Requirements Documentation

Main

This is the requirement documentation for COMPSCI 2212 - Fall 2023 - Group 1's project Spell Checker GUI. It is a comprehensive and detailed description of what this project should achieve, including its objectives, functionalities, features, constraints, and any other important considerations.

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	September, 28th 2023	Fuad Ghareeb	Created the Document
2.0	October, 1st 2023	Yazan, Mohamed, Adam	Finalized all 6 pages
2.5	October, 2nd	Alishba and Adam	Did the diagrams for 3.0
3.0	October, 3rd 2023	Alishba Farhan	Final Review

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 very popular software tools used by almost everyone in today’s society. Ensuring correct spelling in documents is crucial in order to prevent the readers from any misunderstandings. By automatically scanning for any spelling mistakes and detecting them, spell checkers help us produce an error free document, which ultimately improves credibility and effective communication.

The objective of this project is to create a desktop application that will allow us to scan through an entire document, compare it word by word to a dictionary, and provide the user with various options when encountering a spelling error. The options consist of annual corrections and further suggestions for auto-correction. After the user has selected all possible options for auto correct, this special-purpose application should have fulfilled its job 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

2. Domain Analysis

2.1 Description

This sub-page provides an analysis of the domain for the software. That is, the general field of business or technology in which the software will be used. By analyzing and better understanding the domain, the developers will have a better understanding of the needs and expectations of the software to function and deliver value to users in this domain.

2.2 Analysis Answer

What is the domain for this project?

- The software domain for this project is a desktop application that serves as a document spell checker. It is designed to be used in an educational institution, specifically The University of Western Ontario.

What do we know about the domain?

- Spell-checking application is an important tool in today's academic world. The software application made for this project could be used by students, faculty, and staff to ensure that any written content and documents meet academic writing standards. A popular and notable application in this domain is Grammarly, a very popular typing assistant. Grammarly is widely used in many educational institutions for document creation and error checking.

What are the common issues encountered in this domain?

- When discussing the common issues encountered in this domain, several challenges arise. We must ensure the accuracy and correctness of documents while also being able to handle them efficiently, as well as offering tools for collaborative editing. Another issue is ambiguity with words that seem similar to each other like "their" and "there". Finally, addressing the adherence to specific academic style guides and the need for consistent language falls under the criteria.

What are the common solutions to the above issues in this domain?

- There are common solutions to the above issues in this domain. We can develop an advanced spell-checking algorithm that is capable of recognizing academic terminology. Integrating plagiarism detection tools and academic databases in order to support research. And to meet academic writing standards, templates could be provided with guides to adhere to specific styles in order to assist users.

How can we use this domain understanding to improve or accelerate the development of this project?

- A deep understanding of this domain will help us improve or accelerate the development of this project. We will be able to align the spell checker software development to the academic needs and challenges of the institutions. On top of that, understanding the domain will help us face less challenges throughout the developing process.

3. Functional Requirements

In this section, we will delve into the functional requirements of the software project. Functional requirements are essentially statements that describe the behavior that a system must exhibit. These statements outline the necessary actions that the system must perform in order to meet the expectations and needs of the user. Essentially, functional requirements can be viewed as the features that the user will observe and interact with. Our program's functional requirements are categorized into nine parts, each containing use cases which describe the software's specific usage scenarios from the perspective of a particular actor. Additionally, each use case is depicted in a use case diagram. To provide a graphical representation of the interaction flow within a specific scenario, we have used activity diagrams

3.1 General Workflow

Functionality to be delivered:

- Have the program load a start page that explains how to use it. The start page is a text file.
- Upload a text file into the program and open it.
- Spell-check the file that was uploaded.
- Allow the user to save changes made to the file either on the same file or save a copy in a new location.
- Allow the user to discard changes by either loading a new file or exiting the program.
- Warn the user if the file is going to be discarded.
- Allow the user to upload more files to be checked within the same session.

Actors:

- User
- External Dictionary

Actor:	User
Description:	The primary actor that interacts with the software to spell-check files.
Aliases:	End User, Client, Customer
Inherits:	None
Actor Type:	Person - Active
Relationships	None

Actor:	External Dictionary
Description:	Provides the reference for the spell check operation to check the spelling of words.
Aliases:	Dictionary
Inherits:	None
Actor Type:	External System - passive
Relationships	Assists the user in the spell-check process

Use Cases:

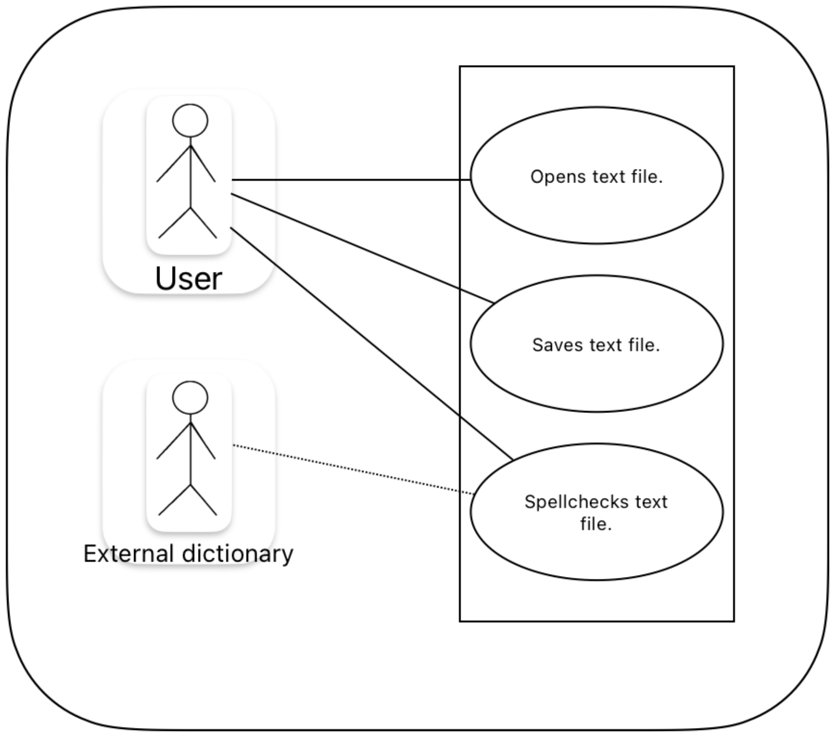
Name:	Open Text File
Primary actor:	User
Secondary actor:	None
Goal in context:	To open a file for the purpose of spell-checking.
Preconditions:	The user has access to the software application. The program is loaded with the start page.

Trigger:	The user selects the option to open a file
Scenario:	1. User starts the application. 2. User chooses to pen a file. 3. File is loaded into the application.
Alternatives:	None
Exceptions:	• Uploading an empty file • Uploading too large of a file for the program to handle

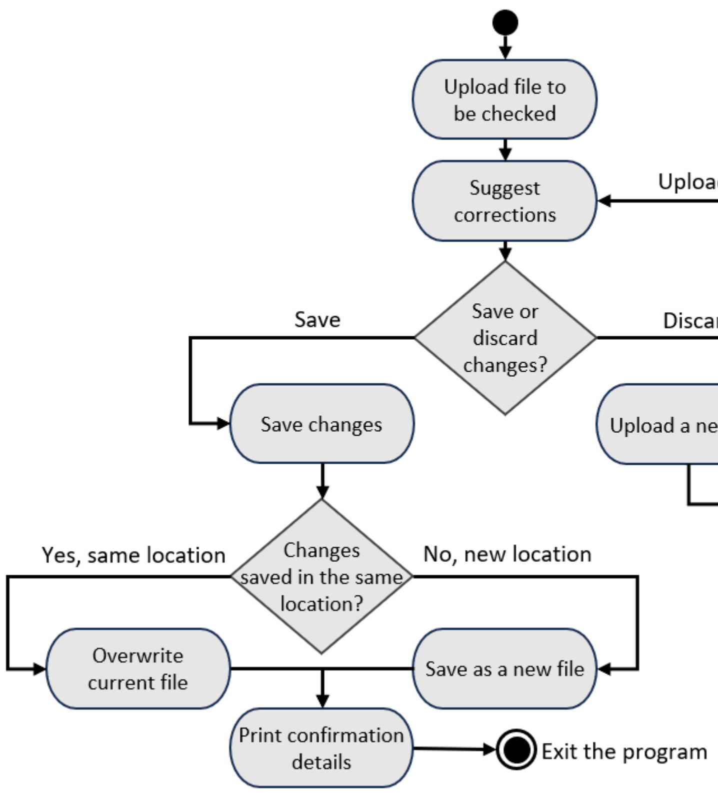
Name:	**Spell Check Text File**
Primary actor:	User
Secondary actor:	External Dictionary
Goal in context:	To check the spelling of words in the opened file
Preconditions:	A file has been opened in the application.
Trigger:	The user initiates the spell-check function
Scenario:	1. The user clicks on the spell-check button 2. The application checks the words with the dictionaries available 3. Any errors are underlined and fixes are suggested
Alternatives:	None
Exceptions:	• The program misses and error or does not recognize it

Name:	**Save text file**
Primary actor:	User
Secondary actor:	None
Goal in context:	To save or discard changes made by the program
Preconditions:	The user has access to the software application. The program is loaded with the start page.
Trigger:	The user clicks on save or discard the file
Scenario:	1. User clicks on save the file 2. User can chose to save it over the original file or in a new location 3. User overwrites the main file 4. The file is saved

Alternatives:	Alternative one: 1. The user clicks on save the file 2. Users can chose to save it over the original file or in a new location 3. The user saves it in a new location 4. The file is saved Alternative two: 1. The user closes the file 2. The user gets a warning 3. File is discarded Alternative three: 1. The user opens a new file 2. The user gets a warning 3. File is discarded 4. New file is opened
Exceptions:	• Saving the file more than once



Activity Diagram:



3.2 Documents

Functionality to be delivered:

- The user must select a dictionary, in English only, for inclusion in the spell checker application.
- The chosen dictionary should serve as the source of correctly spelled words for the spell checker.
- The application must support a user dictionary.
- Users can add words to this user dictionary that are not found in the system dictionary.
- Users should have options to perform various actions on the user dictionary:
- a. Add words to the user dictionary (as a correction option).
- b. Edit or remove words from the user dictionary.
- c. Reset the user dictionary to an empty state.

Actors:

- User
- External Dictionary

Actor:	**User**
Description:	The primary actor of the software, who uses this application to upload various files including markup languages or without.
Aliases:	End User, Client, Customer
Inherits:	None
Actor Type:	Person - Active
Relationships	None

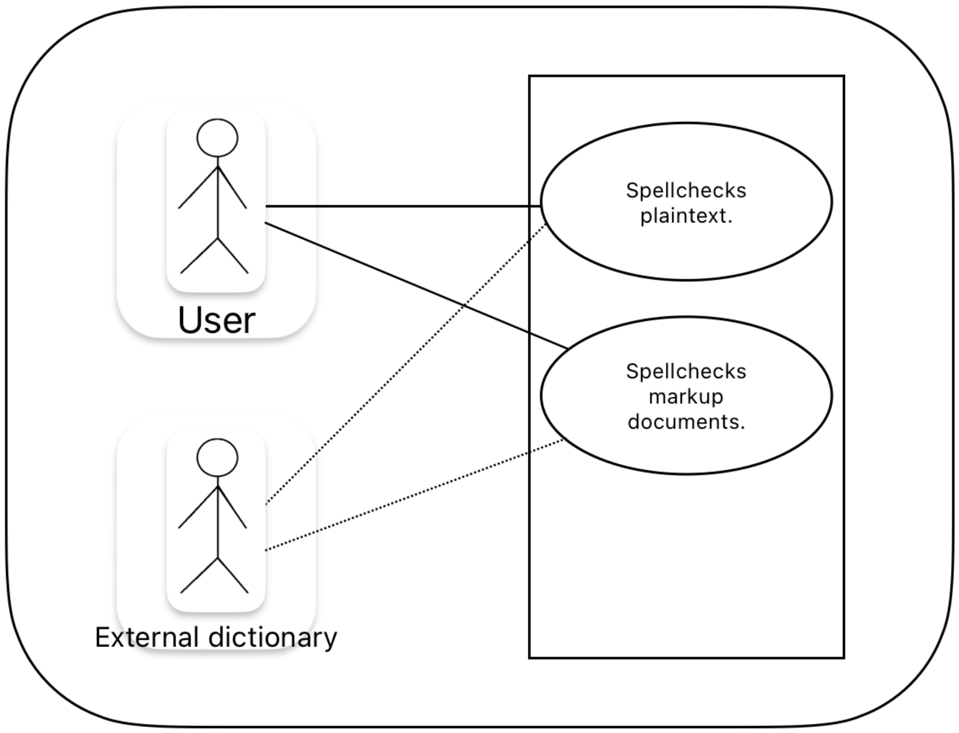
Actor:	**External Dictionary**
Description:	Provides the reference for the spell check operation to check the spelling of words
Aliases:	Dictionary
Inherits:	None

Actor Type:	External System - passive
Relationships	Assists the user in the spell-check process

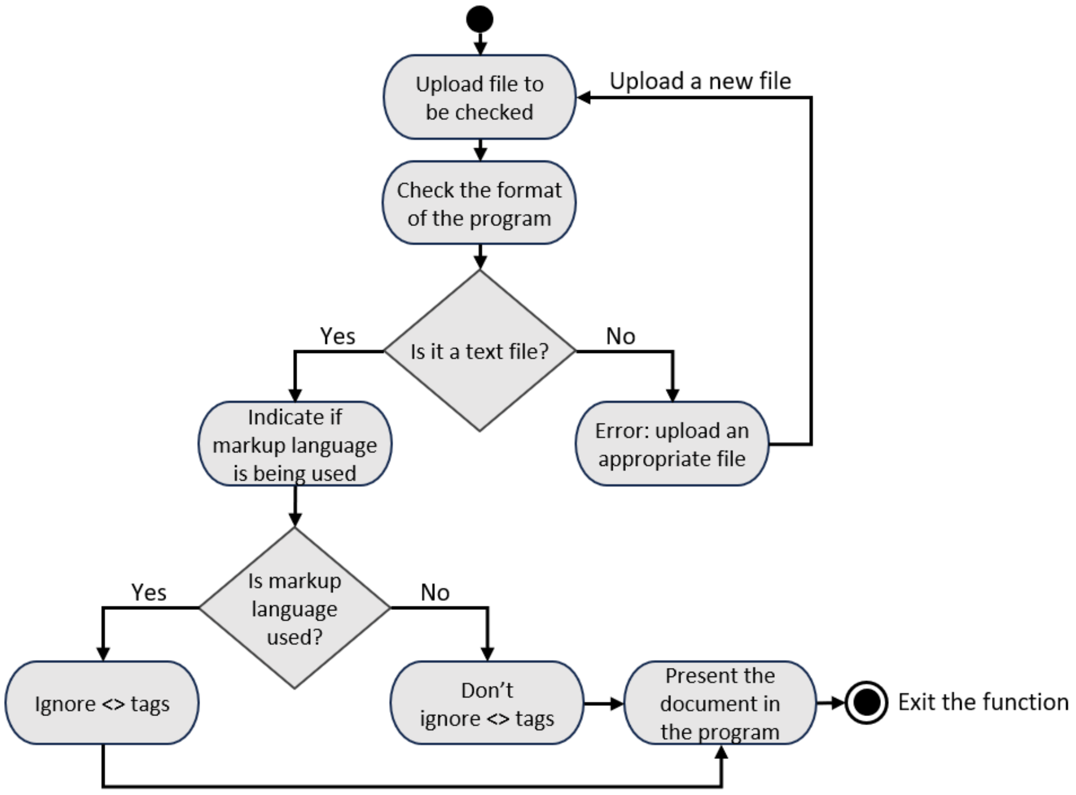
Use Cases:

Name:	**Spellchecks plaintext**
Primary actor:	User
Secondary actor:	External Dictionary
Goal in context:	Accept various files, in various lengths and formats to be spellchecker and export them back.
Preconditions:	The program is loaded with a text file, start page, that explains how the program runs.
Trigger:	The user wants to upload a new file to get checked
Scenario:	1. The user loads an ASCII file into the application. 2. The application performs spell-check, referencing the External Dictionary. 3. Discrepancies or errors are highlighted and presented to the user. 4. The document is presented to the user
Alternatives:	None
Exceptions:	• The program is unable to read the file because the document is corrupted • The program is unable to export the file as the program has no more space

Name:	**Spellchecks Markup Documents**
Primary actor:	User
Secondary actor:	External Dictionary
Goal in context:	Accept various files, in various lengths and formats to be spellchecker and export them back.
Preconditions:	The program is loaded with a text file, start page, that explains how the program runs.
Trigger:	The user wants to upload a new file to get checked
Scenario:	1. The user loads an HTML/XML file into the application. 2. The user selects the option to filter/skip tags during the spell check. 3. The application performs spell-check, ignoring text within <> and referencing the External Dictionary for the rest. 4. Discrepancies or errors are highlighted and presented to the user. 5. Document is presented to the user
Alternatives:	1. The user loads an HTML/XML file into the application. 2. The user does not select the option to filter tags 3. The application performs spell-check including text within <> and referencing the External Dictionary for the rest. 4. Discrepancies or errors are highlighted and presented to the user. 5. The document is presented to the user
Exceptions:	• If the user forgets to opt-out the tags codes written in languages like HTML can get corrupted



Activity Diagram:



Upload file to be checked
Upload a new file
Check the format of the program
Is it a text file?
Yes
No
Indicate if markup language is being used
Error: upload an appropriate file
Is markup language used?
Yes
No
Ignore <> tags
Don't ignore <> tags
Present the document in the program
Exit the function

3.3 Dictionaries

Functionality to be delivered:

- Allow the user to choose a dictionary for the application.
- Load the dictionary into memory at runtime
- Ensure that the system dictionary remains immutable.
- Implement support for a user dictionary.
- Offer the User the following actions for managing the user dictionary:
    - Allow the user to add words to the user dictionary as a correction option.
    - Enable the user to edit or remove words from the user dictionary.
    - Provide the option to reset the user dictionary to its default empty state.

Actors:

- User
- External Dictionary
- Administrator

Actor:	**User**
Description:	The primary actor of the software. Uploads the files and chooses what dictionary to use to spell-check the file.
Aliases:	End User, Client, Customer
Inherits:	None
Actor Type:	Person - Active
Relationships:	None

Actor:	**External Dictionary**
Description:	Provides the reference for the spell check operation to check the spelling of words
Aliases:	Dictionary
Inherits:	None
Actor Type:	External System - Passive
Relationships	Source for the dictionary to be loaded.

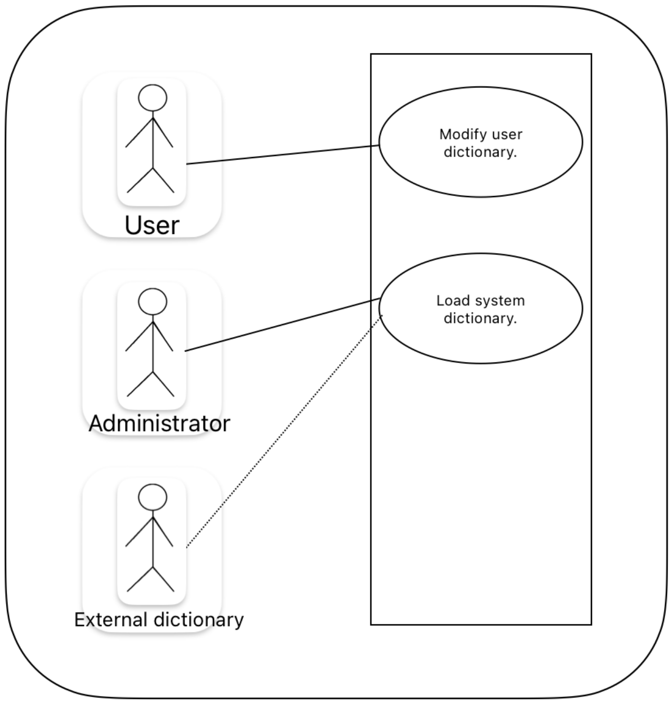
Actor:	**Administrator**
Description:	The administrator is a special account that is responsible for setting up maintaining, and ensuring the proper functioning of the spell-checking software.
Aliases:	Developer
Inherits:	User
Actor Type:	Person - Active
Relationships	None

Use Cases:

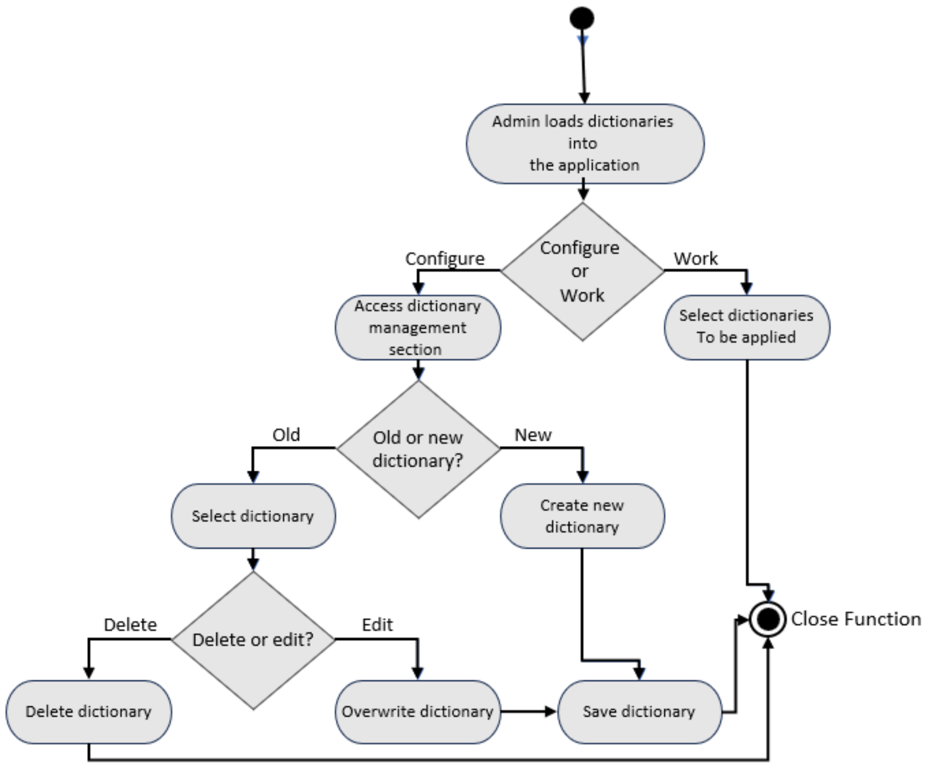
Name:	**Load System Dictionary**
Primary actor:	Administrator
Secondary actor:	External Dictionary
Goal in context:	To initialize and make available the primary dictionary used for standard spell-checking
Preconditions:	Application is initiated and the dictionary is located either online or locally.
Trigger:	The application starts or the administrator initiates the process as a refresher.

Scenario:	1. Application is started 2. As part of the start process, the administrator instructs the application to fetch the system dictionary from the external dictionary file 3. Load user dictionaries presented in the application configuration files 4. The dictionaries is loaded into the memory 5. The Spell-checker is now ready
Alternatives:	1. The administrator decides to fetch the dictionary while the program is running as a refreshment due to an error or due to an update that might have occurred to the file locations 2. The dictionary is fetched from an external dictionary file 3. The dictionary is loaded into the memory 4. The Spell-checker is now ready
Exceptions:	• Administrator fails to find the External Dictionary file • The dictionary file is missing or corrupted.

Name:	**Modify User Dictionary**
Primary actor:	User
Secondary actor:	None
Goal in context:	To add, edit, or remove words from the user-specific dictionary.
Preconditions:	The user dictionary is accessible and originally empty.
Trigger:	The user identifies a word not recognized by the system dictionary.
Scenario:	1. The user adds the word to the user dictionary. 2. Optionally, the user can edit or remove words. 3. The user dictionary is updated.
Alternatives:	1. The user resets the dictionary to its default empty state. 2. The user creates a new dictionary to edit, add, and remove words from
Exceptions:	• Failing to update the user dictionary might cause inconsistency in the work of the user.



Activity Diagram:



3.4 Words

Functionality to be delivered:

- Explicitly define what the program should consider a word, by using one of the following delimiters:
    - Punctuation (periods, commas, colons, semicolons, question marks, exclamation marks, brackets, parentheses, braces, quotation marks, and hyphens)
    - White-space (tab, space, and newline)
- Hyphenated words will be treated as multiple separate words with the hyphen as a delimiter.
- Website addresses and files with extensions will be treated as separate words with the period as their delimiter. This will cause false errors, but the user will be given the option to ignore them.
- The system dictionary will be comprehensive and will be preloaded with all the forms of a word vis-à-vis prefixes and suffixes.
- A string of pure numbers will be ignored entirely.
- Strings of letters containing some numbers will be checked as words (see 3.5 Error Detection and 3.6 Error Correction).

Actors:

- User
- Administrator

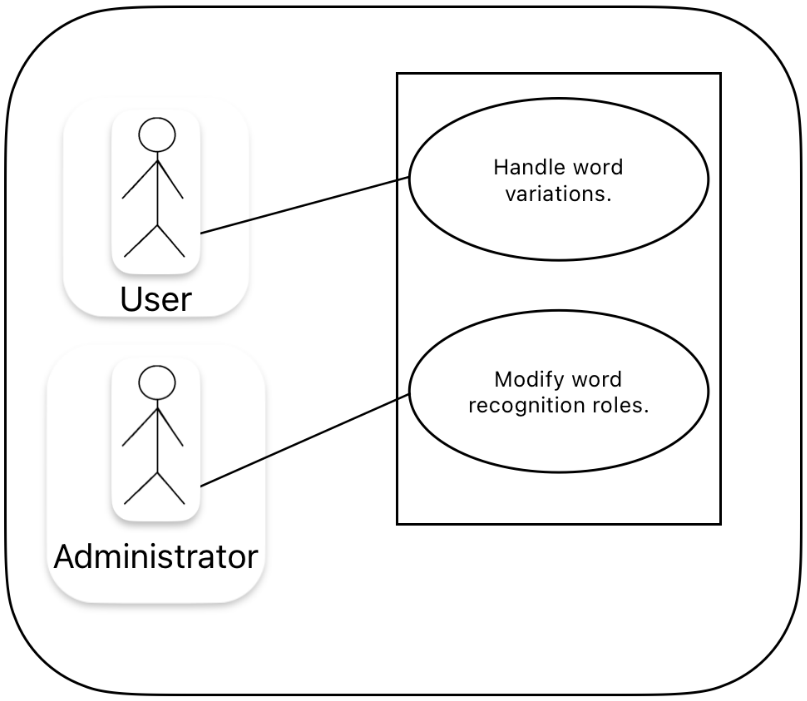
Actor:	**User**
Description:	The user will provide text input, which will get parsed and each word will get individually recorded.
Aliases:	None
Inherits:	None
Actor Type:	Person - Active
Relationships:	None

Actor:	**Administrator**
Description:	The administrator is a special account that is responsible for setting up, maintaining, and ensuring the proper functioning of the spell-checking software.
Aliases:	Developer
Inherits:	User
Actor Type:	Person - Active
Relationships	None

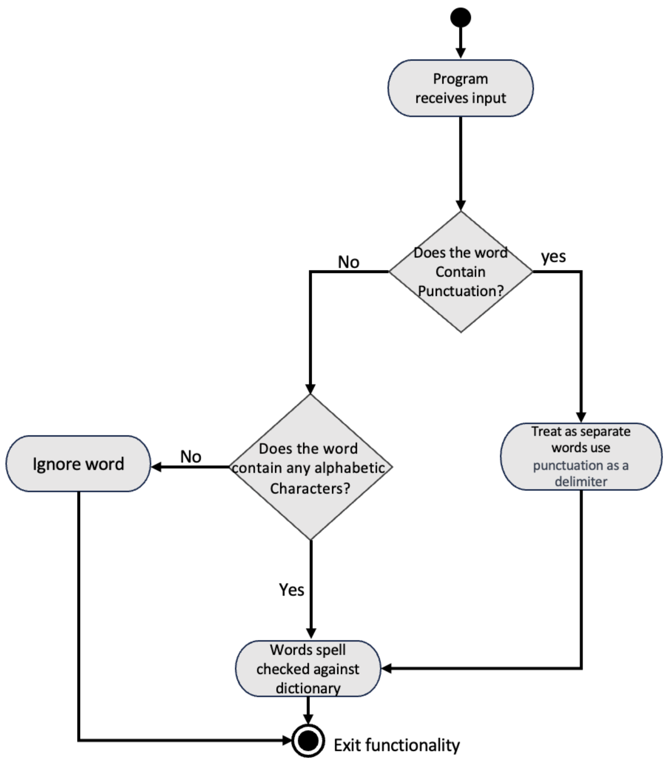
Use Cases:

Name:	**Handle Word Variation**
Primary actor:	User
Secondary actor:	None
Goal in context:	Ensure that the spell-check function works correctly for different word variations
Preconditions:	• A text is loaded into the application
Trigger:	The user clicks on the spell-check button
Scenario:	1. The application identifies words separated by white-space or punctuation. 2. Hyphenated words will be treated as multiple separate words with the hyphen as a delimiter. 3. Words with prefixes and suffixes are preloaded in the dictionary. 4. Numeric words or mixtures, like "l00t", will be checked as words. 5. Words are spell-checked against the dictionary.
Alternatives:	None
Exceptions:	• Fully numeric words are skipped. • Filenames and URLs are split at punctuation for spell-checking.

Name:	**Modify Word Recognition Rules**
Primary actor:	Administrator
Secondary actor:	None
Goal in context:	Adjust word definition criteria.
Preconditions:	• A text is loaded into the application
Trigger:	The user clicks on the spell-check button
Scenario:	1. The administrator reviews word recognition rules. 2. Adjustments are made and saved.
Alternatives:	None
Exceptions:	• If accuracy decreases, adjustments are revisited.



Activity Diagram:


Program receives input
Does the word Contain Punctuation?
No
yes
Does the word contain any alphabetic Characters?
Treat as separate words use punctuation as a delimiter
Ignore word
No
Yes
Words spell checked against dictionary
Exit functionality

3.5 Error Detection

Functionality to be delivered:

- Detect if the first word after a period isn't capitalized, and flag it by underlining it in red.
- Detect letters that are capitalized mid-word, and flag them with a red line.
- Detect words not found in the dictionary (see 3.3 Dictionaries); flag misspelled words.
- Allow mid-word capitals if the entire word is capitalized.
- Detect if two successive words are identical (this is case insensitive), and flag with a red line.
- If a capitalized (either all capitals or just the first letter is capitalized) word isn't in the dictionary, then flag it with a yellow line prompting the user to review it.

Actors:

- User
- External Dictionary

Actor:	**User**
Description:	The primary actor that interacts with the software to spell-check files.
Aliases:	End User, Client, Customer
Inherits:	None
Actor Type:	Person - Active
Relationships	None

Actor:	**External Dictionary**
Description:	Provides the reference for the spell check operation to check the spelling of words
Aliases:	Dictionary
Inherits:	None
Actor Type:	External System

Relationships	Assists the user in the spell-check process

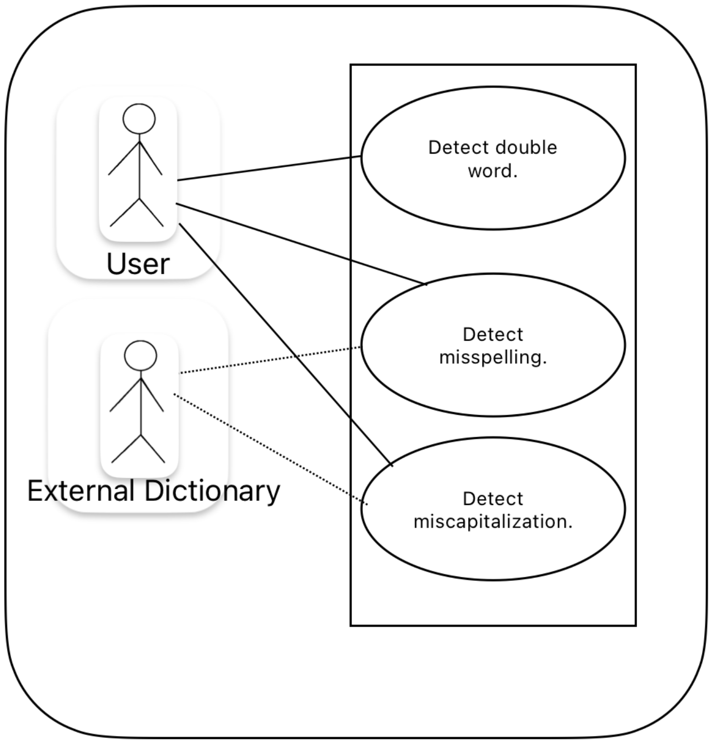
Use Cases:

Name:	**Detect Double Words**
Primary actor:	User
Secondary actor:	None
Goal in context:	Identify repeated consecutive words.
Preconditions:	• The file is uploaded to be checked
Trigger:	The initiation of the spell-check function
Scenario:	1. The user inputs or loads a text document. 2. Application scans for consecutive repeated words. 3. Double words are highlighted.
Alternatives:	1. None
Exceptions:	• Intentional repetitions may be highlighted.

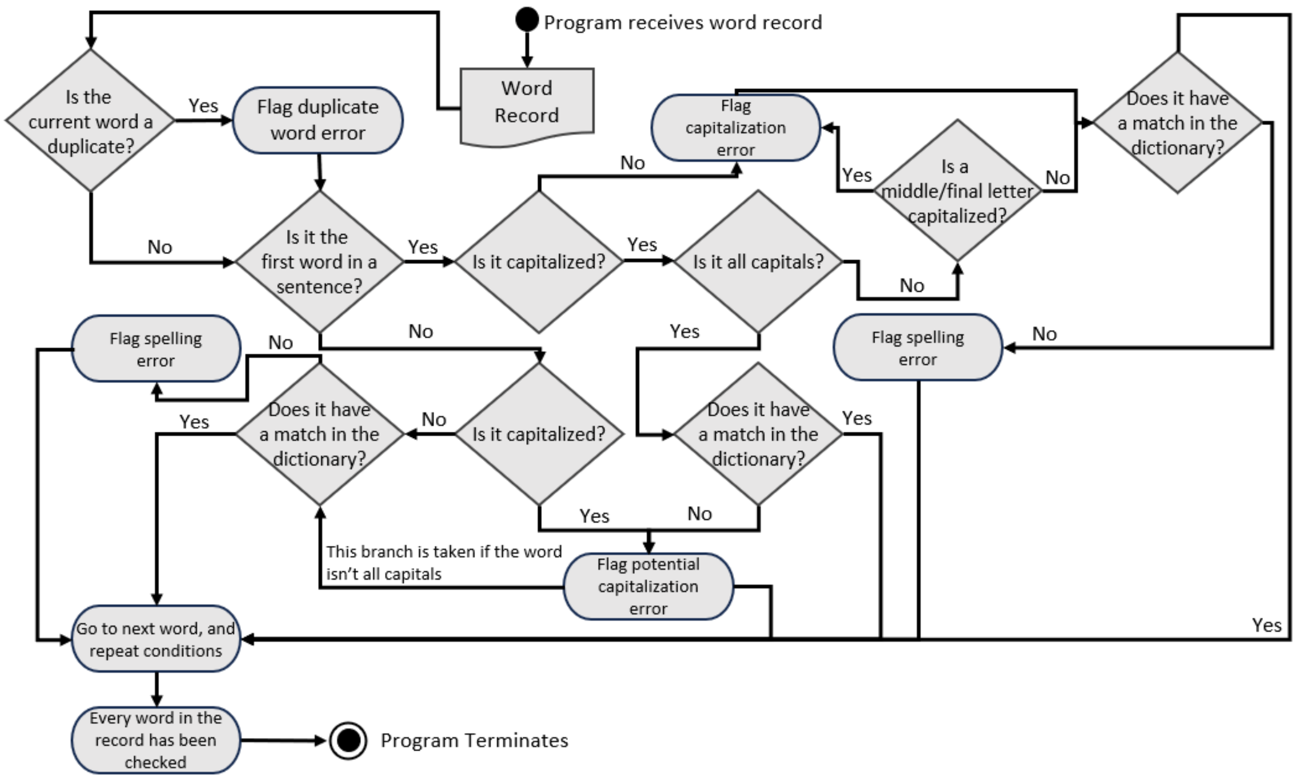
Name:	**Detect Miscapitalizations**
Primary actor:	User
Secondary actor:	None
Goal in context:	Identify words with incorrect capitalization.
Preconditions:	• The file is uploaded to be checked • Detect Double Words
Trigger:	The initiation of the spell-check function
Scenario:	1. The user inputs or loads a text document. 2. Tags <> are ignored if that option was chosen 3. Words are run against a set of rules of capitalization a. Is the word the first in the sentence if yes and the first letter is not capitalized it is flagged with a capitalization error b. If anything other than the first letter is capitalized and does not match a word in the dictionary it is flagged with a capitalization error 4. Words that violate the rules are highlighted and suggestions are presented on the screen.
Alternatives:	1. None
Exceptions:	• Code that is not enclosed within the tag might be mistreated by the software. Examples of such errors will appear in camelCase style and PascalCase style.

Name:	**Detect Misspellings**
Primary actor:	User
Secondary actor:	External Dictionary
Goal in context:	Identify words in the document that are not present in the system or user dictionary.

Preconditions:	• Both dictionaries are loaded into the application • The file is uploaded to be checked • Capitalization detection is done. • Detect Double Words
Trigger:	The initiation of the spell-check function
Scenario:	1. The user inputs or loads a text document. 2. Tags <> are ignored if that option was chosen 3. The file is assumed to all be lowercase 4. The application reviews each word against the External Dictionary and user dictionary. 5. Words not found in dictionaries are highlighted.
Alternatives:	1. None
Exceptions:	• Proper names or acronyms may be highlighted as misspellings if not added by the user into the user dictionary.



Activity Diagram:


Program receives word record
Word Record
Is the current word a duplicate?
Yes
No
Flag duplicate word error
Is it the first word in a sentence?
Yes
No
Is it capitalized?
Yes
No
Flag capitalization error
Is it all capitals?
Yes
No
Is a middle/final letter capitalized?
Yes
No
Does it have a match in the dictionary?
No
Yes
Flag spelling error
Does it have a match in the dictionary?
Yes
No
Is it capitalized?
Yes
No
Flag spelling error
Does it have a match in the dictionary?
Yes
No
Flag potential capitalization error
This branch is taken if the word isn't all capitals
Go to next word, and repeat conditions
Every word in the record has been checked
Program Terminates

3.6 General Correction

Functionality to be delivered:

- Generate a list of suggestions for a misspelled word.
- For miscapitalized words, the program should suggest:
  - The form of the word with all lowercase letters
  - The form of the word with all capital letters
  - Just the first letter capitalized
- For uncapitalized words at the beginning of a sentence, the capitalized form should be suggested.
- All suggestions must come from either the system dictionary or the user dictionary (see 3.3 Dictionaries).
- Correct misspellings by offering the user the ability to:
  - Review and correct an error manually
  - Delete the erroneous word
  - Ignore the error just once
  - Ignore this error for the rest of this document
  - Ignore the error permanently (i.e., add the word to the user dictionary)
  - Choose a replacement from a list of suggestions

Actors:

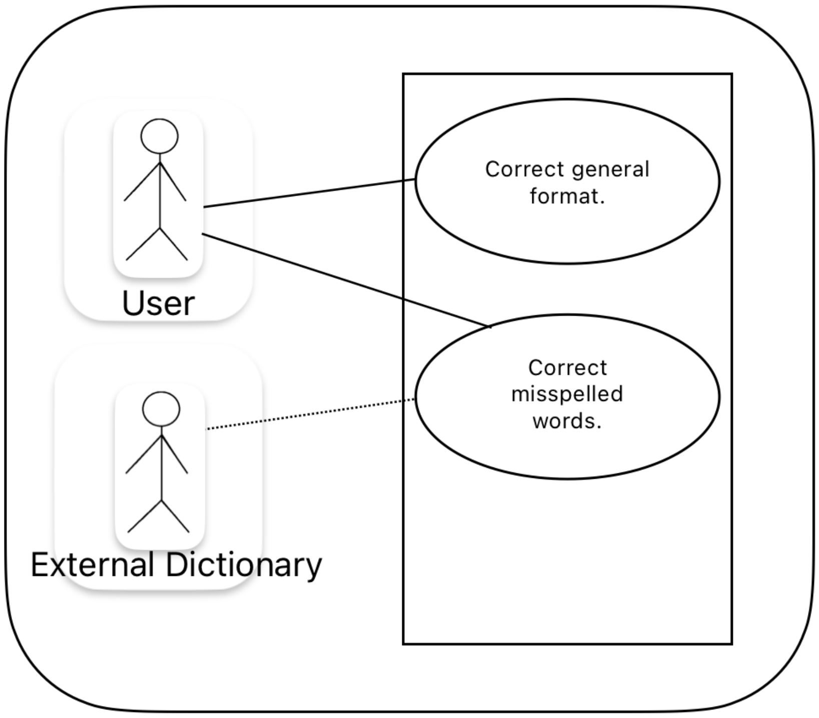
- User
- Administrator

Actor:	**User**
Description:	The user provides the text input that gets analyzed. After analysis is complete, the user is prompted to fix the text according to the program's recommendation, review certain parts, ignore the recommendation, or append the system dictionary with new word.
Aliases:	None
Inherits:	None
Actor Type:	Person - Active
Relationships:	None

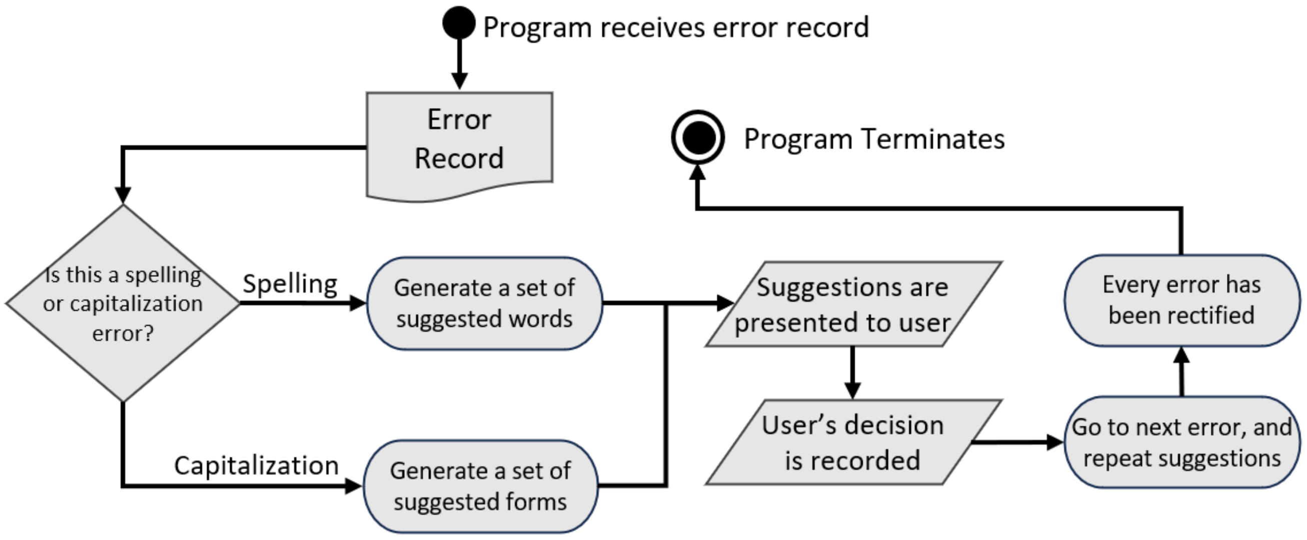
Actor:	**External Dictionary**
Description:	Provides the reference for the spell check operation to check the spelling of words
Aliases:	Dictionary
Inherits:	None
Actor Type:	External System - passive
Relationships	Assists the user in the spell-check process

Use Cases:

Name:	**Error Correction Process**
Primary actor:	User
Secondary actor:	External Dictionary
Goal in context:	Correct detected errors in the document
Preconditions:	• Errors are already detected in the documents
Trigger:	The user opts to correct an identified error
Scenario:	1. The application presents detected errors to the user. 2. The user selects an error to correct. 3. The application provides correction options and possible suggestions. 4. The user selects the desired correction method and proceeds. 5. Error is corrected
Alternatives:	1. The application presents detected errors to the user. 2. The user selects an error to correct. 3. The application provides correction options and possible suggestions. 4. The user adds the word to user's dictionary 5. Error is ignored
Exceptions:	1. If no valid suggestions can be derived from the external dictionary for a detected error, only manual correction is presented.



Activity Diagram:


Program receives error record
Error Record
Program Terminates
Is this a spelling or capitalization error?
Spelling
Generate a set of suggested words
Suggestions are presented to user
Every error has been rectified
User's decision is recorded
Go to next error, and repeat suggestions
Capitalization
Generate a set of suggested forms

3.7 Metrics

Functionality to be delivered:

- Calculate and report characters, lines, and words in the document.
- Identify and report misspellings, misscapitalizations, and double words.
- Track and report manual corrections, accepted suggestions, and word deletions.

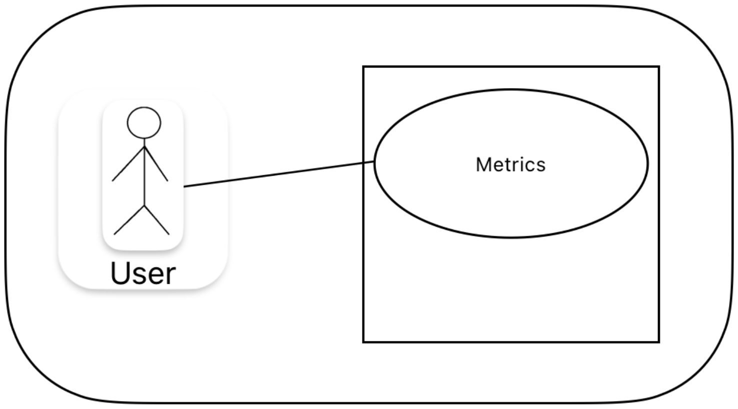
Actors:

- User

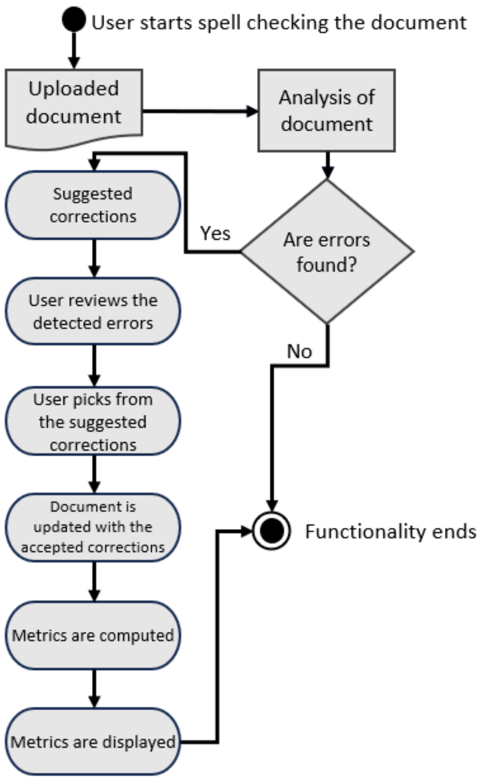
Actor:	**User**
Description:	The primary actor that interacts with the software to spell-check files.
Aliases:	End User, Client, Customer
Inherits:	None
Actor Type:	Person - Active
Relationships	None

Use Cases:

Name:	**Metrics**
Primary actor:	Application/Program
Secondary actor:	None
Goal in context:	To provide users with detailed information on the application's performance during the spell-checking process and help users identify potential areas for improvement and optimize their use of the spell-checker.
Preconditions:	• The user has uploaded a document for spell-checking. • The spell-checking process is completed.
Trigger:	Metrics are requested by the user after spell-checking is completed.
Scenario:	1. the application calculates the number of characters, lines, and words present in the document. 2. The application computes the number of each type of error detected in the document, such as misspellings, miscapitalizations, and double words. 3. After the correction process, the application calculates the number of each type of correction applied to the document (manual corrections, accepted suggestions, word deletions, etc.). 4. The application presents a summary report to the user detailing: a. Total characters, lines, and words. b. Number of each type of error detected. c. The number of each type of correction applied. 5. The user reviews the generated metrics.
Alternatives:	1. the application calculates the number of characters, lines, and words present in the document. 2. The application computes the number of each type of error detected in the document, such as misspellings, miscapitalizations, and double words. 3. After the correction process, the application calculates the number of each type of correction applied to the document (manual corrections, accepted suggestions, word deletions, etc.). 4. The application presents a summary report to the user detailing: a. Total characters, lines, and words. b. Number of each type of error detected. c. The number of each type of correction applied. 5. The user reviews the generated metrics. 6. User exports metrics report for future reference
Exceptions:	• If the same error was corrected and repeated again by the user that might skew the data towards unreal information.



Activity diagram :


User starts spell checking the document
Uploaded document
Analysis of document
Are errors found?
Yes
No
Suggested corrections
User reviews the detected errors
User picks from the suggested corrections
Document is updated with the accepted corrections
Metrics are computed
Metrics are displayed
Functionality ends

3.8 Housekeeping

Functionality to be delivered:

- Users can easily exit the application without data loss.
- Accessible help resources within the app.
- History & Logs
- Optimization Tool

Actors:

- User

Actor:	**User**
Actor Type:	Person - Active
Aliases:	End User, Client, Customer
Description:	The primary actor that interacts with the software to spell-check files.
Inherits:	None
Relationships	None

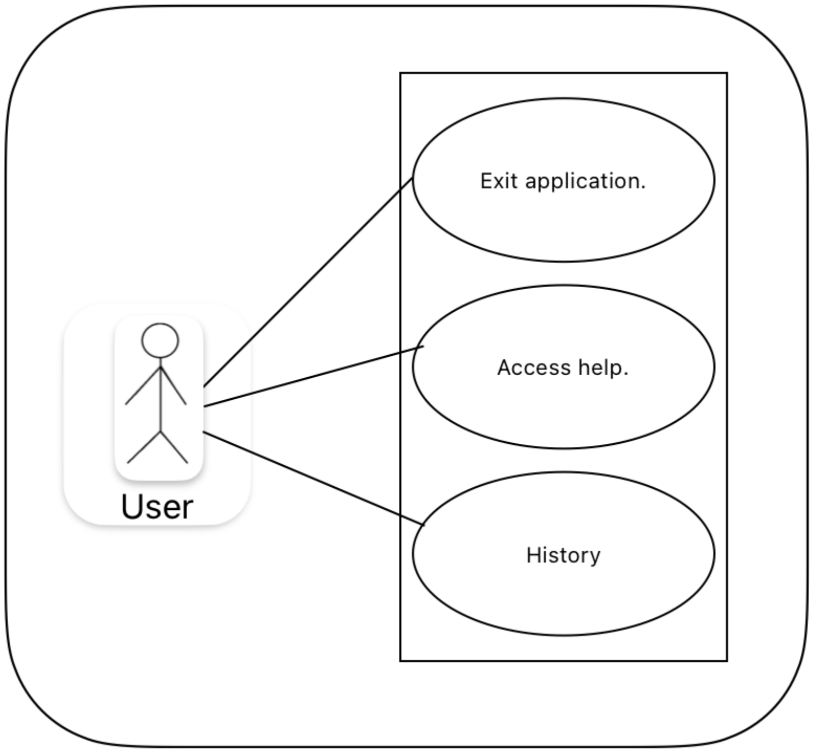
Use Case:

Name:	**Exit Application**
Primary actor:	User
Secondary actor:	None
Goal in context:	Safely close the application without any data loss or corruption
Preconditions:	• The application is up and running.
Trigger:	The user Exits the application
Scenario:	**1.** The user initiates the application exit process. **2.** The application ensures that the user's data is saved if necessary. **3.** The application exits gracefully without causing data loss or unexpected behavior.
Alternatives:	• None
Exceptions:	The application fails to close or hangs. Necessary steps should be taken to ensure data isn't lost.

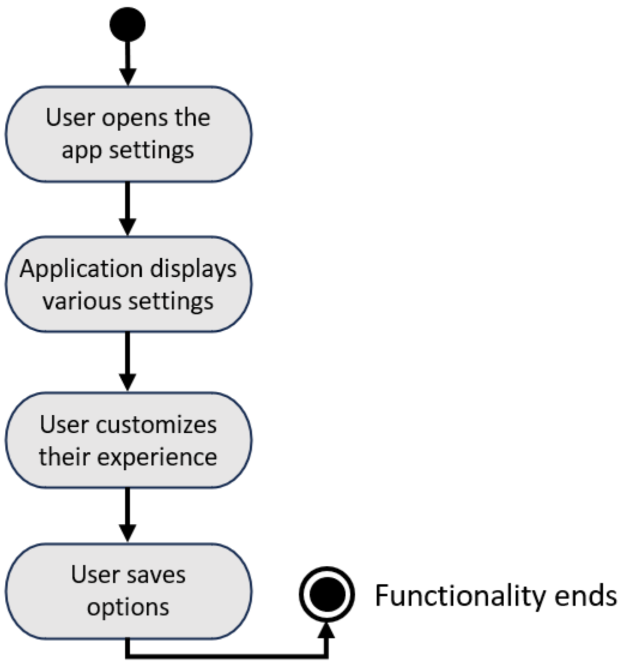
Name:	**Access Help**
Primary actor:	User
Secondary actor:	None
Goal in context:	Obtain assistance or instructions on how to use the application or its features.
Preconditions:	• The application is up and running. • Help is needed
Trigger:	The user clicks on the help icon
Scenario:	**1.** The user navigates to the "Help" section of the application. **2.** User selects the topic or issue they need assistance with. **3.** The application provides detailed steps, FAQs, or guidance.
Alternatives:	• None
Exceptions:	Help content isn't available or fails to load.

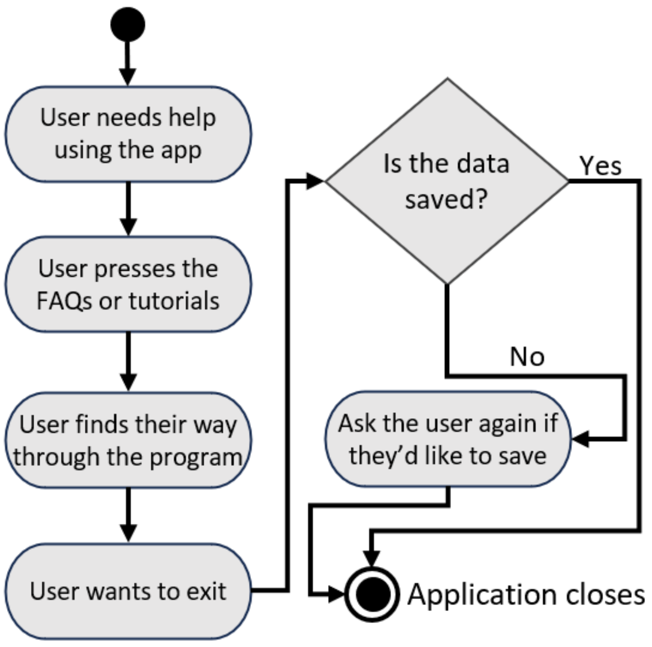
Name:	**History**
Primary actor:	User
Secondary actor:	None
Goal in context:	Adjust preferences, configurations, and general settings of the application. Including the History log and the accessibility Features

Preconditions:	• The application is up and running.
Trigger:	The user Exits the application
Scenario:	1. The user navigates to the settings section of the application. 2. The user adjusts and previews settings including a history tab and an option for a text-voice accessibility tool. 3. The user saves the changes.
Alternatives:	• None
Exceptions:	The application fails to save settings or an error occurs while updating.



Activity diagram:


User opens the app settings
Application displays various settings
User customizes their experience
User saves options
Functionality ends


User needs help using the app
User presses the FAQs or tutorials
User finds their way through the program
User wants to exit
Is the data saved?
Yes
No
Ask the user again if they'd like to save
Application closes

3.9 Persistence

Functionality to be delivered:

- Stores the user dictionary built during a spell-check.
- Stores application settings that need to be maintained between sessions.
- Stores the configuration file internally for easy accessibility during future launches of the application.
- Data is stored persistently and is available between user sessions.

Actors:

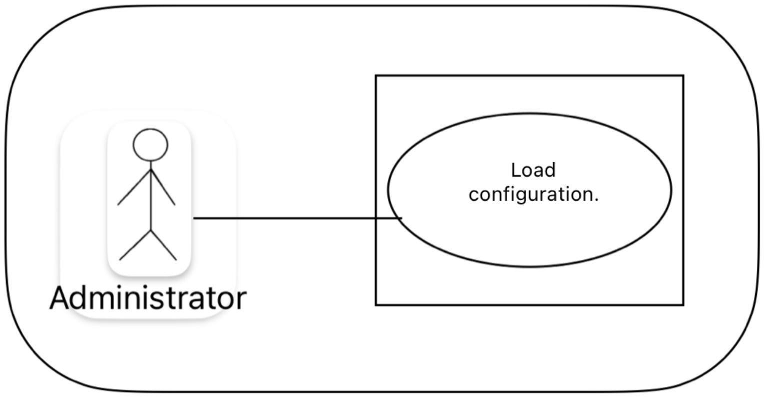
- Administrator

Actor:	**Administrator**
Description:	The administrator is a special account that allows the developer to edit and view the configuration files.
Aliases:	Developer
Inherits:	User
Actor Type:	Person - Active
Relationships	None

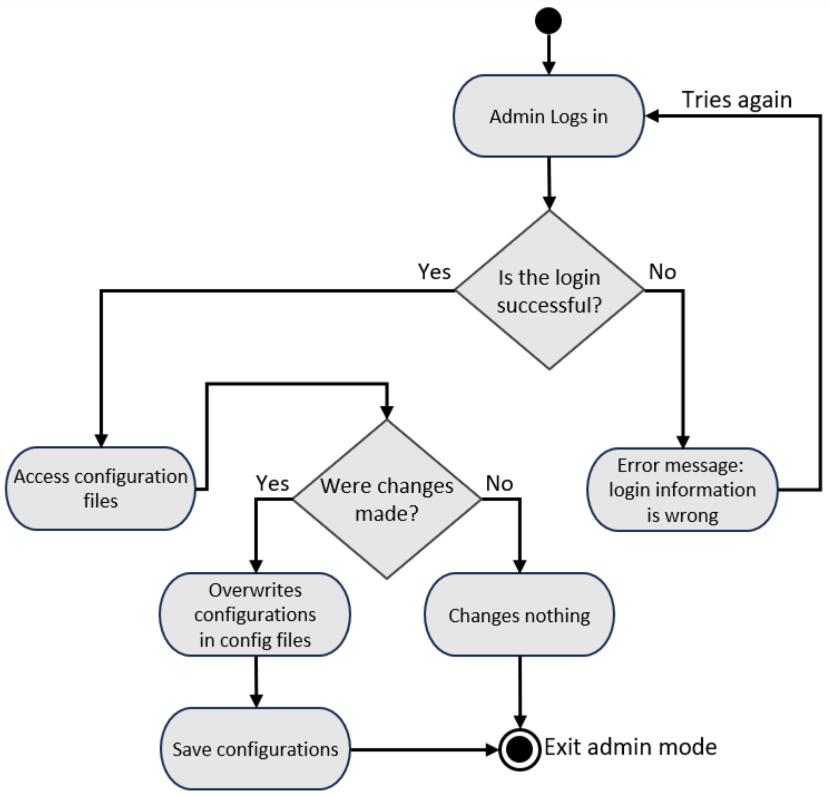
Use Cases:

Name:	**Load Configuration**
Primary actor:	Administrator
Secondary actor:	None
Goal in context:	Set up the application by retrieving the system configuration, user dictionary, and user-specific settings from persistent storage during initiation.
Preconditions:	The application is up and running.
Trigger:	Application is started or restarted.

Scenario:	• Application initiates. • The administrator accesses designated storage locations for default configurations and loads them. • The administrator retrieves user-specific dictionaries and settings from persistent storage. • The application environment, including user-specific settings, is configured based on the retrieved data. • The application is ready for user interaction with spell-check and other features.
Alternatives:	• Application initiates. • The administrator accesses designated storage locations for default configurations and loads them. • The administrator retrieves user-specific dictionaries and settings from persistent storage. • The application environment, including user-specific settings, is configured based on the retrieved data. • If no user-specific data is found, the system uses default settings. • The application is ready for user interaction with spell-check and other features.
Exceptions:	If corrupted system or user data is detected, the system reverts to hardcoded settings and notifies the user about potential discrepancies.



Activity Diagram:


Admin Logs in
Tries again
Is the login successful?
Yes
No
Access configuration files
Error message: login information is wrong
Were changes made?
Yes
No
Overwrites configurations in config files
Changes nothing
Save configurations
Exit admin mode

4. Non-Functional Requirments

4.1 Requirements

1. The application calls for the development of a standalone desktop application using Java as the primary programming language.
2. The application is required to make use of a Java GUI, with Swing or JavaFX being the preferred options. The choice of framework is up to the group, but it must be uniform across the group. Adequate research is essential to gain proficiency in these toolkits.
3. The performance of the application should be optimized to ensure that the dictionary setup for checking and the search for words within the dictionary during a check do not result in excessive load or run time.
4. The application should not require the utilization of any additional libraries. In the event that another library is desired, approval must be obtained from the instructor in written form, specifically via email, prior to implementation.
5. All code in the application must be commented using Javadoc.
6. The group has the freedom to decide on the coding conventions and styles they want to use for naming objects, indenting, and other related aspects. However, it's essential to maintain consistency in applying those conventions and styles throughout all the files in the application.
7. The application must be compatible with systems that have a standard Java installation. Additionally, it is necessary for each member of the group to be able to compile and run the application from their own development environment, which can be of their choice.
8. The application should ensure that it doesn't make any changes outside of its installation directory and its subdirectories. This means that any user dictionary or configuration file must be stored in a suitable location, such as the user's home directory. It's important for the application to be self-contained and not interfere with other files on the system.
9. It is imperative that the application provides a clear and conspicuous response to every user-initiated action. Any action that is erroneous or cannot be completed successfully must be accompanied by a professional and informative error message.
10. The development of the application should adhere with software engineering principles.
11. Regularly committing and pushing project code to the assigned repository is mandatory for team members. It is crucial to ensure that the repository designated for your team is updated with the latest code changes.
12. The group must complete the documentation using Confluence collaboration system
13. All tasks and issues related to the project must be tracked on Jira and updated throughout the project.

5. Summary

5.1 Brief Description

Our team is developing a stand-alone desktop application that provides spell-checking functionalities for text documents. The application will check individual words within a given document against a dictionary of correct words. If an error is detected, the application will present a range of correction options, including manual and automatic suggestions. Additionally, the application will be able to identify and correct instances of improper capitalization and repetition errors. Our software will also support the use of a user dictionary, allowing users to select their preferred dictionary for use within the application. During operation, the application will provide metrics such as word count, misspellings, and repeated words in the document and report these to the user upon completion of the spell-checking process. To manage user data, we will implement a multi-user account system. This system will store user data, including an administrator account that provides access to configuration files and the ability to overwrite existing files. Finally, our application will allow users to exit at any time without losing data and offer accessible help resources within the application if they require assistance.

5.2 Table of Terms

The technical document includes a table of terms, notations, and acronyms that serves as a reference tool for all stakeholders. The definitions provided in the table are clear and concise, and are intended for those without a technical background. The purpose of the table is to facilitate understanding of technical terms and abbreviations used throughout the document.

Term	**Definition**
Use Case	A set of actions that outline the user's interaction with the system in order to accomplish a predetermined objective
Non-Functional Requirements	Specifications outlining the expected performance of a system, including parameters such as response time and reliability.
Functional Requirements	Specifications that outline the functionalities and capabilities a system must possess.