**KWAME NKRUMAH UNIVERSITY OF**

**SCIENCE AND TECHNOLOGY**

**COLLEGE OF SCIENCE**

**DEPARTMENT OF COMPUTER SCIENCE**

**MINI-PROJECT DOCUMENTATION**

****

**PROJECT TITLE: WEBLINGO**

**SUPERVISOR: DR. KATE TAKYI**

**NAME: ABDULLAHI ALHASSAN BAABA**

**Reference Number: 20732782**

**INDEX NUMBER: 4234020**

Table Of Contents

[**Chapter1: Introduction** 4](#_Toc141432576)

[**1.1 Introduction to WebLingo – A Language Enhancement Chrome Extension** 4](#_Toc141432577)

[**1.2 Problem Statement** 4](#_Toc141432578)

[**1.3 Aims** 4](#_Toc141432579)

[**1.4 Key Objectives** 5](#_Toc141432580)

[**1.5 Project Scope** 5](#_Toc141432581)

[**1.6 Project Justification** 6](#_Toc141432582)

[**1.7 Beneficiaries of the Project** 6](#_Toc141432583)

[**1.8 Project Activity Planning and Schedules** 6](#_Toc141432584)

[**1.9 Academic and Practical Relevance of the Project** 6](#_Toc141432585)

[**1.10 Project Limitations** 7](#_Toc141432586)

[**1.11 Structure of the Report** 7](#_Toc141432587)

[**1.12 Project Deliverables** 7](#_Toc141432588)

[**CHAPTER 2: Review of Related Works / Review of Similar Systems** 8](#_Toc141432589)

[**2.1 Introduction** 8](#_Toc141432590)

[**2.2 Existing Systems and Their Features** 8](#_Toc141432591)

[**2.3 The Proposed System: WebLingo** 9](#_Toc141432592)

[**2.4 Conceptual Design** 9](#_Toc141432593)

[**2.5 Architecture of the Proposed System** 10](#_Toc141432594)

[**2.6 Components Designs and Descriptions** 10](#_Toc141432595)

[**2.7 Proposed System/Software Features** 18](#_Toc141432596)

[**2.8 Development Tools and Environment** 18](#_Toc141432597)

[**2.9 Benefits of Implementation of the Proposed System** 19](#_Toc141432598)

[**Chapter 3: Methodology** 20](#_Toc141432599)

[**3.1 Requirement Specification** 20](#_Toc141432600)

[**3.2 Stakeholders of the System** 20](#_Toc141432601)

[**3.3 Requirement Gathering Process** 20](#_Toc141432602)

[**3.4 Functional Requirements** 20](#_Toc141432603)

[**3.5 UML Diagrams** 21](#_Toc141432604)

[**3.6 Non-Functional Requirements** 24](#_Toc141432605)

[**3.7 Project Methods (Agile or Plan Driven)** 24](#_Toc141432606)

[**3.8 Software Process Models** 24](#_Toc141432607)

[**3.9 Chosen Model and Justification** 25](#_Toc141432608)

[**3.10 Project Design Consideration (Logical Designs)** 25](#_Toc141432609)

[**3.8 Developmental Tools** 25](#_Toc141432610)

[**Chapter 4: Implementation and Results** 27](#_Toc141432611)

[**4.1 Mapping Logical Design onto Physical Platform** 27](#_Toc141432612)

[**4.2 Construction** 29](#_Toc141432613)

[**4.3 Testing** 33](#_Toc141432614)

[**4.4 Results:** 34](#_Toc141432615)

[**Chapter 5: Findings and Conclusion** 35](#_Toc141432616)

[**5.1 Conclusions:** 35](#_Toc141432617)

[**Challenges/Limitations of the System:** 35](#_Toc141432618)

[**Lessons Learned** 35](#_Toc141432619)

[**Recommendations for Future Works:** 36](#_Toc141432620)

[**Recommendations for Project Commercialization:** 36](#_Toc141432621)

[**Reference List:** 37](#_Toc141432622)

# **Chapter1: Introduction**

## **1.1 Introduction to WebLingo – A Language Enhancement Chrome Extension**

The purpose of this documentation is to outline the development and implementation of WebLingo, a comprehensive language enhancement Chrome extension that aims to empower internet users with enhanced language capabilities and seamless access to translation, word breakdown, and definition features while browsing the web. In today's interconnected world, language barriers and difficulties in understanding complex words hinder effective communication and information access for many internet users. Existing translation tools and language resources often lack user-friendliness and integration with web browsers, limiting their accessibility and convenience. WebLingo seeks to address these challenges by offering an intuitive and comprehensive language enhancement solution within the Chrome browser environment.

WebLingo aims to revolutionize the way users navigate and comprehend content in different languages by leveraging popular translation APIs and language resources. By seamlessly integrating with the browser, WebLingo provides users with the ability to translate selected text, break down complex words, and access definitions directly within their web browsing experience. The extension's user-friendly interface and customizable display options ensure a seamless and personalized language enhancement experience.

This documentation will outline the project's objectives, methodology, proposed features, resource requirements, and project timeline. It highlights the significance of enhancing language accessibility and comprehension in today's digital landscape and sets forth the key objectives that WebLingo aims to achieve. Through the development and implementation of WebLingo, I strive to empower internet users with the tools and resources they need to overcome language barriers and enhance their browsing experience. Regular updates on the progress and milestones achieved in the project will be provided, keeping stakeholders informed and engaged in the development process.

## **1.2 Problem Statement**

In today’s connected world, language barriers and difficulties of comprehending complex vocabulary make it hard for internet user to communicate effectively and access information online. Existing translation tools and language resources are often not user-friendly and don't work well with web browsers, which makes them less convenient to use. WebLingo solves these problems by offering an easy-to-use language enhancement solution that works seamlessly in the Chrome browser.

## **1.3 Aims**

The aim of this project is to develop a software application that enhances language accessibility and comprehension for internet users. By leveraging translation APIs and language resources, the project aims to empower users with seamless language translation and word breakdown features within the web browsing experience. The project aims to address the following key objectives:

1. Enhance Language Accessibility: WebLingo aims to provide users with seamless access to language translation and word breakdown features within their web browsing experience. By eliminating language barriers, users will be able to communicate and understand content more effectively.
2. Improve Language Comprehension: The project aims to enhance users' comprehension of complex words by providing comprehensive word breakdowns. This includes synonyms, meanings, and usage examples, allowing users to expand their vocabulary and better understand the context in which words are used.
3. Integrate with Browsing Experience: WebLingo seeks to seamlessly integrate its language enhancement features with the browser, ensuring a smooth and intuitive user experience. By integrating with the browser's context menu, users can easily access translation and word breakdown functionality without disrupting their browsing flow.
4. Customize Display Options: The project aims to provide users with customizable display options for translations and word breakdowns. This allows users to choose their preferred format, such as tooltips or pop-up windows, for displaying translated text and word details.
5. Personalize User Settings: WebLingo will implement user settings to enable personalization. Users will have the ability to customize language preferences, including automatic language detection, source and target language selection, and API preferences. This personalization feature ensures a tailored language enhancement experience.

## **1.4 Key Objectives**

The key objectives of the WebLingo project are designed to address the challenges faced by internet users when it comes to language barriers and understanding complex words while browsing the web. These objectives form the foundation of the project and guide its development towards creating a comprehensive language enhancement solution within the Chrome browser environment. With these key objectives in mind, the WebLingo project aims to accomplish the following:

1. Develop a Chrome extension that seamlessly integrates with the browser and provides language translation and word breakdown features.
2. Enable users to translate selected text to their preferred languages using popular translation APIs of their choice.
3. Provide word breakdown functionality to assist users in understanding complex words by including synonyms, meanings, and usage examples.
4. Integrate the translation and breakdown feature with the browser’s context menu for easy access and seamless user experience.
5. Customize the display options of translation and word breakdown features, allowing users to choose between tooltips or pop-up windows.
6. Implement user settings to enable personalization, including automatic language detection, source and target language selection, and API preferences.

## **1.5 Project Scope**

The primary focus of this project is to create WebLingo, a browser extension that seamlessly integrates with widely-used web browsers like Chrome. WebLingo's main features include providing users with swift translation capabilities, language detection, comprehensive translations, contextual word breakdowns, and the flexibility to customize the display options to suit individual preferences.

By concentrating on language translation and comprehension, WebLingo aims to address the challenges posed by language barriers and complex vocabulary while browsing the internet. The extension will be designed to enhance the browsing experience of users by empowering them with effective language tools and resources. Users will be able to easily translate selected text, gain a deeper understanding of complex words, and access relevant language information with just a few clicks.

The project's scope is carefully defined to ensure a focused and efficient development process. While WebLingo aims to be a comprehensive language enhancement solution, the scope explicitly excludes the development of additional browser features unrelated to language translation and comprehension. This decision allows the development team to prioritize the core functionalities, ensuring a robust and user-friendly extension that delivers an intuitive language enhancement experience.

## **1.6 Project Justification**

In the modern digital landscape, WebLingo plays a vital role in enhancing language accessibility and comprehension. With the internet connecting people from various linguistic backgrounds, overcoming language barriers is crucial for effective communication and information access. WebLingo's language translation and comprehension tool aim to improve users' browsing experiences, foster cross-cultural communication, and promote inclusivity in the online space. It seeks to bridge the gaps caused by language differences and empower users with the linguistic tools to navigate the multilingual complexities of the internet. WebLingo's vision extends beyond convenience; it envisions a more united and empathetic online community. Through this project, I endeavor to unlock the true potential of language as a unifying force, breaking down barriers and promoting knowledge exchange without limitations. Together, let us embark on this transformative journey towards a more interconnected, inclusive, and enriched online experience with WebLingo.

## **1.7 Beneficiaries of the Project**

WebLingo, the language translation application, boasts a diverse and widespread user base, impacting individuals and professionals across various sectors. It fills a crucial gap in the digital age, where language barriers hinder effective communication and information access. By offering seamless language translation and comprehension, WebLingo empowers students and researchers with access to global knowledge and research materials. It enables business professionals to communicate effortlessly with international clients, driving cross-cultural partnerships. Travelers find ease in navigating foreign countries, while immigrants overcome language barriers in their new homes. As WebLingo gains widespread adoption, it fosters global understanding and cooperation, bridging linguistic divides and promoting inclusivity.

## **1.8 Project Activity Planning and Schedules**

The project will be executed in several phases, including research, data collection, model development, user interface design, and integration. Below are the 4 main phases:

1. Phase 1 - Research and Planning: Define project requirements, objectives and plan development approach and technologies
2. Phase 2 - Design and Development: This includes creating the UI/UX design for the extension, Implementing the backend functionalities and integrating the APIs
3. Phase 3 - Testing and Refinement: Perform comprehensive testing, gathering feedback and making necessary improvements to optimize performance and user experience
4. Phase 4 - Deployment and Launch: Prepare the extension for deployment and submit to the Chrome Web Store.

## **1.9 Academic and Practical Relevance of the Project**

Academically, the project contributes to the field of natural language processing and machine learning by exploring innovative methods for accurate translation. The application's practical relevance lies in its ability to transform communication dynamics across industries, promoting cross-cultural collaboration, and facilitating inclusive global participation.

## **1.10 Project Limitations**

Despite my best efforts, some limitations may impact the project's outcomes. Limited access to comprehensive language datasets and computational resources might affect the model's training and translation accuracy. Additionally, achieving real-time translations for less common languages could present challenges due to the scarcity of available data. Inability to translate entire webpages and reliance on internet connect to use WebLingo features is a limitation.

## **1.11 Structure of the Report**

The report is organized into distinct chapters, each focusing on different facets of the project.

Chapter 2, an in-depth analysis of related literature and existing translation solutions is provided, shedding light on the context and significance of the current project.

Chapter 3 elucidates the project's methodology, encompassing data collection, model architecture, and the implementation process. The design and development of the user interface are expounded upon in

Chapter 4, highlighting the user-centric approach adopted in creating an intuitive and engaging interface.

Chapter 5 showcases the seamless integration of the translation tool with web browsers, enhancing accessibility and usability for users.

Furthermore, Chapter 6 offers comprehensive details regarding the project's testing and evaluation procedures, ensuring the robustness and accuracy of the developed system.

Lastly, Chapter 7 serves as a conclusive chapter, summarizing key findings and providing valuable recommendations for future enhancements, empowering the project to evolve and cater to evolving user needs and technological advancements. Throughout the report, a comprehensive and cohesive narrative is presented, offering valuable insights into the project's objectives, achievements, and potential for further growth.

## **1.12 Project Deliverables**

The tangible outcomes of the project include:

1. A fully functional language translation application with a user-friendly interface.
2. Integration of the application with chrome web browsers for direct accessibility.
3. A comprehensive report documenting the research, development, and evaluation processes.
4. Presentation materials for showcasing the project to stakeholders and potential users.

# **CHAPTER 2: Review of Related Works / Review of Similar Systems**

## **2.1 Introduction**

In this chapter, I examine existing language improvement tools and related research that aims to help internet users better understand language. To comprehend the benefits and drawbacks of these systems, I evaluate their characteristics, advantages, and disadvantages. I also give a brief overview of the conceptual design, architectural structure, and component descriptions of the proposed system, WebLingo. The chapter's objectives are to provide background information on the development of WebLingo and emphasize its distinctive characteristics that set it apart from competing language improvement products.

## **2.2 Existing Systems and Their Features**

**2.2.1. Google Translate**

Google Translate is a widely used language translation app that provides quick translations for selected text. It seamlessly integrates with popular web browsers and offers a user-friendly interface. Users can highlight text, right-click, and access translation features from the browser's context menu. Google Translate supports a wide range of languages, making it a popular choice for quick translations while browsing the web.

1. Quick Translation: Users can instantly translate selected text to their preferred language.
2. Word Definitions: The system provides definitions, synonyms, and usage examples for selected words.
3. Browser Integration: System A integrates seamlessly with the browser's context menu for easy access.

**Pros of Google Translate**

1. User-Friendly: Google Translate’s simple interface allows users to perform quick translations with ease.
2. Fast Response: The system provides translations and definitions promptly, enhancing user efficiency.

**Cons of Google Translate**

1. Limited Language Support: While Google Translate supports a significant number of languages, it may not cover all linguistic needs.
2. Basic Word Breakdown: Google Translate offers limited word breakdown information, such as synonyms and usage examples.

**2.2.2. Duolingo**

Duolingo is a language enhancement app that combines translation, pronunciation, and language learning resources. It offers an extensive selection of languages for translation and learning purposes. Duolingo provides audio pronunciation of translated text and interactive language lessons to help users improve their language skills. Some key features of Duolingo include:

1. Comprehensive Language Support: Users can choose from a vast selection of languages for translation and learning.
2. Pronunciation Assistance: The system provides audio pronunciation of translated text and language learning content.
3. Interactive Language Lessons: System B offers interactive quizzes and exercises to facilitate language learning.

**Pros of Duolingo**

1. Extensive Language Options: Duolingo caters to a diverse user base with its wide range of supported languages.
2. Pronunciation Support: Users can enhance their language speaking skills with the app's pronunciation assistance.

**Cons of Duolingo**

1. Complex User Interface: Duolingo's extensive features might lead to a cluttered and overwhelming user interface.
2. Learning Focus: The app prioritizes language learning over quick translation, which might not suit all users' immediate translation needs.

## **2.3 The Proposed System: WebLingo**

WebLingo is a language enhancement Chrome extension that aims to address the limitations of existing systems and provide users with a seamless and intuitive language enhancement experience. It combines the convenience of quick translations with comprehensive word breakdowns to offer users a comprehensive language comprehension tool. Key features of WebLingo include:

1. Quick Translation: Users can instantly translate selected text to their preferred languages using popular translation APIs.
2. Word Breakdown: WebLingo provides in-depth word analysis, including synonyms, meanings, pronunciations and usage examples.
3. Chrome Integration: WebLingo seamlessly integrates its language enhancement features with the browser's context menu for easy access.

## **2.4 Conceptual Design**

The conceptual design of WebLingo revolves around providing users with an intuitive and user-friendly language enhancement experience. The extension leverages popular translation APIs and language resources to offer accurate and contextually relevant translations. The conceptual design focuses on the following key aspects:

1. User-Focused Interface: WebLingo prioritizes simplicity and ease of use to cater to a wide range of users, including beginners and language enthusiasts.
2. Comprehensive Word Analysis: The extension aims to empower users with in-depth word breakdowns to enhance their language comprehension.
3. Seamless Integration: WebLingo integrates seamlessly with the Chrome browser, allowing users to access language enhancement features effortlessly.

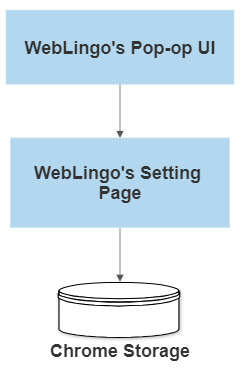
## **2.5 Architecture of the Proposed System**

The architecture of WebLingo is based on a client-server model, where the Chrome extension acts as the client, and the server hosts the translation and word breakdown functionalities. The system's architecture consists of the following components:

1. Chrome Extension Frontend: The frontend component of WebLingo includes the pop-up window, user interface elements, settings page and context menu integration within the Chrome browser. It allows users to interact with the extension and access language enhancement features.
2. Backend Functionality: The backend component encompasses the translation and word breakdown functionalities as well as the chrome storage which acts local as a database. It also connects with popular NodeJS frameworks to produce better results. It interacts with popular translation APIs and language resources to provide accurate translations and word details to users.
3. API Integration: WebLingo integrates with selected translation and word breakdown APIs, such as Google Cloud Translation and Merriam-Webster Dictionary API. These APIs provide the necessary language resources and data for accurate translations and word analysis.
4. User Settings and Preferences: The system includes a settings component that allows users to customize their language preferences, including automatic language detection, source and target language selection, and API preferences.

## **2.6 Components Designs and Descriptions**

WebLingo follows a client-side architecture, with the extension residing and executing within the Chrome browser. The front-end components of WebLingo comprise the user interface elements, such as the pop-up window and context menu integration. The back-end functionality involves utilizing popular translation APIs and language resources to provide accurate translations and comprehensive word breakdowns.



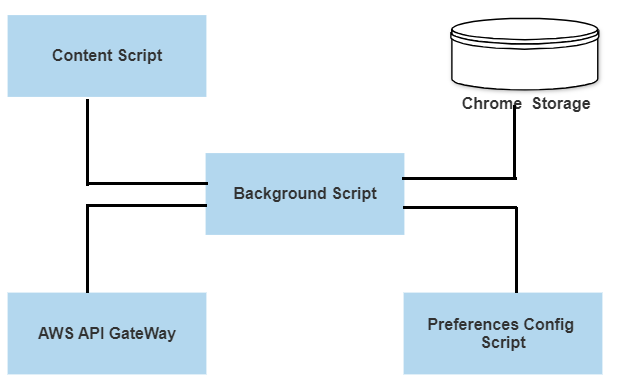
**Figure 2.1 – Frontend Component of WebLingo**

**Front-End Components:**

* 1. Pop-up Window: The pop-up window serves as the user interface for WebLingo. It allows users to interact with the extension, for translation, and view word breakdowns.
  2. Context Menu Integration: WebLingo integrates with the browser's context menu, enabling users to access translation and word breakdown features directly by right-clicking on selected text.
  3. User Interface Elements: The front-end also includes various UI elements like buttons, input fields, and displays to provide a seamless and intuitive user experience.

**Back-End Components:**

1. Translation Component: The translation component handles the process of translating selected text to the user's preferred language. It interacts with chosen translation APIs, such as Google Cloud Translation and Microsoft APIs, to obtain accurate translations.
2. Word Breakdown Component: This component is responsible for providing in-depth word analysis, including synonyms, meanings, and usage examples. It interacts with language resources, such as Merriam-Webster and Oxford Dictionary APIs, to fetch word details.
3. Background Script: The background script is a crucial part of the extension's architecture. It handles communication between the front-end components and back-end functionalities, facilitating seamless data transfer and execution of operations.



**Figure 2.2 – Backend Component of WebLingo**

WebLingo's architecture encompasses essential components that work cohesively to deliver a powerful and user-friendly language enhancement experience. These components are designed to cater to the diverse needs of internet users and address language barriers effectively.

1. **Background Script:**

Description:

The Background Script in WebLingo plays a critical role in orchestrating the communication and coordination between various components of the extension. As the central hub, it serves as the backbone of the entire system, facilitating seamless data transfer and ensuring that each component functions harmoniously to deliver a smooth and efficient user experience.

Functionality:

1. The background script receives user input and requests from the front-end components, processes them, and forwards the necessary information to the translation and word breakdown components. It also sends back the results to be displayed in the pop-up window.
2. Message Passing: One of the primary functions of the Background Script is to handle message passing between the front-end components, such as the Pop-up Window and Context Menu, and the back-end components, including the Translation and Word Breakdown functionalities. When a user interacts with the extension, such as triggering a translation request or selecting a word for breakdown, the Background Script receives these messages and acts as an intermediary to process and relay the information to the appropriate components.
3. API Integration: The Background Script takes charge of integrating with external APIs, such as Lingoscan and other language resources, to facilitate translation and word breakdown requests. It manages API calls and responses, ensuring that user preferences for language selection and feature preferences are communicated effectively to the APIs. Additionally, the Background Script handles error handling and gracefully manages any issues that may arise during API communication.
4. Data Persistence: To enhance user experience and continuity across browsing sessions, the Background Script is responsible for managing data persistence. It stores and retrieves user preferences, language selections, and other settings in the Chrome storage. By doing so, the extension remembers the user's personalized choices and maintains consistency even when the browser is closed or reopened.
5. Background Event Handling: As the extension operates in the background, the Background Script handles various events, such as the resizing of the browser window, to adjust the display and behavior of components accordingly. It optimizes the extension's performance and resource utilization to provide a seamless experience for users without imposing significant overhead on the browser.

In summary, the Background Script in WebLingo serves as the backbone of the extension, facilitating smooth communication and coordination between front-end and back-end components. Its functions range from message passing and API integration to data persistence and event handling, all aimed at delivering a user-friendly, efficient, and personalized language enhancement experience for internet users. Through the Background Script's reliable and robust operation, WebLingo empowers users to overcome language barriers, enhance language comprehension, and seamlessly access language-related features while browsing the web.

**2. Content Script:**

Description:

The Content Script is a crucial component of WebLingo, responsible for injecting custom scripts and styles into webpages when users access them. It enables the extension to interact with the content of webpages and provide language enhancement features directly within the user's browsing experience.

Functionalities:

1. Dynamic Page Interaction: When a user navigates to a webpage, the Content Script automatically runs and injects custom JavaScript code and CSS styles into the page's DOM (Document Object Model). This dynamic interaction allows the extension to access and manipulate the content of the webpage in real-time.
2. Text Selection and Interaction: The Content Script enables users to select text on a webpage, triggering language enhancement options through the Context Menu. When a user selects text, the Content Script sends a message to the Background Script, which then initiates the translation or word breakdown process.
3. Context Menu Integration: The Content Script works in conjunction with the Background Script to integrate language enhancement options into the browser's context menu. It ensures that the extension's features are available to users via right-clicking on selected text, providing a user-friendly and intuitive way to access translation and word breakdown functionalities.
4. Displaying Translated Text: After receiving the translation results from the Background Script, the Content Script modifies the webpage's DOM to display the translated text as a dialogue box, based on the user's settings. This dynamic modification ensures that users can view translated text seamlessly without interrupting their browsing flow. For complex words selected by the user, the Content Script communicates with the Background Script to retrieve detailed word breakdown information from the chosen API (e.g., Lingoscan, Merriam-Webster, Oxford Dictionary). It then formats and displays this information in a tooltip or pop-up window, allowing users to gain comprehensive insights into word meanings, synonyms, antonyms, and usage examples.
5. Dynamic Adaptation: The Content Script is designed to adapt to changes in the webpage's content or structure. It continuously monitors the page and responds to dynamic content loading, ensuring that the language enhancement features remain functional even when webpages change or new elements are added.
6. User Interaction Handling: When users interact with the extension's features on the webpage, the Content Script handles their actions and updates the DOM accordingly. For example, display dialogue box, adjusting settings, or triggering additional language features are managed by the Content Script to maintain a smooth and user-friendly experience.

**3. Pop-up Window:**

Description:

The pop-up window serves as the primary user interface of WebLingo, providing users with quick access to the extension's core functionalities. When users click on the WebLingo extension icon, the pop-up window appears, offering an overview of the extension's features and functionality. The pop-up window is designed to be user-friendly, visually appealing, and informative, ensuring a seamless and intuitive browsing experience.

Functionality:

Upon opening the pop-up window, users are presented with an introductory message that welcomes them to WebLingo and explains the extension's key features. The pop-up window provides users with options to interact with the extension in the following ways:

1. Extension Information: Users can click on the extension icon to have an overview of the extension’s features from the pop-up. The pop-up window contains sections that provides detailed information about the extension. It displays the information about the extension and detailed explanation of the features of the extension.

2. Settings Button: The pop-up window serves as the basic entry point of the user interface, providing users with access to the extension's core functionalities. The pop-up window features a "Settings" button, which allows users to access the user settings section. Here, users can personalize language preferences, API selections, display options, and more, tailoring the extension to suit their language needs.

3. User-Friendly Interface: The pop-up window features an intuitive and visually appealing interface, making it easy for users to navigate and interact with the extension. The interface's layout and design ensure a seamless and enjoyable language enhancement experience.

Diagram: (Include a diagram of the pop-up window interface)

1. **Context Menu Integration:**

Description:

WebLingo seamlessly integrates its language enhancement features into the context menu of the web browser. This integration ensures that users can access translation and word breakdown options with just a right-click on selected text within a webpage. By integrating with the context menu, WebLingo provides users with a convenient and efficient way to enhance their language comprehension while browsing the web.

Functionality:

When users right-click on a selected portion of text within a webpage, WebLingo's context menu Integration will appear, displaying the following options:

1. Translate: The context menu will feature a “Quick Translate” and a “Translate With…” option, allowing users to instantly translate the selected text into their preferred target language. “Quick Translate” does the translation by auto-detecting the source language and using the user’s preferred target language from user settings. Whereas the user chooses a target language when he/she uses the “Translate With…” for translation. By selecting this option, users can quickly access translations without the need to open the pop-up window, streamlining the language enhancement process.
2. Word Breakdown: Additionally, the context menu will provide a "LingoScan" option, enabling users to obtain detailed information about complex words directly from the context menu. By selecting this option, users can access synonyms, meanings, and usage examples, empowering them with comprehensive language understanding within seconds.
3. Non-Intrusive Access: WebLingo's context menu integration operates unobtrusively, appearing only when users perform a right-click action on selected text. This non-intrusive approach ensures that users can access language enhancement options without interrupting their browsing flow or cluttering the interface.

Diagram: (Include a diagram illustrating the context menu integration)

1. **User Settings Component:**

Description:

WebLingo incorporates a user settings feature that allows users to personalize their language enhancement experience. Users can customize language preferences, API choices, and display options to tailor WebLingo to their specific needs and preferences. The user settings are an essential component of WebLingo, ensuring a highly personalized and efficient language enhancement tool.

Functionality:

1. Language Preferences: Users can set their preferred source and target languages for translations. WebLingo will automatically detect the source language when translating, but users have the option to select a different source language if needed. Users can also choose their frequently used target languages for quick and easy translations.
2. API Preferences: WebLingo integrates with multiple translation and word breakdown APIs. In the user settings, users can choose their preferred APIs, providing them with flexibility in selecting the most accurate and reliable translation resources.
3. Automatic Language Detection: Users can enable or disable automatic language detection. When enabled, WebLingo will automatically detect the source language of selected text for translation, eliminating the need for manual language selection.
4. Lemmatization: WebLingo also offers the option to enable or disable Lemmatization within the settings page. When enabled, Lemmatization helps users to identify the base or root form of words, providing a more accurate and concise understanding of the text. By toggling this option on or off, users can customize their language enhancement experience according to their linguistic preferences.
5. Enabling/Disabling Features: Users can enable or disable specific language enhancement features for each selected LingoScan API. This level of granularity allows users to fine-tune their language enhancement experience and control which information they want to access when interacting with translated text or word breakdowns. Once users have chosen their preferred LingoScan APIs, they can further customize the features they want to enable or disable for each API. For example, if a user selects a synonyms API, they can choose whether they want synonyms to be displayed alongside the translated text or the word breakdowns. Similarly, for a pronunciation API, users can decide whether to include pronunciation details in the word breakdowns.
6. Data Persistence in Chrome Store: WebLingo incorporates a robust data persistence mechanism by utilizing the Chrome Storage API to store and retain user settings securely. The Chrome Storage API enables the extension to save user preferences and configuration choices, ensuring that these settings persist across browser sessions and even after the browser is closed. This feature provides users with a seamless and consistent language enhancement experience, as their customized settings are readily available whenever they use WebLingo.
7. Seamless Retrieval and Application of Settings: The data stored in the Chrome Store is seamlessly retrieved by WebLingo each time the extension is initialized. As a result, users do not have to repeatedly set their preferred language enhancement features or API preferences. Instead, the extension intelligently loads the saved settings, applying them to the current browsing session. This functionality allows users to pick up exactly where they left off, without the need for manual reconfiguration.
8. Synchronization: The Settings Script ensures that the front-end components and the back-end functionalities are in sync and up-to-date. For instance, when a user changes language preferences or enables/disables certain features in the settings, the Settings Script ensures that these changes are reflected across all components in real-time.
9. **Translation Component:**

Description:

The translation component is responsible for processing user input and facilitating language translation using chosen APIs.

Functionality:

* + - 1. Multi-API Support: WebLingo integrates with popular translation APIs, giving users the freedom to select their preferred translation service. Users can choose from a range of APIs, such as Google Cloud Translation, Microsoft Translator etc based on their personal preferences and the desired translation quality.
      2. Automatic Language Detection: For added convenience, the Translation Component incorporates automatic language detection. When users do not specify the source language, the component automatically identifies the language of the input text, streamlining the translation process without the need for manual language selection.
      3. Translation Results: Once the translation is initiated, the Translation Component sends the text and selected language preferences to the chosen API. It then receives and displays the translation results in real-time. Users can instantly view the translated text within the dialogue box.
      4. Error Handling: Robust error handling is integrated into the Translation Component to ensure a smooth user experience. In the event of any issues or errors with the selected API, the component provides clear and informative error messages, guiding users on how to address the problem effectively.

Diagram: (Include a diagram representing the flow of translation component)

1. **Word Breakdown Component:**

Description:

The word breakdown component provides users with comprehensive word analysis, including synonyms, meanings, and usage examples. When a user selects the word breakdown option in the pop-up window or context menu, the word breakdown component fetches word details from language resources like Merriam-Webster and Oxford Dictionary APIs. It then displays the breakdown to the user.

Functionality:

* + - 1. Lemmatization Option: As an advanced language comprehension feature, the Translation Component offers users the ability to enable or disable lemmatization. When enabled, the component presents base word forms or lemmas alongside the translated text, providing users with a deeper understanding of word roots and grammatical variations.
      2. Pronunciation Assistance: To aid language learning and pronunciation, the Translation Component offers pronunciation assistance. Users can choose to view phonetic pronunciation guides for translated text or utilize an audio playback feature to hear the pronunciation, improving their language listening skills.
      3. Synonym and Antonym Suggestions: For enhanced vocabulary enrichment, users can enable synonym and antonym suggestions for translated words. When enabled, the Translation Component provides synonyms and antonyms for selected translated words, helping users explore alternative word choices.
      4. Error Handling: The Lingoscan Component in WebLingo is designed with robust error handling to ensure a smooth user experience. In the event of any issues or errors with the Lingoscan API or other selected language APIs, the component provides clear and informative error messages. These messages guide users on how to address the problem effectively, allowing them to navigate any challenges related to language translation with confidence and ease.

Diagram: (Include a diagram illustrating the flow of word breakdown component)

1. **Display Dialogue Box:**

Description:

The Dialogue Box is a user-friendly and visually appealing interface that serves as the medium for displaying translation results, word breakdowns, and other language-related information to the user. It appears as a pop-up window within the browser, ensuring a seamless and intuitive user experience. Overall, the Dialogue Box in WebLingo is an essential component that enhances the user experience by providing a visually appealing and user-friendly interface for accessing language-related results and information. Its seamless integration with the extension's core functionalities makes it a valuable tool in overcoming language barriers and promoting language comprehension while browsing the web.

Functionality:

* + - 1. When the user triggers a translation or word breakdown request through the context menu, the Dialogue Box is activated to display the relevant information. For translation requests, the Dialogue Box shows the translated text along with language details such as the source and target languages. Additionally, for word breakdowns, the Dialogue Box presents in-depth analysis, including synonyms, antonyms, pronunciation, and usage examples.
      2. The Dialogue Box is designed to be customizable and user-friendly. Users can have the option to close it when they no longer need the displayed information. Furthermore, the Dialogue Box remains responsive to changes in the browser window size, ensuring that it adapts well to different screen resolutions and maintains an optimal display.
      3. The Dialogue Box also incorporates a clean and modern design, utilizing easy-to-read fonts and well-organized content. This thoughtful layout ensures that users can quickly comprehend and absorb the language-related information without feeling overwhelmed.

## **2.7 Proposed System/Software Features**

WebLingo offers a range of features designed to enhance language accessibility and comprehension while browsing the web. These features include:

1. Quick Translation: Instantly translate selected text to preferred languages.
2. Language Detection: Automatically detect the source language for translation without manual selection.
3. Lemmatization: Lemmatization is the process of grouping inflected forms together as a single base form. Reduce the different forms of a word to one single form, for example, reducing "builds", "building", or "built" to the lemma "build" before using lingo scan.
4. Accurate Translations: Choose from multiple languages for precise and relevant translation results.
5. Word Breakdown: Access in-depth word analysis, including meanings, synonyms, and usage examples.
6. User-Friendly Settings: Personalize language preferences and API options for a tailored language enhancement experience.
7. Chrome Integration: Access language enhancement features directly from the browser's context menu for easy and seamless use.

## **2.8 Development Tools and Environment**

The development of WebLingo will utilize a set of software tools and a conducive development environment to facilitate efficient implementation. The tools and environment include:

1. **Development Software**:

JavaScript, HTML, CSS, Node.js, Express.js, AWS Lambda, and AWS API Endpoint formed the core technologies for building the WebLingo extension and backend functionality. JavaScript was used to create the dynamic and interactive elements of the extension, while HTML and CSS were utilized for structuring and styling the pop-up window and content. Node.js and Express.js provided a robust server-side environment for handling API requests and processing user input. AWS Lambda and AWS API Endpoint facilitated the deployment of serverless functions, ensuring scalability and cost-effectiveness for the backend implementation.

1. **Development Tools**:

Visual Studio Code (VS Code) served as the primary integrated development environment (IDE) for coding and debugging WebLingo. VS Code's intuitive interface and powerful extensions made it a versatile tool for writing, testing, and maintaining the extension's codebase. It provided essential features such as code highlighting, auto-completion, and version control integration, streamlining the development process and ensuring code quality.

1. **Debugging Tools**:

Chrome Developer Tools and AWS CloudWatch played critical roles in testing and debugging the WebLingo extension and backend APIs. Chrome Developer Tools allowed developers to inspect and manipulate the extension's DOM, monitor network activity, and debug JavaScript code in real-time. AWS CloudWatch provided comprehensive monitoring and logging for AWS Lambda functions and API calls, aiding in identifying and resolving potential issues within the backend infrastructure. Postman was employed for API testing, enabling developers to simulate API requests and responses to verify functionality and ensure API endpoints worked as expected.

1. **API Access**:

Integration with selected translation and word breakdown APIs was fundamental to WebLingo's language enhancement features. Google Cloud Translation API was utilized for seamless and accurate language translation, while the Merriam-Webster Dictionary API provided detailed word breakdowns, including synonyms, antonyms, and usage examples. These APIs allowed WebLingo to access vast language resources and deliver comprehensive language insights to users in real-time.

1. **Hardware and Infrastructure**:

Standard computing devices, reliable internet connectivity, testing devices, and hosting platforms formed the necessary hardware and infrastructure for developing, testing, and deploying WebLingo. Developers utilized their computers and reliable internet connections for day-to-day coding and testing activities. Additionally, testing devices were used to ensure the extension's compatibility and functionality across various devices and browsers. For deployment, the Chrome Web Store served as the hosting platform, enabling users to download and install the WebLingo extension seamlessly.

Overall, the combination of development software, tools, APIs, and infrastructure played a crucial role in the successful implementation of WebLingo, enabling the creation of a robust and user-friendly language enhancement Chrome extension. The seamless integration of these resources ensured that WebLingo achieved its objectives of enhancing language accessibility, improving comprehension, and providing a personalized browsing experience for users across the digital landscape.

## **2.9 Benefits of Implementation of the Proposed System**

The successful implementation of WebLingo will bring forth several benefits for internet users seeking to overcome language barriers and enhance their language comprehension:

1. Enhanced Language Access: WebLingo will provide users with quick translations and in-depth word breakdowns, enabling effective communication and comprehension across different languages.
2. Improved Comprehension: The comprehensive word breakdown feature will empower users to understand complex words better and grasp their contextual usage.
3. Seamless Integration: WebLingo's seamless integration with the Chrome browser ensures a smooth user experience and easy access to language enhancement features.
4. Personalization Options: Users can customize their language preferences and API choices, tailoring the language enhancement experience to their specific needs.
5. User-Friendly Interface: The intuitive and user-friendly design of WebLingo will appeal to a wide range of users, making language enhancement accessible to all.
6. Cross-Cultural Communication: WebLingo will foster cross-cultural communication by breaking down language barriers and promoting inclusivity in the digital space.

By addressing these advantages, WebLingo aims to revolutionize the way users navigate the multilingual landscape of the internet and empower them with a transformative language enhancement solution.

# **Chapter 3: Methodology**

## **3.1 Requirement Specification**

WebLingo was developed to address the need for enhanced language accessibility and comprehension while browsing the web. It aims to empower users with seamless language translation, word breakdown, and definition features within the Chrome browser. The extension targets internet users seeking to overcome language barriers and improve their understanding of complex words and phrases. WebLingo is designed to be user-friendly, customizable, and integrative, offering a personalized language enhancement experience.

## **3.2 Stakeholders of the System**

The stakeholders of the WebLingo system include:

1. End Users: Internet users who install and utilize the WebLingo extension to enhance their language accessibility and comprehension while browsing the web.
2. Developers: The developer responsible for building, testing, and maintaining the WebLingo extension and backend functionalities.
3. Project Supervisor: Dr. Kate Takyi, the supervisor overseeing the development process and providing guidance and support throughout the project.
4. Project Owner: Abdullahi Alhassan Baaba, the creator and owner of the WebLingo project.

## **3.3 Requirement Gathering Process**

The requirement gathering process for WebLingo involved several steps, including:

1. Identifying the Problem: Recognizing the language barriers and difficulties faced by internet users, which hindered effective communication and information access.
2. Stakeholder Consultation: Engaging with potential end users and stakeholders to gather insights into their language enhancement needs and preferences.
3. Market Research: Studying existing language-related Chrome extensions and identifying their strengths, weaknesses, and opportunities for improvement.
4. Establishing Functional and Non-Functional Requirements: Defining the core functionalities and performance expectations for WebLingo along with usability requirements.

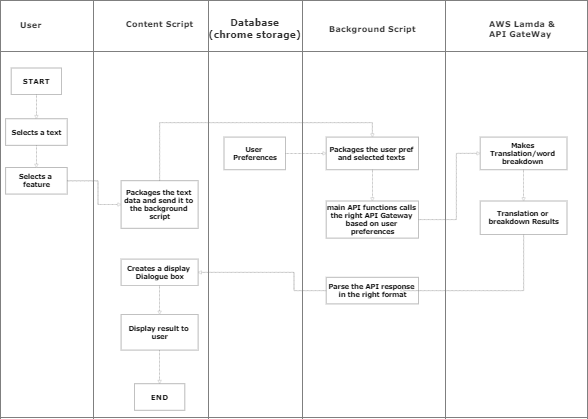
## **3.4 Functional Requirements**

The functional requirements of WebLingo include:

1. Language Translation: Provide users with the ability to translate selected text to their preferred languages using popular translation APIs of their choice.
2. Word Breakdown: Offer in-depth word analysis, including synonyms, meanings, and usage examples, to assist users in understanding complex words.
3. Context Menu Integration: Seamlessly integrate translation and word breakdown features into the browser's context menu for easy access while browsing.
4. Lemmatization: Introduce lemmatization as an additional feature in WebLingo, enabling users to access the base or root form of words. This process reduces words to their base form, providing users with a clearer understanding of the word's core meaning, regardless of its grammatical variations.
5. User Settings: Implement user settings to enable personalization, including automatic language detection, source and target language selection, and API preferences.
6. Display Dialogue Box: Implement a dialogue box in the pop-up window to display the translation and word breakdown results. The dialogue box will present the translated text, along with the in-depth word analysis, including synonyms, meanings, usage examples, and lemmatized form of complex words. This interactive dialogue box enhances the user experience by providing a clear and comprehensive view of the language enhancement results.

## **3.5 UML Diagrams**

The UML diagrams present a visual representation of the use cases, activities, sequences, and class diagrams of WebLingo.



**Figure 3.1. UML diagram (ACTIVITY) for WebLingo’s Front-End**

**3.5.1 Use Case Diagrams**

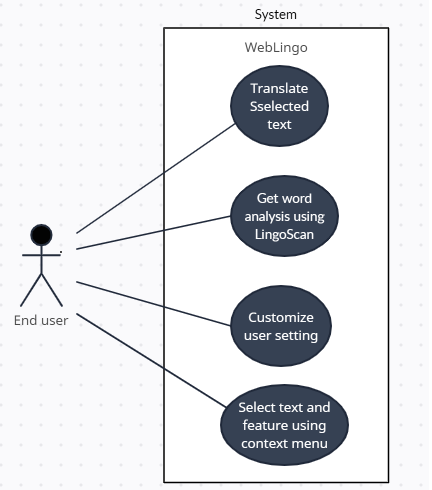
Use case diagrams illustrate the interactions between users and the system. For WebLingo, the following use case diagrams are presented:

**Preconditions for all use cases**: The user has opened a webpage and activated the WebLingo Chrome extension.

1. Use Case Diagram for Front-end Models: This diagram shows the use cases related to the front-end components of WebLingo, including translation, word breakdown, and user settings.
2. Use Case Diagram for Back-end Models: This diagram demonstrates the interactions between the front-end and back-end components of WebLingo, detailing how user input is processed and responses are generated.

**3.5.2 Use Case Descriptions for Front-End**

Each use case description provides a detailed explanation of the roles of actors (users) in the use case diagrams and the functionalities of each use case.

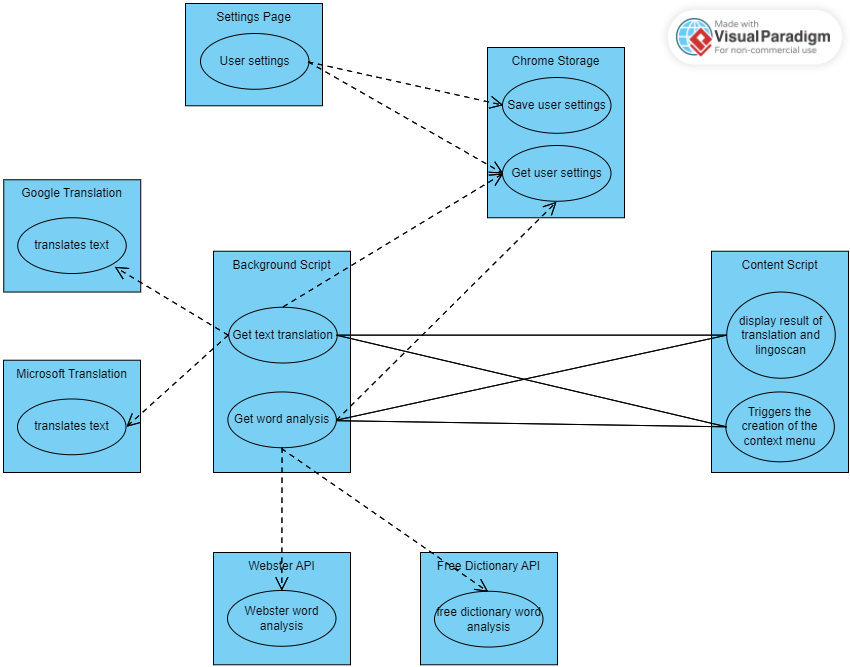


**Figure 3.2. User Case Diagram for WebLingo’s Front-End**

1. **Use Case: Translate Text**
   1. Description: This use case allows users to translate selected text to their preferred languages using the chosen translation API.
   2. Actor: End User
   3. Flow of Events:
      1. User selects text on a webpage.
      2. User triggers the "Translate Text" action from the context menu.
      3. WebLingo's background script receives the request.
      4. The background script sends the selected text and language preferences to the translation component.
      5. The translation component processes the request and sends it to the chosen translation API.
      6. The API translates the text into the target language.
      7. The translation component receives the translation and displays it in the pop-up window.
2. **Use Case: Word Breakdown**
   1. Description: This use case enables users to access in-depth word analysis, including synonyms, meanings, and usage examples for complex words.
   2. Actor: End User
   3. Flow of Events:
      1. User selects a complex word on a webpage.
      2. User triggers the "Word Breakdown" action from the context menu.
      3. WebLingo's background script receives the request.
      4. The background script sends the selected word to the word breakdown component.
      5. The word breakdown component retrieves the word's details from the chosen dictionary API.
      6. The component processes the data and displays the word breakdown in the pop-up window.
3. **Use Case: Customize User Settings**
   1. Description: This use case allows users to customize language preferences according to their preferences.
   2. Actor: End User
   3. Flow of Events:
      1. User opens the WebLingo pop-up window.
      2. User navigates to the "Settings" section.
      3. User makes selections for automatic language detection, source and target language preferences, and API choices.
      4. User makes selections for lemmatization, API choices and edits API features.
      5. WebLingo saves the user settings in the Chrome Store for persistent storage.

**3.5.2 Use Case Descriptions for Back-End**

1. **Use Case: Google Translate**
2. Actor: User
3. Description: This use case represents the interaction between the user and the Google Translate API through the background script. The user requests a translation of selected text or from the pop-up window, and the background script communicates with the Google Translate API to fetch the translation result. The translated text is then displayed in the dialogue box in the pop-up window.
4. **Use Case: Microsoft Translate**
5. Actor: User
6. Description: This use case represents the interaction between the user and the Microsoft Translate API through the background script. Similar to the Google Translate use case, the user requests a translation of selected text or from the pop-up window, and the background script communicates with the Microsoft Translate API to retrieve the translation result. The translated text is displayed in the dialogue box in the pop-up window.
7. **Use Case: Webster API**
8. Actor: User
9. Description: This use case represents the interaction between the user and the Webster API through the background script. The user requests the breakdown of a complex word from the pop-up window, and the background script communicates with the Webster API to fetch the word analysis, including synonyms, meanings, and usage examples. The results are displayed in the dialogue box in the pop-up window.
10. **Use Case: Free Dictionary API**
11. Actor: User
12. Description: This use case represents the interaction between the user and the Free Dictionary API through the background script. Similar to the Webster API use case, the user requests the breakdown of a complex word from the pop-up window, and the background script communicates with the Free Dictionary API to retrieve the word analysis. The results are displayed in the dialogue box in the pop-up window.
13. **Use Case: Get Settings**
14. Actor: User
15. Description: This use case represents the user's interaction with the background script to retrieve their selected settings and preferences. The user accesses the pop-up window, and the background script fetches the user's language preferences, translation options, and other settings from the data storage. The settings are then applied to provide a personalized language enhancement experience to the user.
16. **Use Case: Save Settings**
17. Actor: User
18. Description: This use case represents the user's interaction with the background script to save their selected settings and preferences. The user accesses the pop-up window and customizes their language preferences, translation options, and other settings. The background script then stores these settings in the data storage to ensure that the user's preferences are maintained across sessions and browsing activities.
19. **Use Case: Display Dialogue Box (Content Script and Browser)**
20. Actor: User
21. Description: This use case represents the interaction between the user, the content script, and the browser to display the dialogue box containing translation and word breakdown results**.**

****

**Figure 3.3. User Case Diagram for WebLingo’s Back-End**

## **3.6 Non-Functional Requirements**

WebLingo's non-functional requirements include:

1. Security Concepts: WebLingo implements secure data handling. All API calls are encrypted to safeguard sensitive information.
2. Performance: The extension is designed to provide fast and accurate translation and word breakdown results, ensuring a smooth user experience without delays or disruptions.
3. Usability: WebLingo prioritizes user-friendliness and intuitive design, enabling users to access language enhancement features effortlessly and navigate the extension seamlessly.

## **3.7 Project Methods (Agile or Plan Driven)**

The project followed an agile software development methodology to facilitate flexibility, collaboration, and continuous improvement throughout the development process. Agile principles allowed the development team to respond to changes and evolving requirements effectively.

## **3.8 Software Process Models**

The software process models considered for the development of WebLingo include:

1. Agile Software Development: Agile methodologies, such as Scrum, were considered for their iterative and incremental approach, enabling regular releases and continuous feedback.
2. Waterfall Model: The waterfall model was also evaluated for its linear and structured approach, but its rigid nature made it less suitable for a dynamic project like WebLingo.

## **3.9 Chosen Model and Justification**

The chosen software process model for WebLingo is the Agile Software Development methodology. Agile's iterative and incremental nature allowed for regular progress updates, stakeholder feedback, and rapid adaptations to changing requirements. The development team engaged in sprints to deliver new features and improvements at regular intervals, ensuring a responsive and user-centric approach to development.

## **3.10 Project Design Consideration (Logical Designs)**

WebLingo's logical designs encompassed the following aspects:

1. UI Design: The UI design included wireframes that illustrated the layout and interactions of the WebLingo pop-up window and user interface elements. These wireframes provided a visual representation of how users would interact with the extension's language enhancement features.
2. DB Design (if applicable): WebLingo uses the chrome storage as a form of local database to store user settings.

## **3.8 Developmental Tools**

The developmental tools used in the project were further utilized as follows:

1. Visual Studio Code (VS Code): VS Code served as the primary IDE for coding and debugging WebLingo. Its features, such as code highlighting and version control integration, significantly streamlined the development process.
2. Chrome Developer Tools: Chrome Developer Tools facilitated testing and debugging of the extension's front-end components. It allowed developers to inspect the DOM, monitor network activity, and analyze JavaScript code in real-time.
3. AWS CloudWatch: AWS CloudWatch was employed to monitor and log AWS Lambda functions and API calls in the backend. It provided valuable insights into the performance and health of the backend infrastructure.
4. Postman: Postman was used for API testing, enabling developers to simulate API requests and responses to verify functionality and ensure the correctness of API endpoints.
5. Chrome Storage: Chrome Storage API was employed to store and manage user settings and preferences in the extension. It provided a reliable and persistent storage solution, allowing users to customize their language enhancement experience and maintain their choices across browsing sessions.
6. JavaScript (JS): JavaScript was the core programming language used to develop the front-end components of WebLingo. Its versatility and compatibility with web browsers made it the ideal choice for creating interactive and user-friendly features.
7. ExpressJS: ExpressJS served as the back-end framework for WebLingo, providing a robust and flexible foundation for handling API requests and responses. It streamlined the development of backend functionalities and facilitated smooth communication between the front-end and back-end components.
8. AWS Lambda: AWS Lambda served as the serverless computing platform for hosting the backend functions of WebLingo. It allowed developers to run code without managing servers, providing a scalable and cost-effective solution for executing backend operations.
9. AWS API Gateway: AWS API Gateway was utilized to create, manage, and secure the APIs used in WebLingo. It acted as the gateway for API calls, handling authentication, rate limiting, and request/response transformations.

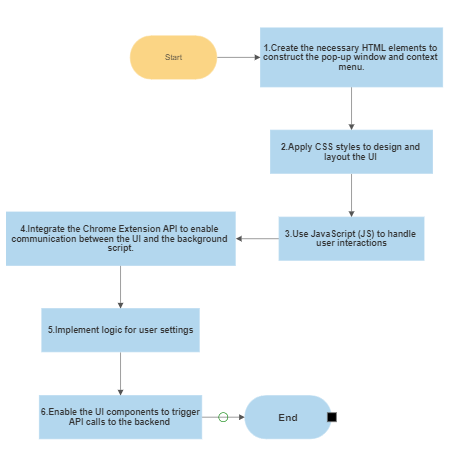
# **Chapter 4: Implementation and Results**

## **4.1 Mapping Logical Design onto Physical Platform**

**4.1.1 Algorithm for Implementing UI:**

The user interface (UI) of WebLingo was implemented using HTML, CSS, and JavaScript (JS) in conjunction with the Chrome Extension API. The following algorithm outlines the high-level steps for implementing the UI:

1. Create the necessary HTML elements to construct the pop-up window and context menu.
2. Apply CSS styles to design and layout the UI components, ensuring a visually appealing and user-friendly interface.
3. Use JavaScript (JS) to handle user interactions, such as mouse clicks and text selection.
4. Integrate the Chrome Extension API to enable communication between the UI and the background script.
5. Implement logic for user settings, language preferences, and display options, allowing users to customize their language enhancement experience.
6. Enable the UI components to trigger API calls to the backend for translation and word breakdown functionalities.
   * 1. **Flowchart Diagram for Implementing UI:**

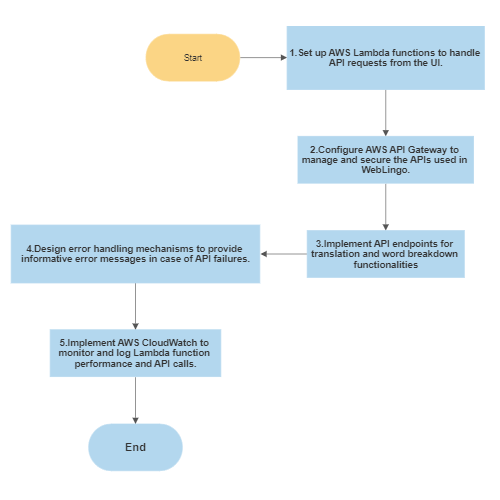
****

**Figure 4.1 – Flow chart for UI design**

**4.1.3 Algorithm for Implementation of Database Development:**

WebLingo does not require a database for its core functionalities as it utilizes Chrome Storage to store user settings and preferences. However, AWS Lambda functions were used to process API calls and manage language resources. The following algorithm outlines the key steps for implementing the backend functionalities:

1. Set up AWS Lambda functions to handle API requests from the UI.
2. Configure AWS API Gateway to manage and secure the APIs used in WebLingo.
3. Implement API endpoints for translation and word breakdown functionalities, integrating popular translation APIs and language resources.
4. Design error handling mechanisms to provide informative error messages in case of API failures.
5. Implement AWS CloudWatch to monitor and log Lambda function performance and API calls.
   * 1. **Flowchart Diagram for Implementation of AWS Lambda and API Gateway:**

****

**Figure 4.1 – Flow chart AWS Lambda and API Gateway**

## **4.2 Construction**

**4.2.1 Snippet Code of the System Logic:**

**CODE SNIPPET - CONTENT SCRIPT**

const go = () => {

chrome.runtime.onMessage.addListener(function (message, sender, sendResponse) {

if (message && message.action === 'displayTranslationResult') {

console.log('message received (cnt)');

console.log("Message from bckgd script: ", message.message);

const messageBck = message.message

createBox(messageBck)

}

});

}

chrome.runtime.sendMessage(

{

action: "createWebLingoContextMenu",

message: "switch off"

}

)

**...**

//this fucntions creates the display dialogue box

const createDialogue = () => {

const box = document.createElement('div')

box.style.boxSizing = `border-box`

box.style.padding = `1.5rem 1rem`

box.style.background = `white`

box.style.minWidth = `300px`

box.style.maxWidth = `300px`

box.style.maxHeight = `300px`

box.style.display = `flex`

box.style.justifyContent = `start`

box.style.alignItems = `flex-start`

box.style.flexDirection = `column`

box.style.overflow = `auto`

box.style.borderRadius = `10px`

box.style.boxShadow = `0 0 20px rgba(0, 0, 0, .9)`

box.style.position = 'fixed';

box.style.top = `10px`

box.style.right = `10px`

box.style.zIndex = `10000000000000000000000`

box.style.fontFamily = `'Montserrat', sans-serif`

return box

}

**CODE SNIPPET - BACKGROUND SCRIPT**

//main function for makling API calls and parsing API response

let fetchApiResp = async (url, method, data, parseType) => {

try {

const response = await fetch(url, {

method: method,

headers: {

“Content-Type”: “application/json”

},

body: JSON.stringify(data)

})

if (response.status != 200) {

throw new Error (“Network response was not ok”, response);

}

const parsedResp = await response.json()

const finalResult = parseType ? parsedResp.translation : parsedResp.lemma

return finalResult

} catch (error) {

return `Error: ${error}`

}

}

**. . .**

//main loop for making translation

if (info.menuItemId === 'quickTranslate' || info.parentMenuItemId === 'translateToTargetLang') {

**. . .**

const googletranslationURL = `https://46nykrhdzl.execute-api.eu-north-1.amazonaws.com/Dev/api/getgoogletranslation`

const mstranslationURL = `https://46nykrhdzl.execute-api.eu-north-1.amazonaws.com/Dev/api/getmstranslations`

const translationURL = (trans\_api === 'googleTranslation') ? googletranslationURL : mstranslationURL

const translation = await fetchApiResp(translationURL, method , translationData, "text")

const translationResultData = {

to: target\_lang,

translatedText: translation

}

const messageData = {

feature: 'translation',

data: translationResultData

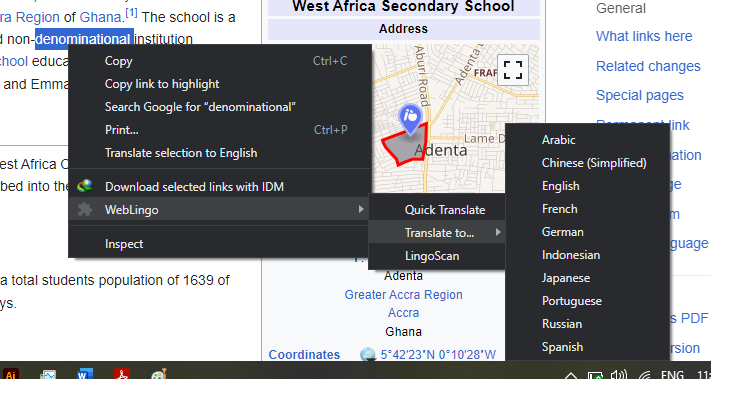
}

sendResp(tab, messageData)

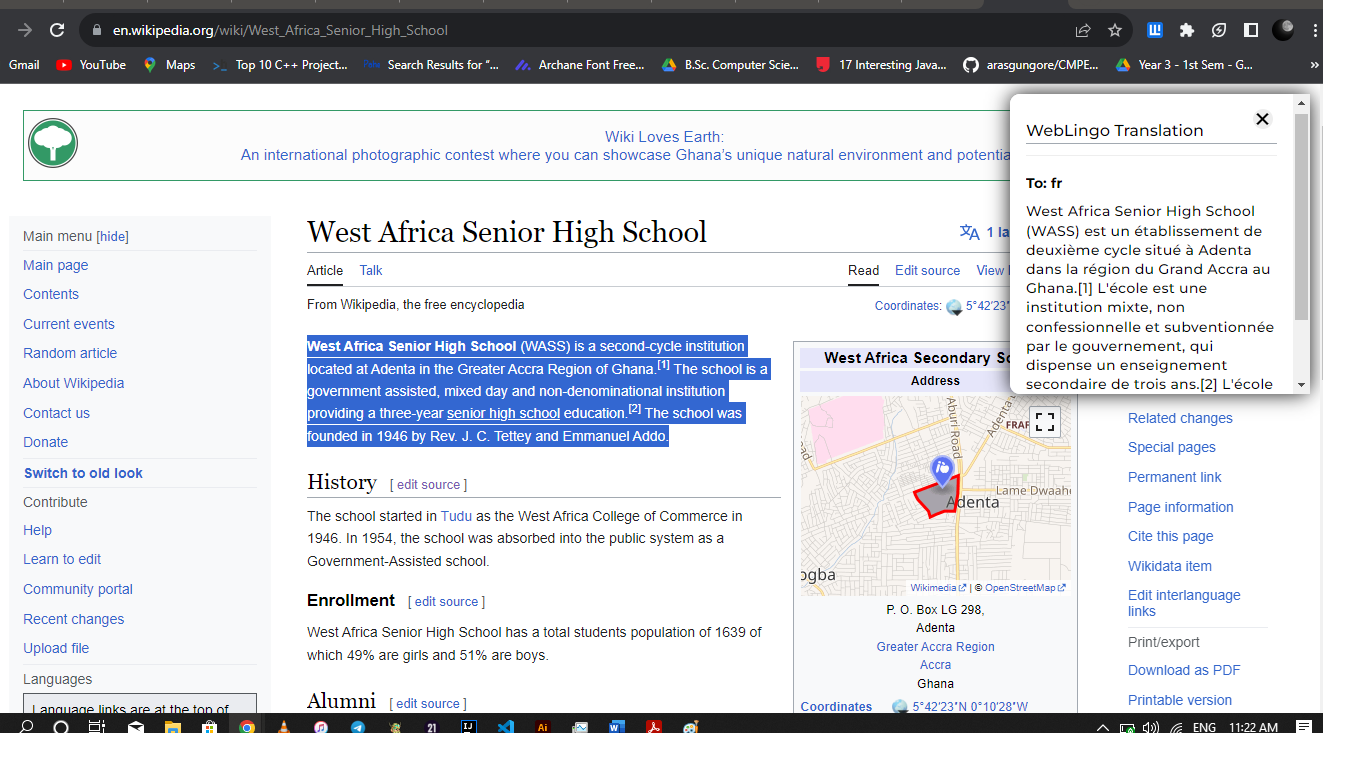
}

**SYSTEM’S COMPONENTS SCREENSHOT**

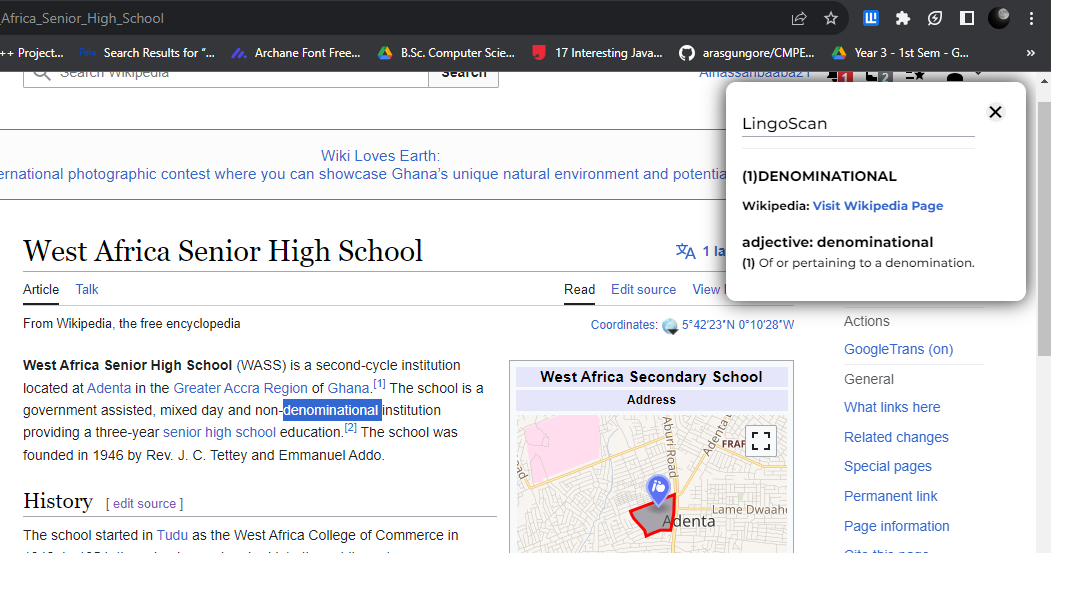
1. **Figure 4.1 - Context Menu**

****

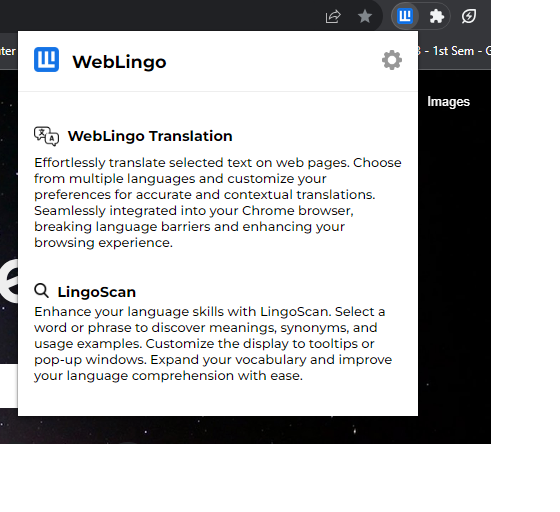
1. **Figure 4.2 - Display Dialogue Box (Translation)**

****

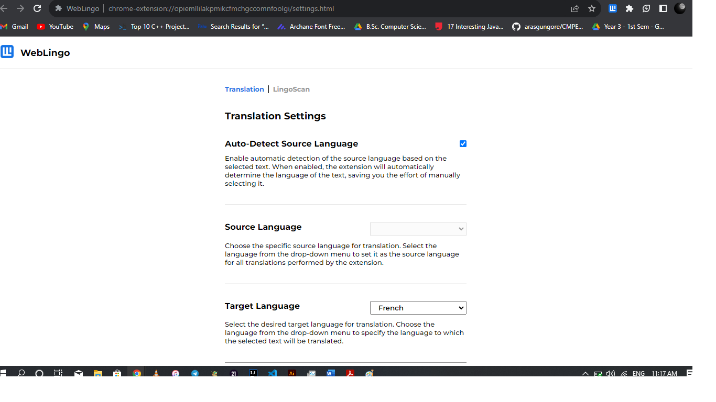
1. **Figure 4.3 - Display Dialogue Box (LingoScan)**

****

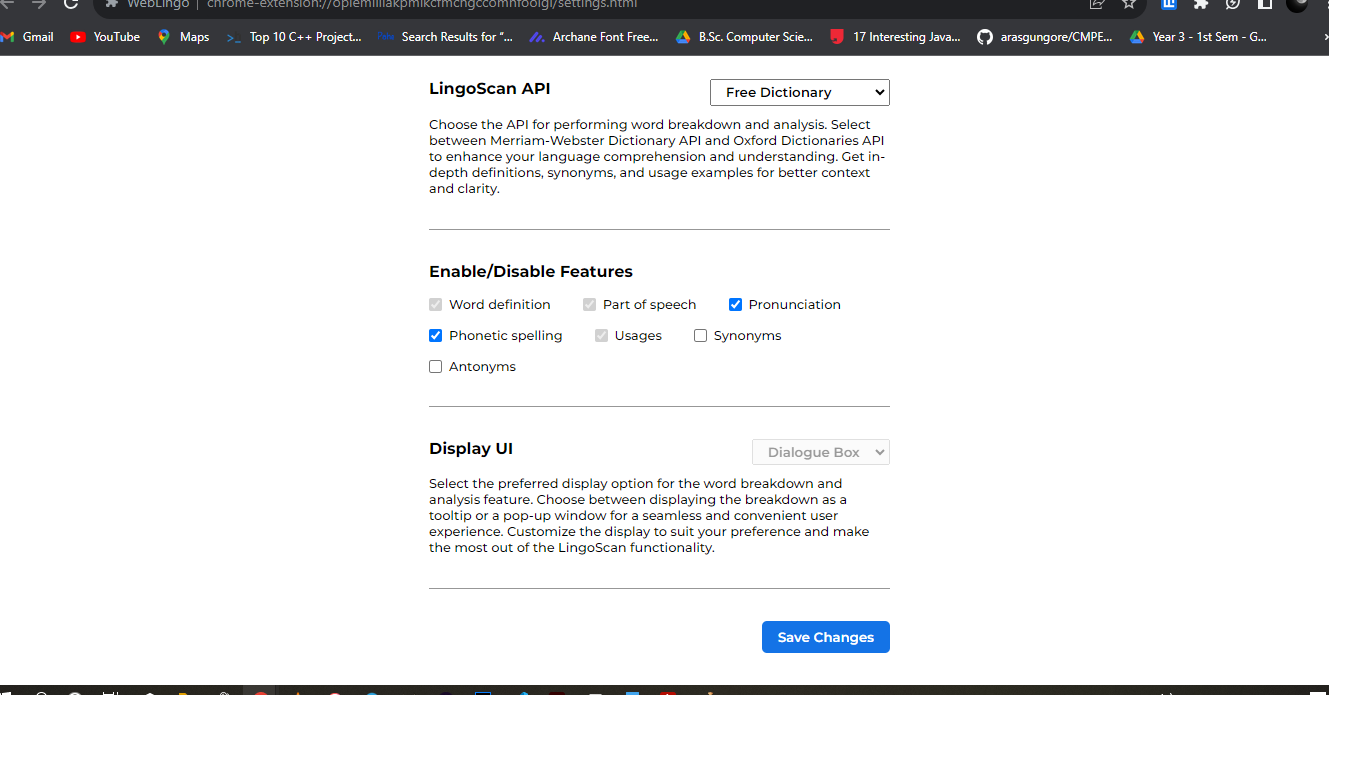
1. **Figure 4.4 - POP UP UI**

****

1. **Figure 4.5 - SETTING PAGE (Translation)**

****

**(6) Figure 4.6 - Settings Page (LingoScan)**

****

## **4.3 Testing**

**4.3.1 Testing Plan:**

WebLingo underwent rigorous testing to ensure its functionality, performance, and reliability. The testing plan encompassed two main areas: components testing and system testing.

**4.3.2 Components Testing Algorithms:**

**4.3.2.1 Algorithm for Testing UI:**

1. Test user interactions in the pop-up window and context menu, including mouse clicks and text selection.
2. Verify that language translation and word breakdown functionalities work as expected for different language selections.
3. Test user settings and ensure that customized preferences are correctly applied and stored in Chrome Storage.
4. Validate that the display options, such as tooltips and pop-up windows, are functioning correctly based on user selections.

**4.3.2.2 Algorithm for Testing Database (Chrome Storage):**

1. Test the storage and retrieval of user settings and preferences using Chrome Storage.
2. Verify that user settings persist across browsing sessions and are correctly updated when modified.

**4.3.3 System Testing Algorithms:**

**4.3.3.1 Algorithm for Verification Testing:**

1. Conduct functional testing to verify that all features and functionalities of WebLingo are working as intended.
2. Validate that language translation results are accurate and match the selected APIs' responses.
3. Verify that word breakdowns provide comprehensive information, including synonyms, antonyms, and usage examples.

**4.3.3.2 Algorithm for Validation Testing:**

1. Gather user feedback and conduct user testing to validate the usability and user-friendliness of WebLingo.
2. Collect feedback on the effectiveness of language translation and word breakdown features in enhancing language comprehension.
3. Analyze user feedback to identify areas for improvement and potential enhancements.

## **4.4 Results:**

The testing phase yielded positive results, showcasing the effectiveness of WebLingo in enhancing language accessibility and comprehension for users. The system demonstrated seamless translation and word breakdown functionalities, providing accurate and comprehensive language resources. User feedback confirmed the user-friendly nature of the extension, allowing users to overcome language barriers and customize their language enhancement experience.

# **Chapter 5: Findings and Conclusion**

## **5.1 Conclusions:**

WebLingo has successfully fulfilled its mission of enhancing language accessibility and comprehension for internet users. Through the seamless integration of powerful translation APIs and language resources, users can effortlessly translate selected text and gain in-depth word breakdowns, fostering effective cross-cultural communication and knowledge acquisition. The extension's user-friendly interface and non-intrusive integration with the browser's context menu have made language enhancement a natural and enjoyable part of users' web browsing experience. WebLingo's impact extends to students, researchers, business professionals, travelers, and immigrants, promoting global understanding and cooperation in the interconnected digital age. The success of WebLingo paves the way for further advancements in language technology, empowering users to navigate the multilingual landscape of the internet with ease and confidence.

## **Challenges/Limitations of the System:**

During the development of WebLingo, some challenges were encountered. One limitation is the inability to translate an entire webpage due to certain technical constraints. Additionally, the performance of selected translation APIs varied based on the language pairs, leading to occasional delays in translations. Further improvements in handling multilingual content and optimizing API performance would address these limitations.

## **Lessons Learned**

The WebLingo project served as a valuable learning experience, offering me insights into the intricacies of effective project planning and execution. Through careful definition of project objectives and a well-structured timeline, I ensured a systematic and organized development process. This approach facilitated seamless collaboration among team members, allowing me to work cohesively towards achieving project milestones.

Throughout the development journey, I recognized the importance of open communication and regular updates on progress. Emphasizing feedback and actively seeking suggestions from team members and stakeholders enabled me to identify areas of improvement and make iterative adjustments to enhance the extension's functionalities.

Resource management played a significant role in the project's success, as I allocated the appropriate tools and technologies to meet the extension's requirements. Leveraging JavaScript, HTML, CSS, Node.js, Express.js, and AWS Lambda allowed me to build a robust and efficient backend infrastructure.

In conclusion, the WebLingo project taught me the value of careful planning, effective communication, and collaborative development. By embracing feedback and continuously refining my approach, I successfully delivered a language enhancement extension that empowers users to overcome language barriers and embrace the diverse content available on the web. The lessons learned from this project will undoubtedly shape my future endeavors and reinforce my commitment to delivering innovative and impactful solutions in the digital landscape.

## **Recommendations for Future Works:**

In the quest to continually improve and enhance WebLingo, several recommendations for future works can be explored. Firstly, the integration of additional language resources, such as specialized dictionaries and domain-specific language databases, could expand the extension's capabilities and offer users more accurate and contextually relevant translations and word breakdowns.

Exploring advanced language processing technologies, such as natural language processing (NLP) and machine learning, could take WebLingo to the next level. These technologies can enable more sophisticated language understanding, allowing the extension to offer better contextual translations and word analysis. Moreover, leveraging machine learning algorithms could lead to improved language detection and automatic language preference settings, providing users with a more personalized language enhancement experience.

To cater to a wider audience, future works could focus on improving the handling of multimedia content for translation and comprehension. Implementing image and video recognition capabilities would enable users to extract and understand text from visual elements within webpages, breaking down language barriers even further.

As a step towards global inclusivity, expanding language support for less commonly spoken languages would be a valuable addition. Offering translation and word breakdown features for a diverse range of languages would make WebLingo more accessible and useful for users around the world.

Looking ahead, the vision for WebLingo includes the ability to translate entire webpages. This would involve extending the extension's functionalities to cover complete webpage translation, providing users with a comprehensive language enhancement solution for their browsing experience.

In summary, the future potential of WebLingo is vast, and ongoing efforts to incorporate new language resources, advanced technologies, and support for diverse languages will continue to make WebLingo a powerful language enhancement tool. With the goal of revolutionizing the way users navigate and comprehend content online, WebLingo will remain a catalyst for promoting effective communication and understanding in an increasingly interconnected world.

## **Recommendations for Project Commercialization:**

For project commercialization, strategic marketing and outreach efforts can be employed to promote WebLingo to a broader audience. Collaborating with educational institutions, language learning platforms, and businesses could facilitate widespread adoption. Offering premium features and subscription models could also be considered to generate revenue and sustain future development.

## **Reference List:**

1. Extension Pop-up Design Inspiration (Adobe Acrobat):

Adobe. (n.d.). Create a custom pop-up in Acrobat DC. <https://helpx.adobe.com/acrobat/using/creating-custom-pop-ups-acrobat.html>

1. Google Cloud Translation Documentation:

Google Cloud. (n.d.). Google Cloud Translation Documentation. <https://cloud.google.com/translate/docs>

1. Microsoft Translation Documentation (MDN Mozilla):

MDN Web Docs. (n.d.). Web APIs. Microsoft Translator. <https://developer.mozilla.org/en-US/docs/Web/API/Microsoft_Translator>

1. Webster API Documentation (Dictionary API):

DictionaryAPI.dev. (n.d.). Webster API Documentation. https://merriam-webster.com

1. Free Dictionary API Documentation (DictionaryAPI.dev):

DictionaryAPI.dev. (n.d.). Free Dictionary API Documentation. <https://dictionaryapi.dev/>

1. AWS Lambda:

Amazon Web Services. (n.d.). AWS Lambda. <https://aws.amazon.com/lambda/>

1. AWS API Gateway:

Amazon Web Services. (n.d.). Amazon API Gateway. <https://aws.amazon.com/api-gateway/>