Software Requirements Specification

For

Hindi Auto-Complete Word Generator

12th November, 2024

Prepared by

Specialization	SAP ID	Name
AI and ML (NH)	500107230	Anshika Sharma
AI and ML (NH)	500107361	Deepali Rana
AI and ML (NH)	500107784	Divi Saxena
AI and ML (NH)	500109666	Kanak Yadav

School Of Computer Science
UNIVERSITY OF PETROLEUM & ENERGY STUDIES,
DEHRADUN- 248007. Uttarakhand

Table of Contents

1. INTRODUCTION:

1.1 Purpose of the Project:

To develop a Hindi Auto Word Complete Generator where users can input a prefix and receive a list of possible word completions in Hindi.

1.2 Target Beneficiary:

The project becomes highly relevant for anyone looking to improve typing efficiency in their local language by incorporating support for various Indic languages. This enhancement would be particularly beneficial for users who work with complex or frequently used words in Indic languages like and others. This approach can make typing faster, reduce errors, and streamline the experience of using Indic languages in digital communication.

1.3 Project Scope:

The scope of this project is to develop an efficient Hindi word completion system using Trie-based algorithms, with potential expansion to other Indic languages. By integrating this system, users can experience faster typing, particularly with frequently used or complex words, enhancing productivity in local languages. This project is valuable for native speakers, content creators, and learners, supporting the broader goal of improving digital accessibility in Indic languages

1.4 References:

Sno.	Link	Resource
1.	https://docs.oracle.com/javase/tutorial/uiswing/	Swing Documentation
2.	https://www.geeksforgeeks.org/trie-insert-and-search/	Trie Data Structure Basics
3.	https://aclanthology.org/2024.lrec-main.568.pdf	Assamese Sentiment Analysis
4.	https://www.utf8-chartable.de/unicode-utf8-table.pl?start=2304&number=128	Unicode Standard for Hindi Documentation
5.	https://github.com/0xproflupin/DeepBhashan/blob/master/dataset/Hindi%20dictionary.txt	Hindi Dictionary

2. PROJECT DESCRIPTION:

2.1 Reference algorithm:

The reference algorithm used for this project is the Trie (prefix tree) data structure. By traversing through the Trie based on the characters of a given prefix, the algorithm can quickly retrieve all possible word completions that start with that prefix.

2.2 Characteristic of Data:

The dataset used in this project is a set of words that are going to be incorporated into the Trie data such that all the word completions can be retrieved quickly.

2.3 SWOT Analysis:

Strengths: Trie data structure is highly efficient for prefix-based search operations, allowing quick retrieval of word suggestions. The project specifically targets Hindi, a language with unique challenges.
Weaknesses: Trie data structures can consume a significant amount of memory when storing a large dictionary of words.
Opportunities: With the growing number of Hindi speakers using digital platforms, there is a high demand for tools that support efficient text input in the native script. This project has the potential to fill this gap.
Threats: Large tech companies like Google and Microsoft already offer Hindi input tools with auto-complete functionality. These tools may overshadow the project if users perceive them to be more reliable or feature-rich.

2.4 Project Features:

It has dynamic user interaction that gives word recommendations based on the given prefix. Users can also add new words to the trie (if the word is not present).

2.5 User Classes:

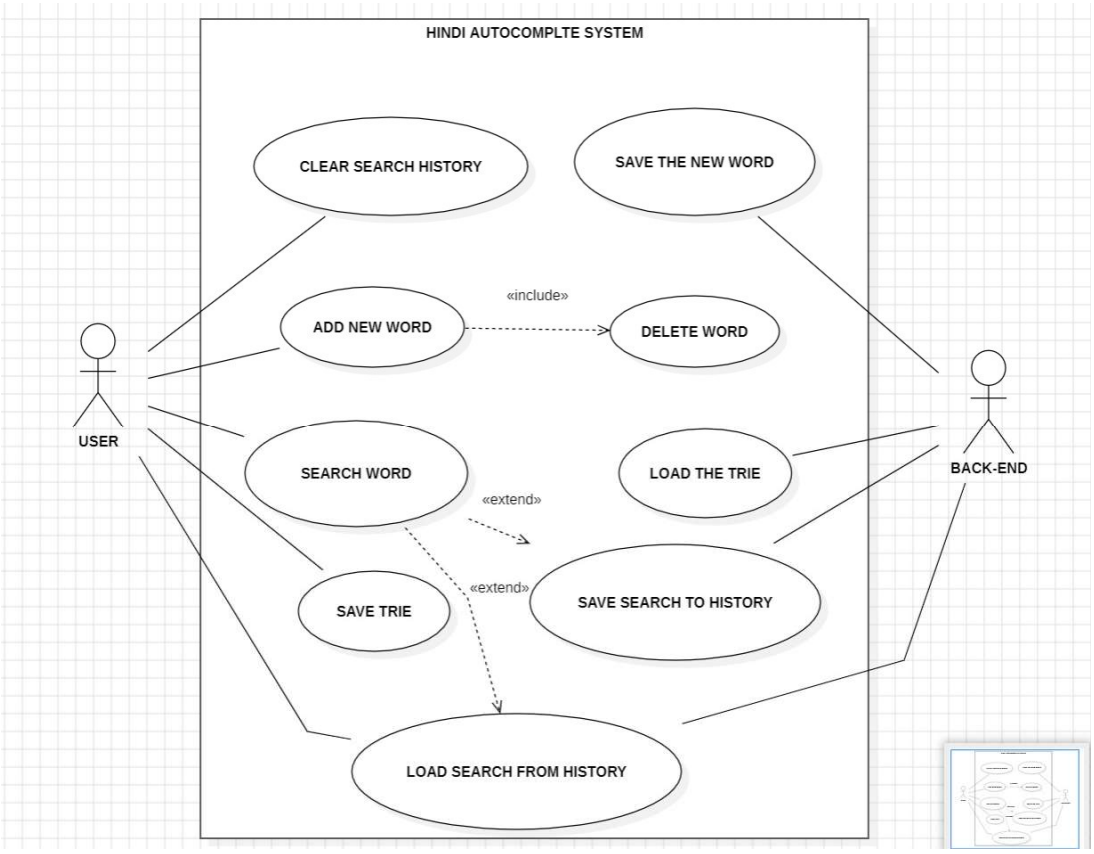
The user classes for this project include Native Language Typists looking to improve typing speed, Students and Language Learners who want assistance with vocabulary and spelling, Content Creators and Writers who need efficient text entry in Hindi, and Software Developers and Researchers who can use the application for NLP projects in Indic languages.

2.6 Design and Implementation Constraints:

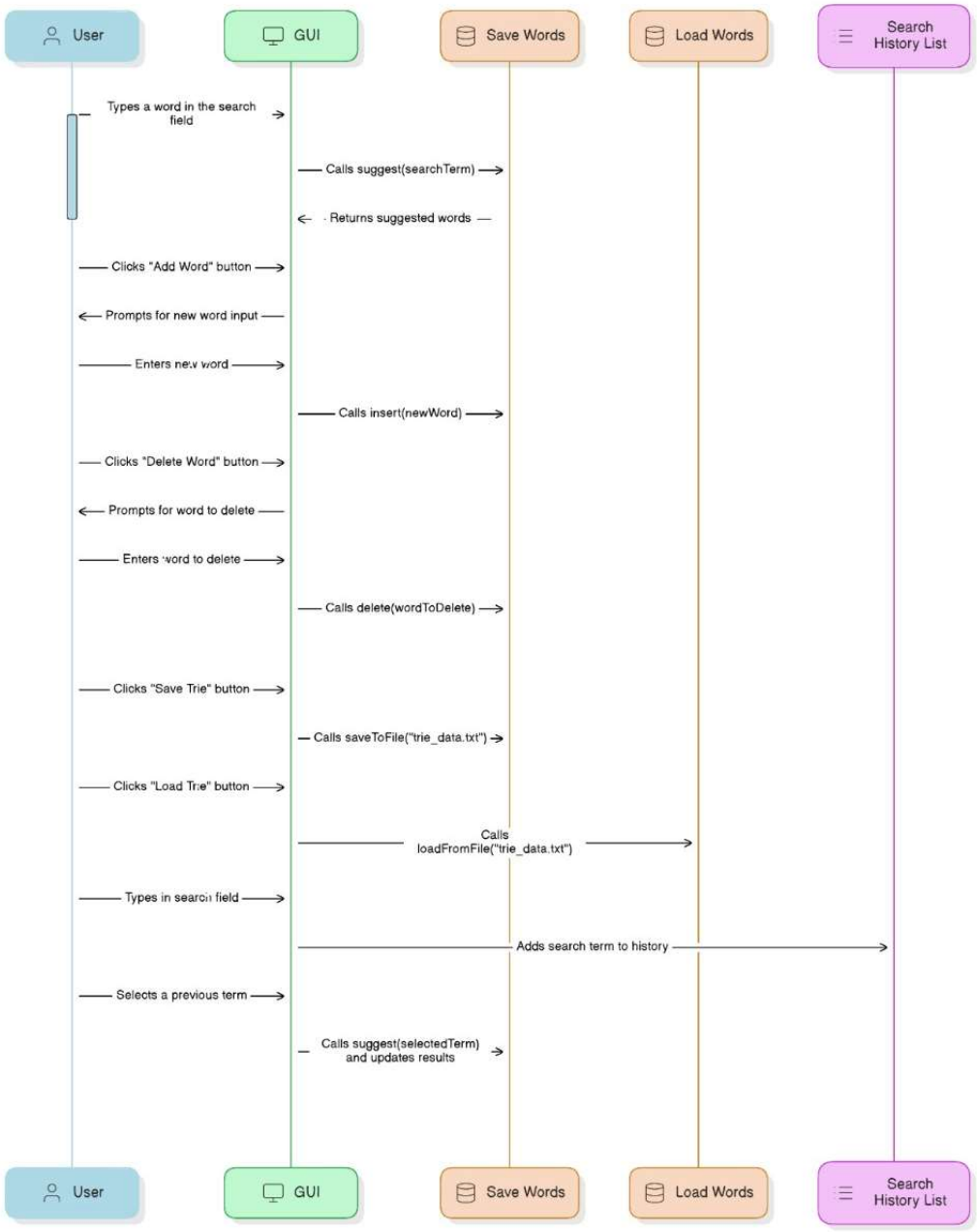
This program can only currently be run on the IntelliJ text Editor along with the requirement of encoding multiple languages and makes use of JAVA programming language hence a lot of setup needs to be done prior running the program.

2.7 Design Diagrams

USE CASE DIAGRAM



SEQUENCE DIAGRAM



2.8 Assumed Factors and Dependencies:

Assumptions: Users are familiar with typing in Hindi or Indic languages, and the initial word dataset is sufficient for common suggestions. A database, if used, is set up and accessible, and a Java runtime environment is available.

Dependencies: The project depends on the Java Runtime Environment (JRE), Swing for the GUI, and JDBC if database integration is included for word storage.

3.SYSTEM REQUIREMENTS:

3.1 User Interface:

The swing package in JAVA is imported which contains all the methods that are used to create a GUI for the User allowing keyboard and mouse interactions with the user.

3.2 Software Interface:

The entire project is implemented in JAVA by using its packages and inbuilt methods to provide the functionality desired for the program in the backend as well as GUI in the frontend.

3.3 Database Interface:

Currently, the project doesn't use a database, but in the future, a database interface could be added to enable users to maintain a personalized search history and save custom words. This feature would allow user preferences and frequently searched words to persist across sessions, enhancing personalization and usability.

3.4 Protocols:

No protocols followed during program implementation and no security requirement needed.

4.NON-FUNCTIONAL REQUIREMENTS:

4.1 Performance Requirements:

Response Time: The autocomplete suggestions should appear within 200 ms of typing, ensuring a smooth user experience.
Load Handling: The system should handle large datasets (10,000+ words) without significant lag.
Database Operations: If a database is used, retrieval and saving operations should not exceed 1 second for efficient data access.

4.2 Security Requirements:

No security requirements due to Abstraction by JAVA ensuring secure data and objects.

4.3 Software Quality Attributes:

Usability: The GUI should be simple, intuitive, and user-friendly for individuals familiar with Hindi typing.
Maintainability: Code should be modular, allowing easy updates or additions (e.g., new languages or features).
Portability: The software should run on different operating systems (Windows, Linux, etc.) with minimal configuration.
Scalability: The system should support expansion to additional Indic languages or larger datasets if needed.

5. OTHER REQUIREMENTS:

The system should be Scalable, accommodating growth in word data and potential expansion to more Indic languages or new features.

APPENDIX A: GLOSSARY

Trie: A tree-like data structure used for efficient retrieval of words based on prefixes, ideal for autocomplete functions.

Autocomplete: A feature that predicts and suggests words based on user input prefixes.

Indic Languages: Languages originating from the Indian subcontinent, such as Hindi, Bengali, Tamil, etc.

GUI (Graphical User Interface): Visual interface allowing users to interact with the application.

APPENDIX B: ANALYSIS MODEL

This model includes diagrams and descriptions of the project's core components and interactions:

User Interactions: Shows how users input prefixes and receive autocomplete suggestions.

Trie Structure: Diagram of how the Trie is structured and used for word insertion, deletion, and search.

Database Interface (if added): Illustrates how the application interfaces with a database for word storage and retrieval.

APPENDIX C: ISSUES LIST

Performance Optimization: Ensure the Trie can handle large datasets efficiently.

Database Integration: Plan for future database support for personalized search history.

Error Handling: Improve robustness to handle invalid inputs and unexpected errors gracefully.

Unicode Compatibility: Ensure full support for all Indic scripts, especially in cross-platform use cases.