2	M6, M9
test-PS-3	All Modules
test-DS-1	Module Independent
test-DS-2	All Modules
test-D-1	M2
test-PR-1	M7, M8
test-PR-2	M2, M7, M9
test-PR-3	M7, M8
test-PR-4	Module Independent
test-OE-1	Module Independent
test-OE-2	Module Independent
test-OE-3	Module Independent
test-M-1	M7, M8

Test ID	Module ID
test-M-2	Module Independent
test-M-3	M7
test-M-4	M7
test-M-5	M7
test-S-1	M1
test-S-2	M1
test-S-3	M1

Table 4: Traceability Matrix

11 Code Coverage Metrics

The below report is for the frontend codebase of SyntaxSentinels. The backend has not been tested with a testing framework yet. Instead we ensure the backend is working as expected by manually testing the API endpoints using Postman and ensuring the server is sent the expected request via the frontend testing suite.

The test coverage report provides the following key metrics:

- **Statements**: 83.52% (218/261)
- **Branches**: 67.61% (48/71)
- **Functions**: 82.46% (47/57)
- **Lines**: 85.14% (212/249)

Below is a breakdown of coverage by individual files:

Directory	**Statements**	**Branches**	**Functions**	**Lines**
Features.tsx	100%	100%	100%	100%
Navbar.tsx	100%	100%	100%	100%
UploadBox.tsx	83.92%	47.36%	93.33%	85.18%
GetStarted.tsx	100%	100%	100%	100%
Button.tsx	100%	66.66%	100%	100%
utils.ts	100%	100%	100%	100%
AnalyzeFiles.tsx	91.56%	73.68%	80%	93.58%
JobsTable.tsx	75.92%	82.35%	66.66%	78.84%

Table 5: Test Coverage Breakdown by Directory

Some uncovered lines include:

- **UploadBox.tsx**: Line 26, 48-53, 66-73.
- **button.tsx**: Line 44.
- **AnalyzeFiles.tsx**: Lines 22, 25, 76.
- **JobsTable.tsx**: Lines 39, 44-55, 128.

These uncovered areas indicate potential gaps in test cases that should be addressed to achieve full coverage.

Appendix — User Data for tests

User ID	**Ease of Use Rating (1-5)**	**Clarity Rating (1-5)**	**Comments**
User1	5	5	"The onboarding process was very intuitive and easy to follow."
User2	4	4	"Clear instructions, but the navigation could be slightly improved."
User3	5	5	"Everything was straightforward and well-explained."
User4	4	4	"Good overall, but I had to look twice to find the upload button."
User5	5	5	"Very user-friendly and clear interface."

Table 6: Usability Test Results for Onboarding Process

User ID	Navigation Time (seconds)	Clicks Required	Comments
User1	15	2	Navigation was smooth and intuitive.
User2	20	3	It took me a moment to find the upload button.
User3	12	2	Very straightforward process.
User4	18	3	Clear instructions, but the layout could be improved slightly.
User5	14	2	Quick and easy to navigate.

Table 7: User Data Collected for Navigation Test

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

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?

   It was relatively straightforward to map out the tests we needed to focus on based on the Verification and Validation (VnV) plan document. The VnV plan provided clear expectations for the necessary tests, making it easier to structure the deliverable around those objectives. We didn't need to perform as much high-level reassessment of the overall design as we initially thought we might and had to for previous deliverables. This made the process feel more smooth, as we could concentrate on detailing how the tests would be executed and aligning them with the deliverables outlined earlier in the project. The clear documentation we had so far played a significant role in creating a sense of continuity, which made writing this deliverable easier.

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

   One of the main challenges we encountered was coming up with compelling data to verify the non-functional requirements for the project. While we were able to conduct usability tests with peers who rated our UI on a scale from 1 to 5, we felt unsure about whether this feedback would provide enough substance to support the NFRs. We sought advice from our TA, who reassured us that such usability tests, particularly when supplemented by qualitative feedback and user comments, could be deemed a reasonable and valid dataset for justifying the passing of NFR tests. This gave us the confidence to proceed with the existing data and treat it as adequate for demonstrating that the UI was user-friendly and met our usability NFRs.

   Another pain point revolved around selecting the right framework for frontend testing. There was some ambiguity about which testing framework would best suit the needs of our application, especially given the variety of tools and options available for testing JavaScript-based web applications. We resolved this by conducting thorough research into

various options. This let us determine that Jest would be the most appropriate choice for our project. Jest's'ease of use, and the fact that it was already well integrated with React (the framework we were using for the frontend) made it a great fit.

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?

   The majority of this document stemmed directly from team discussions. Since we were working closely together and had a good grasp of the technical specifications, most of the deliverable's content was informed by our team's perspectives and experience. The only sections which required client/peer assessment was anything related to the usability tests. As developers, it was difficult for us to maintain impartiality about the UI's intuitiveness and ease of use. We needed unbiased feedback to assess how intuitive and usable our interface truly was, so we conducted usability tests with external users (mainly classmates) to gather their insights. Their feedback on how easily they could interact with the interface was necessary to assign a pass or fail for some tests. Thus, while most of the document was internally driven, the usability testing portion was based on direct feedback from external stakeholders to ensure that our assumptions about the UI held up in real-world use.

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)

   While the VnV plan provided a great starting point, we did encounter some deviations from the original plan as the project evolved. Aspects such as cloud deployment and metrics for the NLP model ended up being much more complex and time-consuming than we originally anticipated. We had not factored in the amount of work required for cloud integration and the creation of monitoring tools for the NLP model's performance, nor how much work it would be to get the project to a

stage where it was ready for either of those components. These changes meant that certain tests had to be postponed to a later phase of the project, and we had to adjust the scope of the tests that could be conducted in the current phase.

Another area of change was related to user notification systems. Initially, the VnV plan assumed that notifications about long processing times (e.g., 10 minutes or more) would be shown directly in the UI. However, after some reflection and considering user behavior, we realized that it was unlikely that users would remain engaged with the UI during long processing times. Therefore, we decided to send these notifications via email only instead, which led to a change in the VnV testing scope. These modifications were a result of our growing understanding of user expectations. We found that such changes can be anticipated to some extent in future projects if we invest more time in the early stages to gather feedback, conduct prototyping, and set clear technical expectations.

While it was difficult to predict all of these changes upfront, the team was able to adjust quickly thanks to open communication and ascertaining of direction. If we had spent more time in earlier phases fully scoping out all potential technical complexities and user scenarios, we might have been able to avoid some of these surprises. That is a goal for future projects we participate in.