Spell Checker

Team Megan

Connor Flok, Megan Mulholland, Ayo, and Jacob Hauff

1. Project Overview

Objective:
Develop a spell checker application that identifies and suggests corrections for misspelled words in a text input. The system should handle common spelling errors, allow users to review and accept/reject suggestions, and support multiple input formats.

Stakeholders:

- **Primary Users:** Students, writers, editors, and general users who need spelling verification.
- **Secondary Stakeholders:** Developers, QA team, project sponsors, and product managers.

Scope:
The application will focus on English-language spell-checking in its first release. Future expansions may include grammar suggestions and support for additional languages.

2. Functional Requirements

The system must:

1. Accept text input (typed, pasted, or loaded from a file).
2. Detect misspelled words using a dictionary or algorithmic approach.
3. Suggest one or more possible corrections for each detected misspelling.
4. Allow the user to accept, reject, or manually edit the correction.
5. Provide a summary report of corrections made.
6. Store and allow user-defined custom dictionary words.

3. Non-Functional Requirements

- **Performance:** Check spelling for a 1,000-word document in < 3 seconds.
- **Usability:** Intuitive UI with clear highlighting of errors and suggestions.

- **Scalability:** Should support files up to 10,000 words.
- **Portability:** Must run on Windows, macOS, and Linux (if desktop-based) or any modern browser (if web-based).
- **Reliability:** Accuracy > 95% for common spelling mistakes.
- **Security:** User data (uploaded files) must not be stored beyond the session.

4. Use Cases

Use Case 1: Check Spelling of Entered Text

Actors: User
Trigger: User clicks “Check Spelling”
Main Flow:

1. User enters text or uploads file.
2. System processes text and finds misspelled words.
3. System highlights errors and displays suggestions.
4. User chooses corrections or ignores them.
5. System outputs final corrected text.

Alternate Flow:

- If no errors are found, system displays “No spelling errors detected.”

6. System Architecture (High-Level)

- **Input Module:** Handles user text/file input.
- **Processing Engine:** Tokenizes text, compares against dictionary, detects errors.
- **Suggestion Engine:** Ranks possible corrections (Levenshtein distance, frequency-based suggestions).
- **UI Layer:** Displays highlighted words, suggestions, and corrected output.
- **Custom Dictionary Manager:** Stores user-specific words.

7. Assumptions and Constraints

Assumptions:

- Users have access to an internet connection (if using online dictionary updates).
- Input text is primarily in English and uses UTF-8 encoding.

Constraints:

- Must use only open-source dictionary data.
- Project must be completed within the semester timeline.

8. Risk Analysis

Risk	Likelihood	Impact	Mitigation
Inaccurate suggestions	Medium	High	Use multiple algorithms and test extensively
Large files cause slow performance	High	Medium	Optimize algorithm and use efficient data structures
Users reject system if UI is confusing	Medium	High	Conduct usability testing early
Dependency on external dictionary APIs	Low	Medium	Include fallback local dictionary