TAMIL WORD SENTIMENT ANALYSIS

A PROJECT REPORT SUBMITTED TO

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

AWARD OF THE DEGREE OF

MASTER OF COMPUTER APPLICATIONS

BY

VISHALI N (REG NO. RA2332014010118)

UNDER THE GUIDANCE OF

DR R THILAGAVATHI M.C.A., M.Phil.,

DEPARTMENT OF COMPUTER APPLICATIONS

FACULTY OF SCIENCE AND HUMANITIES

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

Kattankulathur – 603 203

Chennai, Tamilnadu

OCTOBER - 2024

BONAFIDE CERTIFICATE

This is to certify that the project report titled "**TAMIL WORD SENTIMENT ANALYSIS**" is a bonafide work carried out by **VISHALI N** (RA2332014010118) under my supervision for the award of the Degree of Bachelor of Computer Applications. To my knowledge the work reported herein is the original work done by these students.

Dr.R. THILAGAVATHI M.C.A., M.Phil.,	**Dr. R.Jayashree**
Assistant Professor,	Associate Professor & Head,
Department of Computer Applications	Department of Computer Applications
(GUIDE)	

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

With profound gratitude to the ALMIGHTY, I take this chance to thank the people who helped me to complete this project.

We take this as a right opportunity to say THANKS to my parents who are there to stand with me always with the words “YOU CAN”.

We are thankful to **Dr. T. R. Paarivendhar**, Chancellor, and **Prof. A. Vinay Kumar**, Pro Vice-Chancellor (SBL), SRM Institute of Science & Technology, who gave us the platform to establish me to reach greater heights.

We earnestly thank **Dr. A. Duraisamy**, Dean, Faculty of Science and Humanities, SRM Institute of Science & Technology, who always encourage us to do novel things.

A great note of gratitude to **Dr. S. Albert Antony Raj**, Deputy Dean, Faculty of Science and Humanities for his valuable guidance and constant Support to do this Project.

We express our sincere thanks to **Dr. R. Jayashree**, Associate Professor & Head, for her support to execute all incline in learning.

It is our delight to thank our project guide **Dr.R. Thilagavathi M.C.A., M.Phil.,**

Assistant Professor, Department of Computer Applications, for his help, support, encouragement, suggestions, and guidance throughout the development phases of the project.

We convey our gratitude to all the faculty members of the department who extended their support through valuable comments and suggestions during the reviews.

Our gratitude to friends and people who are known and unknown to me who helped in carrying out this project work a successful one.

VISHALI N

TABLE OF CONTENT

LIST OF FIGURES

ABSTRACT

The **Tamil Word Sentiment Analysis** project is a web-based application designed to evaluate the sentiment of Tamil text inputs. Leveraging the Flask framework, this application provides users with a straightforward interface to enter Tamil words or phrases and receive immediate feedback on their emotional tone—classified as positive, negative, or neutral. By utilizing a predefined list of Tamil words associated with specific sentiments, the application employs a simple algorithm that facilitates efficient analysis without requiring complex machine learning techniques. This tool aims to empower users, including businesses, educators, and social media analysts, by offering insights into public sentiment and emotional dynamics in Tamil language communications. In addition to fostering better engagement with the Tamil language, the project highlights the significance of developing accessible tools for regional languages, promoting inclusivity in technology. Future enhancements may incorporate advanced features such as machine learning models to improve sentiment analysis accuracy, thereby extending the application's capabilities and usability. Overall, this project represents a meaningful step towards bridging the gap in language processing tools for Tamil speakers, making sentiment analysis more accessible and relevant.

1. INTRODUCTION

1.1 Overview of the Project

The **Tamil Word Sentiment Analysis** project is a web application designed to evaluate the sentiment of Tamil words and phrases. Built using the Flask framework, HTML, and CSS, this project aims to make sentiment analysis accessible to Tamil speakers and provide a valuable tool for various applications, including social media monitoring, customer feedback evaluation, and language learning. The core functionality of the application relies on a predefined list of positive and negative Tamil words, enabling it to categorize input text as positive, negative, or neutral.

By employing a straightforward approach to sentiment analysis, this application offers users a quick and efficient way to gauge the emotional tone of their input. It addresses the lack of dedicated sentiment analysis tools for Tamil, which has been a barrier for many individuals and organizations seeking to analyze text in this language. The project demonstrates how simple programming techniques can facilitate language processing tasks, allowing users to engage with technology in their native language.

1.2 Purpose and Scope of the Application

The primary purpose of this application is to provide a user-friendly platform for analyzing sentiment in Tamil text. It caters to individuals, educators, and businesses who need insights into the emotional content of written Tamil. The application serves as an essential tool for understanding public sentiment, particularly in contexts like social media, where gauging emotional reactions can be crucial for brands and influencers.

In terms of scope, the application is designed to handle basic sentiment analysis tasks, focusing specifically on Tamil language inputs. It does not require complex machine learning algorithms, making it lightweight and easy to deploy. The tool aims to empower users by providing them with immediate feedback on their text, fostering a better understanding of language sentiment dynamics. Future enhancements may include the incorporation of advanced analytical features, but the current implementation emphasizes simplicity and accessibility for all users.

2. SOFTWARE REQUIREMENT ANALYSIS

This section provides an overview of the hardware and software resources required to develop and run the **Tamil Word Sentiment Analysis** application.

2.1 Hardware Specification

The project does not demand high-end hardware due to its simple architecture. The recommended hardware includes:

- **Processor**: An Intel Core i3 or equivalent processor to handle the basic processing tasks of the application.
- **Memory (RAM)**: At least 4 GB of RAM to ensure smooth development and testing, especially when running the Flask server locally.
- **Storage**: A minimum of 500 MB of disk space is necessary for storing source code, libraries, and dependencies.

application serves as a vital tool for understanding sentiment in Tamil communications, empowering users to gain insights into emotional expressions in their language.

These hardware specifications allow efficient local development and ensure that the application runs smoothly without performance issues.

2.2 Software Specification

To build and execute this web-based sentiment analysis tool, certain software tools and technologies are required. These include:

- **Operating System**: Compatible with Windows, macOS, or Linux platforms.
- **Programming Language**: Python is the primary language used for the back-end, which provides easy integration with libraries required for web development and text analysis.
- **Web Framework**: Flask is a lightweight Python-based framework chosen for building the web application. It supports simple routing and template rendering, making it ideal for small to medium-sized applications.

Frontend Technologies:

- **HTML**: Used for the structure of the web page.
- **CSS**: Used for styling the web page and improving the user interface.
- **Bootstrap**: This CSS framework ensures the web page is responsive, meaning it works well on different screen sizes and devices (desktops, tablets, and mobile phones)

Other Python Libraries:

- **Jinja2**: A template engine integrated into Flask that renders dynamic HTML content.
- **Werkzeug**: A WSGI utility library used in Flask for handling request and response processing.

2.3 About the Software and Its Features

The software is designed to perform sentiment analysis on Tamil text input using a predefined set of positive and negative words. The core features of the system include:

The sentiment analysis application is crafted using Flask, a flexible and easy-to-use web framework for Python, making it ideal for building web applications. Developed in Visual Studio Code (VS Code), the environment offers powerful coding and debugging tools, enhancing productivity.

The user interface is designed with HTML and CSS, ensuring a clean and intuitive experience for users as they input Tamil text for analysis. This interface facilitates straightforward interaction, allowing users to receive instant feedback on sentiment classification.

At the core of the application lies a main Python file that implements natural language processing (NLP) techniques. It analyzes the sentiment of the input text by referencing a predefined list of positive and negative words. This enables accurate categorization of sentiments as positive, negative, or neutral.

Data interchange between the frontend and backend is handled using JSON files, ensuring efficient communication and response handling. This combination of technologies results in a robust sentiment analysis tool that effectively interprets the emotional tone of Tamil text, providing valuable insights for users.

Flask Application

The Flask application serves as the backbone of the project, managing the web interface and handling user requests. It routes incoming user inputs to the server, processes them, and then returns the corresponding results. Through a straightforward form, users can submit their Tamil text, which is subsequently analyzed by the sentiment analysis function.

Sentiment Analysis Logic

This component is responsible for analyzing the sentiment of the user’s input. It decomposes the text into individual words and matches them against a curated list of positive and negative words specific to Tamil. Based on the frequency of these words, the system categorizes the overall sentiment as "Positive," "Negative," or "Neutral," providing meaningful insights into the emotional tone of the text.

User Interface

The user interface is designed with simplicity in mind, allowing users to easily enter Tamil text and view the results of the sentiment analysis. The layout is intuitive, ensuring that users can navigate the application effortlessly and understand the feedback provided, enhancing overall user experience.

Responsiveness

Utilizing Bootstrap for design, the web application is responsive and adaptable to various screen sizes. This feature ensures that users have an optimal viewing experience, whether they are accessing the application on smartphones, tablets, or desktop computers, making it accessible to a wide audience.

3.SYSTEM ANALYSIS

3.1 Existing System

Currently, there are limited tools specifically designed for sentiment analysis in the Tamil language. Most existing systems are either generic or focused on other languages, lacking the necessary linguistic context for accurate analysis. Users often rely on basic text-processing methods that do not adequately capture the nuances of sentiment in Tamil. This results in unreliable classifications and a lack of resources tailored to the unique characteristics of the Tamil language, hindering effective sentiment evaluation.

Existing System Diagram Explanation

The existing system for a Twitter Sentiment Analysis Web Application using Flask begins with the user entering text into a web form, which serves as the primary input. This input is then directed to the backend server managed by Flask, which acts as the central processing unit, ensuring data flows smoothly between the user interface and the sentiment analysis model. Within the model, the text undergoes analysis and is categorized as positive, negative, or neutral, depending on its content. Some systems may also include a database to log both the inputs and their sentiment outcomes, creating a record that can be used for future model improvements or insights. Finally, the analysis result is displayed back to the user on the web page, providing a straightforward and interactive sentiment evaluation process.

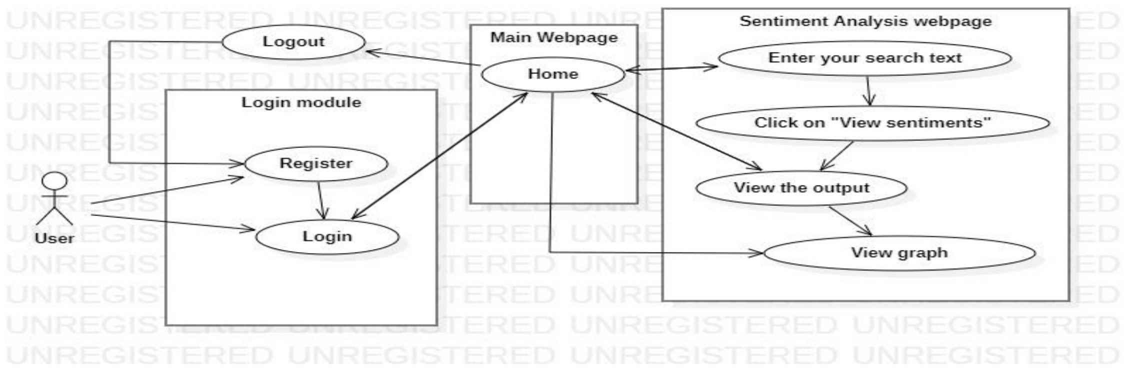


Fig3.1.Existing System

EXISTING SYSTEM OF A TWITTER SENTIMENT ANALYSIS WEB APP USING FLASK

Components:

- User -> Web Form -> Backend Server (Flask or any other API) -> Sentiment Analysis Model (Existing) -> Sentiment Result (Display on web page)

Twitter Sentiment Analysis Web App using Flask

This diagram would represent the current system you are working with, possibly based on a similar sentiment analysis system (if that's what you have). It could involve:

- **User Input**: Text entered by the user.
- **Backend Processing**: Use of existing sentiment analysis algorithms or a framework like Tweepy (Twitter API) for sentiment analysis.
- **Database Interaction**: Where user input is stored and results are logged.
- **Output**: Sentiment result (positive, negative, neutral) is displayed to the user

3.2 Proposed System

The proposed system aims to address the gap in sentiment analysis tools for the Tamil language by developing a dedicated web application capable of accurately categorizing Tamil text based on a predefined list of positive and negative words. Users can input Tamil text into a user-friendly interface, which then processes the input to determine its sentiment—whether positive, negative, or neutral. This solution leverages natural language processing to tokenize the input and compare each word against curated lists of positive and negative words in Tamil, enabling a quick and reliable sentiment assessment. The integration of Flask as a backend framework streamlines the process, handling user input and routing data through sentiment analysis functions to deliver clear feedback.

This system is structured to support various use cases for individuals, researchers, and businesses aiming to understand the emotional tone of Tamil-language content, offering a valuable tool for interpreting user sentiment in regional contexts. The application workflow is straightforward: user inputs text, which is routed through the Flask application to the backend sentiment analysis function, processed by matching tokens against sentiment word lists, and finally outputs the sentiment result to the user on the webpage. By combining practical sentiment analysis with a web-based interface, the system contributes to the development of digital resources for Tamil and supports further advancements in regional linguistic technology.

This diagram will depict the new system you are implementing specifically for Tamil text sentiment analysis using Flask and a list of predefined words.

Proposed System Diagram Explanation

The proposed system for Tamil sentiment analysis uses a web application designed with Flask. Users input Tamil text through a web form, which is then sent to the Flask backend for processing. Flask manages the routing and directs the input to a sentiment analysis function. This function uses predefined lists of Tamil positive and negative words to tokenize and analyze the text's sentiment. Based on this comparison, the text is classified as positive, negative, or neutral. The result is then displayed on the web page, providing users with immediate sentiment feedback.

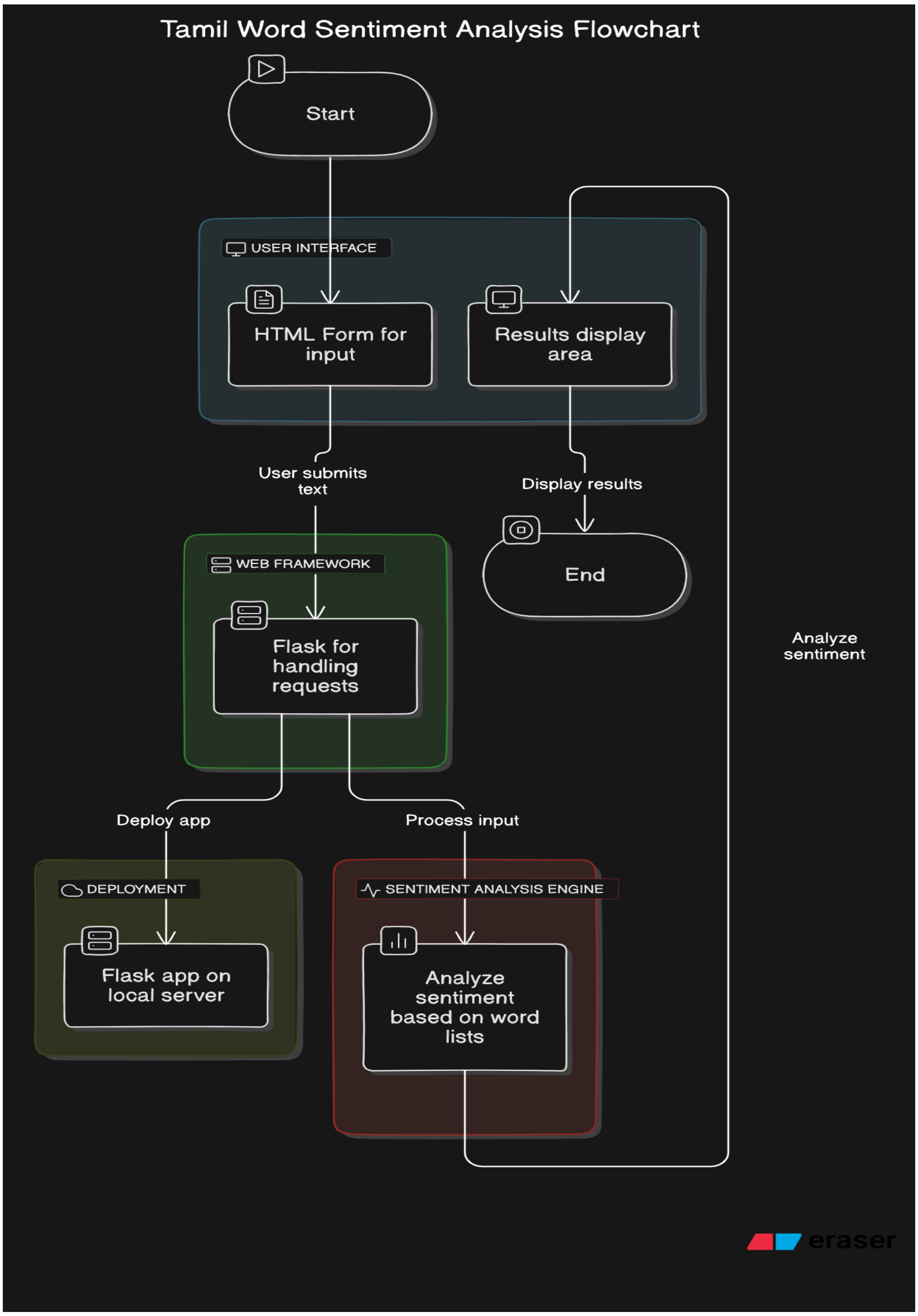


Fig3.2.Proposed System

PROPOSED SYSTEM OF A TAMIL WORD SENTIMENT ANALYSIS

Diagram Flow:

- User -> Web Form -> Flask Backend (Sentiment Analysis Function) -> Sentiment Classifier (Tamil words) -> Sentiment Result -> Web Page Display

Components:

1. **User Input**: Tamil text is input through the web interface.
2. **Flask Web Application**:
   - Handles routing (`GET`, `POST` requests) and serves HTML forms.
   - Passes the input text to the backend for processing.
3. **Sentiment Analyzer (Tamil)**:
   - Python logic using predefined Tamil positive and negative word lists.
   - Tokenizes the input and compares it against the word lists.
4. **Output**: Sentiment result (Positive, Negative, Neutral) is returned and displayed on the web page.

3.3 Feasibility Study

3.3.1 Technical Feasibility

The technical feasibility of the **Tamil Word Sentiment Analysis** application is strong, as it is built on the widely-used Flask framework, which supports rapid development and deployment. The application utilizes basic algorithms for sentiment classification, which do not require extensive computational resources. Additionally, the reliance on predefined word lists simplifies the implementation process. The software can be hosted on standard web servers, making it accessible to users without complex setup requirements. Overall, the technical infrastructure is readily available, ensuring successful execution of the project.

3.3.2 Economic Feasibility

From an economic perspective, the project is considered viable due to its low initial development and operational costs. Utilizing open-source technologies like Flask and HTML reduces software licensing expenses. Moreover, the application's simplicity means that fewer resources are needed for maintenance and updates. Potential revenue streams could include offering premium features or targeted advertising for businesses interested in sentiment analysis tools. This economic model supports the long-term sustainability of the project while providing value to users.

3.3.3 Operational Feasibility

The operational feasibility of the application is promising, as it meets the needs of its target users effectively. The user-friendly interface simplifies interaction, allowing individuals with limited technical knowledge to utilize the sentiment analysis tool. Additionally, the clear guidance provided within the application enhances user experience and engagement. Training sessions or tutorials can further assist users in understanding the functionality. Overall, the project aligns well with user expectations and offers practical solutions for sentiment analysis in the Tamil language.

4. SYSTEM DESIGN

4.1 Data Flow Diagram

Processes:

1. **Receive User Input**
   - Description: Capture text input from the user via the web interface.
   - Input: Text from the user.
   - Output: Sentiment analysis request.
2. **Analyze Sentiment**
   - Description: Process the input text to determine sentiment.
   - Input: User text.
   - Output: Sentiment result (Positive, Negative, Neutral).
3. **Display Results**
   - Description: Show the sentiment analysis result to the user.
   - Input: Sentiment result.
   - Output: Rendered HTML page with results.

Data Stores:

1. **Positive Words List**
   - Description: A list of Tamil words indicating positive sentiment.
   - Data Type: Array of strings.
2. **Negative Words List**
   - Description: A list of Tamil words indicating negative sentiment.
   - Data Type: Array of strings.

External Entities:

1. **User**
   - Description: Individual interacting with the application.
   - Input: Text input for analysis.
   - Output: Sentiment results.

Data Flows:

- **From User to Receive User Input**: User submits text.
- **From Receive User Input to Analyze Sentiment**: Sentiment analysis request.
- **From Analyze Sentiment to Display Results**: Sentiment result.
- **From Display Results to User**: Rendered result page.
- **From Analyze Sentiment to Positive Words List**: Access positive words.
- **From Analyze Sentiment to Negative Words List**: Access negative words.

Data Flows: - User → Receive User Input: User submits text - Receive User Input → Analyze Sentiment: Sentiment analysis request - Analyze Sentiment → Display Results: Sentiment result - Display Results → User: Rendered result page - Analyze Sentiment ↔ Positive Words List: Access positive words - Analyze Sentiment ↔ Negative Words List: Access negative words

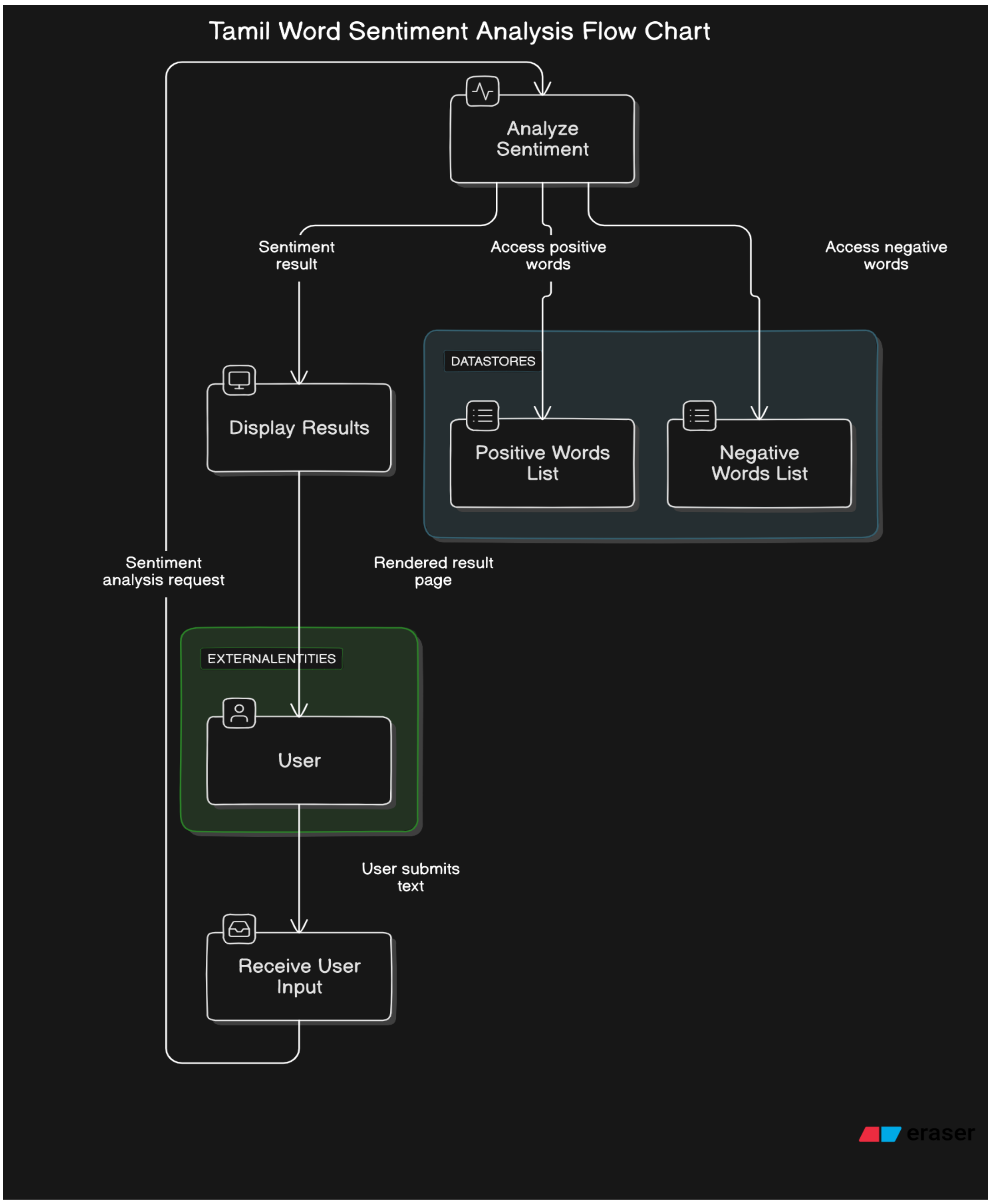


Fig4.1.System Design

SYSTEM DESIGN

4.2 UML Diagrams

4.2.1 Use Case Diagram

1. **User**
    - Description: The individual interacting with the application to analyze sentiment.

Use Cases:

1. **Input Text**
    - Description: User enters Tamil text into the form.
    - Preconditions: User is on the input form page.
    - Postconditions: User text is captured for analysis.
2. **Analyze Sentiment**
    - Description: System processes the input text to determine sentiment.
    - Preconditions: User has submitted text.
    - Postconditions: Sentiment result is generated.
3. **Display Result**
    - Description: System displays the sentiment result to the user.
    - Preconditions: Sentiment analysis is complete.
    - Postconditions: User sees the result on the webpage.

Relationships:

- User → Input Text
- User → Analyze Sentiment
- User → Display Result

Relationships: - User → Input Text - User → Analyze Sentiment - User → Display Result

Use Case Diagram Explanation

The use case diagram for the Tamil sentiment analysis application outlines the user interactions within the system. The user starts by entering Tamil text into an input form, initiating the analysis process. Once the text is submitted, the system processes it to identify the sentiment, determining if it is positive, negative, or neutral.

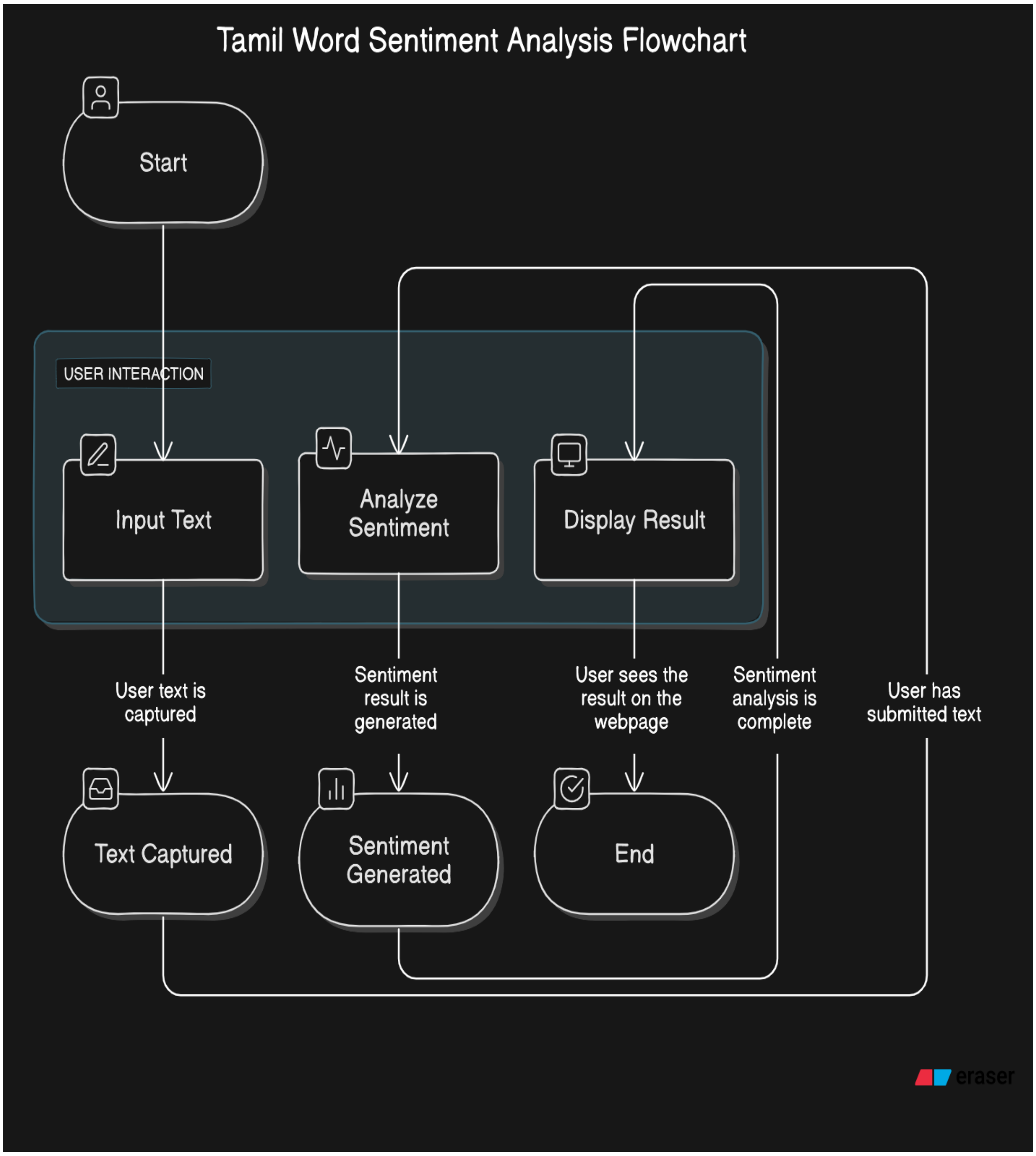


Fig4.2.1.Use Case Diagram

Use Case Diagram of a Tamil Word Sentiment Analysis

4.2.2 Sequence Diagram

Actors:

1. **User**
2. **Flask Application** (Server)
3. **Sentiment Analysis Function**

Sequence of Events:

1. **User Inputs Text**:
   - User enters Tamil text in the form and submits it.
2. **Send Request to Flask**:
   - The form data is sent to the Flask application.
3. **Flask Receives Input**:
   - Flask captures the input text.
4. **Invoke Sentiment Analysis**:
   - Flask calls the sentiment analysis function with the user input.
5. **Analyze Sentiment**:
   - The sentiment analysis function processes the text, checking against positive and negative word lists.
6. **Return Sentiment Result**:
   - The function returns the sentiment result (Positive, Negative, Neutral) to Flask.
7. **Display Result to User**:
   - Flask renders the result on the webpage for the user to see.

Sequence of Events: 1. User → Flask: Inputs and submits Tamil text. 2. Flask → Sentiment Analysis Function: Sends user input for analysis. 3. Sentiment Analysis Function → Flask: Returns sentiment result. 4. Flask → User: Displays sentiment result on the webpage.

Sequence Diagram Explanation

Flask captures this input and forwards it to the sentiment analysis function for processing. The function analyzes the text, determines the sentiment, and sends the result back to Flask. Finally, Flask displays the sentiment result on the webpage for the user to view.

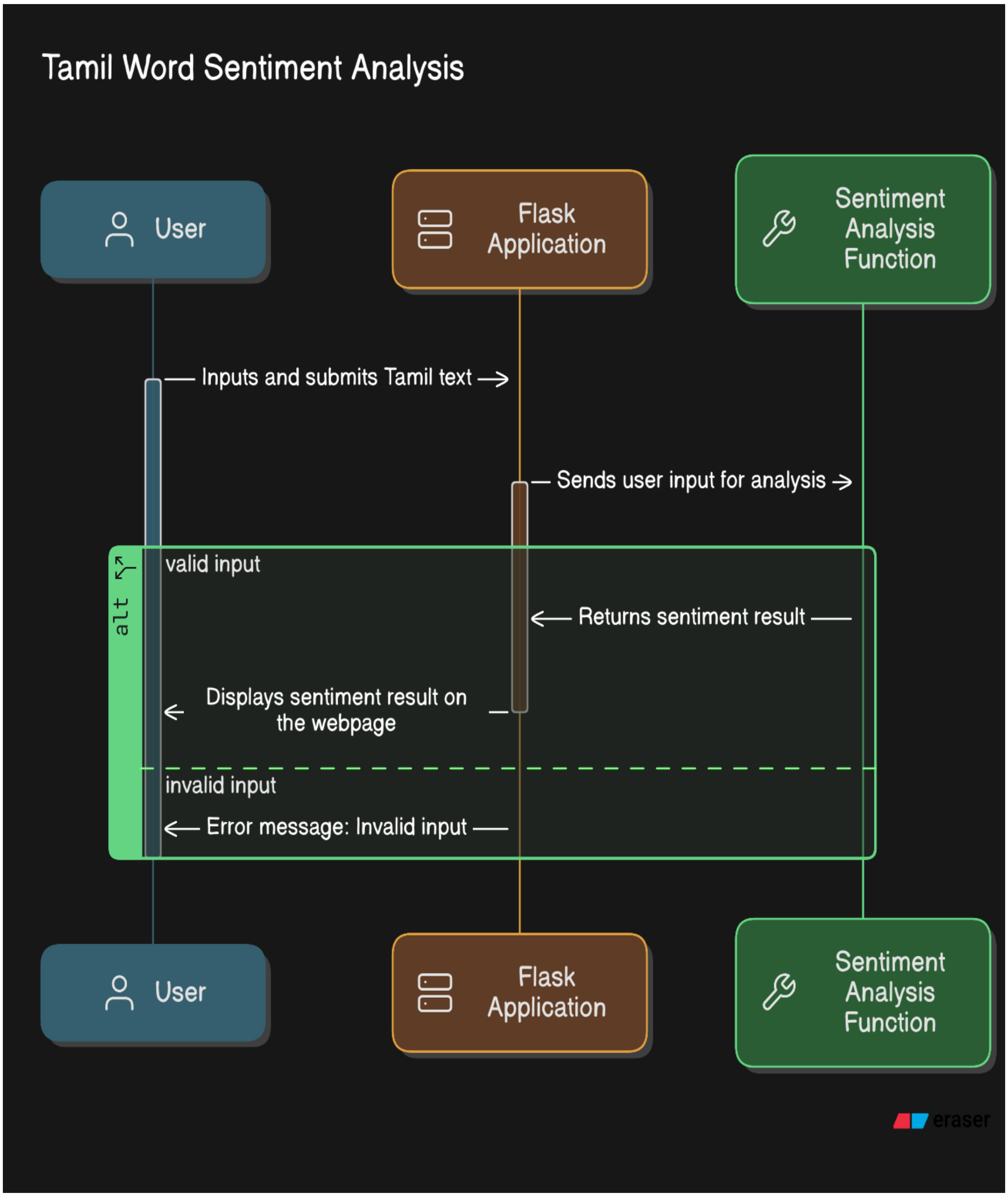

Tamil Word Sentiment Analysis
User
Flask Application
Sentiment Analysis Function
Inputs and submits Tamil text
Sends user input for analysis
alt
valid input
Returns sentiment result
Displays sentiment result on the webpage
invalid input
Error message: Invalid input
User
Flask Application
Sentiment Analysis Function
eraser

Fig4.2.2.Sequence Diagram

Sequence Diagram of a Tamil Word Sentiment Analysis

4.2.3 Class Diagram

Classes:

1. **User**
   - **Attributes**:
     - `user_id`: String
     - `input_text`: String
     - `sentiment_result`: String
   - **Methods**:
     - `enterText(text: String)`: void
     - `submitText()`: void
     - `viewResult()`: String
2. **SentimentAnalyzer**
   - **Attributes**:
     - `positive_words`: List<String>
     - `negative_words`: List<String>
   - **Methods**:
     - `analyze(text: String): String`: returns sentiment (Positive, Negative, Neutral)
     - `loadWords()`: void
3. **WebApp (Flask)**
   - **Attributes**:
     - `host`: String
     - `port`: Integer
   - **Methods**:
     - `renderForm()`: void
     - `handleSubmission(user_input: String)`: void
     - `displayResult(result: String)`: void

Relationships:

- **User** interacts with **WebApp** (composition).
- **WebApp** uses **SentimentAnalyzer** (dependency).
- **User** may interact with **Database** (optional for persistence).

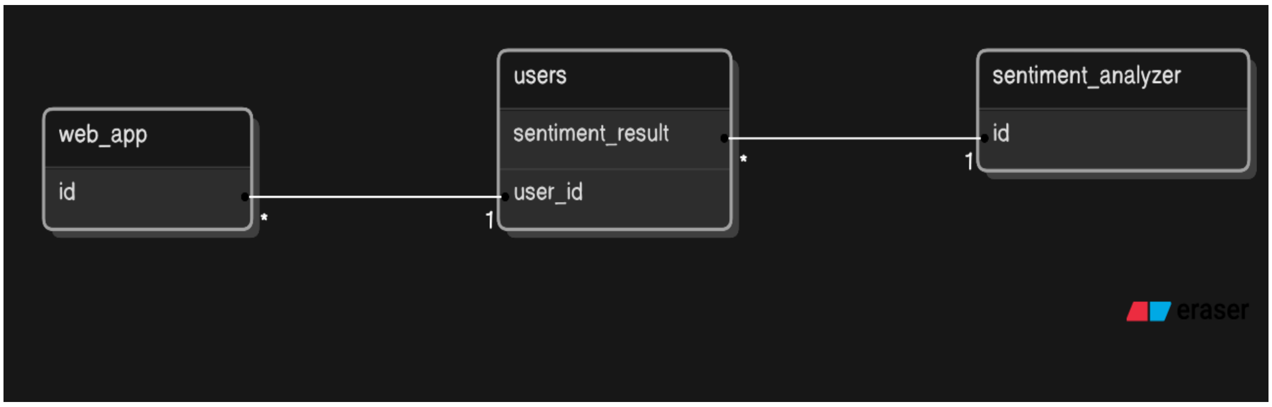


Fig4.2.3.Class Diagram

4.3 Database Design

The Tamil Word Sentiment Analysis project is structured without a database, as it relies on a static set of predefined Tamil words, categorized into positive and negative sentiments. These word lists are embedded within the code and are referenced during the sentiment analysis process when users input text. Since the sentiment classification is done in real-time, the system does not need to store or retrieve data, and thus, there is no requirement for persistent data storage.

Despite the simplicity of the current design, adding a database could enhance the system's scalability and flexibility. For instance, a database could be used to store user inputs and the corresponding sentiment results, allowing for future retrieval and analysis. This would be particularly useful in cases where user data needs to be logged, analyzed over time, or used to generate insights on trends in sentiment. Additionally, a database could allow for updates and management of the word lists, making it easier to adapt the tool as new sentiment-bearing words emerge.

If a database were to be integrated, options like MySQL or PostgreSQL (for relational database systems) could be considered for structured data storage. Alternatively, a NoSQL database such as MongoDB might be more appropriate if the application needs to handle large volumes of unstructured data, such as user-generated content. The database could be connected to the Flask application through an ORM (Object-Relational Mapping) tool like SQLAlchemy, which would simplify data manipulation and querying.

Incorporating a database would also open up possibilities for future enhancements, such as tracking user behavior, adding machine learning models for improved sentiment analysis, or enabling personalized analysis based on stored user data. This would make the system more robust, offering not only real-time sentiment evaluation but also historical insights, trend analysis, and enhanced user experiences

5. CODE TEMPLATES

5.1 Module Description

User Interface Module

This module serves as the front-end of the application, facilitating user interactions. It provides a user-friendly interface where individuals can input Tamil text for analysis. The design is aimed at ensuring ease of use, allowing users to seamlessly enter their text and navigate through the application while receiving real-time feedback on their submissions.

Sentiment Analysis Module

The heart of the application, this module is responsible for processing the user-inputted text and determining its emotional tone. It employs a set of predefined positive and negative words in Tamil to evaluate the sentiment of the text. By comparing the input against these word lists, the module classifies the sentiment into categories such as Positive, Negative, or Neutral, providing a foundational analysis for user feedback.

Feedback Generation Module

Once the sentiment has been analyzed, this module takes charge of displaying the results back to the user. It presents the sentiment classification clearly alongside the original input text, enabling users to understand the analysis. The feedback is designed to be informative, helping users interpret the sentiment of their text effectively.

Error Handling Module

This module ensures robust application performance by managing any errors or exceptions that may arise during user interactions. It validates inputs to catch any invalid entries and provides meaningful feedback to users. By handling errors gracefully, this module enhances the user experience, allowing for a smoother interaction with the application even when issues occur.

5.2 Tables

In the current implementation of the Tamil Word Sentiment Analysis project, there are no database tables utilized, as the system operates without a database. However, the project does incorporate predefined lists of Tamil words categorized by sentiment, which function similarly to lookup tables within the code.

These predefined lists, often referred to as lexicons, consist of Tamil words associated with positive and negative sentiments. They are integral to the sentiment analysis process, enabling the system to evaluate the sentiment of input text by comparing words against these lists. The lexicons are typically structured as simple data structures, such as dictionaries or lists, within the Python code.

For example, a lexicon may be represented as a dictionary where each key is a sentiment category (e.g., 'positive' or 'negative'), and the corresponding value is a list of words associated with that sentiment:

```
sentiment_lexicon = {

    'positive': ['சிறந்த', 'அருமை', 'மிகவும் நல்லது'],

    'negative': ['மோசமான', 'அருவருப்பு', 'மிகவும் ,”கெட்டது']

}
```

These lexicons are curated based on linguistic resources and are crucial for the accurate assessment of sentiment in Tamil text. While not stored in a traditional database table, they effectively serve as in-memory tables that guide the sentiment analysis process.

In future iterations of the project, transitioning these lexicons to a database format could enhance manageability, especially as the lists expand. This would allow for more dynamic updates and efficient handling of large datasets. For instance, using a relational database, one could create tables to store words along with their associated sentiments, facilitating more complex queries and analyses.

5.3 Source Code

The source code for the sentiment analysis application is structured into multiple key components, each playing a vital role in the overall functionality and performance of the system.

1. **Main Python File**: This file serves as the heart of the application, encapsulating the core sentiment analysis logic. It begins by importing necessary libraries, including Flask for web handling and other NLP libraries for text processing. The main function processes user input from the web form, converting the Tamil text into a list of words. It then compares these words against predefined lists of positive and negative sentiment words. Based on the frequency of matches, the application classifies the input sentiment as "Positive," "Negative," or "Neutral." Additionally, the code includes error handling to manage invalid inputs gracefully, ensuring a smooth user experience. This modular design allows for easy updates to the sentiment analysis logic, making it adaptable to future improvements or expansions.

Source Code For main python file:

```
from flask import Flask, request, render_template


app = Flask(__name__)


# Define the sentiment analysis function

def analyze_sentiment(text):

    # Define positive and negative words in Tamil

    positive_words = ['அருமை', 'வெற்றி', 'மிகவும்',
'அதிர்ஷ்டமான','சிறப்பு','நன்மை','நலம்தர','பேரின்பம்','மிகச்சிறப்பு',
'மகிழ்ச்சி','வணக்கம்','நன்றி','நல்லா','நம்பிக்கை','புகழ்','உற்சாகம்',",'
',",",",",",",",","]

    negative_words = ['அசிங்கமான',
'மோசமான','மோசம்','கெட்ட','எரிச்சல்','தீய','தீங்கு',",",",",",",",",","]

    # Tokenize the text into words

    words = text.split()


    # Initialize sentiment scores
```

```
        positive_score = 0
        negative_score = 0
       # Analyze sentiment based on positive and negative words
        for word in words:
            if word in positive_words:
                positive_score += 1
            elif word in negative_words:
                negative_score += 1



        # Determine sentiment
        if positive_score > negative_score:
            sentiment = 'Positive'
        elif positive_score < negative_score:
            sentiment = 'Negative'
        else:
            sentiment = 'Neutral'
return sentiment
# Home route to display the form
    @app.route('/')
    def my_form():
        return render_template('form.html')


    # Post route to process the user input
    @app.route('/', methods=['POST'])
```

```
def my_form_post():
    # Get user input from the form
    text1 = request.form['text1'].lower()


    # Analyze sentiment of the user input
    sentiment = analyze_sentiment(text1)


    # Render the result in the HTML template
    return render_template('form.html', final=sentiment, text1=text1)


if __name__ == "__main__":
    app.run(debug=True, host="127.0.0.1", port=8080)
```

2. **Flask Framework**: Utilizing Flask, the application efficiently manages routing and server-side logic. The code defines specific routes that handle various user requests, such as rendering the main input form and processing the submitted text. When a user submits their input, Flask routes the request to the appropriate function in the Python file, which then executes the sentiment analysis. This framework simplifies the management of HTTP requests and responses, allowing for rapid development and deployment of web applications. Additionally, Flask's ability to work with templates ensures that dynamic content can be rendered seamlessly, providing users with real-time feedback based on their input.

3. **HTML Templates**: The user interface is built using HTML, providing a structured layout for the application. The templates include forms where users can input their Tamil text and sections for displaying the results of the sentiment analysis. This separation of structure and logic allows for easier updates to the user interface without affecting the underlying functionality. The HTML code is designed to be clean and semantic, ensuring that it is both easy to read and search-engine friendly. By incorporating elements such as headers, paragraphs, and buttons, the interface is not only functional but also visually appealing, encouraging user interaction.

Source Code For HTML file:

Form.html

```
 <html>

    <head>

            <style>

table, th, td {

  border: 1px solid black;

}




</style>

        <title>{{ title }} TAMIL WORD Sentiment Analysis </title>

        <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css')
}}">

<!-- <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet"> -->

                        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></scri
pt>

    </head>

    <body><div class="container-c11" style="background-color:DarkSeaGreen" >

            <div class="container-fluid p-5 bg-success text-white text-center">

                        <centre><h1>தமிழ் WORD SENTIMENT ANALYSIS</h1></centre>



            </div>

        <!--h1 align="center">A machine learning end to end flask web app for
<b>"Sentiment Analysis" </b>model created using Scikit-learn &amp; VADER
Sentiment.</h1-->
```

```
                    </br></br><div align="center" id="a1">

                            <form method="POST">

                                    <textarea name="text1" placeholder="Give any word in
Tamil......" rows="10" cols="109"></textarea><br><br>



                                    <input class="btn btn-success" type="submit">

                            </form>

                    </div>

     {% if final %}



<!--result start -->

    </br></br>  <div align="center">

       </br>  </br>

                      <h2 class="bg-success text-white">The Sentiment of</h2>

                      '{{ text1 }}'

                      <h2>is {{ final }}</h2>



<div class="container table-responsive-sm">

<table class="table table-bordered">

</div>



        {% else %}

        <p></p>

        {% endif %}

     </div></br></br>
```

```
<!--result end -->



  </div>  </body>
</html>
```

Form.html.bak

```
<html>
    <head>
            <style>
table, th, td {
  border: 1px solid black;
}
.c11{
  border: 3px solid blue;
}

</style>
        <title>{{ title }} Sentiment Analysis</title>
        <!--link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}"-->
                        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
                        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
    </head>
    <body><div class="container c11" >
            <div class="container-fluid p-5 bg-success text-white text-center">
```

```
                    <h1>Sentiment Analysis</h1>
                    <p>A machine learning end to end flask web app for <b>"Sentiment
Analysis" </b>model created using Scikit-learn &amp; VADER Sentiment.</p>
          </div>
       <!--h1 align="center">A machine learning end to end flask web app for
<b>"Sentiment Analysis" </b>model created using Scikit-learn &amp; VADER
Sentiment.</h1-->
                    </br></br><div align="center">
                              <form method="POST">
                                        <textarea name="text1" placeholder="Give any word in
Tamil: ...." rows="10" cols="109"></textarea><br><br>

                                        <input class="btn btn-primary" type="submit">
                              </form>
                    </div>
       {% if final %}

<!--result start -->
     </br></br>  <div align="center">
         </br>  </br>
                      <h2 class="bg-primary text-white">The Sentiment of</h2>
                      '{{ text1 }}'
                      <h2>is {{ final }}</h2>
                              <h2>Score table</h2>



<div class="container table-responsive-sm">
<table class="table table-bordered">
  <tr>
    <th>SENTIMENT METRIC</th>
    <th>SCORE</th>
  </tr>
  <tr>
```

```
    <td>Positive</td>
    <td>{{text2}}</td>

  </tr>
  <tr>
    <td>Neutral</td>
    <td>{{text3}}</td>

  </tr>
  <tr>
    <td>Negative</td>
    <td>{{text5}}</td>

  </tr>
  <tr>
    <td>Compound</td>
    <td>{{text4}}</td>

  </tr>
</table>
</div>

            {% else %}
            <p></p>
            {% endif %}
        </div></br></br>
<!--result end -->



  </div>  </body>
</html>
```

4. **CSS Styling**: CSS is employed to enhance the visual presentation of the application, making it more engaging and user-friendly. The styling includes various elements such as colors, fonts, and layouts to create a cohesive design that aligns with modern web standards. Responsive design principles are applied, ensuring that the application adapts smoothly to different screen sizes, from smartphones to desktop monitors. This approach enhances accessibility, allowing users to interact with the application regardless of their device. The CSS also includes transitions and hover effects that improve the user experience by providing visual feedback during interactions.

```
@charset "UTF-8";
#header {
    background-color: #0a53be;
}

:root {
    --bs-blue: #0d6efd;
    --bs-indigo: #6610f2;
    --bs-purple: #6f42c1;
    --bs-pink: #d63384;
    --bs-red: #dc3545;
    --bs-orange: #fd7e14;
    --bs-yellow: #ffc107;
    --bs-green: #198754;
    --bs-teal: #20c997;
    --bs-cyan: #0dcaf0;
    --bs-white: #fff;
```

```
--bs-gray: #6c757d;

--bs-gray-dark: #343a40;

--bs-gray-100: #f8f9fa;

--bs-gray-200: #e9ecef;

--bs-gray-300: #dee2e6;

--bs-gray-400: #ced4da;

--bs-gray-500: #adb5bd;

--bs-gray-600: #6c757d;

--bs-gray-700: #495057;

--bs-gray-800: #343a40;

--bs-gray-900: #212529;

--bs-primary: #0d6efd;

--bs-secondary: #6c757d;

--bs-success: #198754;

--bs-info: #0dcaf0;

--bs-warning: #ffc107;

--bs-danger: #dc3545;

--bs-light: #f8f9fa;

--bs-dark: #212529;

--bs-primary-rgb: 13, 110, 253;

--bs-secondary-rgb: 108, 117, 125;

--bs-success-rgb: 25, 135, 84;

--bs-info-rgb: 13, 202, 240;

--bs-warning-rgb: 255, 193, 7;

--bs-danger-rgb: 220, 53, 69;
```

```
    --bs-light-rgb: 248, 249, 250;

    --bs-dark-rgb: 33, 37, 41;

    --bs-white-rgb: 255, 255, 255;

    --bs-black-rgb: 0, 0, 0;

    --bs-body-color-rgb: 33, 37, 41;

    --bs-body-bg-rgb: 255, 255, 255;

    --bs-font-sans-serif: system-ui, -apple-system, "Segoe UI", Roboto, "Helvetica Neue",
Arial, "Noto Sans", "Liberation Sans", sans-serif, "Apple Color Emoji", "Segoe UI Emoji",
"Segoe UI Symbol", "Noto Color Emoji";

    --bs-font-monospace: SFMono-Regular, Menlo, Monaco, Consolas, "Liberation Mono",
"Courier New", monospace;

    --bs-gradient: linear-gradient(180deg, rgba(255, 255, 255, 0.15), rgba(255, 255, 255, 0));

    --bs-body-font-family: var(--bs-font-sans-serif);

    --bs-body-font-size: 1rem;

    --bs-body-font-weight: 400;

    --bs-body-line-height: 1.5;

    --bs-body-color: #212529;

    --bs-body-bg: #fff

}



*,

::after,

::before {

    box-sizing: border-box

}
```

```
@media (prefers-reduced-motion:no-preference) {
    :root {
            scroll-behavior: smooth
    }
}

body {
    margin: 0;
    font-family: var(--bs-body-font-family);
    font-size: var(--bs-body-font-size);
    font-weight: var(--bs-body-font-weight);
    line-height: var(--bs-body-line-height);
    color: var(--bs-body-color);
    text-align: var(--bs-body-text-align);
    background-color: var(--bs-body-bg);
    -webkit-text-size-adjust: 100%;
    -webkit-tap-highlight-color: transparent
}

hr {
    margin: 1rem 0;
    color: inherit;
    background-color: currentColor;
    border: 0;
    opacity: .25
}

hr:not([size]) {
      height: 1px
}

.h1,
.h2,
.h3,
.h4,
.h5,
.h6,
h1,
h2,
h3,
h4,
h5,
h6 {
      margin-top: 0;
      margin-bottom: .5rem;
      font-weight: 500;
      line-height: 1.2
}

.h1,
h1 {
    font-size: calc(1.375rem + 1.5vw)
}

@media (min-width:1200px) {

    .h1,
    h1 {
            font-size: 2.5rem
    }
}

.h2,
h2 {
    font-size: calc(1.325rem + .9vw)
}

@media (min-width:1200px) {

    .h2,
    h2 {
            font-size: 2rem
    }
}
```

```
.h3,
h3 {
    font-size: calc(1.3rem + .6vw)
}

@media (min-width:1200px) {

    .h3,
    h3 {
            font-size: 1.75rem
    }
}

.h4,
h4 {
    font-size: calc(1.275rem + .3vw)
}

@media (min-width:1200px) {

    .h4,
    h4 {
            font-size: 1.5rem
    }
}
```

```
.h5,
h5 {
    font-size: 1.25rem
}

.h6,
h6 {
    font-size: 1rem
}

p {
    margin-top: 0;
    margin-bottom: 1rem
}

abbr[data-bs-original-title],
abbr[title] {
    -webkit-text-decoration: underline dotted;
    text-decoration: underline dotted;
    cursor: help;
    -webkit-text-decoration-skip-ink: none;
    text-decoration-skip-ink: none
}
```

```
address {
    margin-bottom: 1rem;
    font-style: normal;
    line-height: inherit
}

ol,
ul {
    padding-left: 2rem
}

dl,
ol,
ul {
    margin-top: 0;
    margin-bottom: 1rem
}

ol ol,
ol ul,
ul ol,
ul ul {
    margin-bottom: 0
}
```

```
dt {
    font-weight: 700
}

dd {
    margin-bottom: .5rem;
    margin-left: 0
}

blockquote {
    margin: 0 0 1rem
}

b,
strong {
    font-weight: bolder
}

.small,
small {
    font-size: .875em
}

.mark,
mark {
    padding: .2em;

    background-color: #fcf8e3

}


sub,

sup {

    position: relative;

    font-size: .75em;

    line-height: 0;

    vertical-align: baseline

}


sub {

    bottom: -.25em

}


sup {

    top: -.5em

}


a {

    color: #0d6efd;

    text-decoration: underline

}
```

```
a:hover {
    color: #0a58ca
}

a:not([href]):not([class]),
a:not([href]):not([class]):hover {
    color: inherit;
    text-decoration: none
}

code,
kbd,
pre,
samp {
    font-family: var(--bs-font-monospace);
    font-size: 1em;
    direction: ltr;
    unicode-bidi: bidi-override
}

pre {
    display: block;
    margin-top: 0;
    margin-bottom: 1rem;
    overflow: auto;
    font-size: .875em
}

pre code {
    font-size: inherit;
    color: inherit;
    word-break: normal
}

code {
    font-size: .875em;
    color: #d63384;
    word-wrap: break-word
}

a>code {
    color: inherit
}

kbd {
    padding: .2rem .4rem;
    font-size: .875em;
    color: #fff;
    background-color: #212529;
    border-radius: .2rem
```

```
}

kbd kbd {
    padding: 0;
    font-size: 1em;
    font-weight: 700
}

figure {
    margin: 0 0 1rem
}

img,
svg {
    vertical-align: middle
}

table {
    caption-side: bottom;
    border-collapse: collapse
}

caption {
    padding-top: .5rem;
    padding-bottom: .5rem;
    color: #6c757d;

    text-align: left

}


th {

    text-align: inherit;

    text-align: -webkit-match-parent

}


tbody,

td,

tfoot,

th,

thead,

tr {

    border-color: inherit;

    border-style: solid;

    border-width: 0

}


label {

    display: inline-block

}


button {
    border-radius: 0
}

button:focus:not(:focus-visible) {
    outline: 0
}

button,
input,
optgroup,
select,
textarea {
    margin: 0;
    font-family: inherit;
    font-size: inherit;
    line-height: inherit
}

button,
select {
    text-transform: none
}

[role=button] {
    cursor: pointer
```

```
}

select {
    word-wrap: normal
}

select:disabled {
    opacity: 1
}

[list]::-webkit-calendar-picker-indicator {
    display: none
}

[type=button],
[type=reset],
[type=submit],
button {
    -webkit-appearance: button
}

[type=button]:not(:disabled),
[type=reset]:not(:disabled),
[type=submit]:not(:disabled),
button:not(:disabled) {
    cursor: pointer
}

::-moz-focus-inner {
    padding: 0;
    border-style: none
}

textarea {
    resize: vertical
}

fieldset {
    min-width: 0;
    padding: 0;
    margin: 0;
    border: 0
}

legend {
    float: left;
    width: 100%;
    padding: 0;
    margin-bottom: .5rem;
    font-size: calc(1.275rem + .3vw);
    line-height: inherit
}

@media (min-width:1200px) {
    legend {
            font-size: 1.5rem
    }
}

legend+* {
    clear: left
}

::-webkit-datetime-edit-day-field,
::-webkit-datetime-edit-fields-wrapper,
::-webkit-datetime-edit-hour-field,
::-webkit-datetime-edit-minute,
::-webkit-datetime-edit-month-field,
::-webkit-datetime-edit-text,
::-webkit-datetime-edit-year-field {
    padding: 0
}

::-webkit-inner-spin-button {
    height: auto
```

```
}

[type=search] {
    outline-offset: -2px;
    -webkit-appearance: textfield
}

::-webkit-search-decoration {
    -webkit-appearance: none
}

::-webkit-color-swatch-wrapper {
    padding: 0
}

::-webkit-file-upload-button {
    font: inherit
}

::file-selector-button {
    font: inherit
}

::-webkit-file-upload-button {
    font: inherit;
    -webkit-appearance: button
}

output {
    display: inline-block
}

iframe {
    border: 0
}

summary {
    display: list-item;
    cursor: pointer
}

progress {
    vertical-align: baseline
}

[hidden] {
    display: none !important
}

.lead {
    font-size: 1.25rem;

    font-weight: 300

}


.display-1 {

    font-size: calc(1.625rem + 4.5vw);

    font-weight: 300;

    line-height: 1.2

}


@media (min-width:1200px) {

    .display-1 {

            font-size: 5rem

    }

}


.display-2 {

    font-size: calc(1.575rem + 3.9vw);

    font-weight: 300;

    line-height: 1.2

}


@media (min-width:1200px) {

    .display-2 {

            font-size: 4.5rem
    }
}

.display-3 {
    font-size: calc(1.525rem + 3.3vw);
    font-weight: 300;
    line-height: 1.2
}

@media (min-width:1200px) {
    .display-3 {
            font-size: 4rem
    }
}

.display-4 {
    font-size: calc(1.475rem + 2.7vw);
    font-weight: 300;
    line-height: 1.2
}

@media (min-width:1200px) {
    .display-4 {
            font-size: 3.5rem
    }
```

```
}

.display-5 {
    font-size: calc(1.425rem + 2.1vw);
    font-weight: 300;
    line-height: 1.2
}

@media (min-width:1200px) {
    .display-5 {
            font-size: 3rem
    }
}

.display-6 {
    font-size: calc(1.375rem + 1.5vw);
    font-weight: 300;
    line-height: 1.2
}

@media (min-width:1200px) {
    .display-6 {
            font-size: 2.5rem
    }
}
```

```
.list-unstyled {
    padding-left: 0;
    list-style: none
}

.list-inline {
    padding-left: 0;
    list-style: none
}

.list-inline-item {
    display: inline-block
}

.list-inline-item:not(:last-child) {
    margin-right: .5rem
}

.initialism {
    font-size: .875em;
    text-transform: uppercase
}

.blockquote {
    margin-bottom: 1rem;
    font-size: 1.25rem
}

.blockquote>:last-child {
    margin-bottom: 0
}

.blockquote-footer {
    margin-top: -1rem;
    margin-bottom: 1rem;
    font-size: .875em;
    color: #6c757d
}

.blockquote-footer::before {
    content: "— "
}

.img-fluid {
    max-width: 100%;
    height: auto
}

.img-thumbnail {
    padding: .25rem;
    background-color: #fff;
    border: 1px solid #dee2e6;
    border-radius: .25rem;
    max-width: 100%;
    height: auto
}

.figure {
    display: inline-block
}

.figure-img {
    margin-bottom: .5rem;
    line-height: 1
}

.figure-caption {
    font-size: .875em;
    color: #6c757d
}

.container,
.container-fluid,
.container-lg,
.container-md,
.container-sm,
.container-xl,
.container-xxl {
    width: 100%;
    padding-right: var(--bs-gutter-x, .75rem);
    padding-left: var(--bs-gutter-x, .75rem);
    margin-right: auto;
    margin-left: auto
}
```

JSON for Data Exchange: JSON (JavaScript Object Notation) is used for efficient data interchange between the frontend and backend of the application. When users submit their input, the application sends this data in JSON format to the server for processing. After the sentiment analysis is complete, the results are also returned in JSON format, allowing for easy integration with the frontend. This lightweight data format is ideal for web applications, as it enables quick parsing and minimizes bandwidth usage. The use of JSON ensures that the application can handle real-time updates effectively, providing users with instant feedback on their inputs.

launch.json

```
    "version": "0.2.0",
      "configurations": [

        {
          "name": "Python: Current File",
          "type": "python",
          "request": "launch",
    "program": "${file}",
    "console": "integratedTerminal"
},
{
    "type": "pwa-chrome",
    "request": "launch",
    "name": "Launch Chrome against localhost",
    "url": "http://localhost:8080",
    "webRoot": "${workspaceFolder}"
}]}
```

6. TESTING

6.1 Testing Methodologies

6.1.1 Unit Testing

Unit testing involves testing individual functions or components of the application in isolation to ensure they work as intended. In the context of the Tamil Word Sentiment Analysis project, each function within the Python code, particularly in the Sentiment Analysis Module**,** is subjected to rigorous testing. For example, the `analyze_sentiment` function is tested with various input scenarios, including:

- Valid inputs containing known positive and negative words.
- Inputs with no sentiment-related words to check for neutral output.
- Edge cases such as empty strings or very short inputs to ensure robust handling.

This testing ensures that each function performs accurately and reliably, which is essential for the integrity of the overall application.

6.1.2 Integration Testing

Integration testing focuses on the interactions between different modules of the application to ensure they work together seamlessly. For the Tamil Word Sentiment Analysis project, this includes testing the communication between the Flask backend and the HTML frontend. Key aspects of integration testing involve:

- Verifying that user inputs from the HTML form are correctly received by the Flask application.
- Ensuring the sentiment analysis results generated by the backend are accurately displayed on the frontend.
- Testing data flow between the User Interface Module**,** Sentiment Analysis Module, and Feedback Generation Module to confirm they integrate smoothly.

This step is crucial to identify any issues that may arise when different parts of the system interact with each other.

6.1.3 System Testing

System testing is the final phase where the complete application is tested as a whole to evaluate its compliance with the specified requirements. In the Tamil Word Sentiment Analysis project, end-to-end testing encompasses the following:

- Conducting real-world scenarios where users submit Tamil text for sentiment analysis and reviewing the accuracy of the output.
- Testing the application's performance under various conditions, such as high user load or large input sizes, to ensure stability and responsiveness.
- Checking for any security vulnerabilities, ensuring that user inputs are sanitized to prevent issues like injection attacks.

6.2 Test Cases

Test Case 1: Valid Positive Input
Test Description: This test checks if the application can accurately identify positive sentiment from a known positive word.
Input: "அருமை"
Expected Output: "Positive"
Explanation: The system should recognize "அருமை" as a positive word and return a sentiment classification of "Positive."

Test Case 2: Valid Negative Input
Test Description: This test verifies the application's ability to correctly identify negative sentiment using a recognized negative word.
Input: "மோசமான"
Expected Output: "Negative"
Explanation: The application should classify "மோசமான" as a negative word and output "Negative."

Test Case 3: Neutral Input
Test Description: This test assesses the application's performance when no sentiment-related words are present in the input.

Input: "கண்கள்"

Expected Output: "Neutral"

Explanation: The input "கண்கள்" does not contain any sentiment words, so the system should return "Neutral."

Test Case 4: Mixed Sentiment Input

Test Description: This test examines how the application handles a sentence containing both positive and negative words.

Input: "அருமை ஆனால் மோசமான"

Expected Output: "Neutral"

Explanation: Since the positive and negative sentiments balance each other out, the application should classify this input as "Neutral."

Test Case 5: Empty Input

Test Description: This test checks the application's behavior when given no input at all.

Input: ""

Expected Output: "Neutral"

Explanation: The system should gracefully handle an empty input and return "Neutral" as a default response.

Test Case 6: Long Input

Test Description: This test evaluates how well the application manages longer text inputs.

Input: "இந்த நுரை எவ்வளவு அருமை! ஆனால் இது சில நேரங்களில் மோசமாக இருக்கும்."

Expected Output: "Neutral"

Explanation: Although the input contains both positive and negative sentiments, the overall classification should be "Neutral."

Test Case 7: Invalid Characters

Test Description: This test assesses the application's handling of inputs with special characters or numbers.

Input: "சிறப்பு #123"

Expected Output: "Neutral"

Explanation: The application should ignore the invalid characters and classify the sentiment as "Neutral."

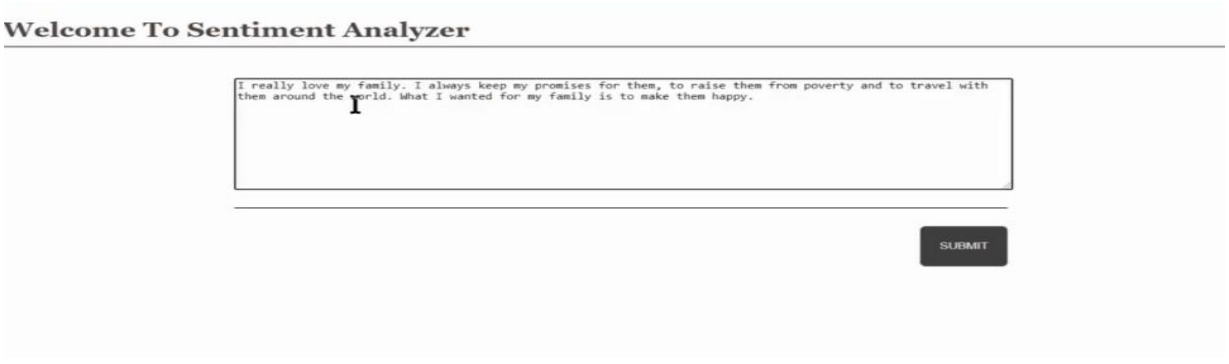
Test Case 8: Performance Under Load

Test Description: This test measures the application’s performance when handling multiple simultaneous user requests.
Input: Simulate various users submitting different inputs at the same time.
Expected Output: The application should remain responsive and functional without crashing or exhibiting significant delays.
Explanation: This test ensures that the system can handle high traffic efficiently.

7. OUTPUT SCREENS

Screenshot of the application:

- Initial input form.it is the initial screen which was made in the English language which only classify the English text with a simple structure and and with a submit button at the bottom of the right side.



I really love my family. I always keep my promises for them, to raise them from poverty and to travel with them around the world. What I wanted for my family is to make them happy.

Fig7.1.Initial Page

Final Display of sentiment analysis result.

The final display which classify the tamil words and with a white text background and coloured webpage using html, and css and the submit button is place at the centre of the webpage.

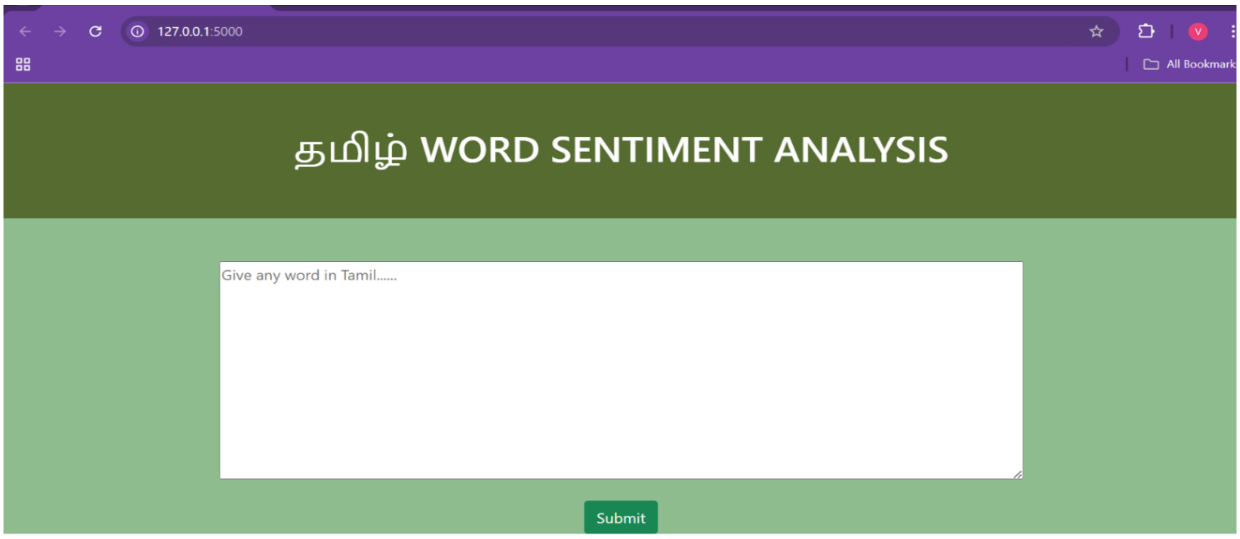


Fig7.2.Final Page Display

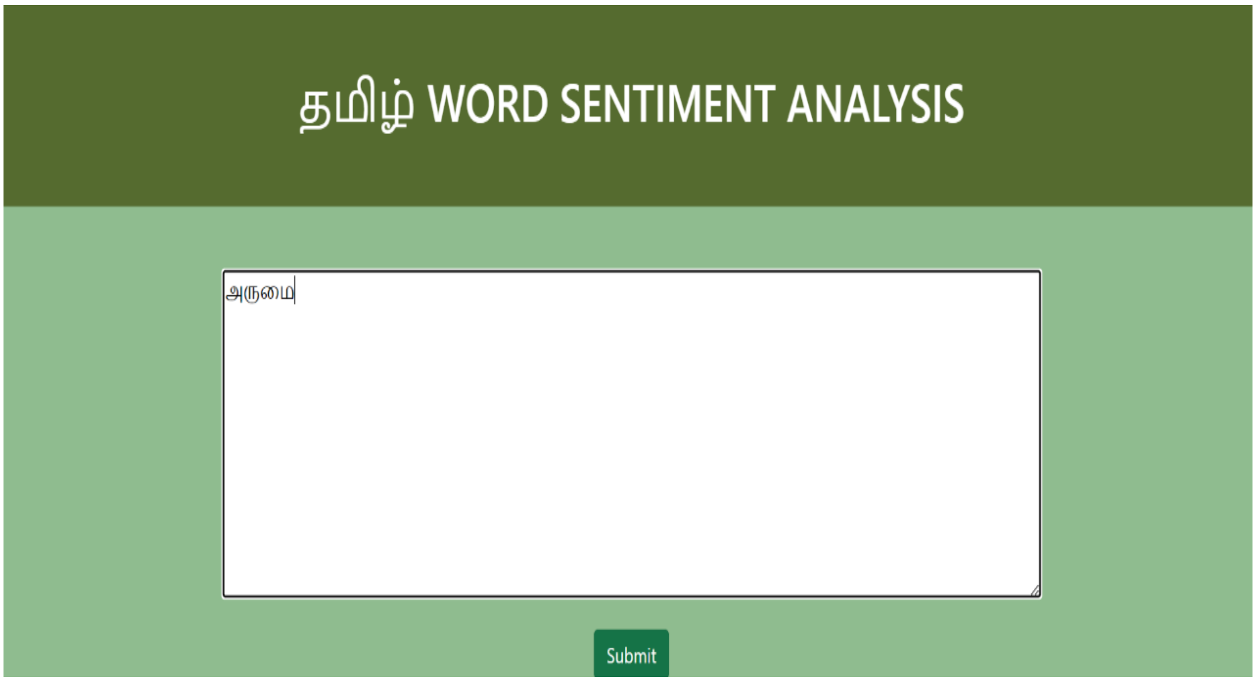
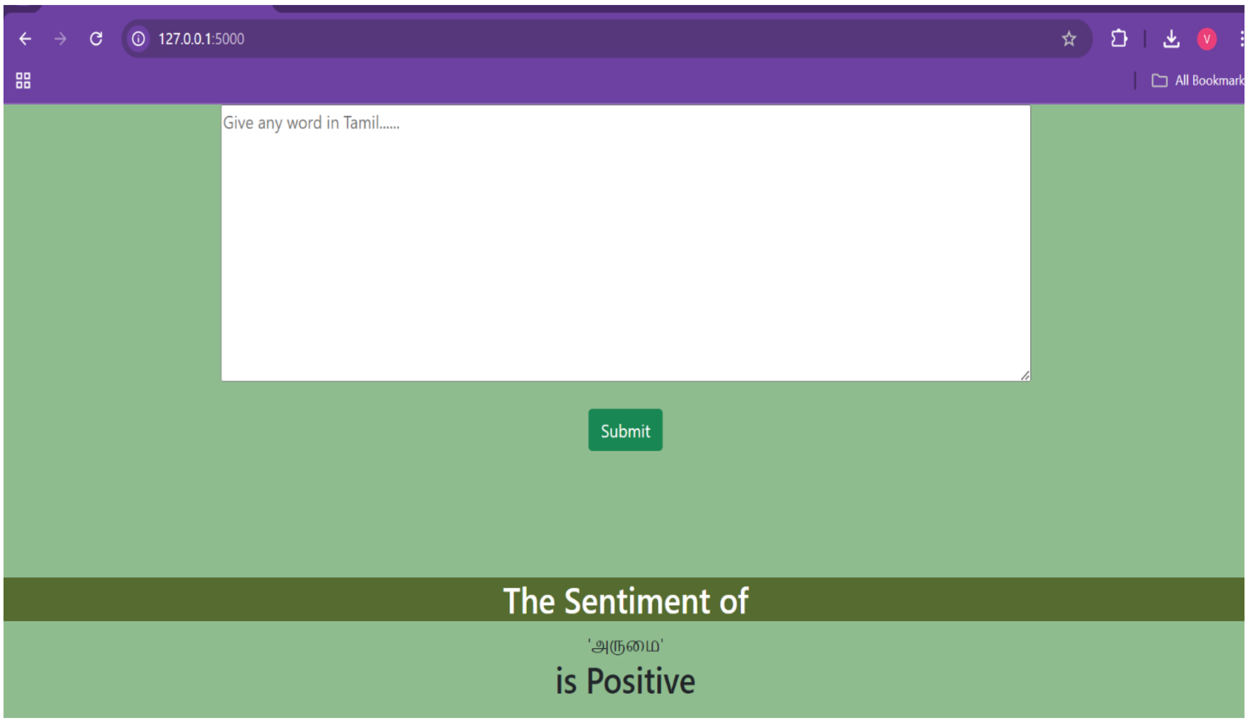


Fig7.3.Final Input Text Page

The final page of the tamil word classification under the submit button



The Sentiment of

'அருமை'

is Positive

Fig7.4.Final Sentiment Page

8. CONCLUSION

8.1 Summary of the Project

The **Tamil Word Sentiment Analysis** project is designed to provide a user-friendly web application that evaluates the sentiment of Tamil text. Utilizing Flask for the backend and HTML/CSS for the frontend, this application allows users to input Tamil words and receive instant feedback on whether the sentiment expressed is positive, negative, or neutral. The core functionality hinges on a predefined list of Tamil words categorized by sentiment, enabling efficient and accurate analysis.

The project aims to bridge the gap in sentiment analysis tools specifically tailored for the Tamil language, addressing the growing need for linguistic resources in various applications, such as social media monitoring and customer feedback analysis. By implementing a modular structure, the application is not only easy to maintain but also allows for future enhancements, such as the inclusion of machine learning algorithms for improved accuracy.

Testing methodologies, including unit testing, integration testing, and system testing, were employed to ensure that each component functions correctly and that the application performs well under various conditions. The project demonstrates the effectiveness of using simple algorithms for sentiment classification while providing a foundation for further development.

In summary, this project not only showcases the capabilities of basic sentiment analysis techniques but also emphasizes the importance of language-specific tools in the field of natural language processing. With a clear potential for expansion, the Tamil Word Sentiment Analysis application stands as a valuable resource for researchers, businesses, and developers interested in Tamil language applications.

8.2 Key Achievements

Accurate Sentiment Classification

The application successfully categorizes Tamil text into three distinct sentiment categories: positive, negative, and neutral. By utilizing a carefully curated list of words, it provides reliable sentiment analysis, allowing users to gain insights into the emotional tone of their text.

Intuitive User Interface

An engaging and user-friendly web interface has been developed, enabling users to easily input Tamil text. The design prioritizes usability, ensuring that users receive prompt feedback on sentiment analysis results, which enhances the overall experience.

Comprehensive Testing

A robust testing framework has been established that encompasses unit testing, integration testing, and system testing. This comprehensive approach ensures the application's reliability and performance across various scenarios, validating its effectiveness under different conditions.

Modular Design

The architecture of the application is modular, which significantly improves maintainability. This design choice not only allows for easier updates and modifications but also sets the stage for future enhancements, such as the integration of advanced machine learning techniques.

Contribution to Linguistic Resources

By addressing the scarcity of sentiment analysis tools specifically for the Tamil language, this project provides a valuable asset for users, researchers, and businesses. It fills a crucial gap in linguistic resources, promoting greater understanding and analysis of Tamil sentiment.

Foundation for Future Development

The project lays a strong foundation for ongoing development, including plans to expand the word database and refine sentiment detection algorithms. Additionally, there are opportunities to introduce support for multiple languages, further broadening its applicability.

Real-time Analysis

Real-time sentiment analysis capabilities have been implemented, allowing users to quickly gauge the emotional context of their inputs. This feature is particularly beneficial for applications in social media monitoring and customer feedback analysis.

Positive User Feedback

Initial users have provided encouraging feedback, highlighting the application's effectiveness and ease of use in interpreting Tamil sentiment. This validation reinforces the project’s relevance and utility in the field of sentiment analysis.

Educational Resource

The application serves as a practical educational tool for students and developers interested in natural language processing and sentiment analysis. It provides hands-on experience in these fields, fostering learning and innovation.

Community Engagement

The project has sparked interest in developing resources for the Tamil language within the tech community. It encourages collaboration and discussion, paving the way for future projects aimed at enhancing linguistic tools and technologies.

9. FURTHER ENHANCEMENTS

9.1 Possible Future Features

- **Machine Learning Integration:** Implementing machine learning algorithms could enhance the sentiment analysis capabilities by allowing the system to learn from user inputs and improve accuracy over time, moving beyond simple keyword matching.

- **Expanded Word Database:** Expanding the list of positive and negative words to include slang, idiomatic expressions, and contextually relevant phrases would enable a more nuanced understanding of sentiment in diverse Tamil dialects.

- **Multi-language Support:** Adding support for additional languages could broaden the application's reach, allowing users to analyze sentiments in languages beyond Tamil and making it a versatile tool for multilingual sentiment analysis.

- **Contextual Analysis:** Developing features to analyze the context of sentences more effectively could help distinguish between different sentiments expressed in similar phrases, improving the application’s reliability.

- **User Feedback Mechanism:** Introducing a feedback system where users can report inaccuracies in sentiment classification could provide valuable data for refining algorithms and improving the application’s performance.

- **Sentiment Trend Analysis:** Implementing features to track sentiment trends over time for specific keywords or phrases would provide users with insights into how sentiments evolve in different contexts, such as social media campaigns or product reviews.

- **Visualization Tools:** Adding graphical representations of sentiment data, such as charts and graphs, would help users visualize sentiment trends and make data-driven decisions based on the analysis.

- **Mobile Application Development:** Creating a mobile version of the application could increase accessibility, allowing users to perform sentiment analysis on the go and reach a wider audience.

- **API Development:** Developing an API would enable third-party applications to integrate sentiment analysis functionalities, expanding the potential use cases and applications of the tool.

- **Enhanced User Customization:** Providing users with options to customize the sentiment analysis parameters, such as adjusting the sensitivity of sentiment detection, would allow for a more tailored experience based on individual needs.

9.2 Scaling the Project

At present, the Tamil Word Sentiment Analysis project operates as a locally hosted web application without cloud deployment or user account systems. The sentiment analysis is performed in real-time based on user input, using predefined lists of positive and negative words in Tamil. Since the project runs locally, it does not require internet connectivity for access, and no data is stored after the analysis is complete.

Currently, there are no features that allow users to create accounts or save their analysis results. Once the sentiment analysis is performed, the results are displayed immediately but are not saved for future reference. The system processes each input individually and does not provide functionalities like data persistence or tracking user interaction history.

While the project is simple and efficient in its current form, it does not yet support scalability through cloud deployment or advanced features such as user authentication and data storage. These features, if integrated, could enhance the project by making it more accessible and allowing users to store and revisit previous analysis results. However, these improvements would require additional development and infrastructure.

10.REFERENCES

10.1 Documentation and Sources

1. geeksforgeeks website: Twitter sentiment analysis, https://www.geeksforgeeks.org/twitter-sentiment-analysis-using-python/

2.Javatpoint website: Creating a sentiment analysis web app using streamlit, https://www.javatpoint.com/create-a-simple-sentiment-analysis-webapp-using-streamlit

3.ProjectsWorld: Sentiment analysis ML web App using Flask, https://projectworlds.in/sentiment-analysis-ml-flask-python-web-app-project-with-source-code/

4.CodersPro Youtube Channel: Sentiment Analysis web app using python, https://youtu.be/RPU11g-0ByE?feature=shared

5.ResearchGate:Developing a Web application Analysis using the Flask Framework and python, https://www.researchgate.net/publication/363780327_DEVELOPING_A_WEB_APPLICATION_FOR_SENTIMENT_ANALYSIS_USING_THE_FLASK_FRAMEWORK_AND_PYTHON

6.Medium:Sentiment Analysis ML Flask Python Web App, https://medium.com/@sairaghav.ganesh/sentiment-analysis-ml-flask-python-web-app-e5dcda7be6db

7.The AI University Youtube Channel: Sentiment Analysis using LSTM Model& Flask Web App, https://youtu.be/L5pNL2Ac6EE?feature=shared

8.Edureka:A Quick Guide to Sentiment Analysis, https://youtu.be/O_B7XLfx0ic?feature=shared

9. Akrati Saxena, Harita Reddy, Pratishtha Saxena, "Introduction to Sentiment Analysis Covering Basics, Tools, Evaluation Metrics, Challenges, and Applications", *Principles of Social Networking*, vol.246, pp.249, 2022.

10. Kanika Sharma, Dhyanendra Jain, Ashu Jain, Yogendra Singh Rathore, Shruti Agarwal, "Sentimental Analysis based on Machine Learning Technique", *2024 3rd International Conference for Innovation in Technology (INOCON)*, pp.1-7, 2024.

11. APPENDICES

11.1 User Documentation

Step-by-step guide on how to use the application.

STEP 1: Click on the url to open the Homepage, http://127.0.0.1:5000/

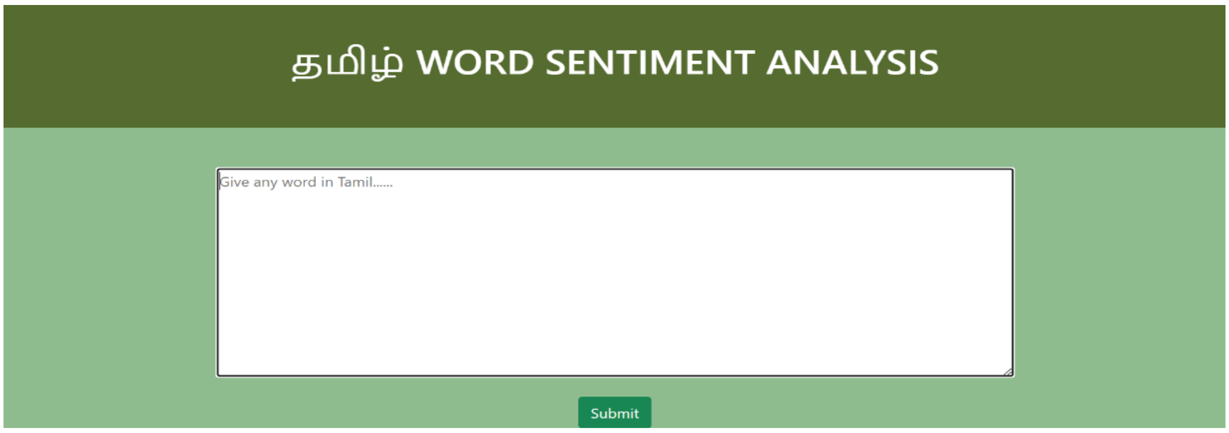


Fig11.1.Homepage Display

HOMEPAGE OF THE TAMIL WORD SENTIMENT ANALYSIS

STEP 2: Enter the Tamil Word in the Text Area

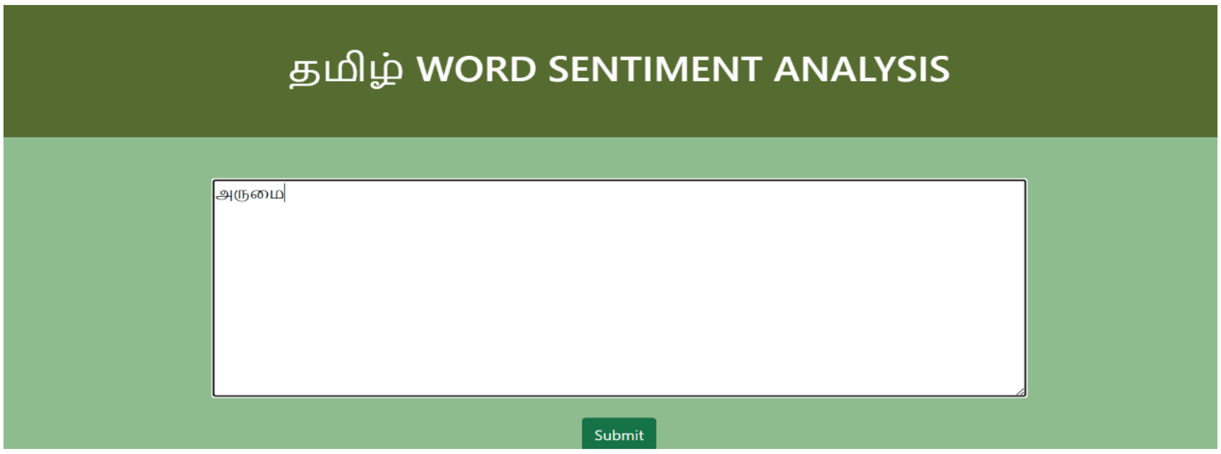


Fig11.2.Home page Text Area

STEP 3: Click on the submit button

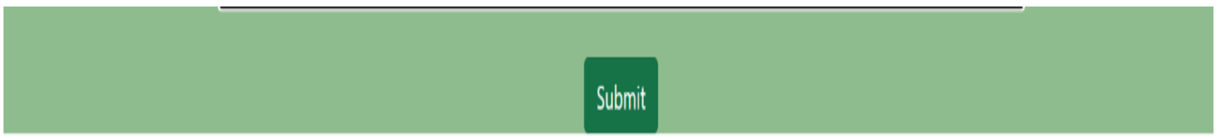
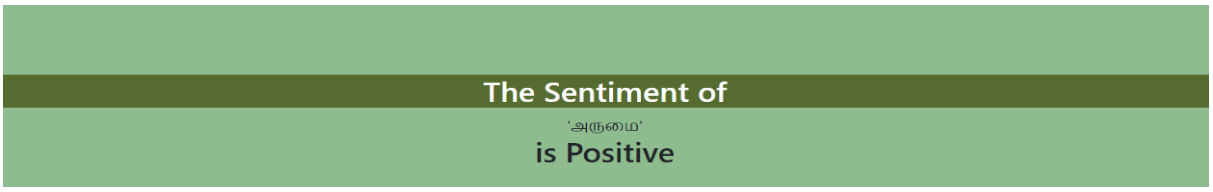


Fig11.3.Submit button display

STEP 4:The Sentiment of the Text that is Displayed at the bottom of the HomePage


The Sentiment of

'அருமை'

is Positive

Fig11.4.Sentiment display

11.2 README

The README provides developers with basic instructions for setting up and running the Tamil Word Sentiment Analysis project. The project is built using Python, the Flask framework for the web interface, and predefined lists of Tamil words for sentiment analysis. To set up the project, developers need to ensure that Python is installed on their system, along with necessary dependencies like Flask. A virtual environment is recommended to manage the project's dependencies. The setup steps typically include:

1. Clone the project repository.
2. Set up a virtual environment using `venv` or another tool.
3. Install the required Python libraries from the `requirements.txt` file using `pip install -r requirements.txt`.
4. Run the Flask server by executing the main Python file.
5. Access the application in a web browser via the local server address (e.g., `http://127.0.0.1:5000`).