Architecture Document:

CorrSpell - Spelling Corrector

Written By	Amit Ranjan
Document Version	0.3
Last Revised Date	29– September -2023

Document Control

Change Record:

Version	Date	Author	Comments
0.1	13-August -2023	Amit Ranjan	Introduction & Architecture defined
0.2	15-September -2023	Amit Ranjan	Content & Architecture Description appended and updated
0.3	29-september-2023	Amit Ranjan	Content & Architecture Description appended and updated

Contents

1.Introduction........1
1.1 Purpose........1
1.2 Scope........1

2.System Overview........1
2.1 High-Level Architecture........1
2.2 Key Components........1

3.Detailed Architecture........2
3.1 Start........2
3.2 Input Data........2
3.3 Tokenization........3
3.4 Bigram Dictionary........3
3.5 Lookup........4
3.6 Start Web App........4
3.7 Enter Text........4
3.8 Predict........5
3.9 Send to Web Server........5
3.10 Render Result........6
3.11 End........6

4.omponent Interactions........6
4.1 User Request Flow........6
4.2 Spell Correction Process........7

5.Key Design Decisions........7
5.1 Use of SymSpell........7
5.2 User Interface Choice........7

6.Data Flow Diagram........8
6.1 Data Flow Overview........8

6.2 Data Flow Diagram……………………………………………………………………………………………………..8

7.Deployment Architecture……………………………………………………………………………………………..8
7.1 Setting Up the Render Environment………………………………………………………………………….8

7.2 Render Platform Deployment………………………………………………………………………………….9

8.Non-Functional Requirements…………………………………………………………………………………….10
8.1 Performance…….10
8.2 Scalability………..10
8.3 Reliability……10
8.4 Security………10

9.Conclusion………..11

1. Introduction

1.1 Purpose

The purpose of this document is to provide an in-depth architectural overview of the "corrspell" spelling correction system. It describes the system's components, their interactions, and key design decisions.

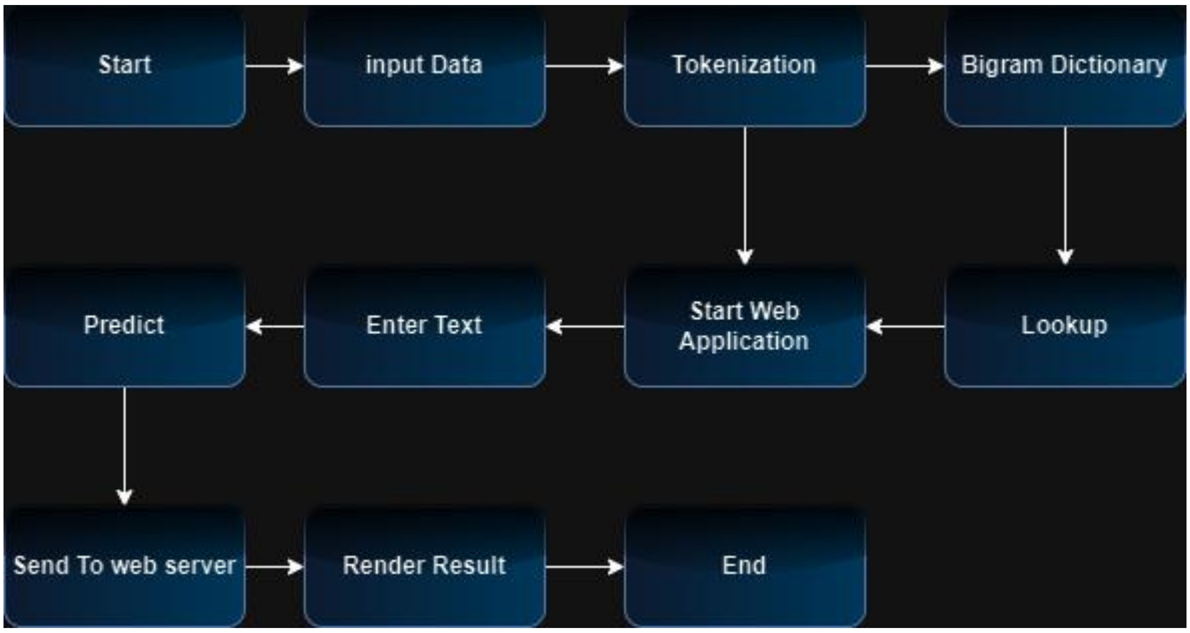
1.2. Scope

This document covers the architecture of the "corrspell" system, focusing on its high-level structure, detailed component descriptions, data flow, and deployment.

2.System Overview

2.1 High-Level Architecture

The "corrspell" system follows a client-server architecture where the client is a web-based user interface and the server hosts the spell correction logic.



2.2 Key Components

- Start: The system initialization point.
- Input Data: The user-provided input text.
- Tokenization: The process of breaking input text into words.
- Bigram Dictionary: A dictionary used for bigram-based spell correction.
- Lookup: The component responsible for looking up word suggestions.
- Start Web App: Initialization of the web-based user interface.
- Enter Text: User input of text to be corrected.
- Predict: The spell correction prediction process.
- Send to Web Server: Sending corrected text to the web server.
- Render Result: Displaying corrected text to the user.
- End: The termination point of the system.

3. Detailed Architecture

3.1 Start

Description:

The "Start" component serves as the initialization point for the "corrspell" system. It's responsible for configuring and initializing various components and resources required for the spell correction process.

Functionality:

Initializes the spell correction engine (e.g., SymSpell).

Loads language dictionaries and models.

Sets up the web server to serve the user interface.

Importance:

This component is crucial for the system's startup and ensures that all necessary resources are available for spell correction.

3.2 Input Data

Description:

The "Input Data" component represents the user-provided text that needs to be checked for spelling correctness and corrected if necessary. This text serves as the raw input for the spell correction process.

<u>Functionality:</u>

Accepts user input through the web-based user interface.

Passes the user's text to the tokenization component for further processing.

<u>Importance:</u>

This component is the starting point for spell correction and provides the source of data to be corrected.

3.3 Tokenization

Description: The "Tokenization" component is responsible for breaking down the input text into individual words or tokens. It prepares the text for dictionary lookups and correction.

<u>Functionality:</u>

Splits the input text into words, considering spaces and punctuation.

Generates a list of tokens for further processing.

<u>Importance:</u>

Tokenization is a crucial step as it prepares the text for spell checking by making individual words accessible for lookup.

3.4 Bigram Dictionary

<u>Description:</u>

The "Bigram Dictionary" is a specialized dictionary used for bigram-based spell correction. It stores word pairs and their frequencies, enabling context-aware corrections.

<u>Functionality:</u>

Provides context-based word suggestions by considering adjacent word pairs.

Enhances the accuracy of corrections by analyzing word combinations.

Importance:

The use of a bigram dictionary improves the system's ability to correct words based on their context within the input text.

3.5 Lookup

Description:

The "Lookup" component is responsible for searching dictionaries and databases to find suggestions for correcting misspelled words.

Functionality:

Utilizes dictionaries and language models to find word suggestions.

Determines edit distances between input words and dictionary entries.

Importance:

The lookup process is central to spell correction, as it identifies potential corrections and their relevance.

3.6 Start Web App

Description:

The "Start Web App" component initiates the web-based user interface (UI) that interacts with users. It configures the server to serve the UI.

Functionality:

Initializes the Flask web application.

Configures routes and endpoints for user interactions.

Importance:

This component allows users to input text for correction and receive corrected results through the web interface.

3.7 Enter Text

Description: The "Enter Text" component represents the user interface's functionality where users can input the text they want to correct.

Functionality:

Presents a text input field to users.

Accepts user-provided text and passes it to the prediction process.

Importance:

This component facilitates user interaction by enabling them to enter text easily for correction.

3.8 Predict

Description:

The "Predict" component is responsible for processing the user-provided text, identifying misspelled words, and suggesting corrections.

Functionality:

Analyzes each token in the input text.

Identifies misspelled words using dictionaries and models.

Provides a list of suggested corrections for misspelled words.

Importance:

Prediction is the core function of the system, as it corrects misspelled words and enhances text accuracy.

3.9 Send to Web Server

Description:

The "Send to Web Server" component handles the corrected text generated by the prediction process. It sends the corrected text to the web server for rendering and display.

Functionality:

Transmits the corrected text to the appropriate web server endpoint.

Provides the result for rendering on the user interface.

Importance:

This component ensures that the corrected text is delivered to the user interface for presentation.

3.10 Render Result

Description:

The "Render Result" component is responsible for displaying the corrected text on the user interface. It presents the final result to the user.

Functionality:

Renders the corrected text in a readable format.

Displays the corrected text to the user for review.

Importance:

This component completes the user interaction loop by presenting the corrected text to the user.

3.11 End

Description:

The "End" component marks the termination point of the system's operation. It finalizes any necessary cleanup or shutdown procedures.

Functionality:

Performs system cleanup tasks, if required.

Concludes the spell correction process.

Importance:

This component ensures that the system gracefully concludes its operations.

4. Component Interactions

4.1 User Request Flow

- The user initiates the system.
- User input data is tokenized.
- Bigram dictionary is used for lookup.
- Suggestions are generated.
- Corrected text is rendered for the user.

4.2 Spell Correction Process

- User input is tokenized.
- Bigram dictionary is used for lookup.
- Suggestions are generated.
- Corrected text is rendered for the user.

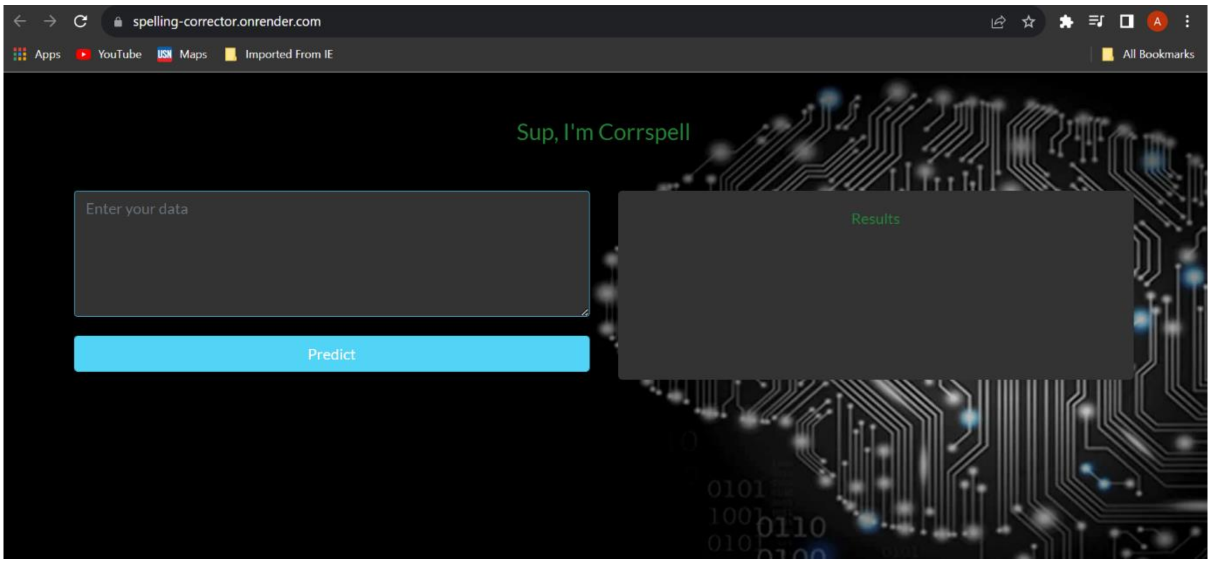
5. Key Design Decisions

5.1 Use of SymSpell

The decision to integrate SymSpell was made due to its efficiency in spelling correction. Its dictionaries and algorithms improve correction accuracy.

5.2 User Interface Choice

A web-based user interface using Flask was chosen for its simplicity and ease of use.

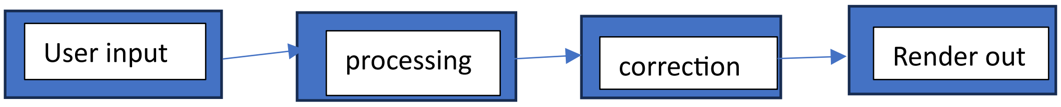


6. Data Flow Diagram

6.1 Data Flow Overview

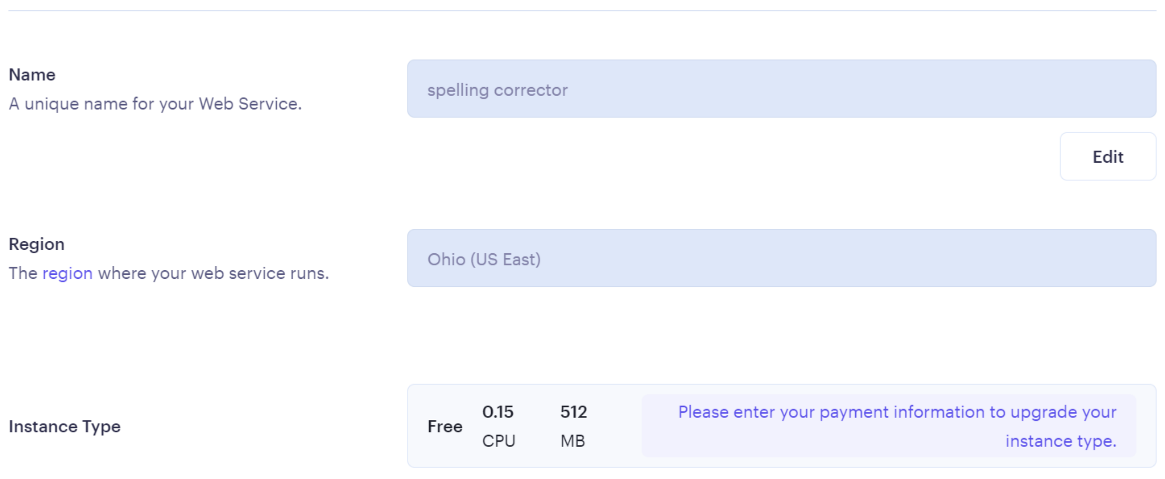
The primary data flow in the system involves user input, processing within the spell correction logic, and corrected output.

6.2 Data Flow Diagram



7. Deployment

7.1 Setting Up the Render Environment



Name
A unique name for your Web Service.

spelling corrector

Edit

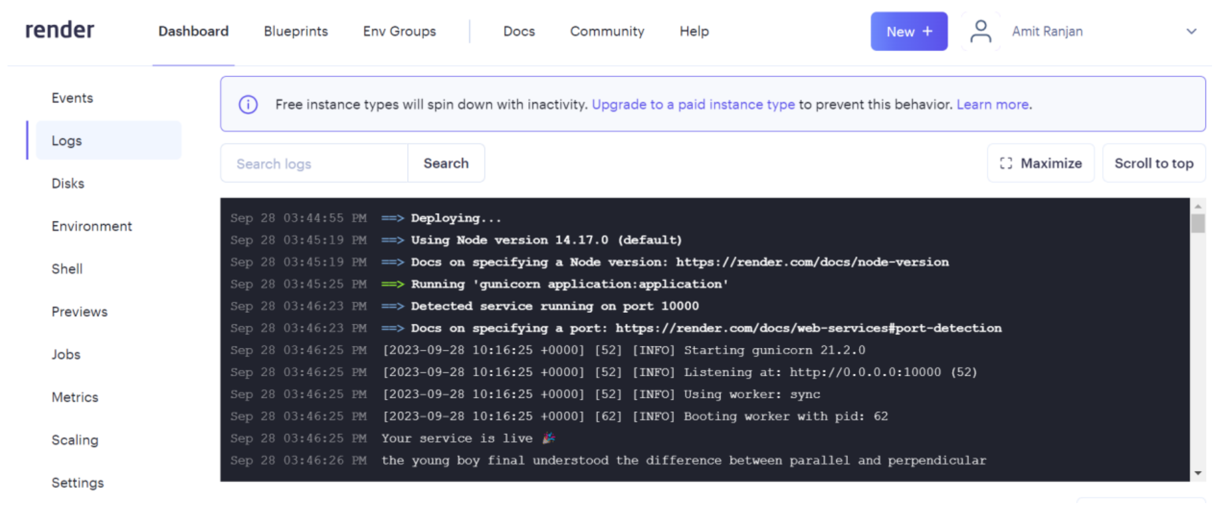
Region
The region where your web service runs.

Ohio (US East)

Instance Type

Free 0.15 CPU 512 MB

Please enter your payment information to upgrade your instance type.

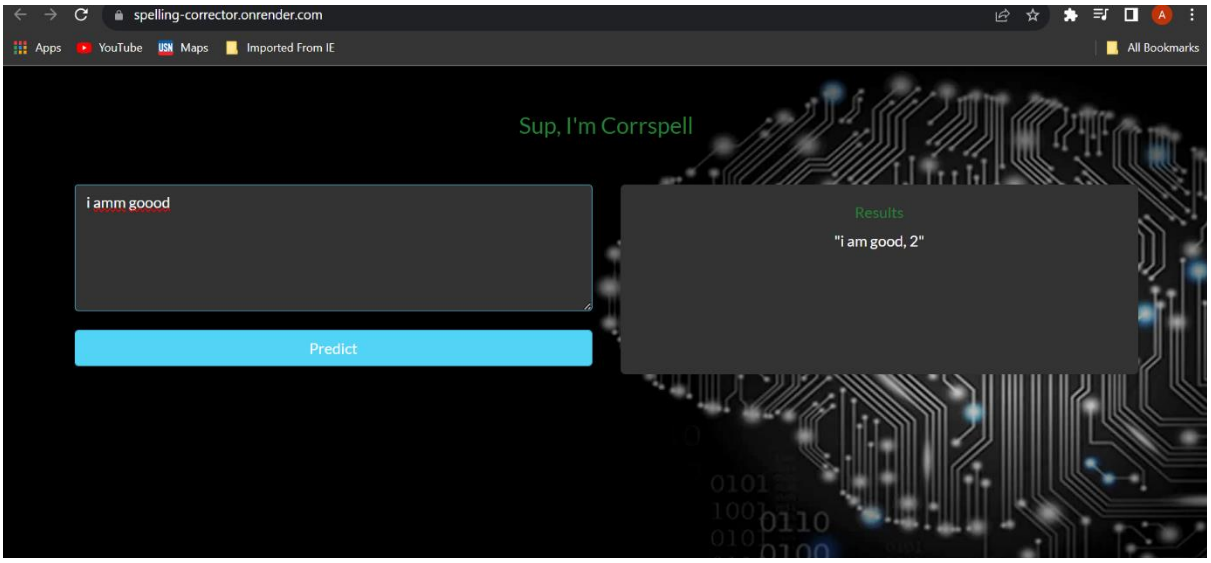


Free instance types will spin down with inactivity. Upgrade to a paid instance type to prevent this behavior. Learn more.

```
Sep 28 03:44:55 PM  ==> Deploying...
Sep 28 03:45:19 PM  ==> Using Node version 14.17.0 (default)
Sep 28 03:45:19 PM  ==> Docs on specifying a Node version: https://render.com/docs/node-version
Sep 28 03:45:25 PM  ==> Running 'gunicorn application:application'
Sep 28 03:46:23 PM  ==> Detected service running on port 10000
Sep 28 03:46:23 PM  ==> Docs on specifying a port: https://render.com/docs/web-services#port-detection
Sep 28 03:46:25 PM  [2023-09-28 10:16:25 +0000] [52] [INFO] Starting gunicorn 21.2.0
Sep 28 03:46:25 PM  [2023-09-28 10:16:25 +0000] [52] [INFO] Listening at: http://0.0.0.0:10000 (52)
Sep 28 03:46:25 PM  [2023-09-28 10:16:25 +0000] [52] [INFO] Using worker: sync
Sep 28 03:46:25 PM  [2023-09-28 10:16:25 +0000] [62] [INFO] Booting worker with pid: 62
Sep 28 03:46:25 PM  Your service is live 🎉
Sep 28 03:46:26 PM  the young boy final understood the difference between parallel and perpendicular
```

7.2 Render Platform Deployment

The "corrspell" system is deployed on the Render platform to make it accessible via the web. It is hosted as a web service with the Flask-based user interface.



Deployed_link: https://spelling-corrector.onrender.com

8. Non-Functional Requirements

8.1 Performance

Performance testing ensures efficient and responsive spelling correction, even for large texts.

8.2 Scalability

The system can be designed to handle increased user load by scaling vertically or horizontally. Instance type can be upgraded and then it can handle more user efficiently. It is running on very poor instance type.

8.3 Reliability

Reliability is maintained through robust error handling and SymSpell integration.

9. Conclusion

This architecture document provides a comprehensive overview of the "corrspell" spelling correction system, including its components, interactions, design decisions, data flow, deployment, and non-functional requirements. It serves as a valuable resource for project development and understanding the system's structure.