**Language Detection using NLP**

**1. Introduction**

This document provides comprehensive documentation for the **Language Detection System**, which identifies the language of a given text input using **Natural Language Processing (NLP) and Machine Learning**. The system is developed using **Flask** for the backend API, **HTML & JavaScript** for the frontend interface, and **Scikit-Learn** for the machine learning model.

The ability to detect languages accurately is crucial in applications such as multilingual chatbots, automated content moderation, and cross-lingual communication tools. This project leverages modern machine learning techniques to classify input text into multiple languages based on textual patterns and vocabulary structures.

**2. Project Structure**

|-- app.py # Flask API for language detection

|-- index.html # Frontend web page for user interaction

|-- language\_detection.py # Script for training the language detection model

|-- Language\_Detection.csv # Dataset containing text samples in multiple languages

|-- language\_model.pkl # Trained machine learning model & vectorizer

**3. Installation and Setup**

**Prerequisites:**

* Python 3.x
* Flask
* Scikit-learn
* Pandas
* NumPy
* Flask-CORS

**Installation Steps:**

1. Install dependencies using pip:

pip install flask flask-cors scikit-learn pandas numpy

1. Run the model training script to generate language\_model.pkl:

python language\_detection.py

1. Start the Flask server:

python app.py

1. Open index.html in a web browser to access the frontend interface.

**4. Deployment on Render and GitHub**

This application is deployed using **Render** and **GitHub**, allowing seamless version control and automated deployments.

**Steps to Deploy on GitHub and Render:**

1. **Create a GitHub Repository:**
   * Push your project files to a new repository on [GitHub](https://github.com/).
2. **Create an Account on Render:**
   * Sign up at [Render](https://render.com/) and create a new **Web Service**.
3. **Connect GitHub Repository:**
   * Link your GitHub repository to Render for continuous deployment.
4. **Define Service Configuration:**
   * Use render.yaml to specify the service settings.
5. **Set Environment Variables (if needed):**
   * Configure any necessary environment variables within the Render dashboard.
6. **Deploy the Application:**
   * Click **Deploy** on Render to host the Flask API.
7. **Access Public URL:**
   * Once deployed, obtain the public URL to use the service globally.

**5. Backend (Flask API)**

**app.py**

* Loads the trained model (language\_model.pkl).
* Provides an API endpoint /detect to process text input and return the detected language.
* Routes:
  + GET / → Returns a message indicating that the API is running.
  + POST /detect → Accepts a JSON input { "text": "sample text" } and returns { "language": "English" }.

The API is designed to be lightweight, ensuring quick response times while maintaining high accuracy in language classification.

**6. Frontend (index.html)**

* Provides a user-friendly interface for entering text and detecting its language.
* Uses JavaScript to send an AJAX request to the Flask API.
* Displays the detected language on the webpage dynamically.
* Implements error handling for invalid responses or network failures.

The interface has been designed with accessibility and ease of use in mind, making it suitable for a wide range of users.

**7. Model Training**

**language\_detection.py**

* Loads the dataset from Language\_Detection.csv.
* Preprocesses text and removes unwanted languages.
* Utilizes **CountVectorizer** for feature extraction.
* Trains an **SGDClassifier** for classification.
* Saves the trained model as language\_model.pkl.

The dataset consists of multiple text samples labeled with their corresponding languages. Text preprocessing includes tokenization, stopword removal, and vectorization. The classifier is optimized using **Stochastic Gradient Descent (SGD)** to balance speed and accuracy.

**8. API Testing**

* Use **Postman** or curl to test the API:

curl -X POST http://localhost:5000/detect -H "Content-Type: application/json" -d '{"text": "Bonjour"}'

* The response should return the predicted language in JSON format.
* Error handling ensures meaningful responses in case of invalid input.

**9. Future Enhancements**

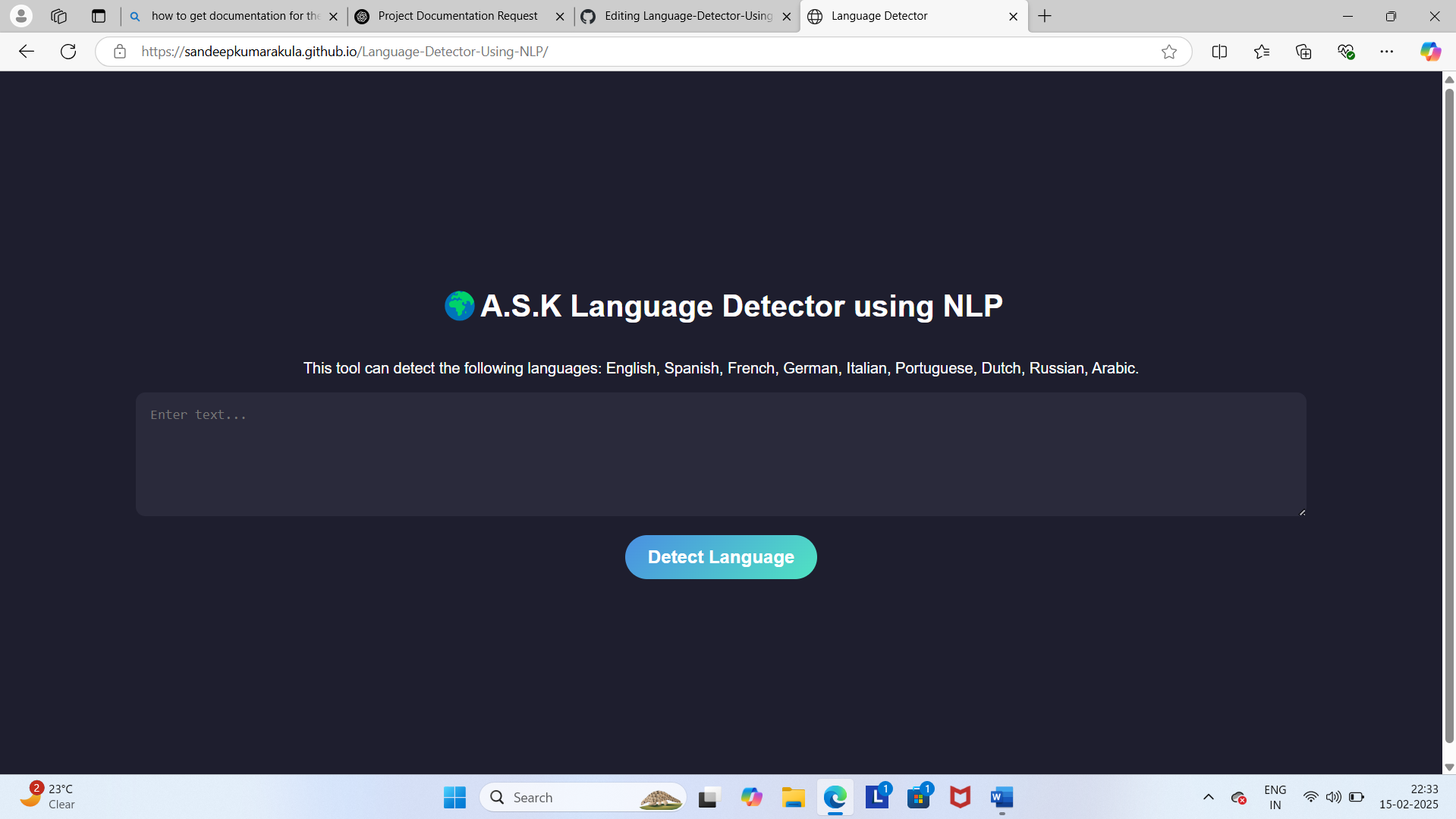
* Enhance model accuracy using advanced vectorization techniques such as **TF-IDF** and **Word Embeddings**.
* Deploy the application on additional cloud platforms such as **Heroku**, **AWS Lambda**, or **Google Cloud Functions** for scalability.
* Expand language support by incorporating additional training data from diverse linguistic sources.
* Implement a confidence scoring mechanism to indicate prediction certainty.
* Add support for real-time language detection in live chat applications.
* Develop a mobile-friendly version of the frontend.

**10. Conclusion**

This project offers a robust and scalable solution for detecting languages using **NLP and machine learning**. By leveraging feature extraction techniques and efficient classification algorithms, the system achieves high accuracy while maintaining fast processing speeds.

By integrating **GitHub** and **Render**, the deployment process is automated, ensuring continuous delivery with minimal effort. This system can be expanded for real-world applications, including customer service automation, multilingual search engines, and content filtering systems. By improving language recognition capabilities, this tool contributes to breaking language barriers and fostering global communication.

**11.Result**

****