Testing Document

Main

This is the Design documentation for COMPSCI 2212 - Fall 2023 - Group 1's project Spell Checker GUI. This document outlines a comprehensive plan for the software system's design, including specific goals and technical specifications to ensure successful implementation.

Tabular Revision History

Below is a table that records the changes made to the requirement documentation. It provides a summary of the modifications made, who made them, and when they were made.

Version	Date	Author(s)	Summary of Changes
1.0	November, 28th 2023	Alishba Farhan	Created the Document
2.0	November, 30th 2023	Yazan, Alishba, Fuad, Adam	Final Changes

Table of Contents

The following is an outline of the major sections and sub-sections of the document. It serves as a roadmap for the reader, providing a quick overview of the structure and content of the document

1. Introduction

1.1 Overview

Spell checkers are ubiquitous software tools that have become essential in modern society. They play a critical role in ensuring accurate spelling, which is crucial for effective communication and preventing misunderstandings. By automatically scanning documents for spelling errors, spell checkers improve the credibility of written text and enhance the overall quality of communication.

The purpose of this project is to develop a desktop application that can analyze entire documents by comparing them word-by-word to a dictionary. The application will provide users with a range of options when it encounters spelling errors, including automatic corrections and suggestions for further auto-correction. Once the user has selected all the necessary options, the application will have fulfilled its primary objective of producing an error-free document.

1.2 Objectives

- Applying effective software engineering principles as a team to solve a real-word issue
- Adhering to all the specifications presented
- Referring to the specifications to produce clear models of the design and requirements
- Ensuring good design implementation in Java and dealing with decisions made earlier in the process
- Creating graphical, user-facing content and user-friendly applications
- Writing code clearly and efficiently
- Documenting the Java code in a well-ordered fashion that adheres to best practices
- Constantly reflecting on the good/bad decisions made over the course of the project collectively

1.3 References

- CS2212 Group Project Specification - Fall session 2023
- CS2212 Group 1 Required Documentation
- CS2212 Group 1 Design Documentation

2. Test Plan

Unit testing is a software testing method where individual units or components of a software application are tested independently. The primary purpose of unit testing is to validate that each unit of the software performs as designed. A "unit" in this context typically refers to the smallest testable part of an application, such as a function, method, or class. Unit testing is critical for ensuring the quality of software because it helps identify and fix bugs early in the development process, promotes cleaner and more modular code, and aids in maintaining and refactoring code with confidence.

JUnit, on the other hand, is an open-source framework used for unit testing in the Java programming language. It provides annotations to identify test methods, and assertions for testing expected results, making it easier for developers to write and run tests. JUnit plays a significant role in the development of test-driven development (TDD), where tests are written before the actual code. It helps developers write more efficient and error-free code and is a crucial component of a continuous integration and continuous deployment (CI/CD) pipeline. JUnit has become the standard testing framework for Java developers due to its ease of use, extensive feature set, and integration with various development tools.

Dictionary Class)

Junit Test cases

1. **Load Main Dictionary from file ('testLoadMainDictionaryFromFile')**

- **Objective**: Ensures that words are loaded into the main dictionary properly from file
- **Justification:** Validates that the main dictionary accurately reflects content of its file, which is crucial for the system's operation functionality

2. **Load user Dictionary from file ('testLoadUserDictionaryFromFile')**

- **Objective:** Tests ability to load user-specified words from file into the user dictionary.
- **Justification:** Verifies customizability of software, allowing for user-specific modifications.

3**. Load Ignore dictionary ('testLoadIgnoreDictionaryFromFile')**

- **Objective:** Asses loading of words Into the ignore dictionary from a file.
- **Justification:** Ensures that the system correctly recognizes and omits words designated to be ignored

Document Class)

Junit Test cases

1. **Document creation ('testDocumentCreation')**

- **Objective:** Validates constructor's ability to correctly parse and store content of the document
- **Justification:** Ensures functionality of reading and interpreting document data properly.

2. **File Path and Name Verification ('testFilePathAndName')**

- **Objective:** Confirms correctness of setting and getting the file path and file name in 'Document' class.
- **Justification:** Verifies integrity of file handling within the class.

3. **Singleton Instance Verification ('testDocumentSingleton')**

- **Objective:** Checks singleton behaviour of 'Document' class, to verify only one instance is created and used
- **Justification:** Singleton pattern verification is important to maintain a single state across the application.

4. **Character count accuracy ('testCharacterCount')**

- Objective: Verifies character count functionality of the 'Document' class is accurate

- Justification: Makes sure character counting is reliable, which is a vital feature of document processing

5. **Markup Flag Functionality ('testMarkupFlag')**

- Objective: Tests functionality of setting and getting markup flag in 'Document'

- Justification: Checks capability of class to properly handle document markup properties, affective how documents are displayed.

6. **Document refreshing ('testRefreshingDocument')**

- Objective: Asses functionality of refreshing the document

- **Justification:** Ensures that instance can be updated correctly, a necessary feature for dynamic document handling.

Error Class)

Junit Test cases

1. **Correct Spelling Check ('testCheckSpellingCorrectWord')**

- Objective: Validate function's ability to correctly identify a correctly spelled word

- Justification: Ensures that spell check functionality does not falsely flag correct words.

2. **Incorrect Spelling Check ('testCheckSpellingIncorrectWord')**

- Objective: Confirms functional capability to identify an incorrectly spelled word.

- Justification: Important for spell check system to detect and flag spelling errors accurately.

3. **Spelling Correction by Substitution ('testCorrectSpellingSubstitution')**

- Objective: Tests accuracy of suggesting spelling corrections through substitution.

- Justification: Ensures reliability of correcting spelling by substitution.

4. **Capitalization Check ('testIsFirstLetterCapitalized')**

- Objective: Verify function's ability to identity capitalization status of the first letter in a word.

- Justification: Important for grammatical checks especially in processing text.

5. **Spell Correction by Omission ('TestCorrectSpellingWithOmission')**

- Objective: Verify ability to fix spelling via omission.

- Justification: Validates spell correction feature for cases where characters are mistakenly added to words.

6. **Spelling Correction with Insertion ('testCorrectSpellingWithInsertion')**

- Objective: Check accuracy of suggesting spelling correction by inserting missing characters.

- Justification: Ensures class can handle and correct words with missing characters which is a common typing error.

7. **Spelling Correction With Reversal ('testCorrectSpellingWithReversal')**

- Objective: Test Accuracy of suggesting spelling corrections for words with reverse characters.

- Justification: Validates ability to handle as well as correct words where characters are in reverse, which is less common but possible when typing.

Node Class)

Junit Test cases

1. **Node Creation and Getters verification ('testNodeCreationAndGetters')**

- Objective: Validate creation of 'Node' object and accuracy of the getters.

- Justification: Confirms that 'Node' class correctly initializes with provided values and that getters accurately retrieve those values.

- 2. **Setters functionality verification ('testSetters')**

- Objective: Confirms setters of 'Node' class effectively update attributes of 'Node' object.

- Justification: Verifies setters is essential to ensure class can adapt to changes in state or context which is key to dynamic data structures.

Integration Testing: Integration testing focuses on validating the collaboration and connectivity between different units or components within the spellchecker application. It ensures that the individual modules, implemented as source code, seamlessly come together as per the overall design and architecture of the software. During integration testing, the interactions between units are examined to identify and resolve any potential issues related to data flow, control flow, and the exchange of information. This phase verifies that the integrated components work cohesively, reflecting the intended behavior outlined in the requirements and design documentation. For our spellchecker application, integration testing would involve confirming the smooth interaction between the spellchecking algorithm, user interface components, and dictionary management system to guarantee the holistic functionality of the application.

This section focuses on testing the integration of multiple GUI components and their connected back-end functionality. These test cases were chosen as they're all the result of user input and/or interaction.

Test Case: Buttons on Main Page connect to all other pages on mouse action and can be accessed by the user.

- Note: the pages are Metrics, Settings, Upload, Save, Information

Justification: To ensure that the application is able ton display the correct screen that corresponds to the button.

Test Case: Buttons on Help Page connect to How To Use and About pages on mouse action

Justification: To ensure that the pages are connected to the right buttons and are displayed when prompted.

Test Case: Upload file offers the user the option to identify the document as a markup language.

Justification: To ensure that the back end code is able to recognize when the user prompts that application that the text file contains markup language and for then back end to ignore the markup language when checking for errors in the text file. This ensures that the front end and back end of the application are able to communicate without errors

Test Case: Textbox displays "Loading...." upon file upload

Justification: The front end is able to recognize when the back end is executing a task.

Test Case: Text from uploaded document is displayed in textbox

Justification: Ensures that the front end and back end are able to transfer data between each other without fail and mistakes.

Test Case: Grammatical and spelling errors are displayed once file is uploaded

Justification: Ensures that the data transfer and the functions for both the front and back end are working and well integrated to work together.

Test Case: Suggestions are displayed for each error on mouse action

Justification: The front end is able to take interactions done by the user and implement actions accordingly.

Test Case: User is warned before overwriting a document

Justification: Application is able to identify the users actions and prompt appropriate error messages

Test Case: User is given the option to save in a different location

Justification: Ensure that the application is able to handle files in an orderly manner and be able to save files without corrupting them and causing harm to the user files.

Validation Testing ensures that the spellchecker application aligns with the specified functional and non-functional requirements. It involves confirming that the software correctly identifies and suggests corrections for misspelled words, follows the defined workflow for document processing, and seamlessly integrates system and user dictionaries. The testing process verifies the accuracy of error detection mechanisms, including identifying misspellings, miscapitalizations, and double words. Additionally, it ensures that the application provides appropriate correction options and computes metrics, such as the number of errors and corrections. Performance criteria, such as reasonable load times and efficient run times, are also validated to guarantee the application meets its intended purpose effectively.

Spell Checking Workflow

Objective: Validate that the general workflow (open file, spell check, save results) functions as expected.
Test Cases:
- Open a file and verify that the spell check process initiates.
- Manually introduce misspellings and confirm the application detects and offers corrections.
- Save the results and verify that changes are applied to the document.

Document Support

Objective: Ensure the application supports spell-checking of plain text files and optional filtering of HTML/XML tags.
Test Cases:
- Spell check a plain text file and confirm correct functionality.
- Spell check an HTML/XML file with tag filtering enabled and verify accurate results.

Dictionary Functionality
Objective: Validate loading and using system and user dictionaries.
Test Cases:
- Load a system dictionary and verify that words from the dictionary are recognized.
- Add words to the user dictionary during spell checking and confirm their recognition in subsequent checks.

Words and Definitions

Objective: Validate the application's definition of a word, considering whitespace, punctuation, hyphenation, prefixes, suffixes, and embedded numbers.
Test Cases:
- Introduce hyphenated words and confirm correct handling.
- Test words with prefixes and suffixes, ensuring proper recognition.
- Validate consistent handling of words with embedded numbers.

Error Detection
Objective: Validate detection of misspellings, miscapitalizations, and double words.
Test Cases:
- Introduce misspelled words and confirm detection.
- Create sentences with miscapitalizations and verify correct identification.
- Add double words and ensure the application detects and flags them.

Error Presentation

Objective: Validate error presentation in context.
Test Cases:
- Confirm that errors are presented with sufficient surrounding text.
- Verify the highlighting or formatting of errors for easy identification.

Error Correction Options
Objective: Validate manual correction, suggestion generation, and correction options.
Test Cases:
- Manually correct errors and confirm changes are applied.
- Verify the generation of relevant suggestions for misspellings.
- Test various correction options, including deletion, ignoring, and adding to the user dictionary.

Metrics Calculation

Objective: Validate the computation and reporting of document metrics.
Test Cases:
  - Spell-check a document and verify the accurate calculation of characters, lines, words, error types, and correction types.

Housekeeping Functions

Objective: Validate basic housekeeping functions, including clean application exit and access to help.
Test Cases:
  - Exit the application and confirm a clean exit.
  - Access help resources and ensure they are informative and accessible.

Persistence

Objective: Validate persistent storage of user dictionaries and settings.
Test Cases:
  - Spell-check multiple documents, restart the application, and confirm the persistence of user dictionary changes.
  - Modify application settings, restart, and validate the persistence of changes.

Non-Functional Requirements

Objective: Validate compliance with non-functional requirements:

- Develop a standalone desktop application using Java.
- Implement Java GUI with Swing or JavaFX, ensuring uniformity across the group.
- Optimize performance for dictionary setup and word search.
- Avoid additional libraries without prior approval.
- Comment all code using Javadoc.
- Decide and maintain consistent coding conventions.
- Ensure compatibility with standard Java installations.
- Limit file system changes to the installation directory.
- Provide clear user feedback and informative error messages.
- Adhere to software engineering principles.
- Regularly commit code to the designated repository.
- Complete documentation using Confluence.
- Track tasks and issues on Jira throughout the project.

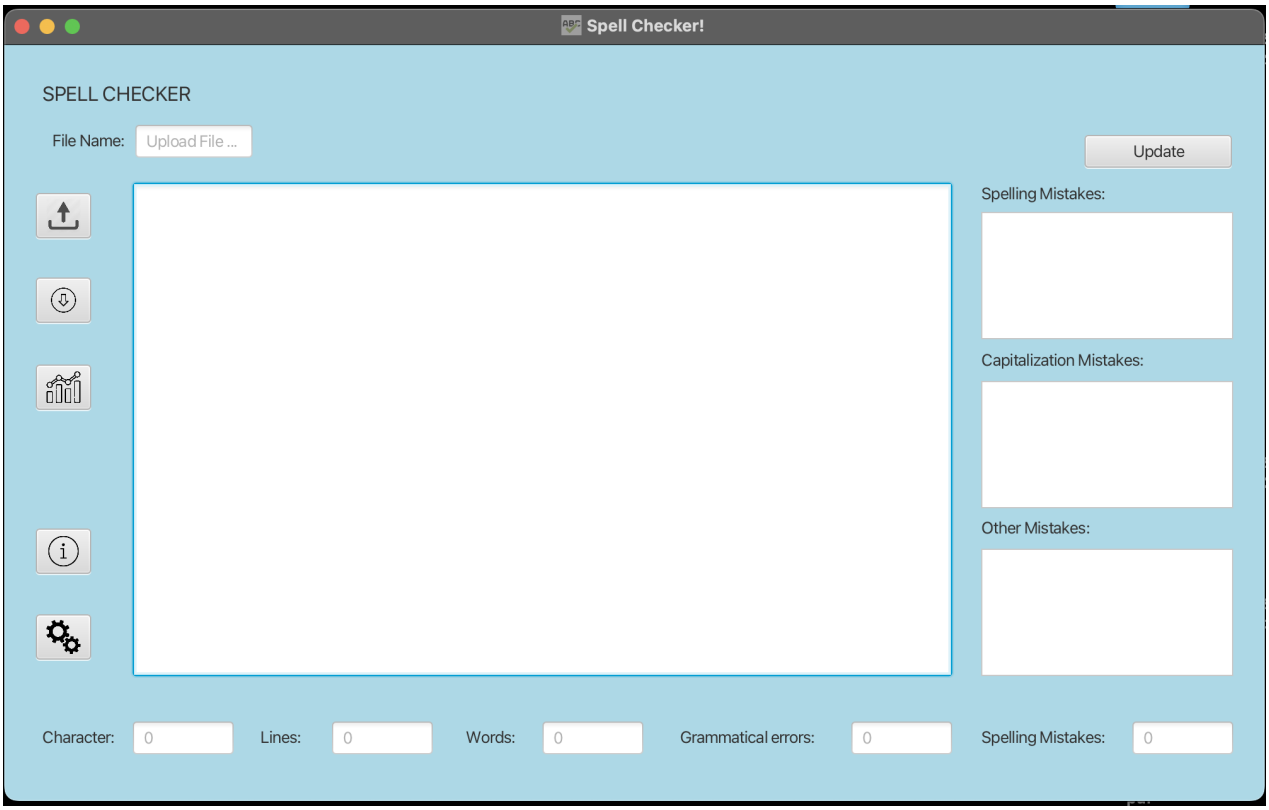
Test Cases:

- Test the process of committing and pushing code to the designated repository.
- Verify that the latest code changes are reflected in the repository.
- Check the creation and updating of Confluence pages for project documentation.
- Validate the creation and updating of tasks and issues on Jira throughout the project.

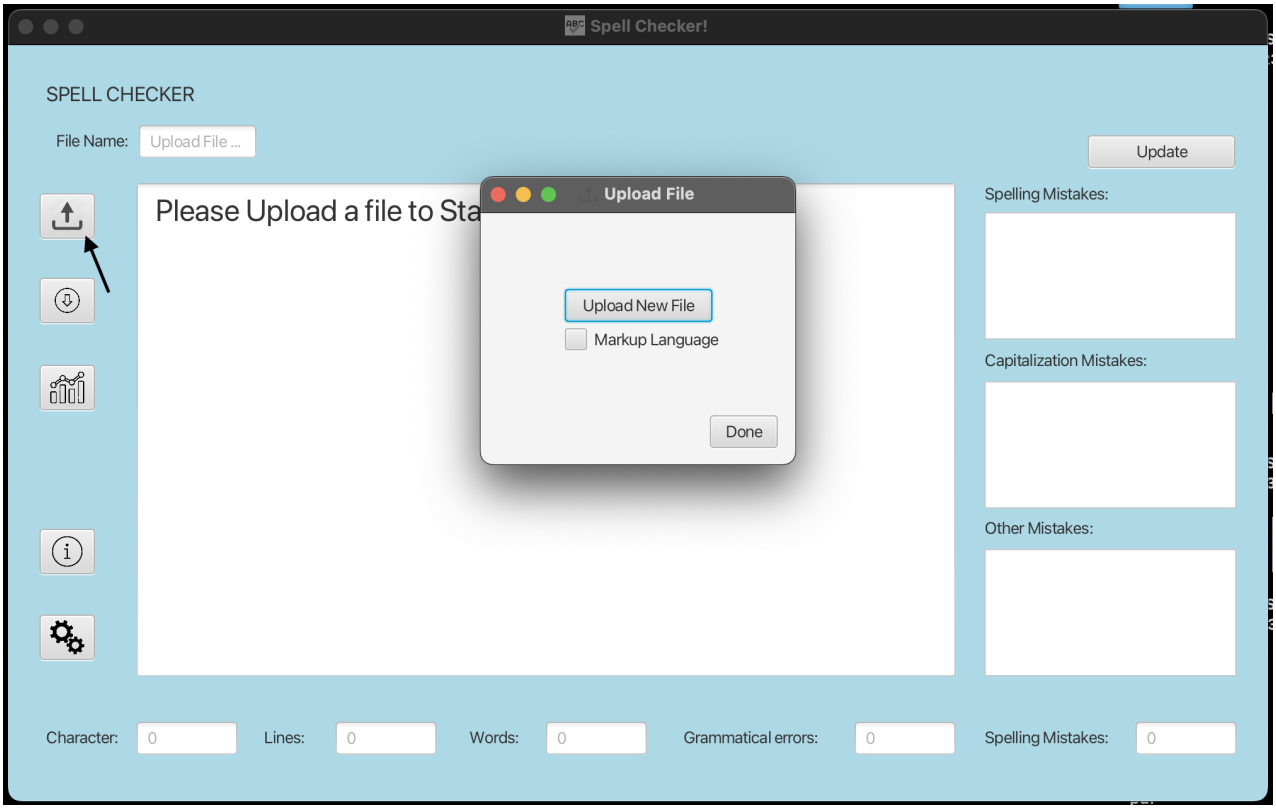
System Testing is a critical phase of software testing that evaluates the entire integrated software product, including all hardware and software components. It verifies and validates the system's behavior as per the end-to-end system specifications. In a complex computer-based system, the software is usually one component among many others. System Testing helps ensure that the software interacts seamlessly with all other components, including hardware and software systems, and that it meets the desired functional and non-functional requirements. In essence, System testing is a comprehensive series of tests that aim to exercise the entire computer-based system, including hardware, software, and user interface, to ensure that it functions as intended and meets the quality standards.

Usability Testing

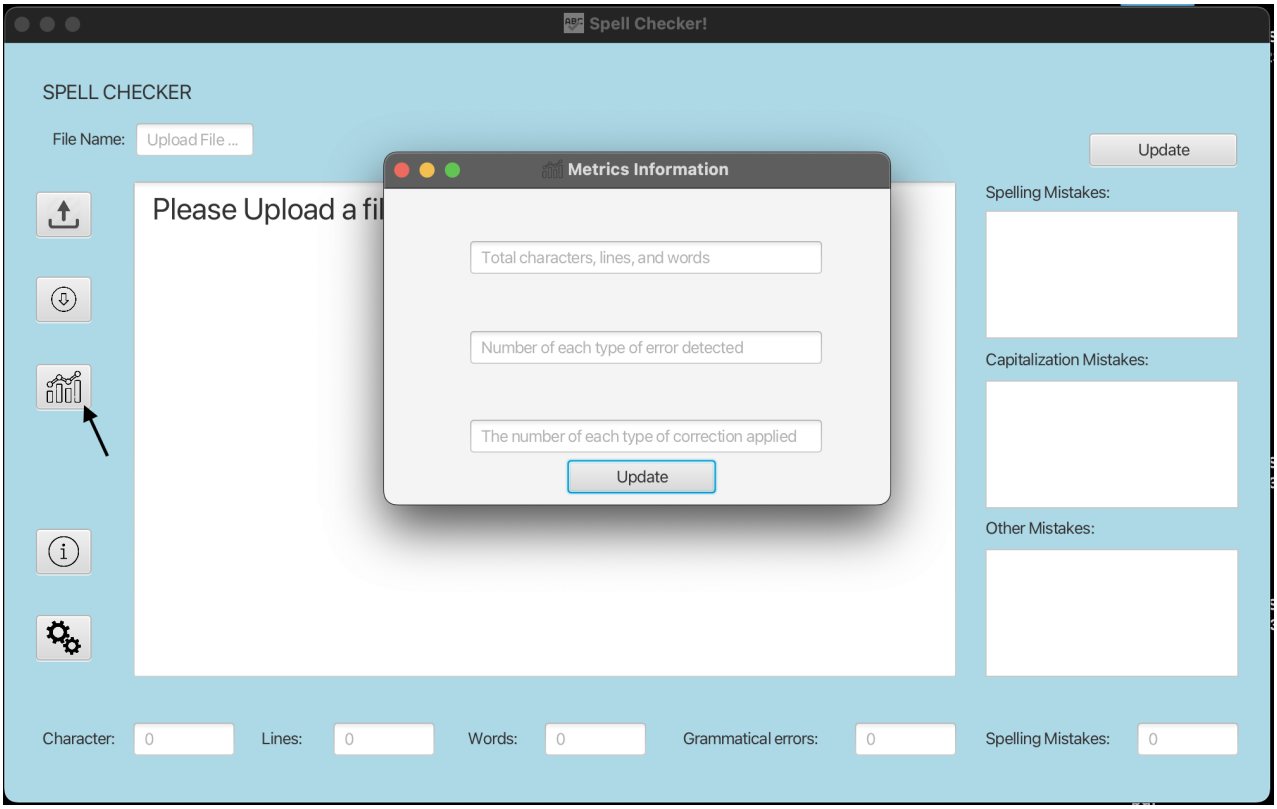
The application's buttons are appropriately sized and each icon has a clear representation of its function, enhancing usability. The visually appealing combination of the black text and light blue background adds to the application's aesthetics. The application's main screen displays all the necessary information, eliminating the need for users to navigate through multiple pages, streamlining the user experience.



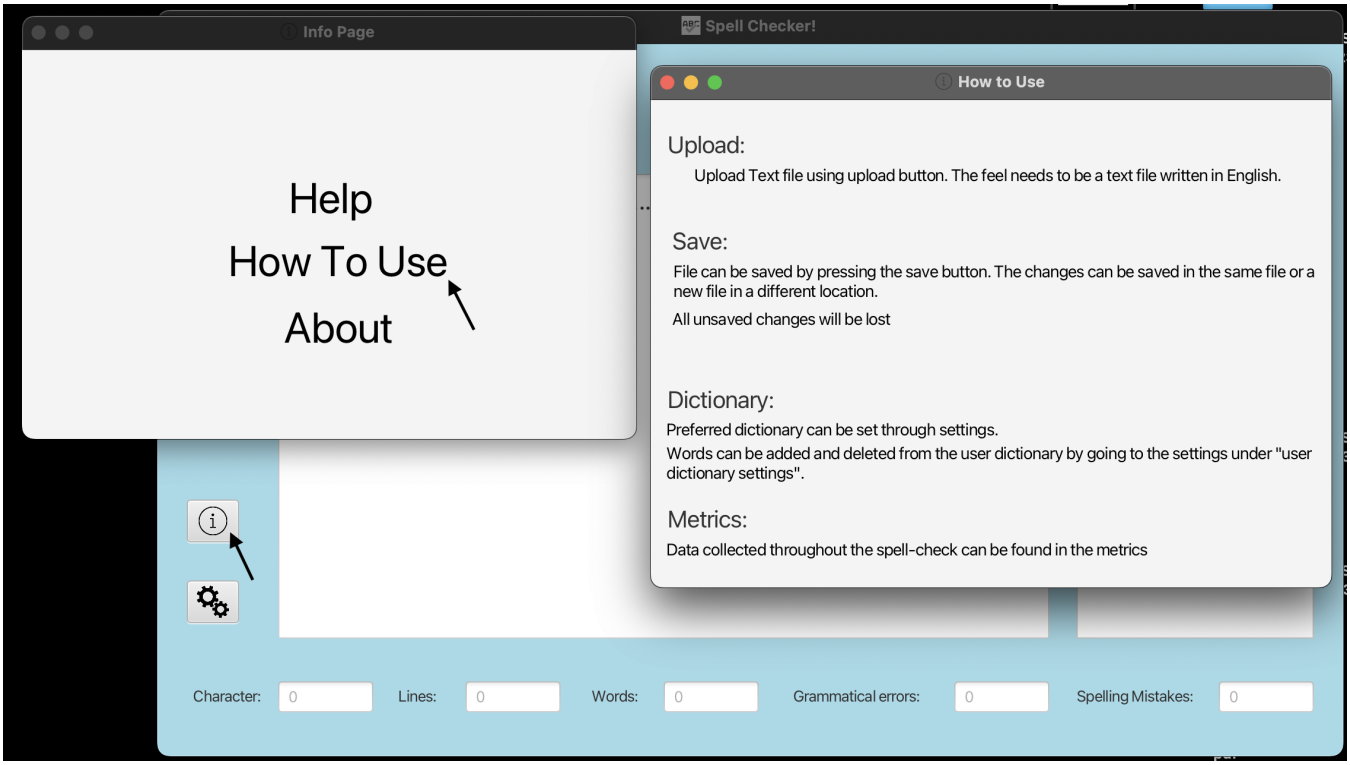
Upon clicking the upload file button, a pop-up window for file upload appears with a simple design that presents only two buttons to minimize user confusion. The pop-up also includes an option for Markup language, as specified in the Functionality requirements.



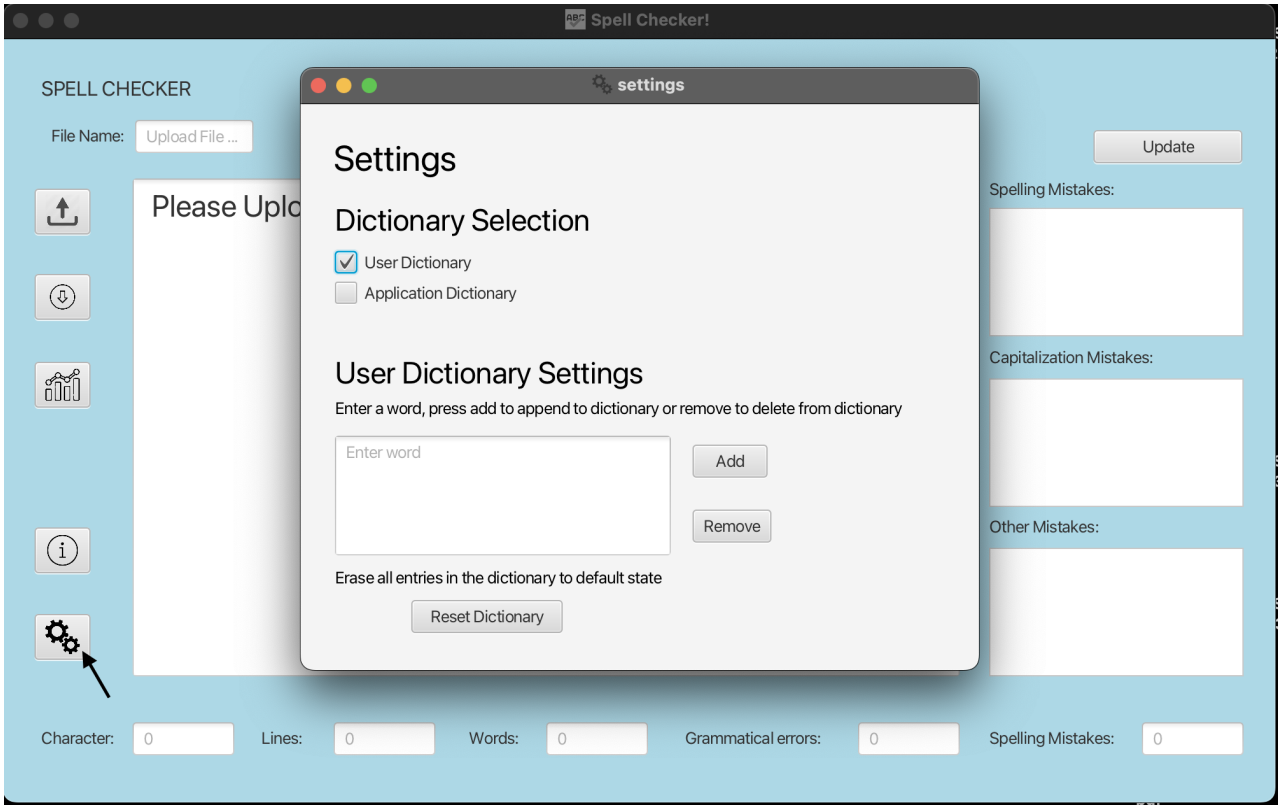
Upon clicking the metrics button, the application presents various sections with distinct information, facilitating comprehension for the user. Additionally, the application allows for updating the metrics. Further, the main page of the application contains additional comprehensive details pertaining to metrics.



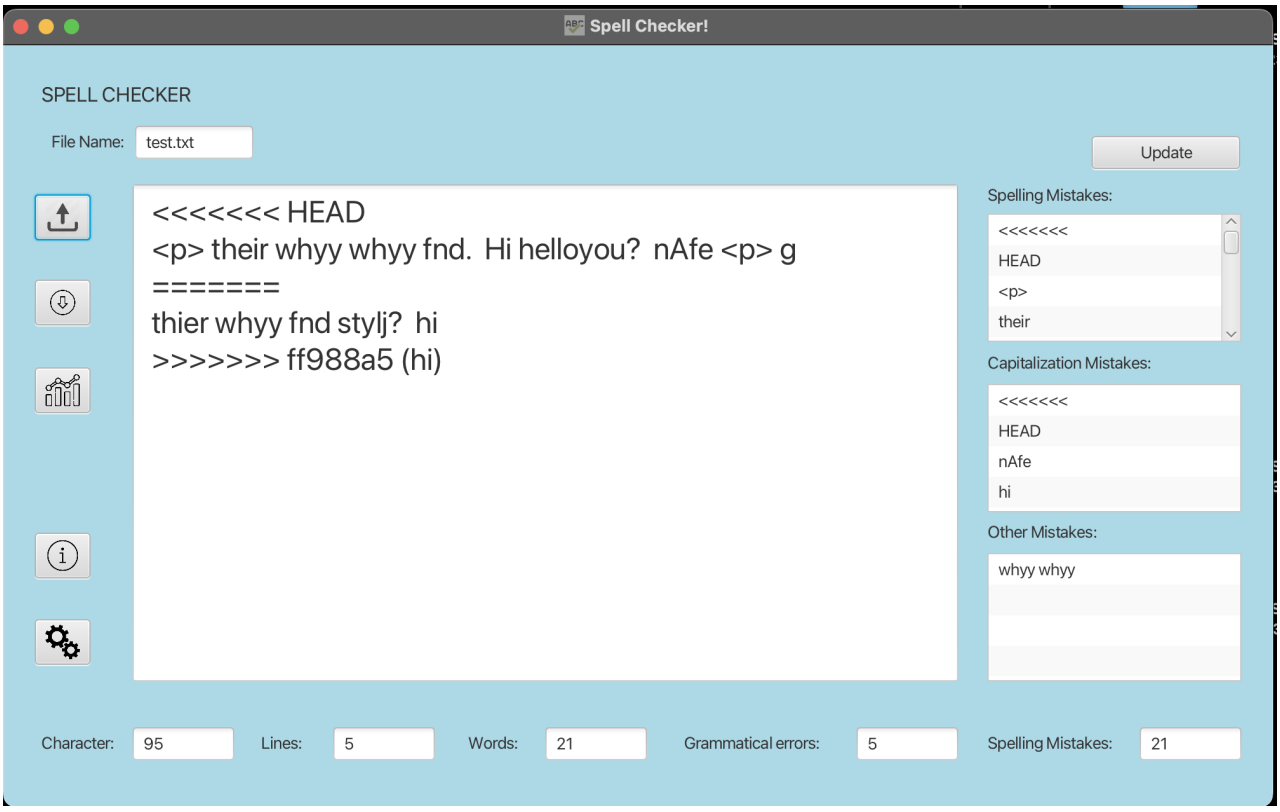
Upon selecting the information button, three options are presented to the user. Opting for the "How to Use" option triggers a pop-up window that offers both simplistic and detailed instructions on the functionality of each component of the application. This feature significantly enhances the user experience by providing valuable assistance that is accessible and easy to understand. As a result, the application becomes more user-friendly and appealing to the target audience.



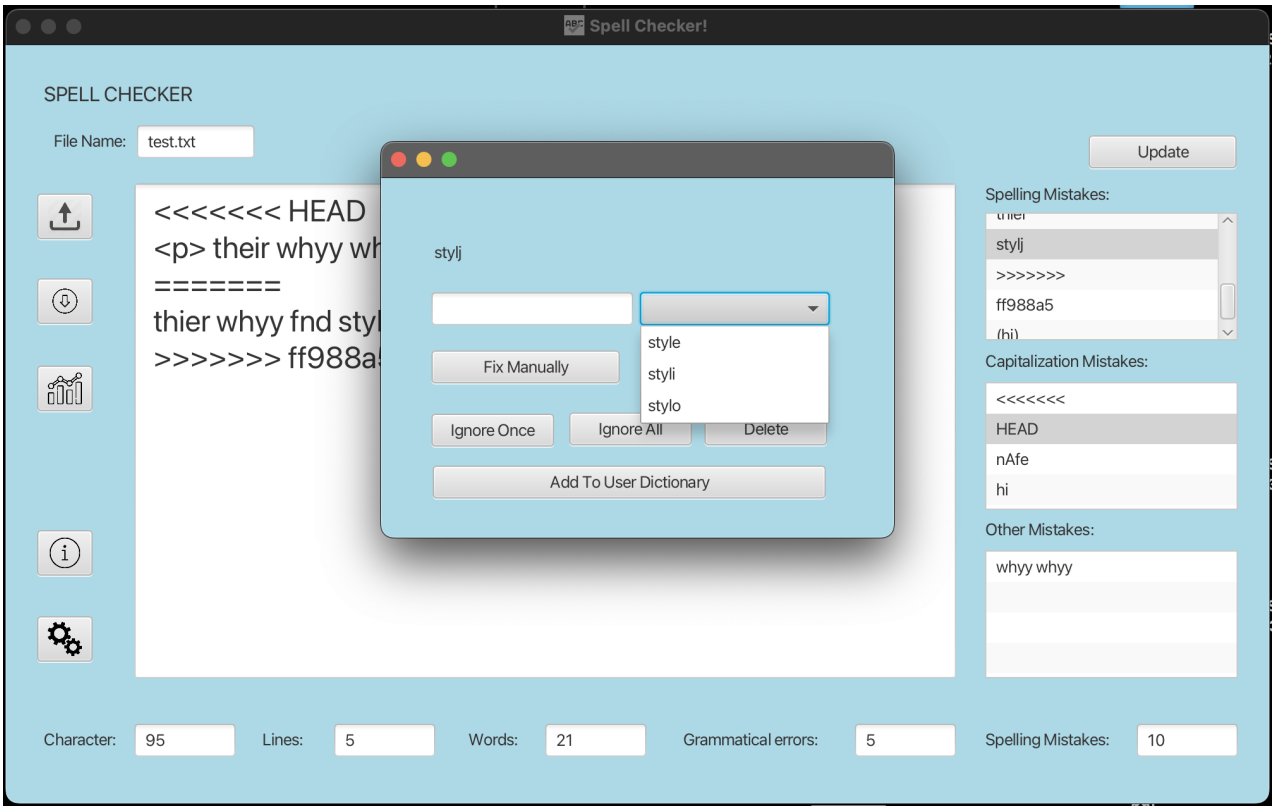
Upon clicking the settings button, a pop-up is triggered, presenting all available options, including the choice for the user to select their preferred dictionary. The selected option is visually distinguished with a tick mark, providing ease of use and reducing potential confusion. Additionally, a dedicated user dictionary section is available, accompanied by straightforward instructions to enhance user accessibility.



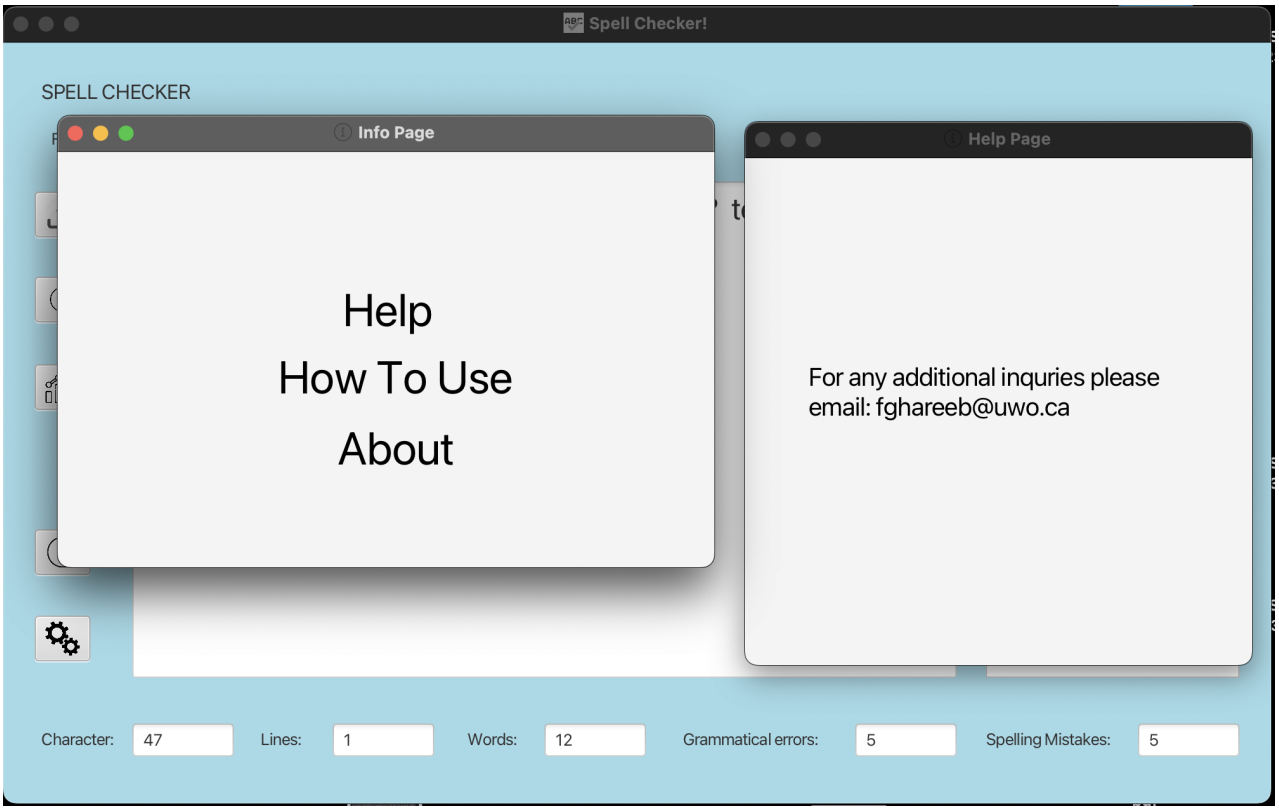
After a file is uploaded, the application presents the relevant information in a well-organized and visually appealing manner. All error messages are displayed in their respective locations, allowing users to easily access the necessary information in a structured format.



When a user selects a mistake to be fixed, they are presented with multiple options to rectify the issue. The user interface is designed with neatly placed buttons and a drop-down menu, which provides a range of options to choose from, making it easy to understand and navigate. The interface also includes arrows in appropriate locations, enhancing its user-friendliness. Overall, the user interface is intuitive and user-friendly, ensuring a seamless experience for the user.



The application provides the user with the option to get in contact with the developers for any concerns regarding the application

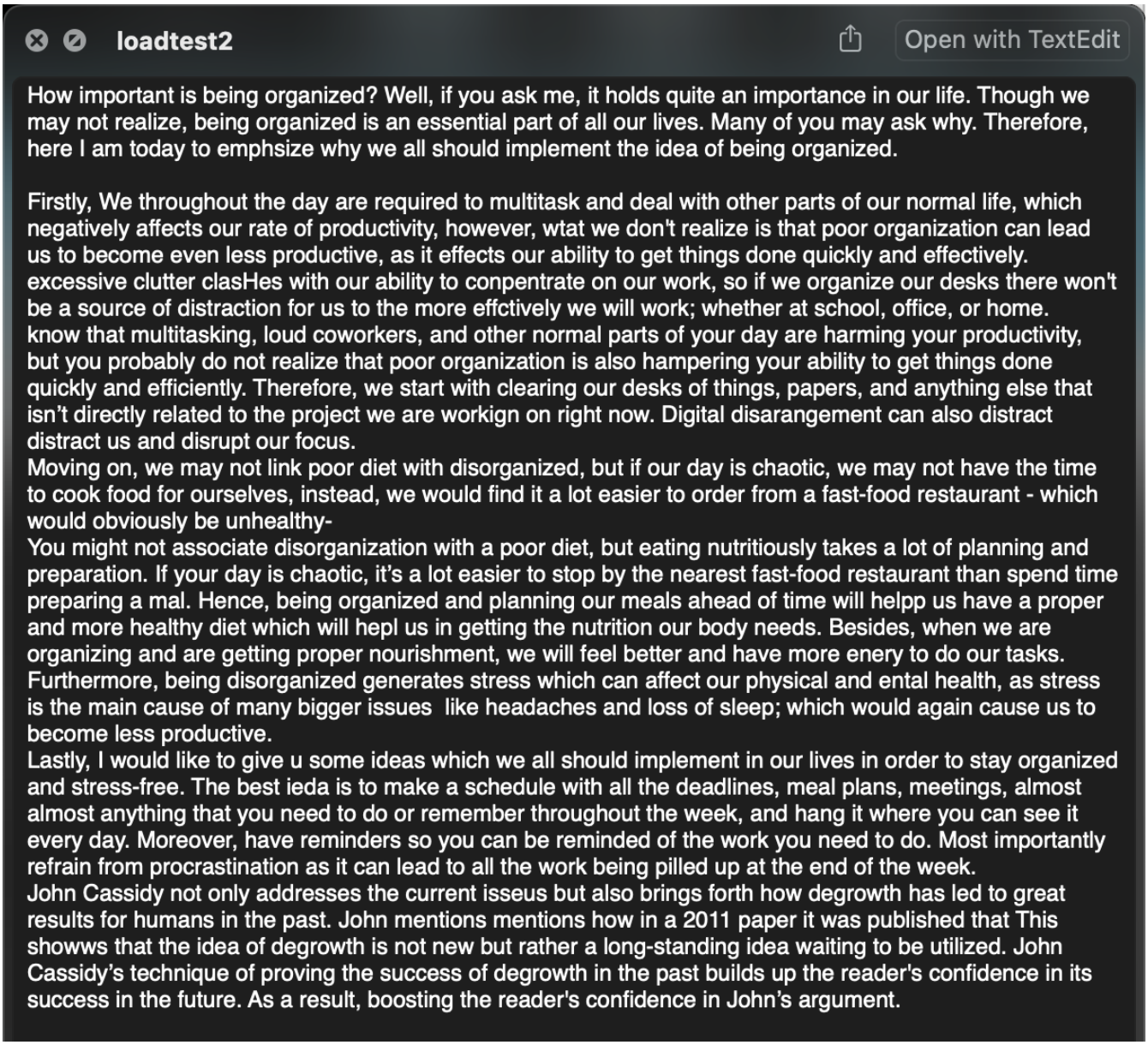


Load Testing

We tested the time it took for the application to upload a text file containing 48 characters. We also tested the time it took for the application to do a single task at a time. Every task related to spell checking is done by the check function in the program. Therefore, the time execute time for check is the time it took the application to do a. single task.


```
Total execution time to upload file in millis: 130
2023-11-30 19:44:18.973 java[30660:8539373] +[CATransaction synchronize] called within transaction
5
Total execution time to execute check in Java in millis: 2
Nov 30, 2023 7:45:04 PM javafx.fxml.FXMLLoader$ValueElement processValue
WARNING: Loading FXML document with JavaFX API of version 21 by JavaFX runtime of version 19.0.5
Total execution time to execute check in Java in millis: 1
```

We tested the time it took for the application to upload a text file containing 3,363 characters. We also tested the time it took for the application to do a single task at a time. Every task related to spell checking is done by the check function in the program. Therefore, the time execute time for check is the time it took the application to do a. single task.



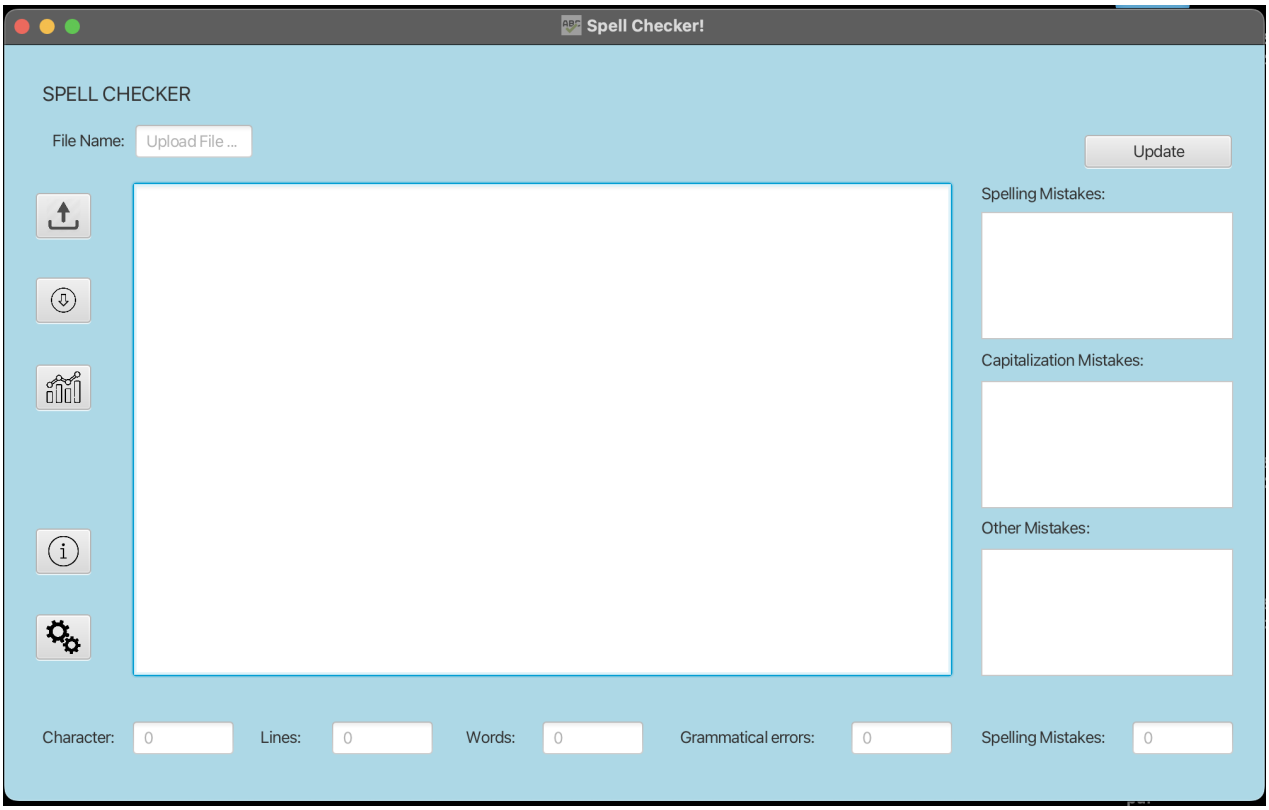

```
Total execution time to upload file in millis: 141
2023-11-30 20:23:50.209 java[34741:8604620] +[CATransaction synchronize] called within transaction
15
Total execution time to execute check in Java in millis: 7
```

Conclusion

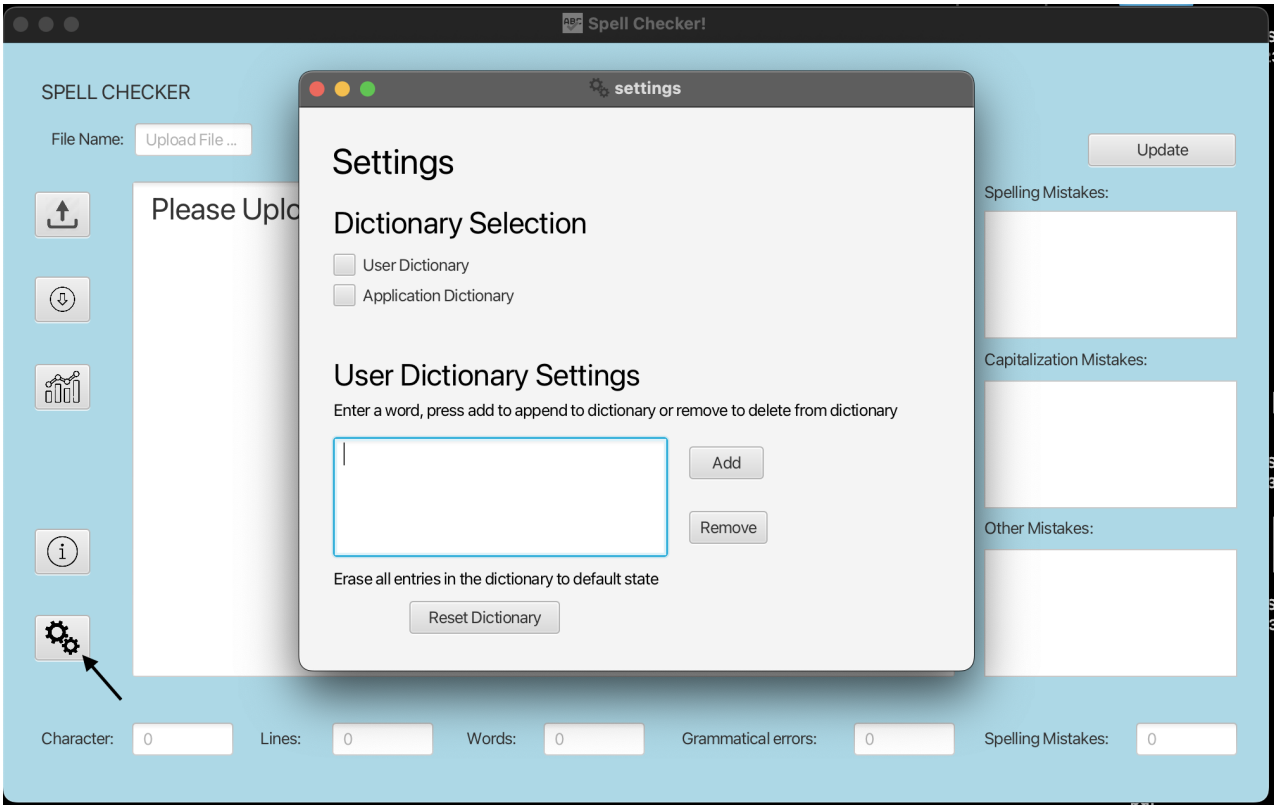
Despite the significant difference in character count between the two files, the discrepancy in execution time is negligible. This implies that the application has the capability to process large files without encountering any crashes or taking an excessive amount of time to complete the operation.

Recovery Testing

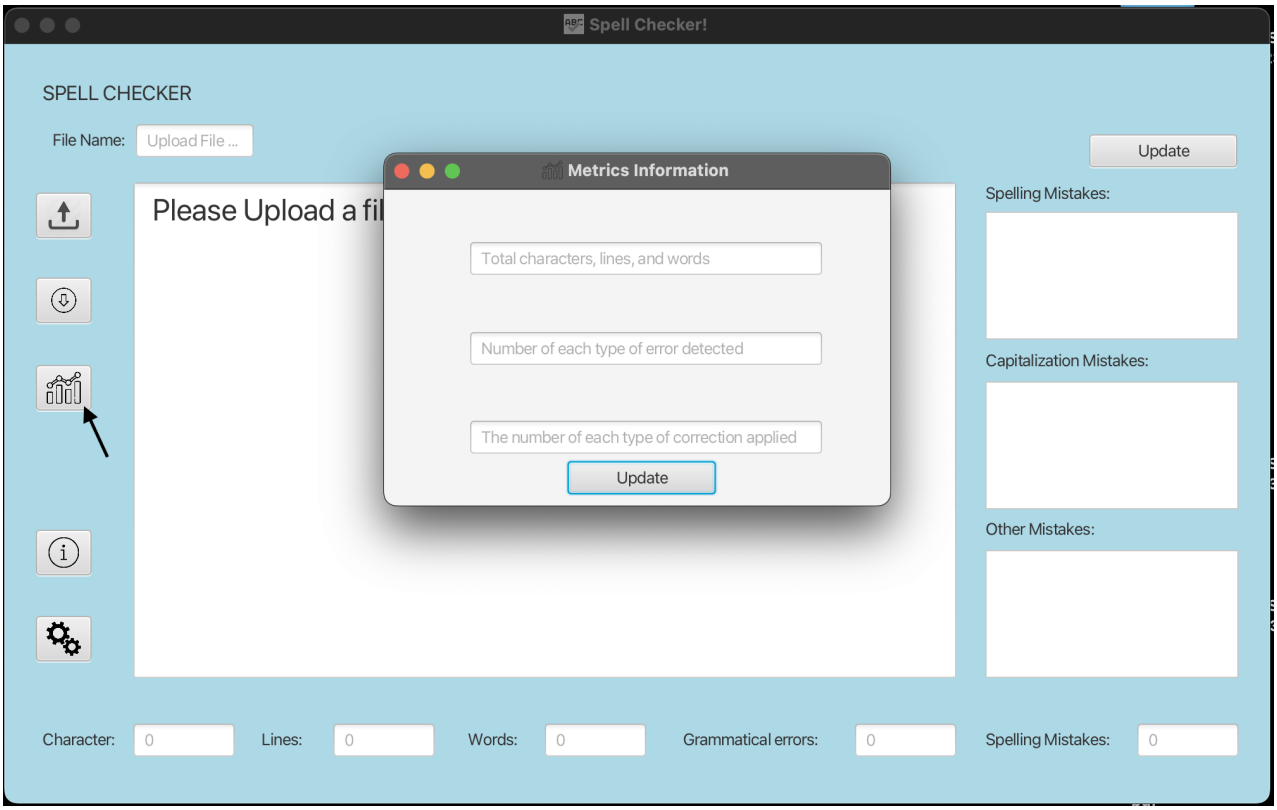
The application includes update buttons that allow the user to recover from any possible issues that may arise, such as unresponsiveness or failure to update changes on the screen. These buttons allow the program to refresh and update its state in real-time, ensuring that it remains accurate and responsive to user input. This feature is critical in maintaining a seamless user experience and preventing potential issues from disrupting workflow.



The application offers a reset dictionary button incase the user wants to reset the user dictionary. The button comes along with a clear and concise description of its functionality. This ensures that the user is not left uncertain about what the button does.



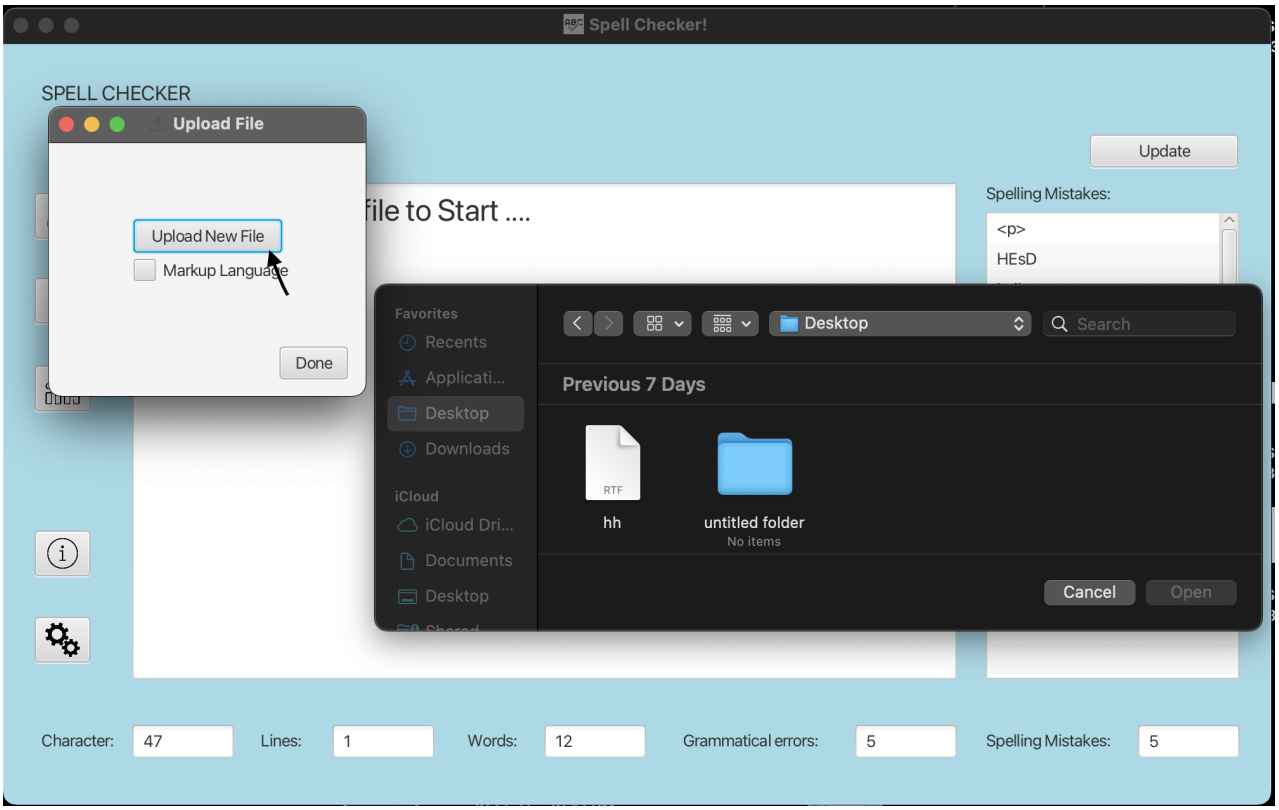
Users can access the update button on the metrics pages to refresh the metrics that may be incomplete. This feature is useful when encountering issues with metric updates.



Functionality Testing

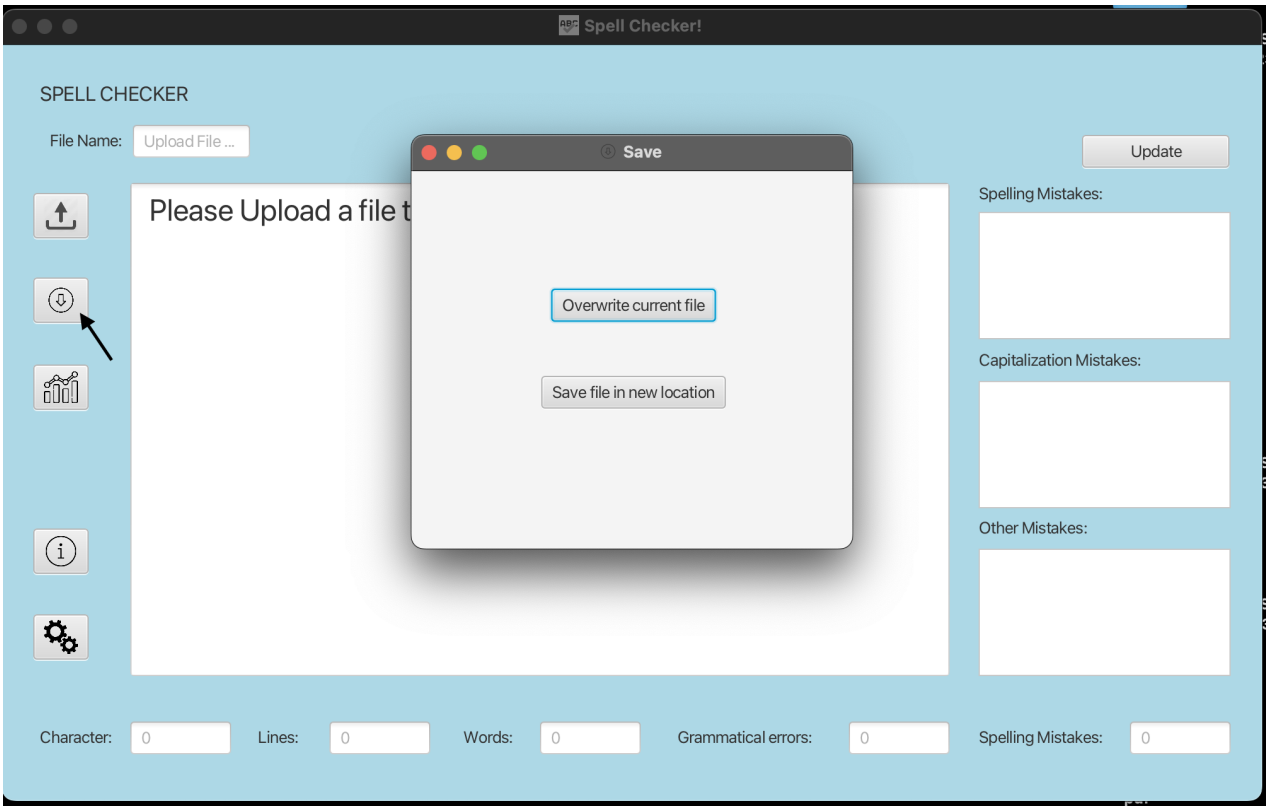
Upload Functionalities:

The application allows the user to upload a text file of any size and length without any limitations. The file format must be strictly text-based. Upon clicking the upload file button, users are granted the liberty to select a file from any location within their computer. After choosing a file when the user clicks done the text is displayed onto the screen to be spell-checked.



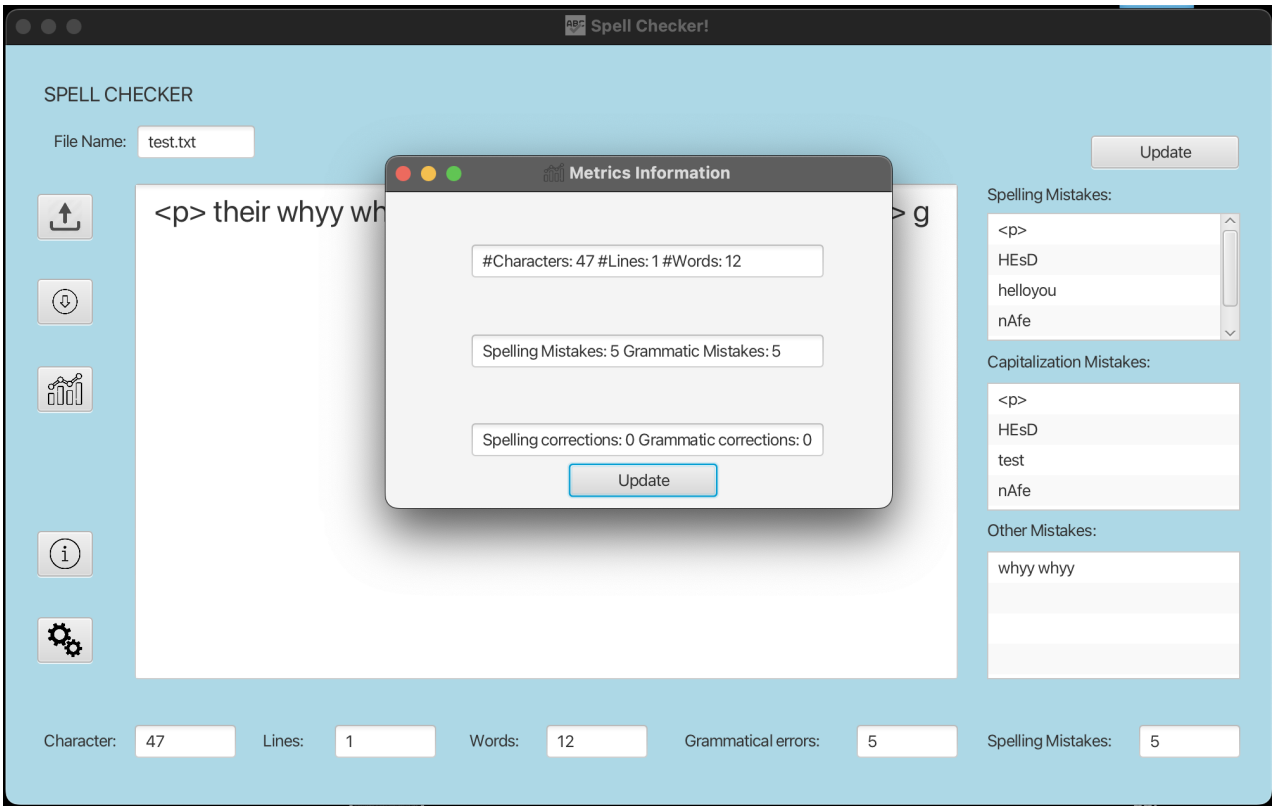
Save Functionalities:

The file-saving feature provides two options to the user: overwriting the current file or saving it in a new location. In the latter case, a pop-up window will appear, showing desktop locations, from where the user can choose any location to save their file. Additionally, users can name their file as per their preference.



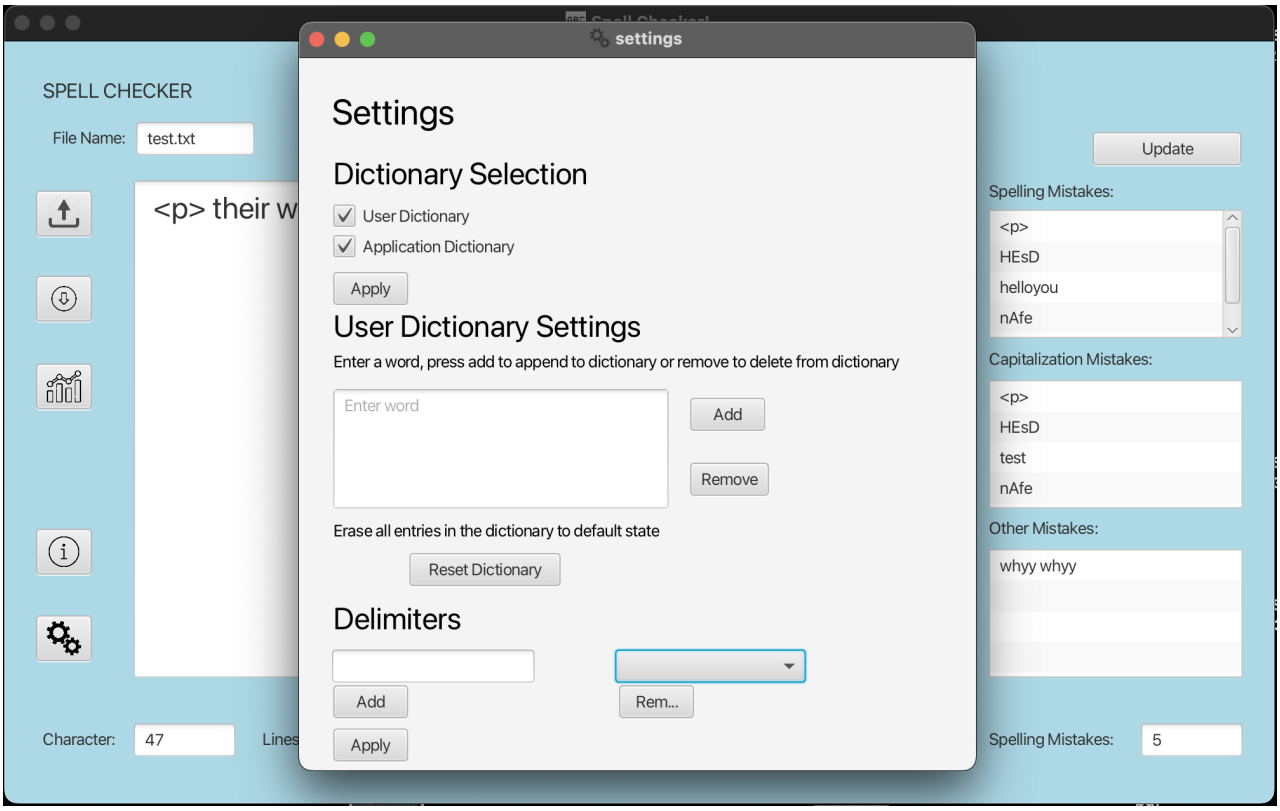
Metrics Functionalities:

In addition to the main page, there is a dedicated section where users can access detailed information about the program, including the number of lines, characters, and other metrics such as the number of detected errors and corrected mistakes. Whenever the user corrects any type of mistake the corresponding metrics is updated.



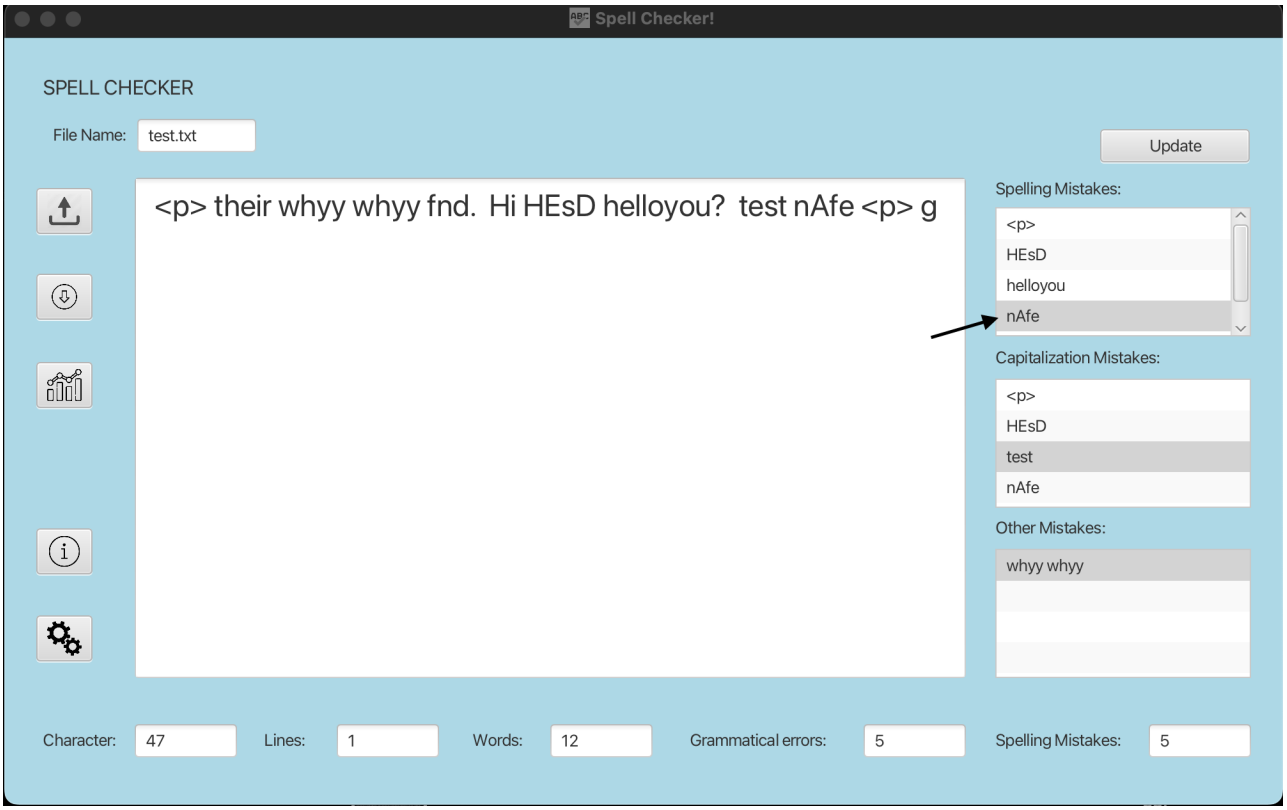
Dictionary Functionalities:

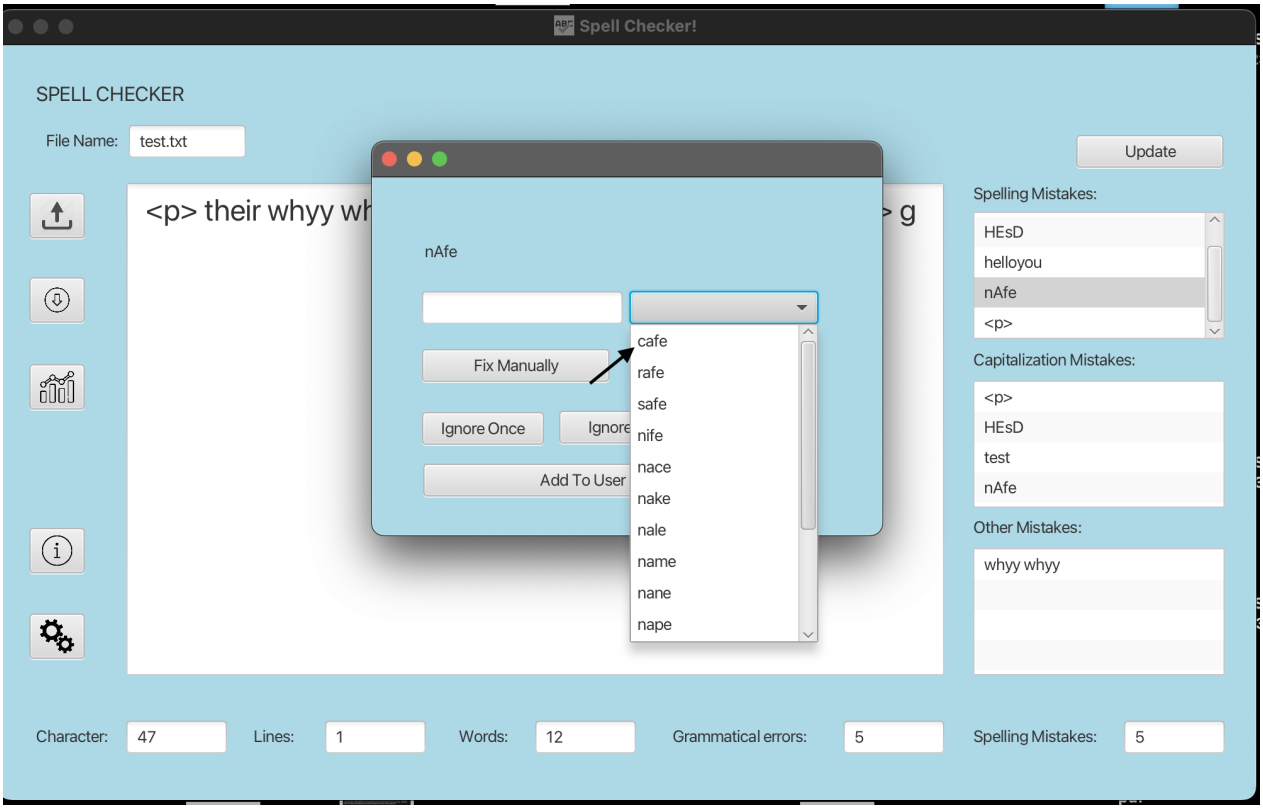
The program offers the users the flexibility to choose between using their own dictionaries or program's dictionaries. They can modify these preferences anytime during the program's usage and can save the changes by using the update button. Furthermore, users can also edit their own dictionary by adding, removing, or resetting the words.

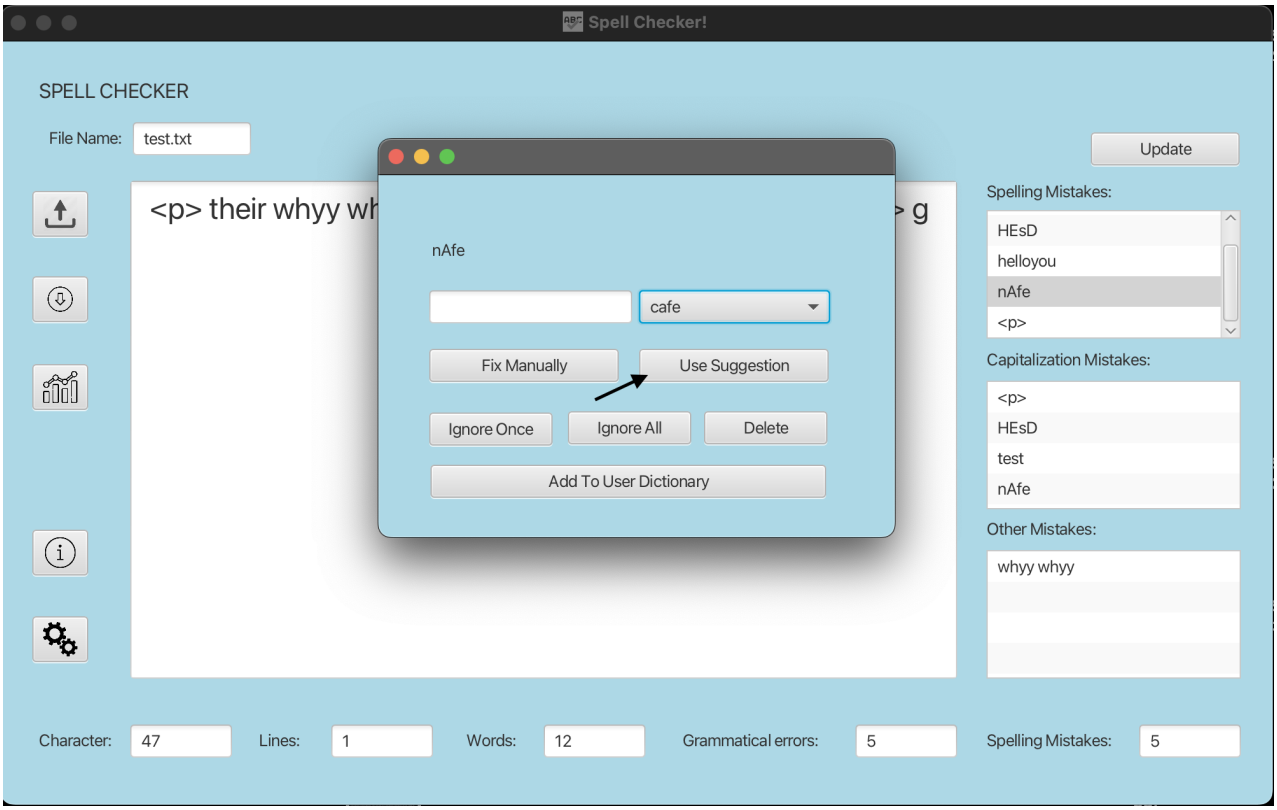


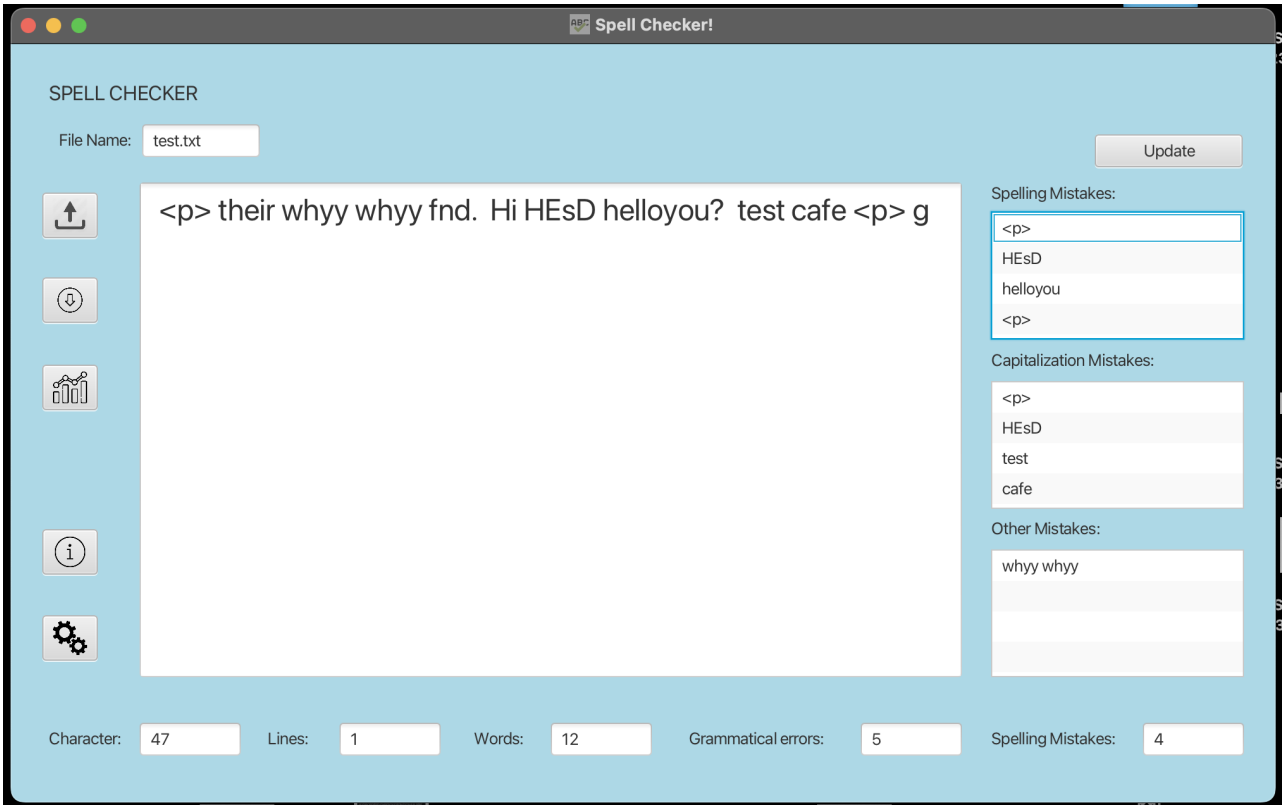
Spell-check Functionalities:

After the file upload, the Application initiates the process of identifying and displaying all spelling mistakes in the appropriate box. The user has the flexibility to fix the words in any desired order. The Application provides multiple options to the user, including the ability to use one of the suggested corrections provided by the Application. The user can select an option and apply the changes by pressing the "use suggestion" button. The mistake is then removed from the spelling mistakes section once the change is applied.

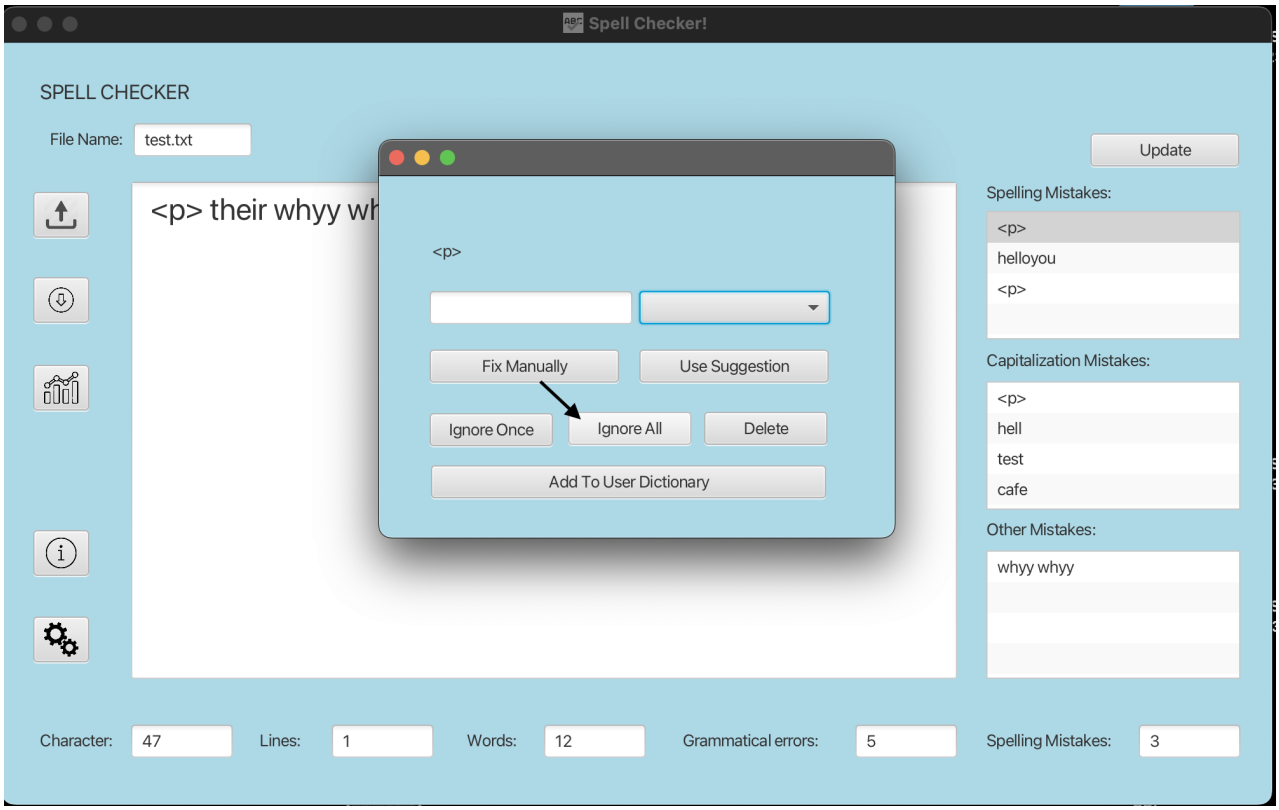

Spell Checker!
SPELL CHECKER
File Name:
test.txt
Update
<p> their whyy whyy fnd. Hi HEsD helloyou? test nAfe <p> g
Spelling Mistakes:
<p>
HEsD
helloyou
nAfe
Capitalization Mistakes:
<p>
HEsD
test
nAfe
Other Mistakes:
whyy whyy
Character:
47
Lines:
1
Words:
12
Grammatical errors:
5
Spelling Mistakes:
5

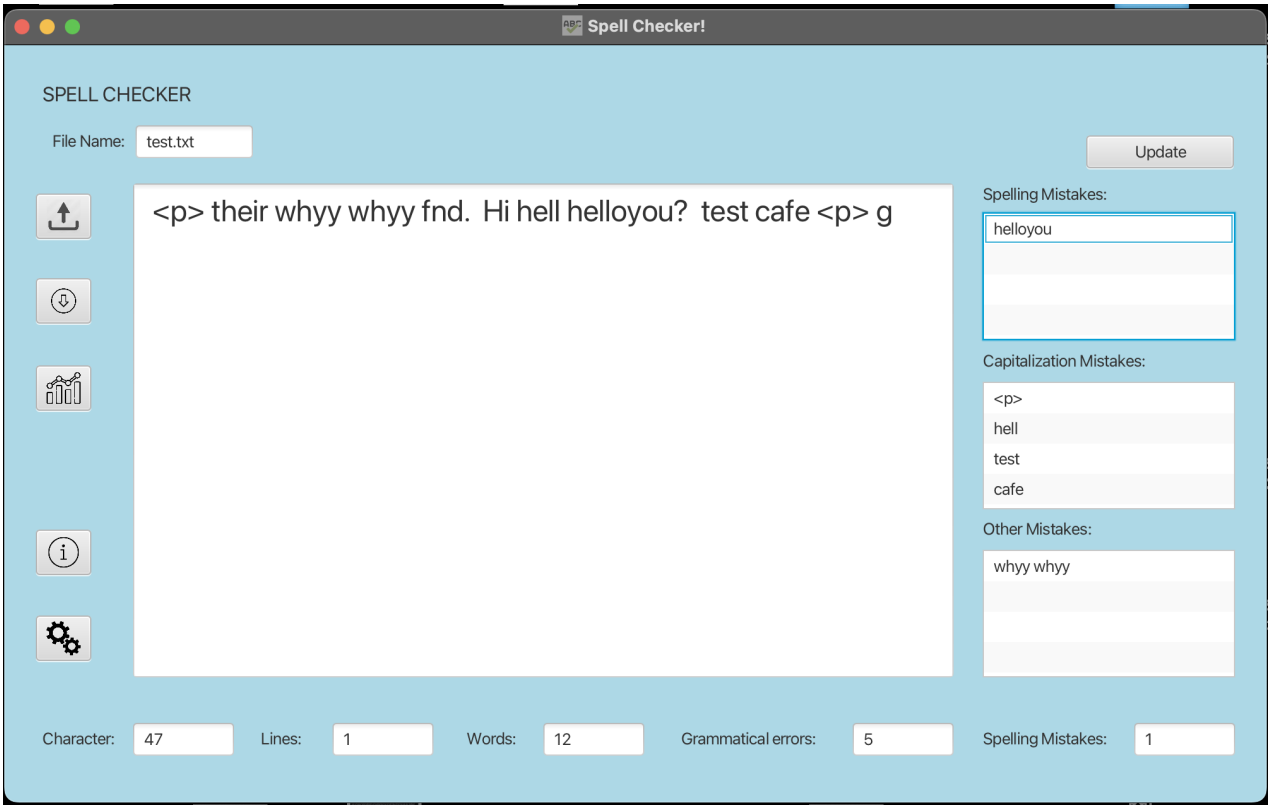

Spell Checker!
SPELL CHECKER
File Name:
test.txt
Update
<p> their whyy wh
> g
nAfe
Fix Manually
Ignore Once
Ignore
Add To User
cafe
rafe
safe
nife
nace
nake
nale
name
nane
nape
Spelling Mistakes:
HEsD
helloyou
nAfe
<p>
Capitalization Mistakes:
<p>
HEsD
test
nAfe
Other Mistakes:
whyy whyy
Character:
47
Lines:
1
Words:
12
Grammatical errors:
5
Spelling Mistakes:
5



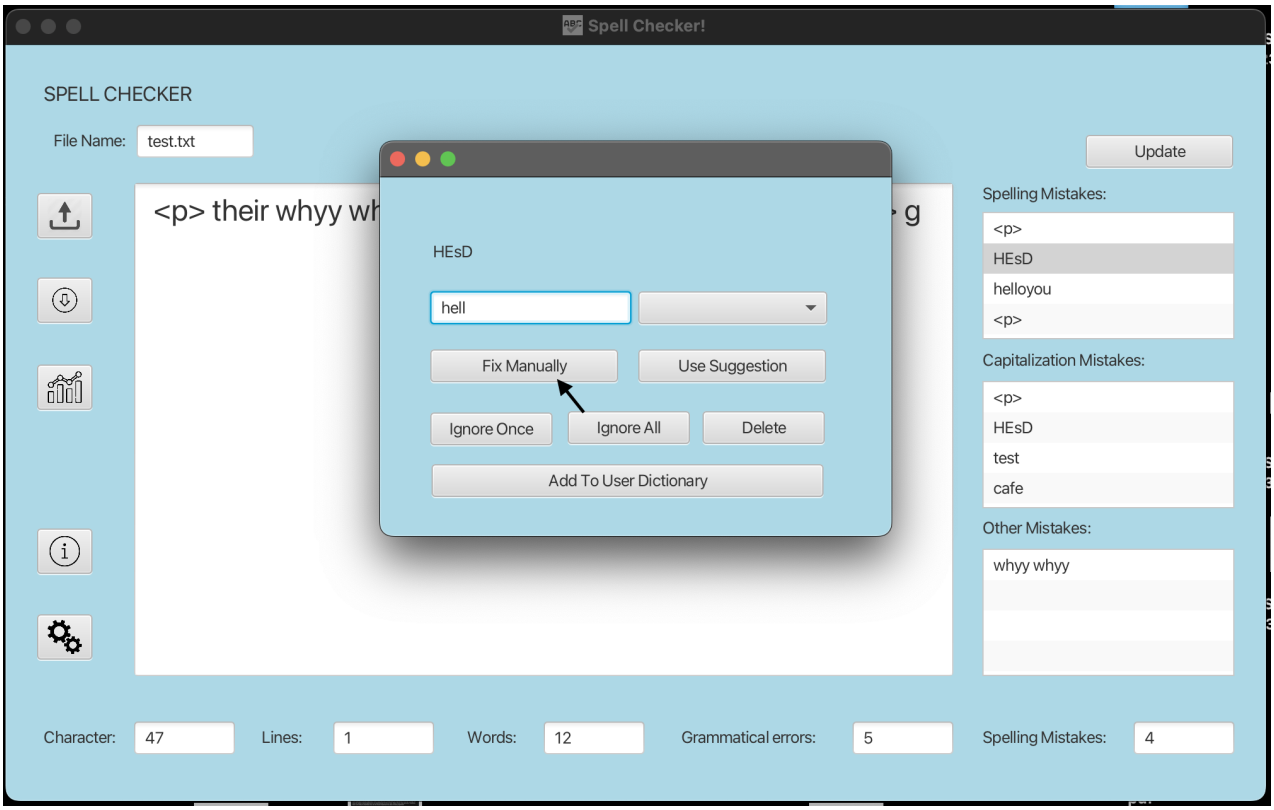


The user has the option of rectifying a mistake by either ignoring it once or ignoring all occurrences in the document. The latter is achieved through the 'ignore all' feature, which effectively removes all identified errors from the spelling mistake section.

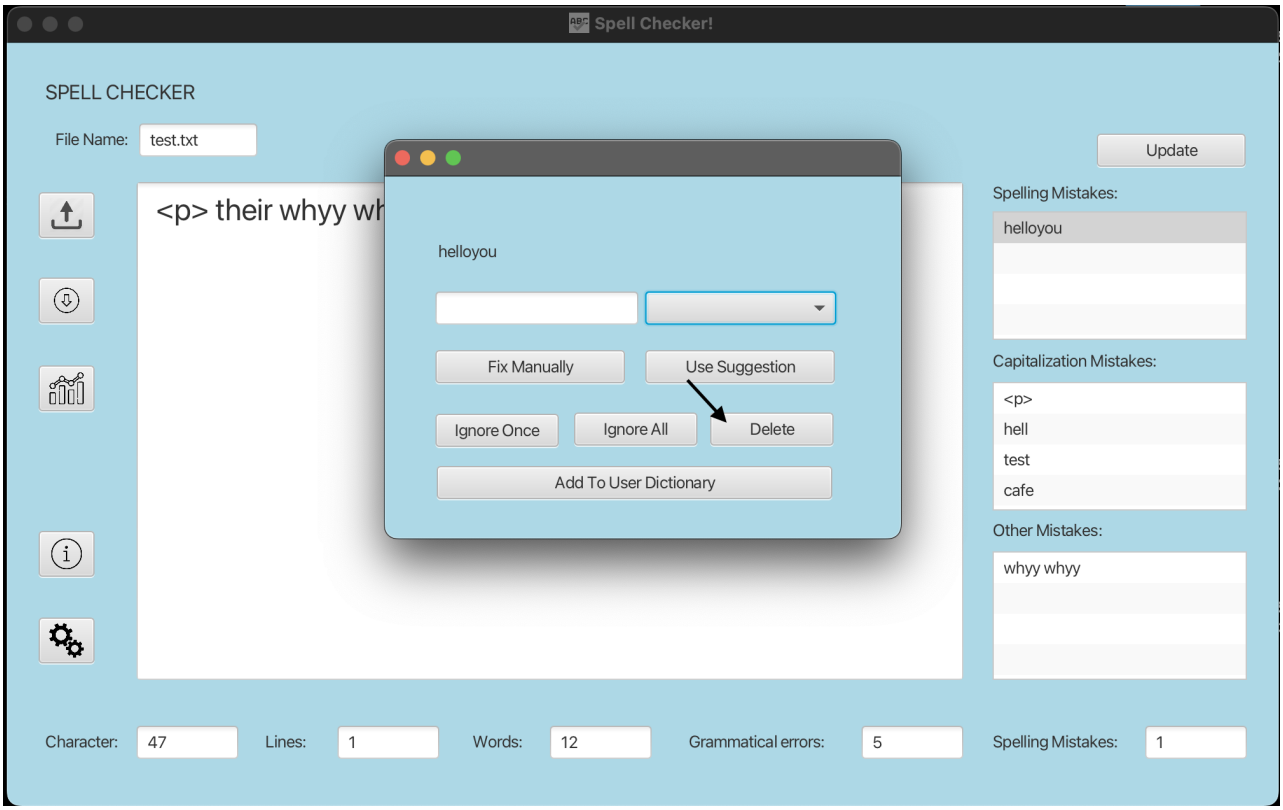


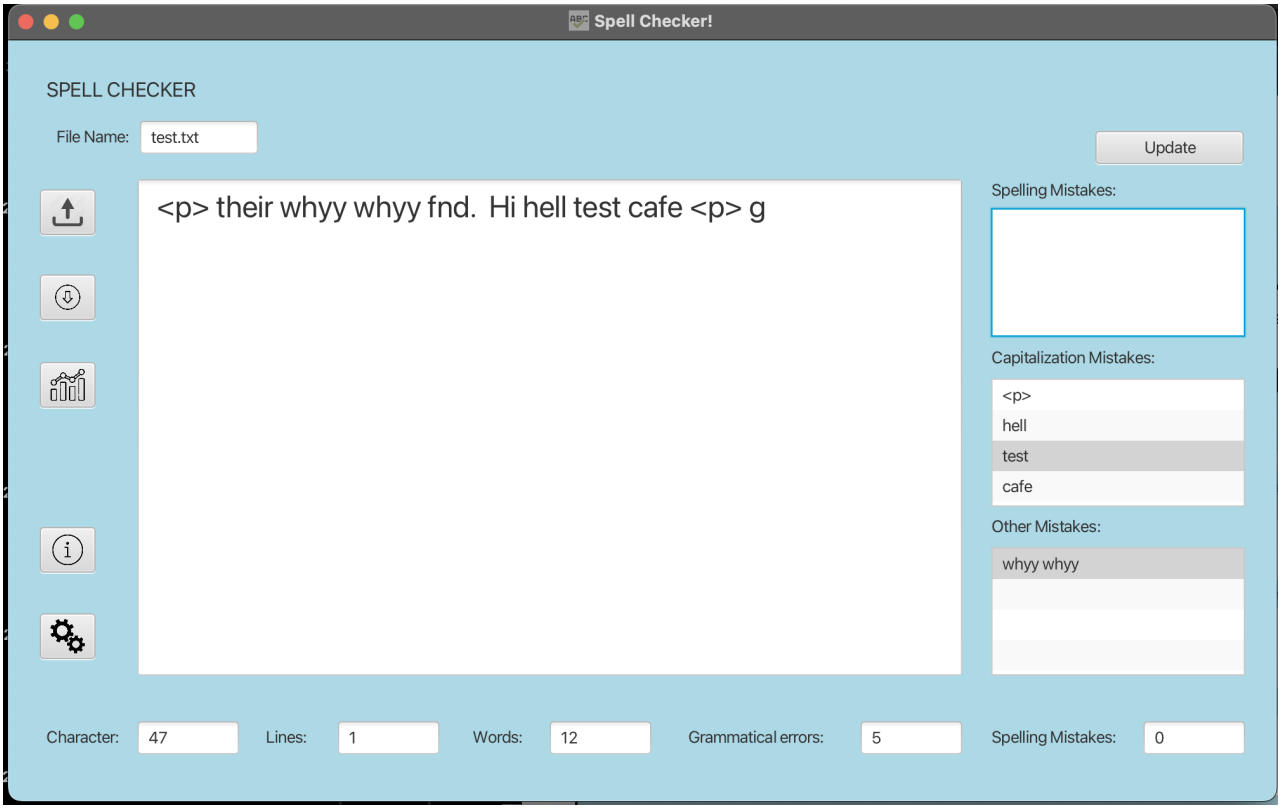


In addition, users have the option to manually correct any errors by inputting the desired replacement and selecting the "fix manually" function.The software has the ability to correct capitalization errors. It is capable of detecting sentences that begin with lowercase letters or words with capital letters in the middle.



Lastly, the user also has the option to delete the mistake by clicking delete and the mistake is then removed from the spelling mistakes section.

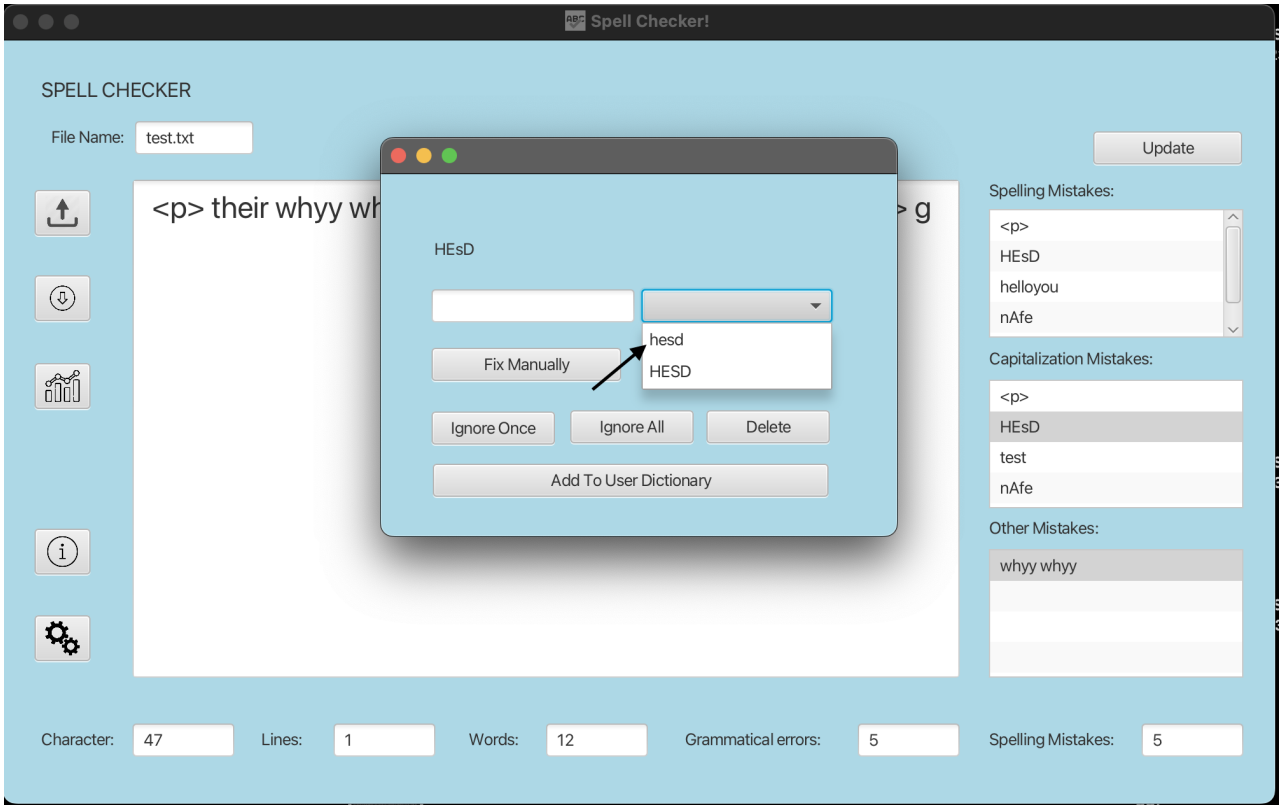


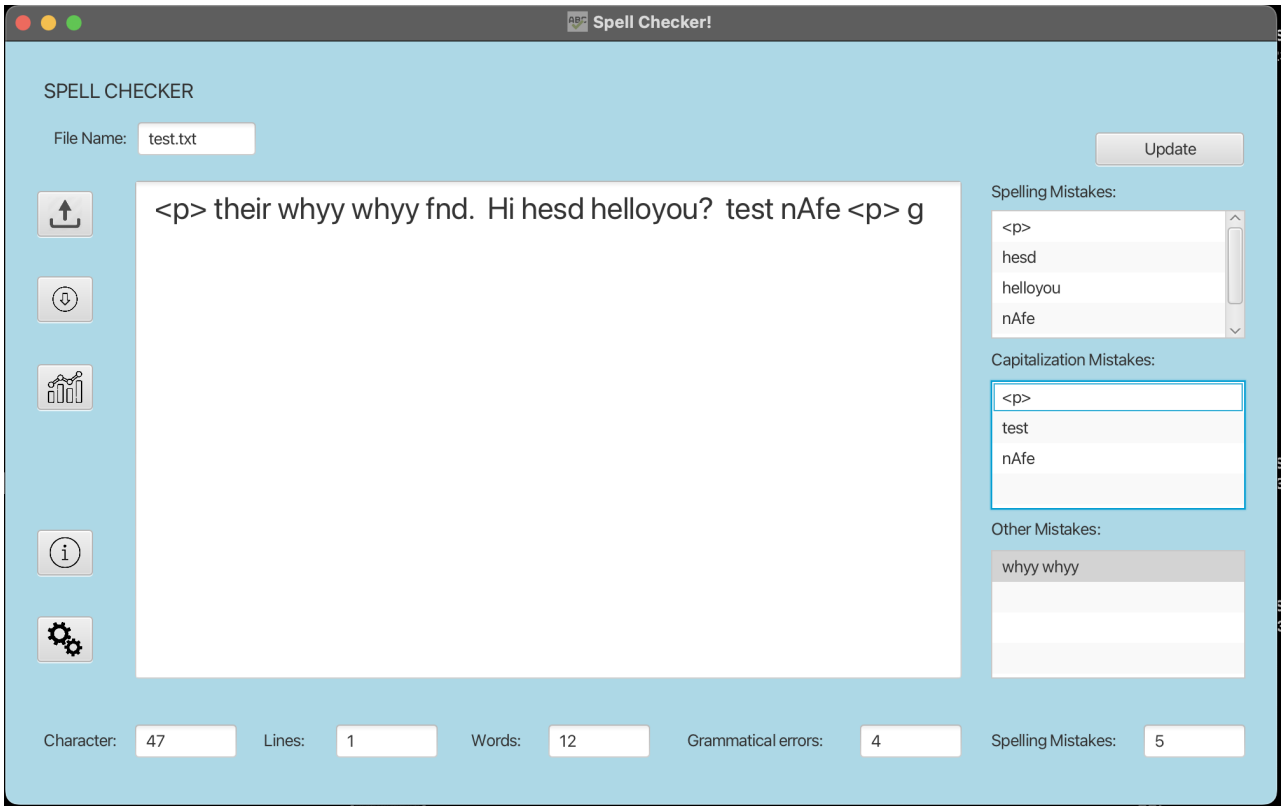


Capitalization Functionalities:

The software has the ability to correct capitalization errors. It is capable of detecting sentences that begin with lowercase letters or words with capital letters in the middle of the word. The mistake is then removed from the capitalization mistakes section once the change is applied.

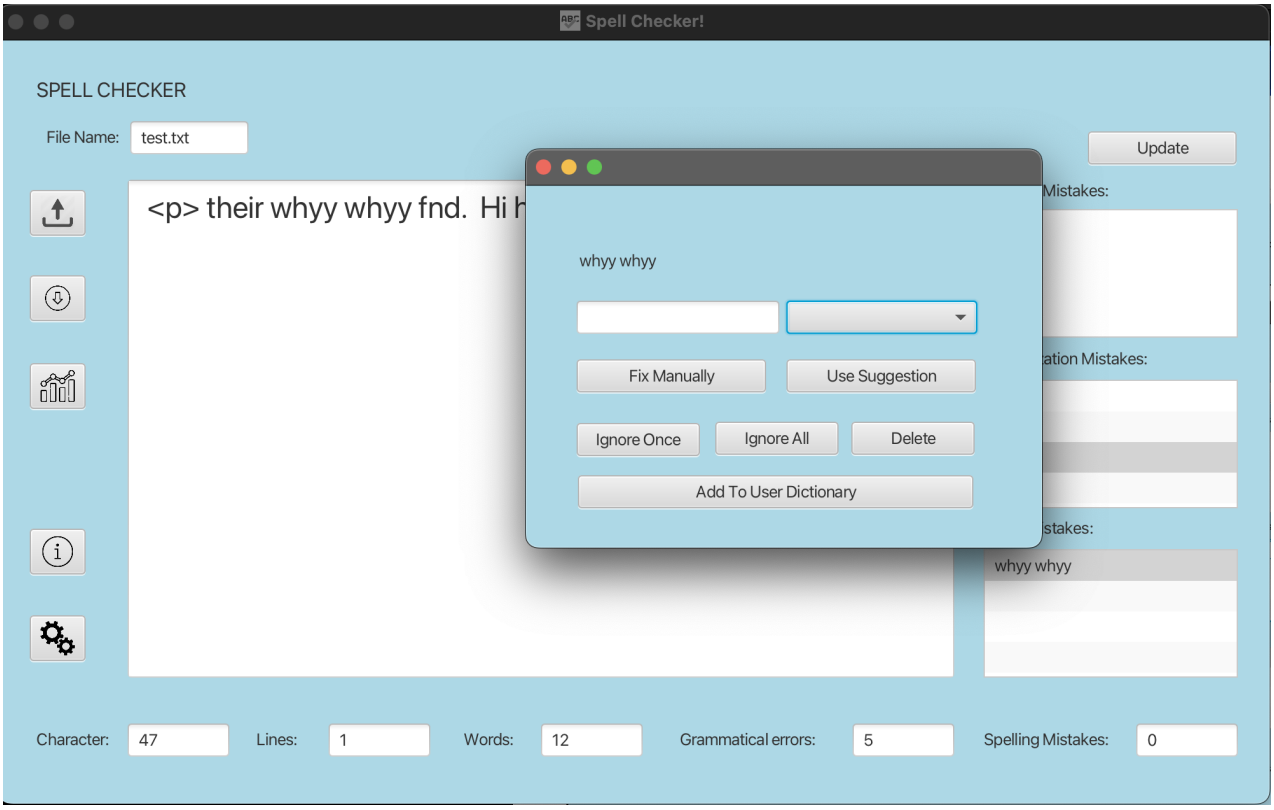
The user is given similar options to Spell-Checking, however this time the corrections are to capitalize the first letter when a sentence is detected for started with a lowercase letter. When the program detects a capitalization in the middle of a word the user is given the option to change the whole word be capital or lowercase

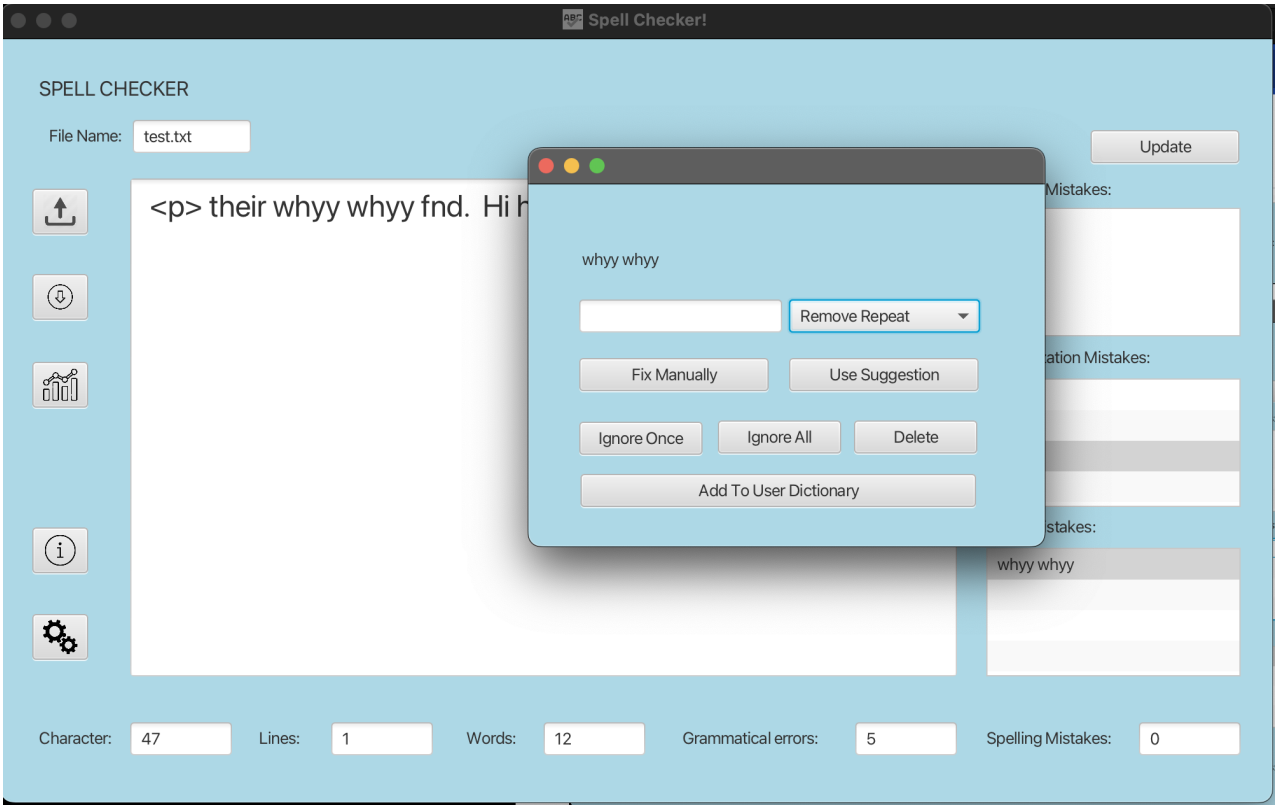


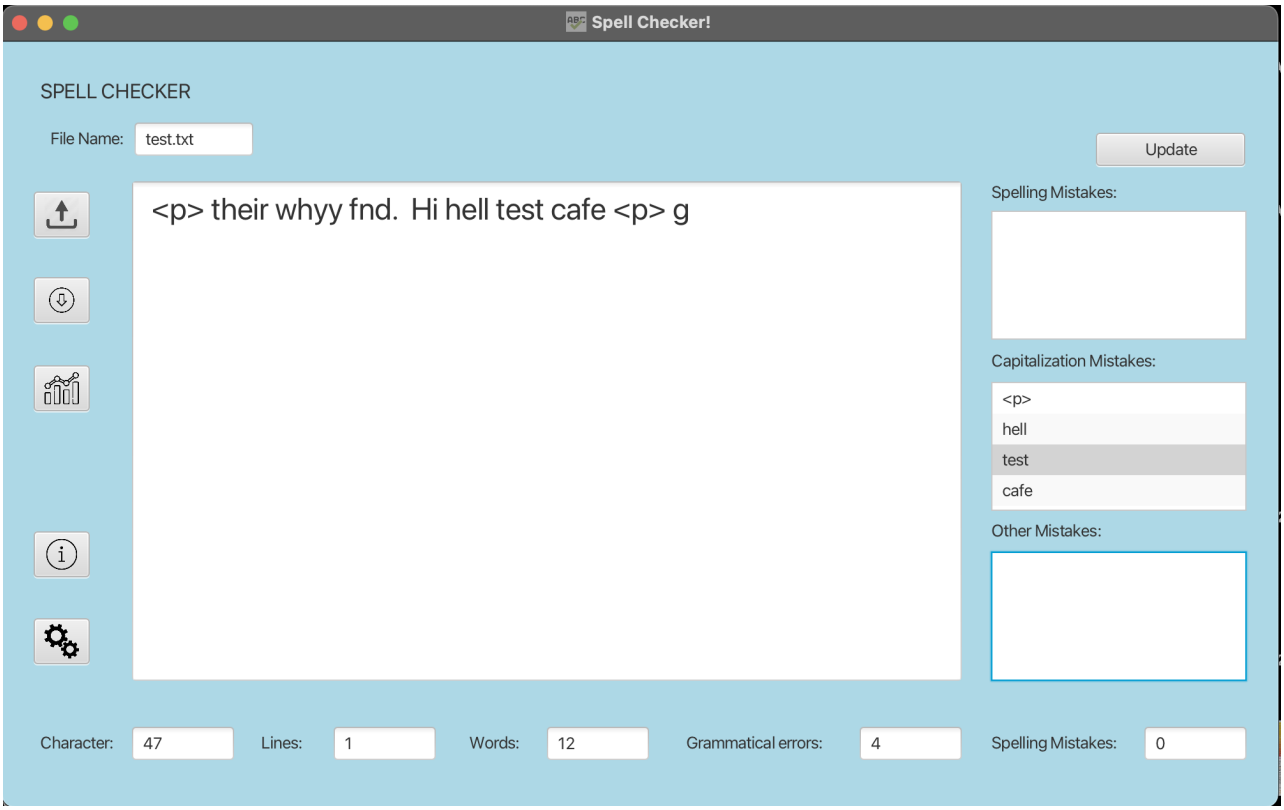


Other-mistake Functionalities:

The application has the capability to identify instances of double words in the text input and offers the user an option to rectify them. The available options to the user are similar to those of spelling mistakes and capitalization errors with the suggested correction being removal of the additional word. The mistake is then removed from the other mistakes section once the change is applied.

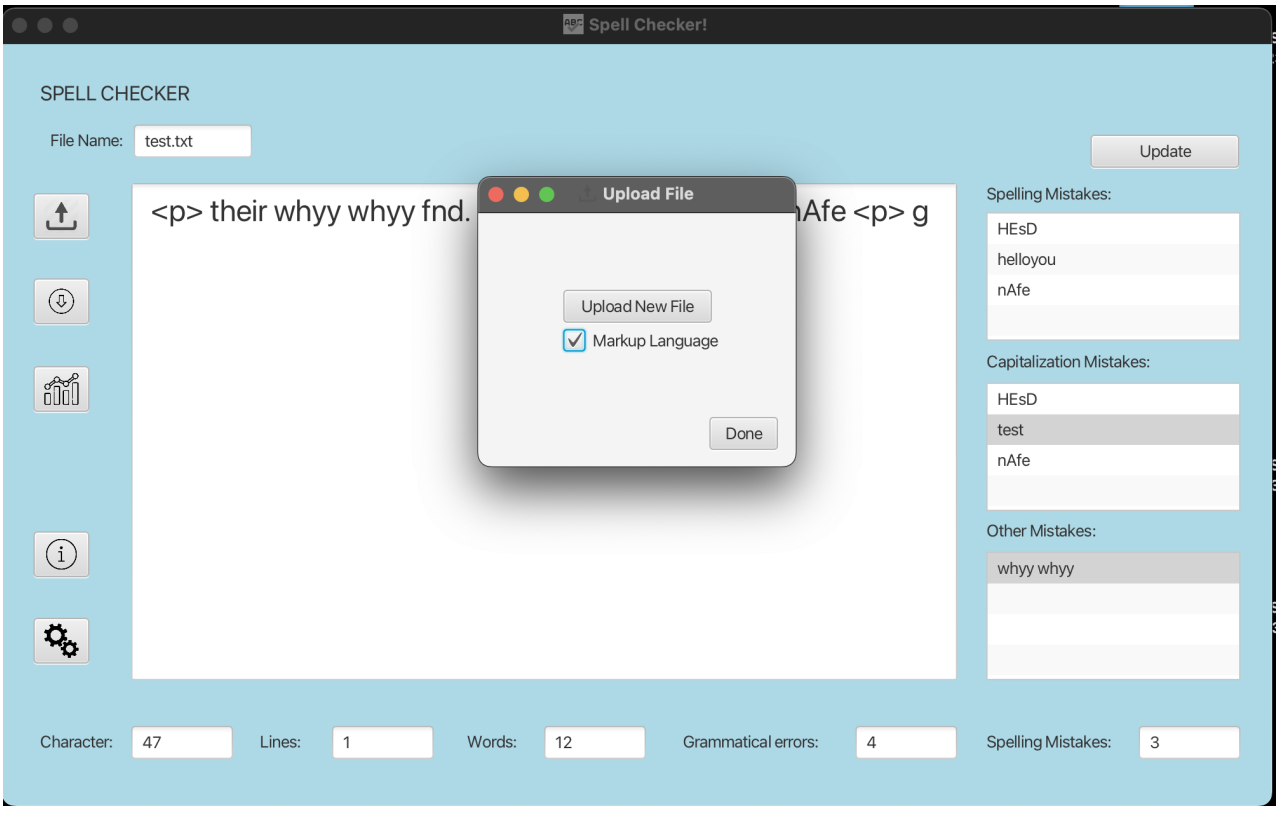


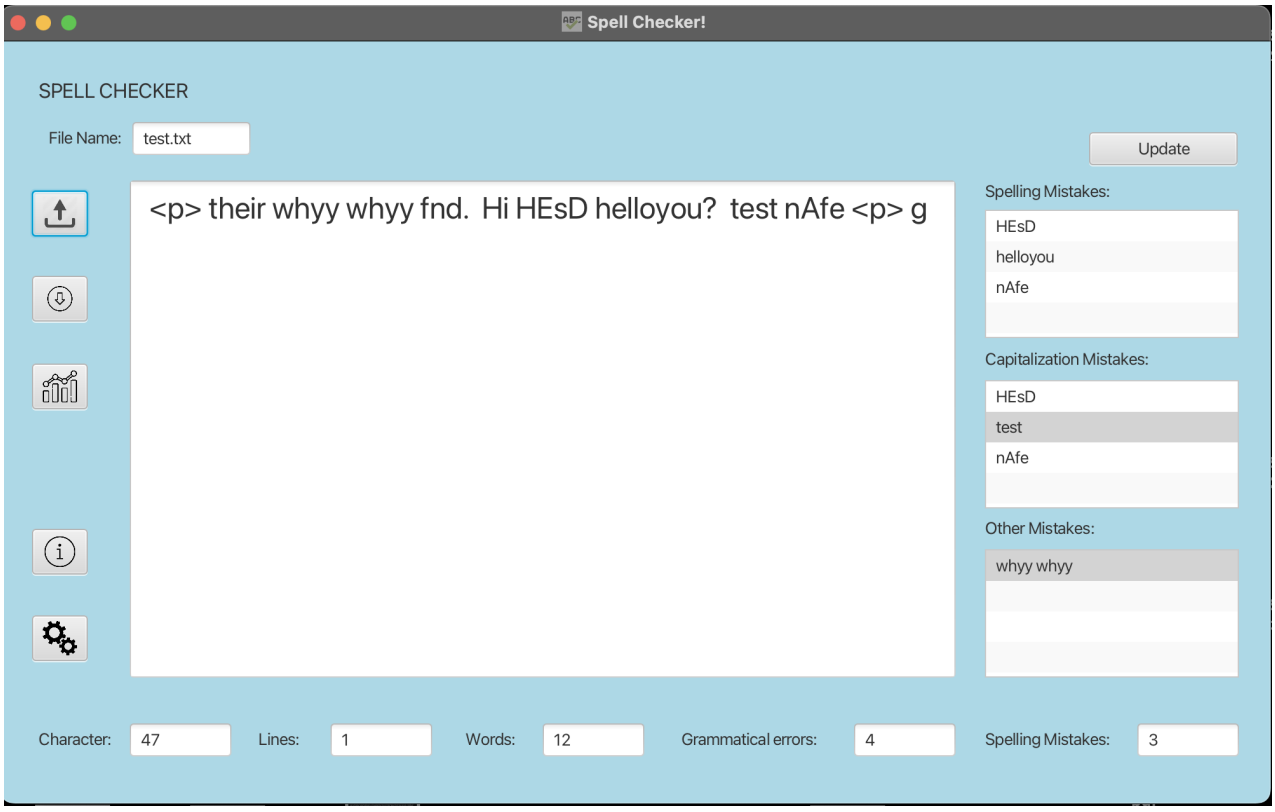




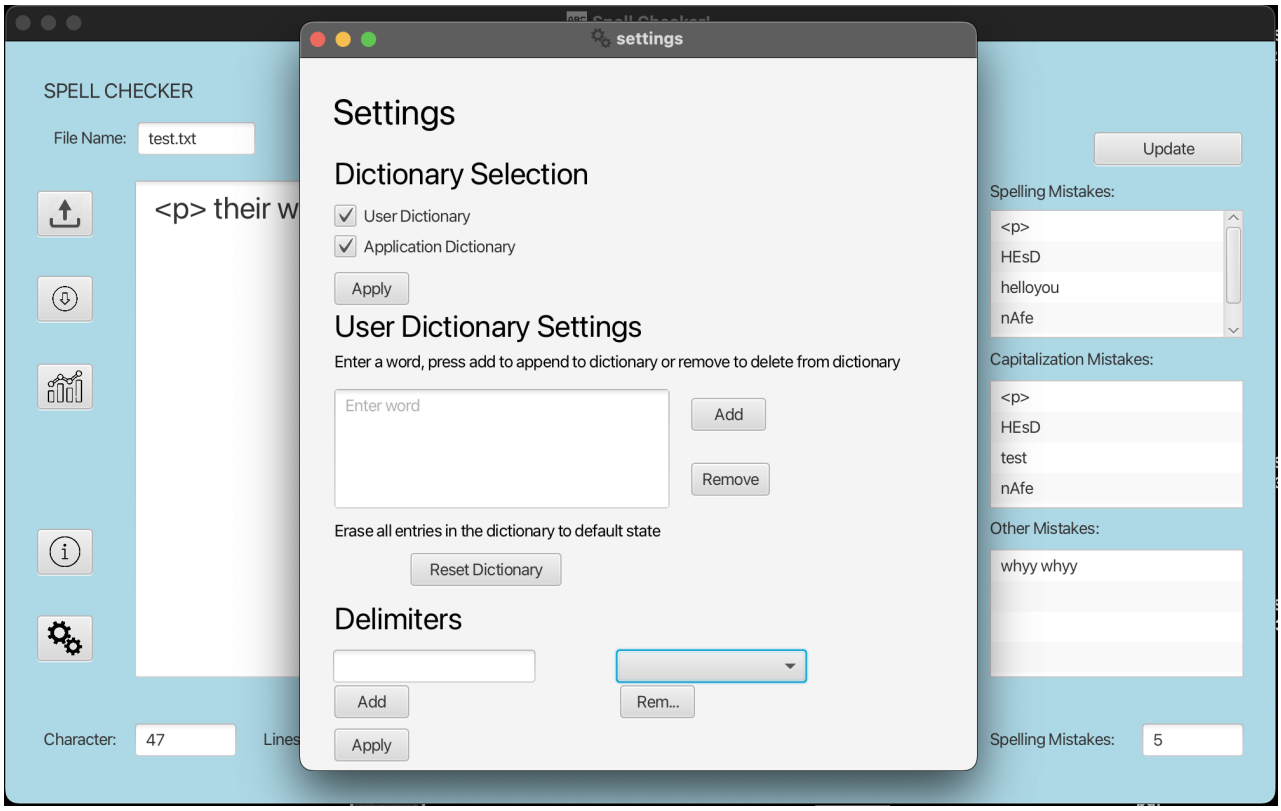
Other Functionalities:

The software grants the user the choice to indicate whether the text file comprises markup language. If the user selects so, the software will exclude the markup and not categorize it as a spelling error in the system.





The user can also specify the delimiters that they want to use throughout the spell-checking process. They can add and remove the list of delimiters whenever they want and can apply the changes by clicking the button apply.



3. Summary

3.1 Brief Description

This report outlines the four distinct software testing approaches we employed to guarantee the functionality of our program in line with the specified requirements. We have included a detailed explanation of the rationale behind our coding decisions and our method selection. Additionally, we have provided a thorough breakdown of our application's inner workings.

Unable to render {include} The included page could not be found.