**TEXT SUMMARIZATION**

*A*

*Mini Project Report*

*Submitted in partial fulfilment of the*

*Requirements for the award of the Degree of*

**BACHELOR OF ENGINEERING**

IN

**INFORMATION TECHNOLOGY**

By

**AVANISH REDDY (1602-19-737-006)**

**GOVIND CHANDAK (1602-19-737-014)**

**G SREE MAHALAXMI (1602-19-737-045)**

****

**Department of Information Technology**

**Vasavi College of Engineering (Autonomous)**

**VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)**

**(AFFILIATED TO OSMANIA UNIVERSITY)**

**HYDERABAD - 500 030**

**Department of Information Technology**

****

**DECLARATION BY CANDIDATE**

We, **AVANISH REDDY, GOVIND CHANDAK, G.SREE MAHALAXMI,** bearing hall ticket number, **1602-19-737-006, 1602-19-737-014, 1602-19-737-045** hereby declare that the project report entitled **”TEXT SUMMERIZATION”** Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfillment of the requirement for the award of the degree of **Bachelor of** **Engineering** in **Information Technology.**

This is a record of bonafide work carried out by me and the results embodied in this project report has not been submitted to any other university or institute for the award of any other degree or diploma.

**AVANISH REDDY (1602-19-737-006)**

**GOVIND CHANDAK (1602-19-737-014)**

**G.SREE MAHALAXMI (1602-19-737-045)**

**VASAVI COLLEGE OF ENGINEERING (AUTONOMOUS)**

**(AFFILIATED TO OSMANIA UNIVERSITY)**

**HYDERABAD - 500 030**

**Department of Information Technology**

****

**BONAFIDE CERTIFICATE**

This is to certify that the project entitled “**TEXT SUMMERIZATION**” being submitted by **AVANISH REDDY, GOVIND CHANDAK, MAHALAXMI** bearing **1602-19-737-006, 1602-19-737-014, 1602-19-737-045**, in partial fulfillment of the requirements for the completion of MINI PROJECT of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

Dr. N Anil Kumar

Internal Guide

**ACKNOWLEDGEMENT**

We thank the department of INFORMATION TECHNOLOGY, for introducing the subject “Mini project-2” in BE 3rd year that let us learn and explore features in “Web Development with Machine Learning”.

We would also like to show our appreciation to our Honourable Principal, Dr. S V RAMANA sir, for supporting us and our mini project lecturer, Dr. Nagam Anil Kumar sir for letting us correctly understand the process of doing a project using Web Development with Machine Learning and for providing valuable insight and expertise that has greatly assisted us in the making of the project.

THANK YOU!

**CHAPTER 1**

**ABSTRACT**

Automatic text summarization is the task of producing a concise and fluent summary while preserving key information of the content and overall meaning using machine learning. It takes care of choosing the most significant portions of text and generates coherent summaries that expresses the main intent of the given document. The main idea of summarization is to find a subset of data which contains the “information” of the entire set.

In today’s world, data generation and consumption are exploding at an exponential rate. Due to this, text summarization has become the necessity of many applications such as search engines , business analysis, market review etc. Automatic text summarization involves producing a summary of the given text document without any human help. It is a viable solution to reduce the information overload problem on the web and to reduce time instead Of reading the whole text file .It provides quick Information from large documents.

Text summarization are of two types- Abstractive summarization and Extractive summarization .In this project we focus on Extractive summarization . It focuses on extracting objects directly from the entire collection without modifying the content Itself. Extractive summarizers takes sentences as the input and produces a probability vector as output. To produce the final summary best sentences are chosen according to the required summary length.

**CHAPTER 2**

**INTRODUCTION**

**2.1 PURPOSE**

The ability to summarize a text is very important in the modern day. In our day to day activities we actively try to find shortcuts and ways to make our work more efficient. This is where our text summarizer comes into play.

**2.2 INTENDED AUDIENCE**

The intended audience for this document is the development team & for enthusiasts who wish to further work on this project.

**2.3 PRODUCT SCOPE**

There are a few other websites that provide text summarization , but our Text summarization provides more options to the users.

**2.4 PROBLEM DEFINATION**

**Scenario 1 :**

For example, if a student has to write a maximum amount of words for an article, essay, report, story you name it, The Automatic Text Summarization website will help to make sure its word count reaches a specific requirement or stays within a certain limit.

This helps the student form precise and accurate summaries, which gives a recap of the entire text in a condensed manner saving time needed for understanding the text.

This is also handy if a student is preparing for an exam which has loads and loads of material to study from.

**Scenario 2 :**

In our daily life, we can see many articles on the same concept in different newspapers. Everyone has different expository writing styles.

For example, if there is an article related to the Ukraine war in The Hindu newspaper and the same article in the Times of India with a different way of narrating.

Then our website Text Summarization is used to compare the text of different articles, summarize the contents of all text, and provide the overall meaning of the idea.

And also identifies which one is the best among them.

**CHAPTER 3**

**3.1 LITERATURE SURVEY**

This paper contains a large literature review in the research field of Text Summarization (TS) based on Human Language Technologies (HLT). TS helps users manage the vast amount of information available by condensing documents’ content and extracting the most relevant facts or topics included in them. The rapid development of emerging technologies poses new challenges to this research field, which still needs to be solved. Therefore, it is essential to analyse its progress over the years and provide an overview of the past, present, and future directions, highlighting the main advances achieved and outlining remaining limitations. For this purpose, several important aspects are addressed within the scope of this survey. On the one hand, the paper aims at giving a general perspective on the state-of-the-art, describing the main concepts, as well as different summarization approaches and relevant international forums. Furthermore, it is important to stress the fact that the birth of new requirements and scenarios has led to the creation of new types of summaries with specific purposes (e.g. sentiment-based summaries) and novel domains within which TS has proven to be suitable for (e.g. blogs). In addition, TS is successfully combined with a number of intelligent systems based on HLT (e.g., information retrieval, question answering, and text classification). On the other hand, a deep study of the evaluation of summaries is also conducted in this paper, where the existing methodologies and systems are explained, as well as new research that has emerged concerning the automatic evaluation of summaries’ quality. Finally, some thoughts about TS in general and its future will encourage the reader to think of novel approaches, applications, and lines to conduct research in the next few years. The analysis of these issues allows the reader to have a broad and useful background on the main important aspects of this research field.

**CHAPTER 4**

* 1. **Existing Method/System**

**LATENT SEMANTIC ANALYSIS :**

The existing method of implementing Text Summarization is Latent Semantic Analysis, picking the most important phrases and lines from the document using named entities and the LSA algorithm.

• LSA is an algorithm that uses for extractive summarization.

• In Latent semantic analysis, sentences are sorted according to semantic order.

• Implementing LSA in python using genism.

1. Tokenization, splitting the text into sentences and individual tokens.

2. POS –tagging, marking each token with its POS tag.

3. Graph construction, a node consists of a keyword or a sentence.

4. Calculating keyword/sentence node weights and rank.

5. Sorting by rank.

**4.2 DRAWBACKS:**

* Might miss out on certain sentences affecting the summary’s meaning: Certain sentences that contribute to the summary might be omitted which in return might affect the generated summary.
* Grammatical mistakes - abstractive algorithms are prone to grammatical mistakes: Abstractive methods rewrite certain portions of sentences to generate the summary. There is a chance that these sentences might contain grammatical errors affecting the overall readability.
* Efforts put into training the models might not exactly meet the required standards: Neural Network-based models require large resources and time to train. The results might not exactly meet the required standards or the level of manual text summarization.

**CHAPTER 5**

**SOFTWARE REQUIREMENTS**

5.1 Visual Code Studio

5.2 Streamlit for BACKEND Framework

5.3 Web Browser for Web App

5.4 Front End Technology

5.4.1 Django

5.4.2 HTML

5.4.3 CSS

5.5 Back End Technology

5.5.1 Python

**HARDWARE REQUIREMENTS**

5.7 Operating system – Linux/Windows

5.8 Hard Drive – 16 GB

5.9 RAM – 1GB

**REQUIREMENT ANALYSIS**

5.9 Python is used in our program. It uses the following libraries.

* *gensim==3.8.3*
* *nltk==3.5*
* *Pattern==3.6*
* *spacy==3.2.1*
* *spacy\_streamlit==1.0.2*
* *streamlit==1.3.0*
* *textblob==0.15.3*

5.10 OS: Program is tested on windows 10.

5.11 Laptop: Used to run our code.

**CHAPTER 6**

**6.1 Proposed Method/System**

**TF-IDF ALGORITHM :**

**TF-IDF (term frequency-inverse document frequency)** is a method that gives us a numerical weightage of words that reflects how meaningful the particular word is to a document in a corpus.

**TERM FREQUENCY (TF):** The weight of a term that occurs in a document is proportional to the term frequency.

TF (t , d) = count of term in document / number of words in document

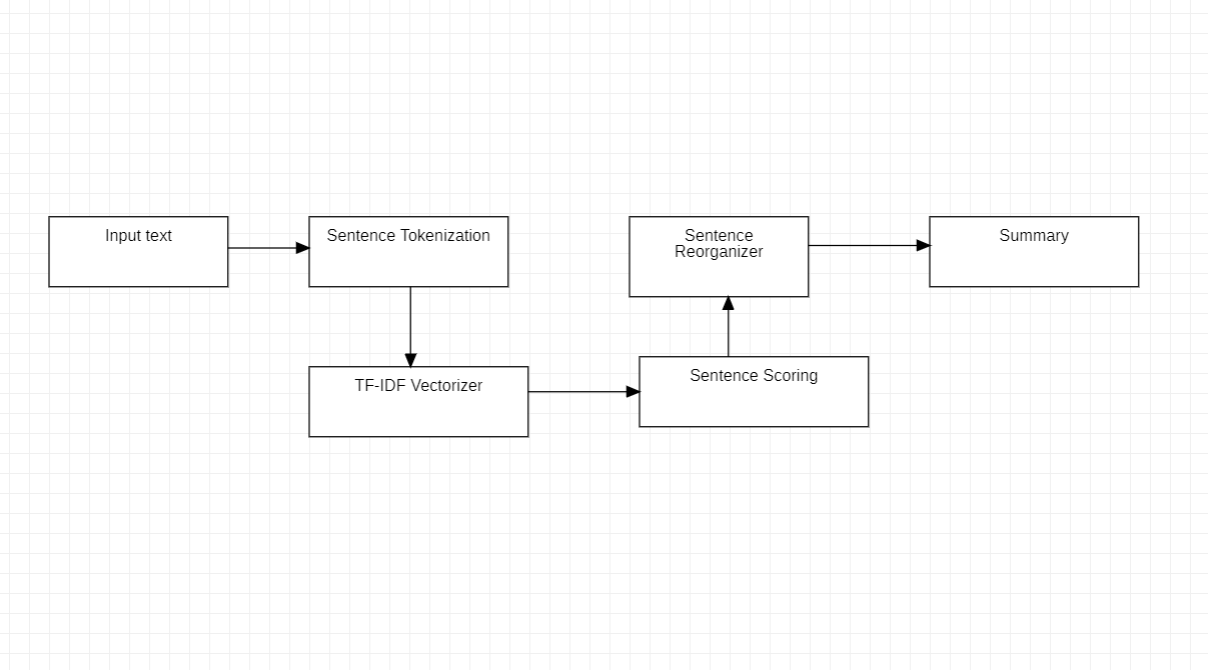
**DOCUMENT FREQUENCY (DF):** DF is the count of occurrences of term t in the document set N

DF (t) = occurrence of t in documents

**Inverse Document frequency (IDF):** The inverse document frequency factor is incorporated, which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.(ex: Stop words)

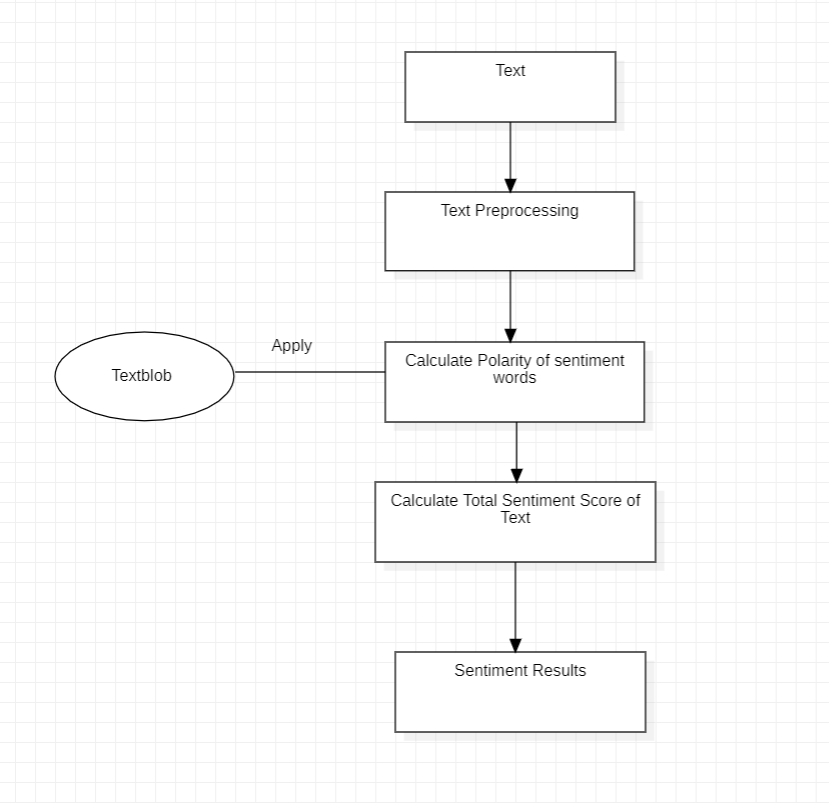
IDF (t) = log (N/(DF + 1))

Overall Formula for finding weightage of each sentence:

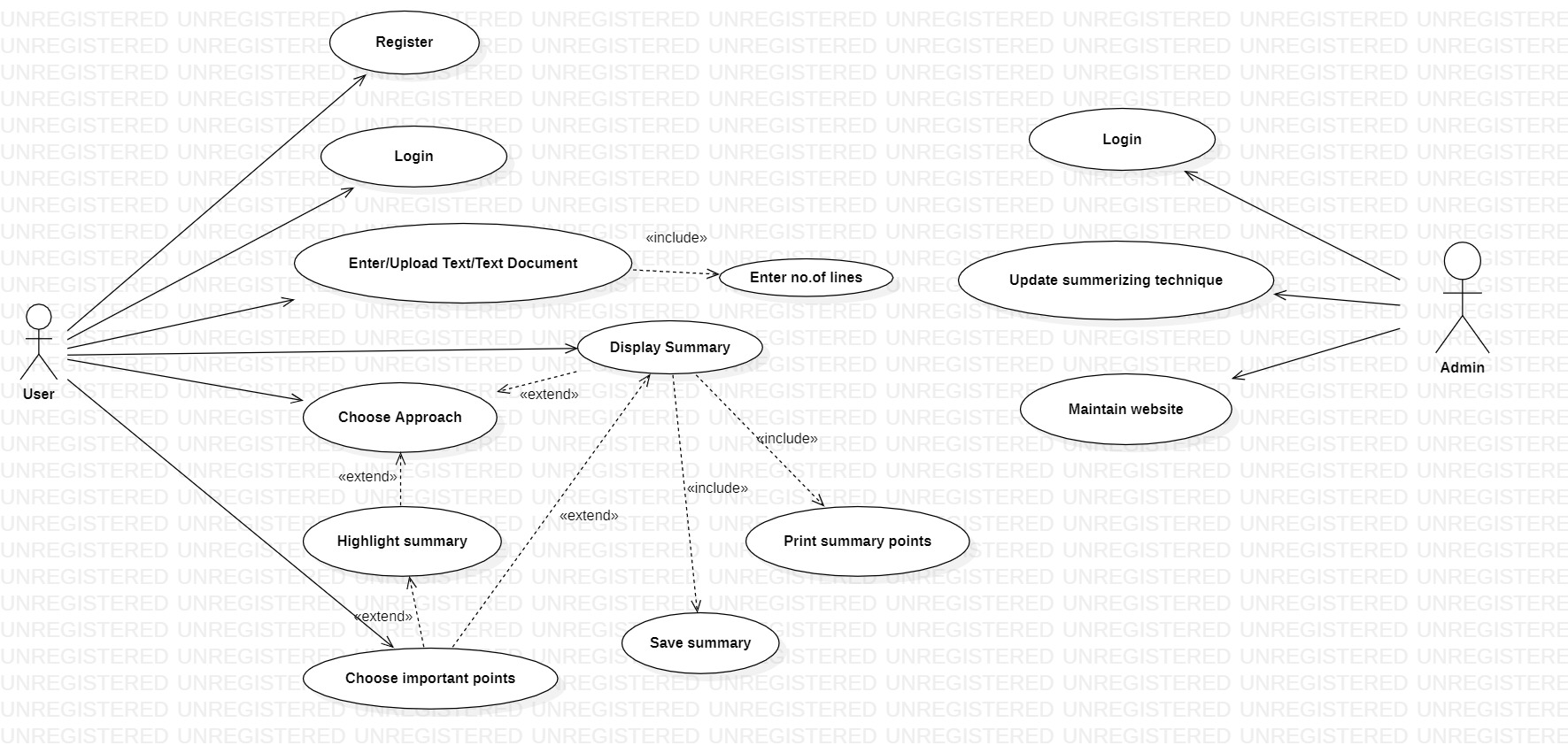
**TF-IDF (t, d) = TF (t, d) \* log (N / (DF + 1))**

**6.2 System Architecture**

**6.2.1 : Text blob architecture**



**6.2.2 : Use case diagram**



**CHAPTER 7**

**7.1 Implementation & Code**

**(FRONT END)**

Index.html

{% load static %}

<!DOCTYPE html>

<html lan="en" dir="ltr">

<head>

<meta charset="UTF-8">

<title>TextSummerization</title>

<!-- <link rel="stylesheet" type="text/css" href="{% static 'index.css' %}"> -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.1.0/css/font-awesome.min.css";>

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.3.1/css/all.css">

<link rel="stylesheet" href="path/to/font-awesome/css/font-awesome.min.css">

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.1/css/all.css"

integrity="sha384-gfdkjb5BdAXd+lj+gudLWI+BXq4IuLW5IT+brZEZsLFm++aCMlF1V92rMkPaX4PP" crossorigin="anonymous">

<meta name="viewport" content="width=device-width,intial-scale=1.0">

<script

src="https://kit.fontawesome.com/64d58efce2.js"

crossorigin="anonymous">

</script>

</head>

<body>

<nav>

<div class="navbar">

<div class="logo"><a href="#">Text Summerization</a></div>

<ul class="menu">

<li><a href="#Home">Home</a></li>

<li><a href="#Services">Services</a></li>

<li><a href="#Mission">Our mission</a></li>

<li><a href="#Contact">Contact us</a></li>

<li><a href="{% url 'login' %}"></a></li>

<span class="hello-msg">Hello,{{request.user}}</span>

<span ><a class="hello-msg" href="{% url 'logout' %}">Logout</a></span>

</ul>

</div>

</nav>

<section id="Home">

<pre> </pre>

<b><p>Welcome To Our Summerizer!<br><br>

Valuing Your Time Always<br><br>

Upload Any File And Get Its Summary instantly<br><br>

<a href="{% url 'login' %}">

<button class="btn1" style="width:130px; height:40%;">Get Started</button></p></b><br><br><br>

</a>

</section>

<section id="Services">

<div class="services">

<h1>Our Services</h1>

<div class="cen">

<div class="service">

<i class="fas fa-file-alt"></i>

<h2>TEXT SUMMERIZATION</h2><br>

<p>Providing you the summary of text using Extraction method.</p>

<p>We value your time.</p>

</div>

<div class="service">

<i class="fas fa-volume-up"></i>

<h2>SPEECH</h2><br>

<p>You can listen your summerized text.</p>

</div>

<div class="service">

<i class="fas fa-info-circle"></i>

<h2>DICTIONARY</h2><br>

<p>It provides the meaning,Parts of Speech,Synonym and Antonym of each word with example of summerized text.</p>

</div>

<br>

<div class="service">

<i class="fas fa-tags"></i>

<h2>NAMED ENTITY RECOGNITION</h2><br>

<div class="right">

<h2>Contact Us</h2>

<input type="text" class="field" placeholder="Your Name">

<input type="text" class="field" placeholder="Your Email">

<input type="text" class="field" placeholder="Phone">

<textarea placeholder="Message" class="field"></textarea>

<button class="btn">Send</button>

</section>

<div class "button">

<a href="#Home"><i class="fa fa-arrow-up fa-2x"></i>

</div>

</body>

</html>

**(BACK END )**

SUMMARY.PY

import streamlit as st

import nltk

import spacy

import spacy\_streamlit

from nltk.corpus import wordnet

from textblob import TextBlob

import streamlit.components.v1 as components

from pattern.web import Google

import codecs

import streamlit.components.v1 as stc

import webbrowser

from sklearn.feature\_extraction.text import TfidfVectorizer

from spacy.lang.en import English

import numpy as np

nlp = spacy.load('en\_core\_web\_sm')

try:

os.mkdir("temp")

except:

pass

input\_language = "en"

output\_language = "en"

tld = "com"

st.set\_page\_config(page\_title='Text Summarization',

page\_icon=':book:',

layout='wide')

st.markdown(

"""

<style>

.reportview-container {

background-color: #D3D3D3;

}

</style>

""",

unsafe\_allow\_html=True

)

def text\_downloader(raw\_text):

b64 = base64.b64encode(raw\_text.encode()).decode()

new\_filename = "new\_text\_file\_{}\_.txt".format(timestr)

st.markdown("#### Download File ###")

href = f'<a href="data:file/txt;base64,{b64}" download="{new\_filename}">Click Here!!</a>'

st.markdown(href,unsafe\_allow\_html=True)

def convert\_pdf\_to\_txt(path):

'''Convert pdf content from a file path to text

:path the file path

'''

rsrcmgr = PDFResourceManager()

codec = 'utf-8'

laparams = LAParams()

with io.StringIO() as retstr:

with TextConverter(rsrcmgr, retstr, codec=codec,

laparams=laparams) as device:

with open(path, 'rb') as fp:

interpreter = PDFPageInterpreter(rsrcmgr, device)

password = ""

maxpages = 0

caching = True

pagenos = set()

for page in PDFPage.get\_pages(fp,

pagenos,

maxpages=maxpages,

password=password,

caching=caching,

check\_extractable=True):

interpreter.process\_page(page)

return retstr.getvalue()

def main():

activites = ["Summary with text","Analyse with URL","Summarizing a file","Comparing different articles","Best of All","Text Cleaning",

"Tokenization","Grammar","Search"]

choice = st.sidebar.selectbox("Select Activity",activites)

#Summary with text

if choice == "Summary with text":

html\_temp = """

<div style= "background-color:#778899;"><p style="color:white; font-size:60px; width:70%;">Text Summarization</p></div>

"""

components.html(html\_temp)

text5 = st.text\_area("Input Text For Summary",height=300)

if st.button("summarize"):

doc = nlp(text5.replace("\n", ""))

sentences = [sent.text.strip() for sent in doc.sents]

sentence\_organizer = {k:v for v,k in enumerate(sentences)}

tf\_idf\_vectorizer = TfidfVectorizer(min\_df=2, max\_features=None,

strip\_accents='unicode',

analyzer='word',

token\_pattern=r'\w{1,}',

ngram\_range=(1, 3),

use\_idf=1,smooth\_idf=1,

sublinear\_tf=1,

stop\_words = 'english')

tf\_idf\_vectorizer.fit(sentences)

sentence\_vectors = tf\_idf\_vectorizer.transform(sentences)

sentence\_scores = np.array(sentence\_vectors.sum(axis=1)).ravel()

N = 3

top\_n\_sentences = [sentences[ind] for ind in np.argsort(sentence\_scores, axis=0)[::-1][:N]]

mapped\_top\_n\_sentences = [(sentence,sentence\_organizer[sentence]) for sentence in top\_n\_sentences]

# Ordering our top-n sentences in their original ordering

mapped\_top\_n\_sentences = sorted(mapped\_top\_n\_sentences, key = lambda x: x[1])

ordered\_scored\_sentences = [element[0] for element in mapped\_top\_n\_sentences]

summary2 = " ".join(ordered\_scored\_sentences)

def summarizer(text, tokenizer, max\_sent\_in\_summary=3):

# Create spacy document for further sentence level tokenization

doc = nlp(text5.replace("\n", ""))

# sentences = [sent.string.strip() for sent in doc.sents]

sentences = [sent.text.strip() for sent in doc.sents]

# Let's create an organizer which will store the sentence ordering to later reorganize the

# scored sentences in their correct order

sentence\_organizer = {k:v for v,k in enumerate(sentences)}

# Let's now create a tf-idf (Term frequnecy Inverse Document Frequency) model

tf\_idf\_vectorizer = TfidfVectorizer(min\_df=2, max\_features=None,

strip\_accents='unicode',

analyzer='word',

token\_pattern=r'\w{1,}',

ngram\_range=(1, 3),

use\_idf=1,smooth\_idf=1,

sublinear\_tf=1,

stop\_words = 'english')

# Passing our sentences treating each as one document to TF-IDF vectorizer

tf\_idf\_vectorizer.fit(sentences)

# Transforming our sentences to TF-IDF vectors

sentence\_vectors = tf\_idf\_vectorizer.transform(sentences)

# Getting sentence scores for each sentences

sentence\_scores = np.array(sentence\_vectors.sum(axis=1)).ravel()

# Getting top-n sentences

N = max\_sent\_in\_summary

top\_n\_sentences = [sentences[ind] for ind in np.argsort(sentence\_scores, axis=0)[::-1][:N]]

# Let's now do the sentence ordering using our prebaked sentence\_organizer

# Let's map the scored sentences with their indexes

mapped\_top\_n\_sentences = [(sentence,sentence\_organizer[sentence]) for sentence in top\_n\_sentences]

# Ordering our top-n sentences in their original ordering

mapped\_top\_n\_sentences = sorted(mapped\_top\_n\_sentences, key = lambda x: x[1])

ordered\_scored\_sentences = [element[0] for element in mapped\_top\_n\_sentences]

# Our final summary

summary = " ".join(ordered\_scored\_sentences)

return summary

st.success(summarizer(text=text5, tokenizer=nlp, max\_sent\_in\_summary=3))

text\_range= st.sidebar.slider("Summarize sentences Range",5,50)

text1 = st.text\_area("Input Text For Summary",height=250)

if st.button("summarize\_text\_with\_wordcount"):

doc = nlp(text1.replace("\n", ""))

sentences = [sent.text.strip() for sent in doc.sents]

sentence\_organizer = {k:v for v,k in enumerate(sentences)}

tf\_idf\_vectorizer = TfidfVectorizer(min\_df=2, max\_features=None,

strip\_accents='unicode',

analyzer='word',

token\_pattern=r'\w{1,}',

ngram\_range=(1, 3),

use\_idf=1,smooth\_idf=1,

sublinear\_tf=1,

stop\_words = 'english')

tf\_idf\_vectorizer.fit(sentences)

sentence\_vectors = tf\_idf\_vectorizer.transform(sentences)

sentence\_scores = np.array(sentence\_vectors.sum(axis=1)).ravel()

N = 3

top\_n\_sentences = [sentences[ind] for ind in np.argsort(sentence\_scores, axis=0)[::-1][:N]]

mapped\_top\_n\_sentences = [(sentence,sentence\_organizer[sentence]) for sentence in top\_n\_sentences]

# Ordering our top-n sentences in their original ordering

mapped\_top\_n\_sentences = sorted(mapped\_top\_n\_sentences, key = lambda x: x[1])

ordered\_scored\_sentences = [element[0] for element in mapped\_top\_n\_sentences]

summary2 = " ".join(ordered\_scored\_sentences)

def summarizer(text, tokenizer, max\_sent\_in\_summary=3):

# Create spacy document for further sentence level tokenization

doc = nlp(text1.replace("\n", ""))

# sentences = [sent.string.strip() for sent in doc.sents]

sentences = [sent.text.strip() for sent in doc.sents]

# Let's create an organizer which will store the sentence ordering to later reorganize the

# scored sentences in their correct order

sentence\_organizer = {k:v for v,k in enumerate(sentences)}

# Let's now create a tf-idf (Term frequnecy Inverse Document Frequency) model

tf\_idf\_vectorizer = TfidfVectorizer(min\_df=2, max\_features=None,

strip\_accents='unicode',

analyzer='word',

token\_pattern=r'\w{1,}',

ngram\_range=(1, 3),

use\_idf=1,smooth\_idf=1,

sublinear\_tf=1,

stop\_words = 'english')

# Passing our sentences treating each as one document to TF-IDF vectorizer

tf\_idf\_vectorizer.fit(sentences)

# Transforming our sentences to TF-IDF vectors

sentence\_vectors = tf\_idf\_vectorizer.transform(sentences)

# Getting sentence scores for each sentences

sentence\_scores = np.array(sentence\_vectors.sum(axis=1)).ravel()

# Getting top-n sentences

N = max\_sent\_in\_summary

top\_n\_sentences = [sentences[ind] for ind in np.argsort(sentence\_scores, axis=0)[::-1][:N]]

# Let's now do the sentence ordering using our prebaked sentence\_organizer

# Let's map the scored sentences with their indexes

mapped\_top\_n\_sentences = [(sentence,sentence\_organizer[sentence]) for sentence in top\_n\_sentences]

# Ordering our top-n sentences in their original ordering

mapped\_top\_n\_sentences = sorted(mapped\_top\_n\_sentences, key = lambda x: x[1])

ordered\_scored\_sentences = [element[0] for element in mapped\_top\_n\_sentences]

# Our final summary

summary = " ".join(ordered\_scored\_sentences)

return summary

st.markdown(f"## Summary :")

# Summarizing

st.warning(summarizer(text=text1, tokenizer=nlp, max\_sent\_in\_summary=text\_range))

elif choice == "Summarizing a file":

html\_temp2 = """

<div style= "background-color:#778899;"><p style="color:white;font-size:60px;">Summarizing a file</p></div>

"""

components.html(html\_temp2)

text\_range= st.sidebar.slider("Summarize sentences Range",5,50)

docx\_file = st.file\_uploader("Upload File",type=['docx','pdf'])

if st.button("Process"):

if docx\_file is not None:

file\_details = {"Filename":docx\_file.name,"FileType":docx\_file.type,"FileSize":docx\_file.size}

st.write(file\_details)

if docx\_file.type == "application/pdf":

try:

with pdfplumber.open(docx\_file) as pdf:

result=convert\_pdf\_to\_txt(docx\_file.name)

doc = nlp(result.replace("\n", ""))

sentences = [sent.text.strip() for sent in doc.sents]

sentence\_organizer = {k:v for v,k in enumerate(sentences)}

tf\_idf\_vectorizer = TfidfVectorizer(min\_df=2, max\_features=None,

strip\_accents='unicode',

analyzer='word',

token\_pattern=r'\w{1,}',

ngram\_range=(1, 3),

use\_idf=1,smooth\_idf=1,

sublinear\_tf=1,

stop\_words = 'english')

tf\_idf\_vectorizer.fit(sentences)

sentence\_vectors = tf\_idf\_vectorizer.transform(sentences)

sentence\_scores = np.array(sentence\_vectors.sum(axis=1)).ravel()

N = 3

top\_n\_sentences = [sentences[ind] for ind in np.argsort(sentence\_scores, axis=0)[::-1][:N]]

mapped\_top\_n\_sentences = [(sentence,sentence\_organizer[sentence]) for sentence in top\_n\_sentences]

# Ordering our top-n sentences in their original ordering

mapped\_top\_n\_sentences = sorted(mapped\_top\_n\_sentences, key = lambda x: x[1])

ordered\_scored\_sentences = [element[0] for element in mapped\_top\_n\_sentences]

summary2 = " ".join(ordered\_scored\_sentences)

def summarizer(text, tokenizer, max\_sent\_in\_summary=3):

# Create spacy document for further sentence level tokenization

doc = nlp(result.replace("\n", ""))

# sentences = [sent.string.strip() for sent in doc.sents]

sentences = [sent.text.strip() for sent in doc.sents]

# Let's create an organizer which will store the sentence ordering to later reorganize the

# scored sentences in their correct order

sentence\_organizer = {k:v for v,k in enumerate(sentences)}

# Let's now create a tf-idf (Term frequnecy Inverse Document Frequency) model

tf\_idf\_vectorizer = TfidfVectorizer(min\_df=2, max\_features=None,

strip\_accents='unicode',

analyzer='word',

token\_pattern=r'\w{1,}',

ngram\_range=(1, 3),

use\_idf=1,smooth\_idf=1,

sublinear\_tf=1,

stop\_words = 'english')

# Passing our sentences treating each as one document to TF-IDF vectorizer

tf\_idf\_vectorizer.fit(sentences)

# Transforming our sentences to TF-IDF vectors

sentence\_vectors = tf\_idf\_vectorizer.transform(sentences)

# Getting sentence scores for each sentences

sentence\_scores = np.array(sentence\_vectors.sum(axis=1)).ravel()

# Getting top-n sentences

N = max\_sent\_in\_summary

top\_n\_sentences = [sentences[ind] for ind in np.argsort(sentence\_scores, axis=0)[::-1][:N]]

# Let's now do the sentence ordering using our prebaked sentence\_organizer

# Let's map the scored sentences with their indexes

mapped\_top\_n\_sentences = [(sentence,sentence\_organizer[sentence]) for sentence in top\_n\_sentences]

# Ordering our top-n sentences in their original ordering

mapped\_top\_n\_sentences = sorted(mapped\_top\_n\_sentences, key = lambda x: x[1])

ordered\_scored\_sentences = [element[0] for element in mapped\_top\_n\_sentences]

# Our final summary

summary = " ".join(ordered\_scored\_sentences)

return summary

hide\_st\_style = """

<style>

#MainMenu {visibility: hidden;}

footer {visibility: hidden;}

header {visibility: hidden;}

</style>

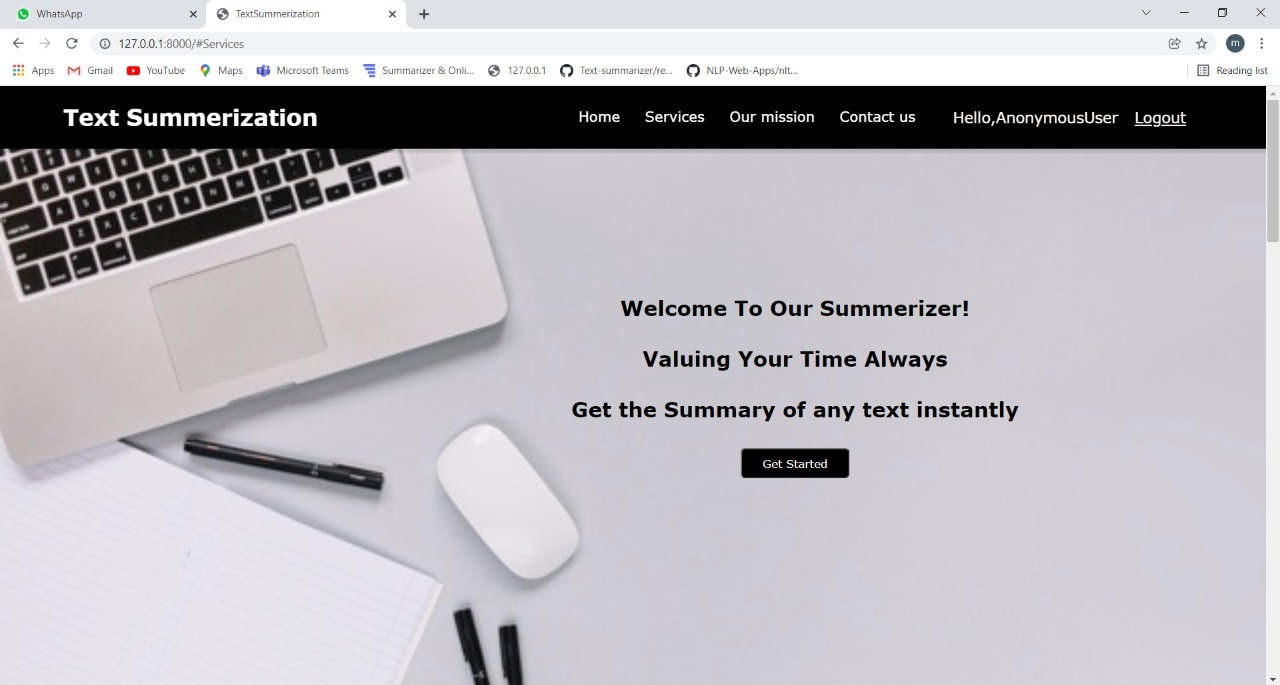
"""

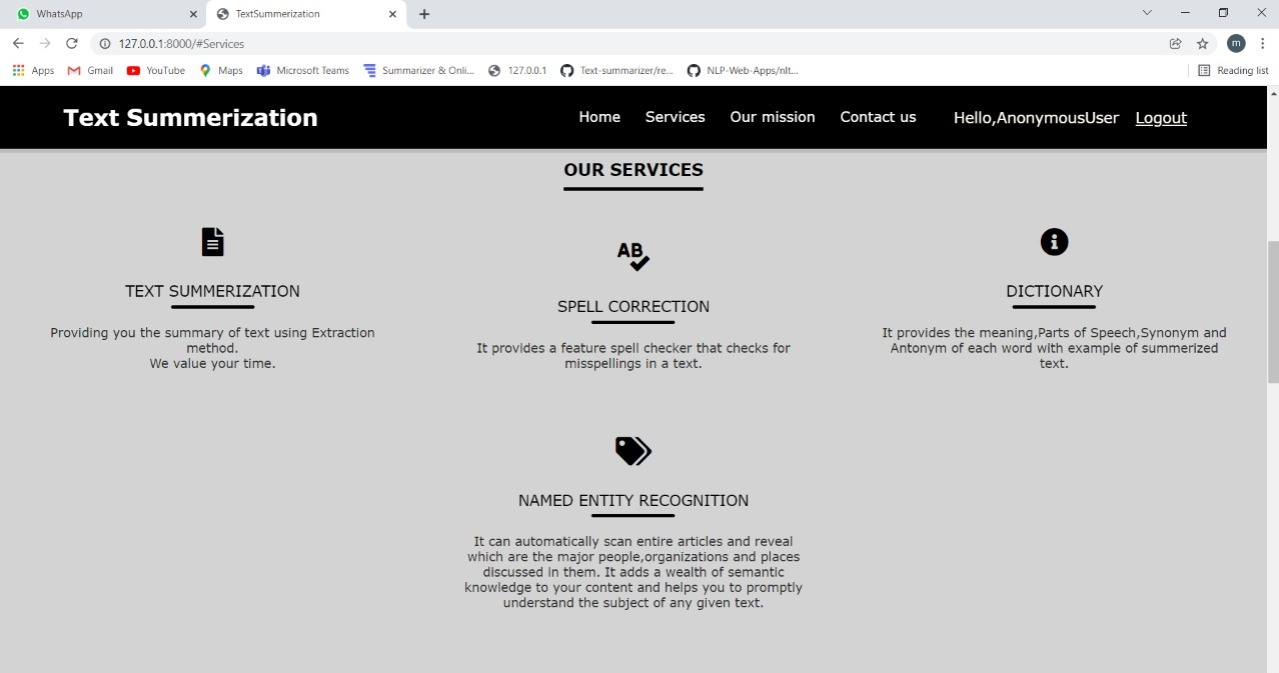
st.markdown(hide\_st\_style, unsafe\_allow\_html=True)

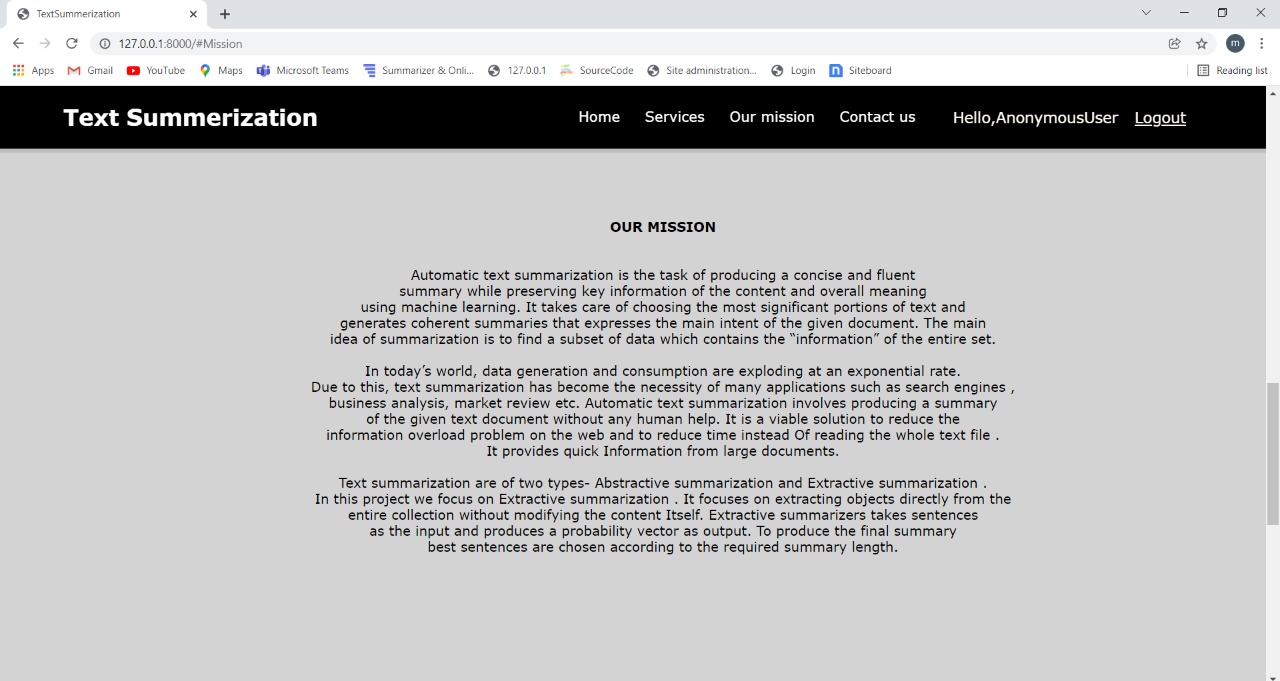
if \_\_name\_\_ == '\_\_main\_\_':

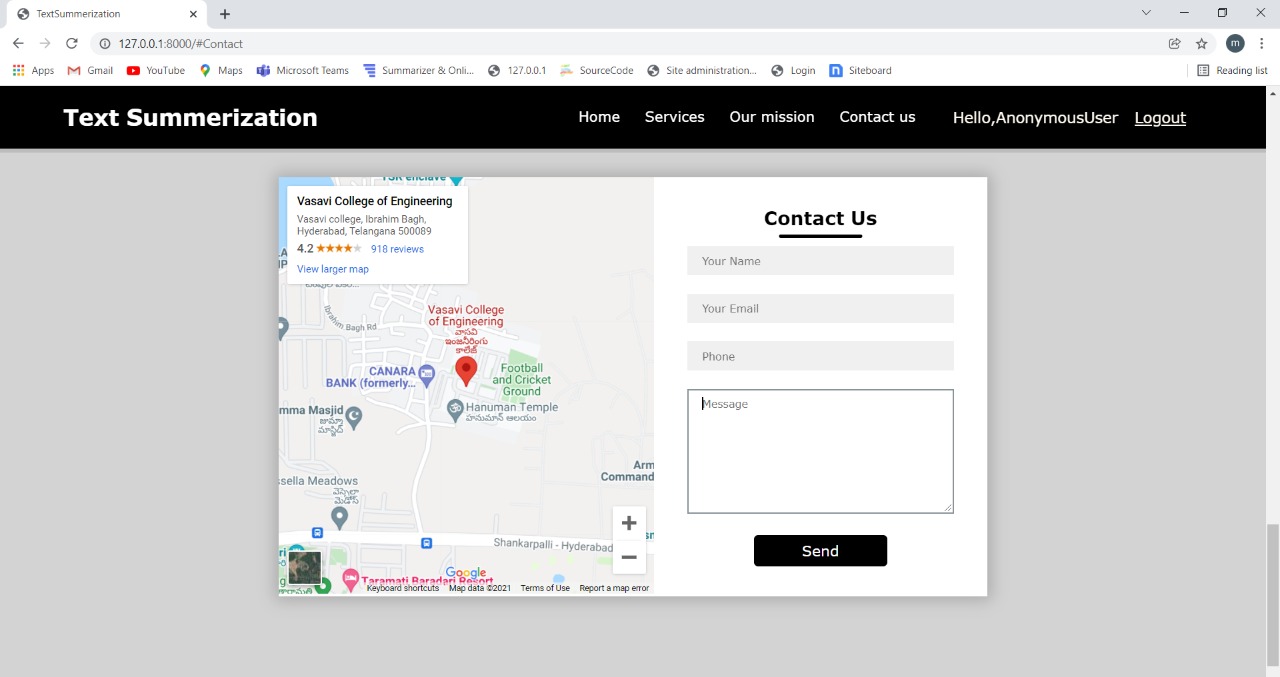
main()

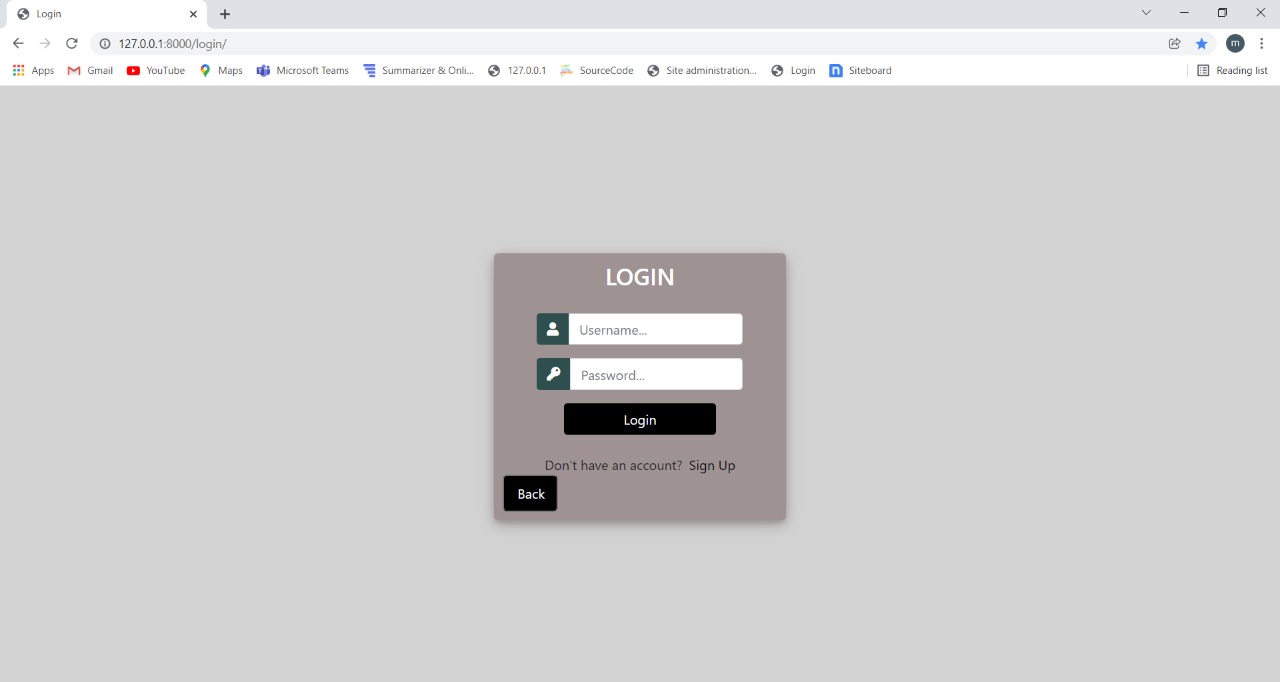
**CHAPTER 8: RESULTS**

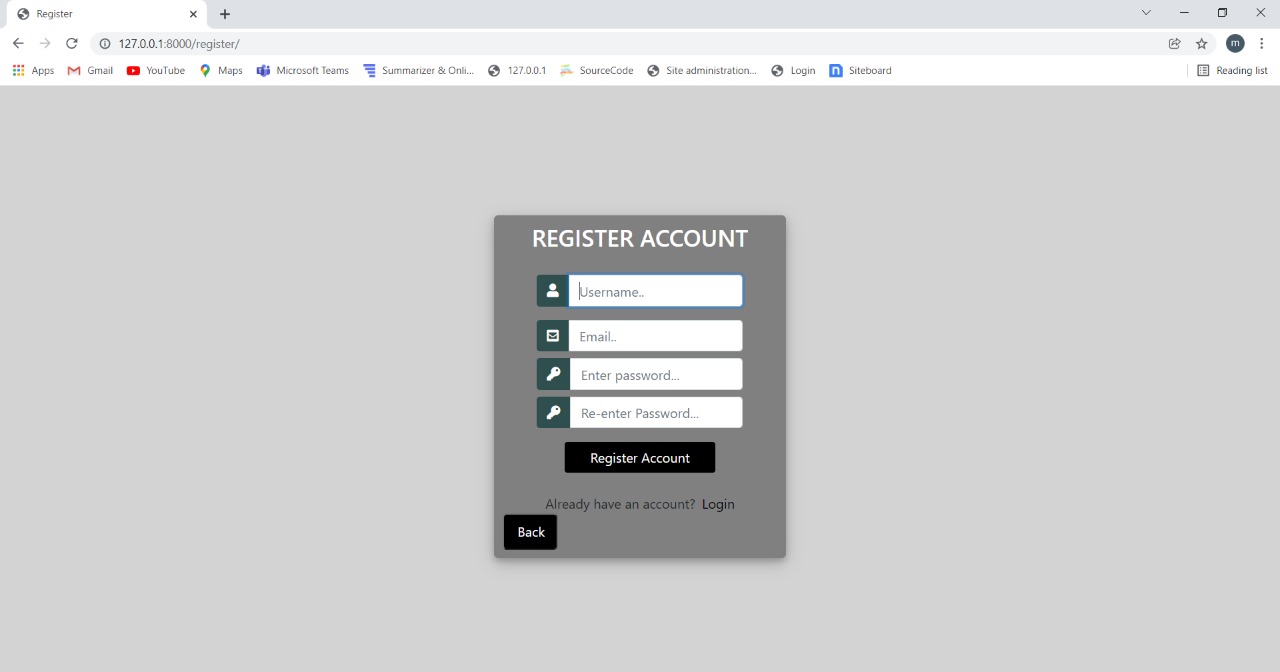
****

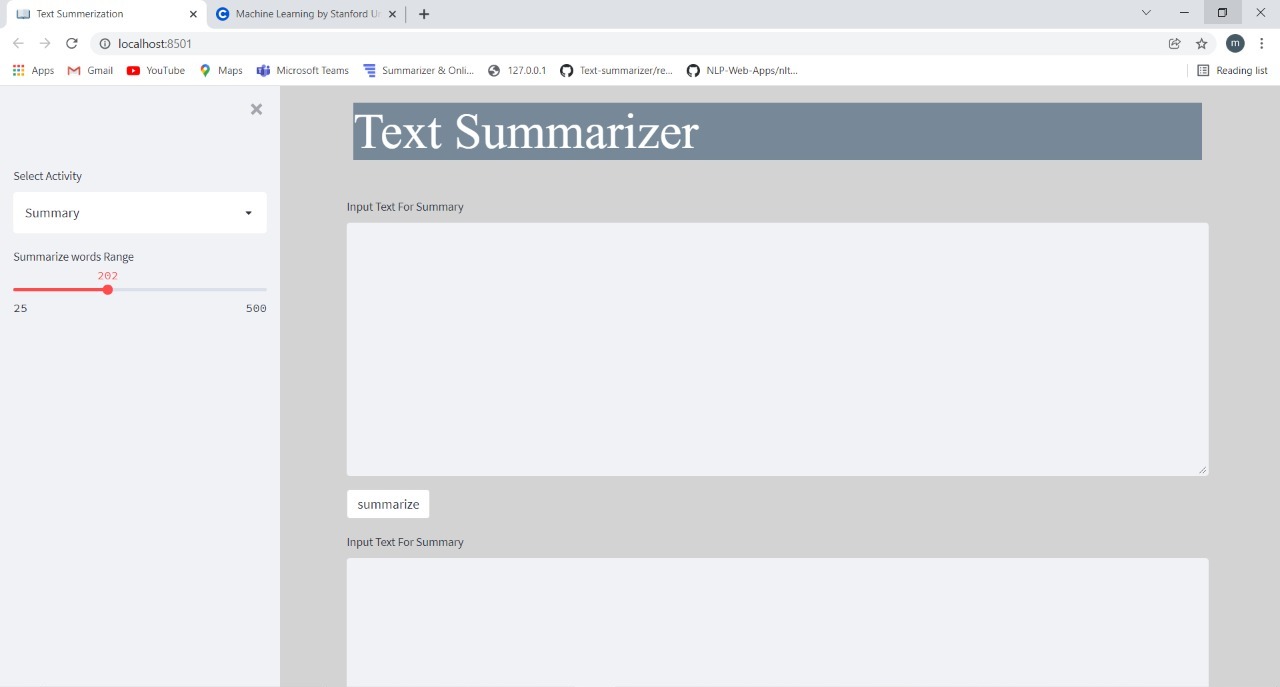
****

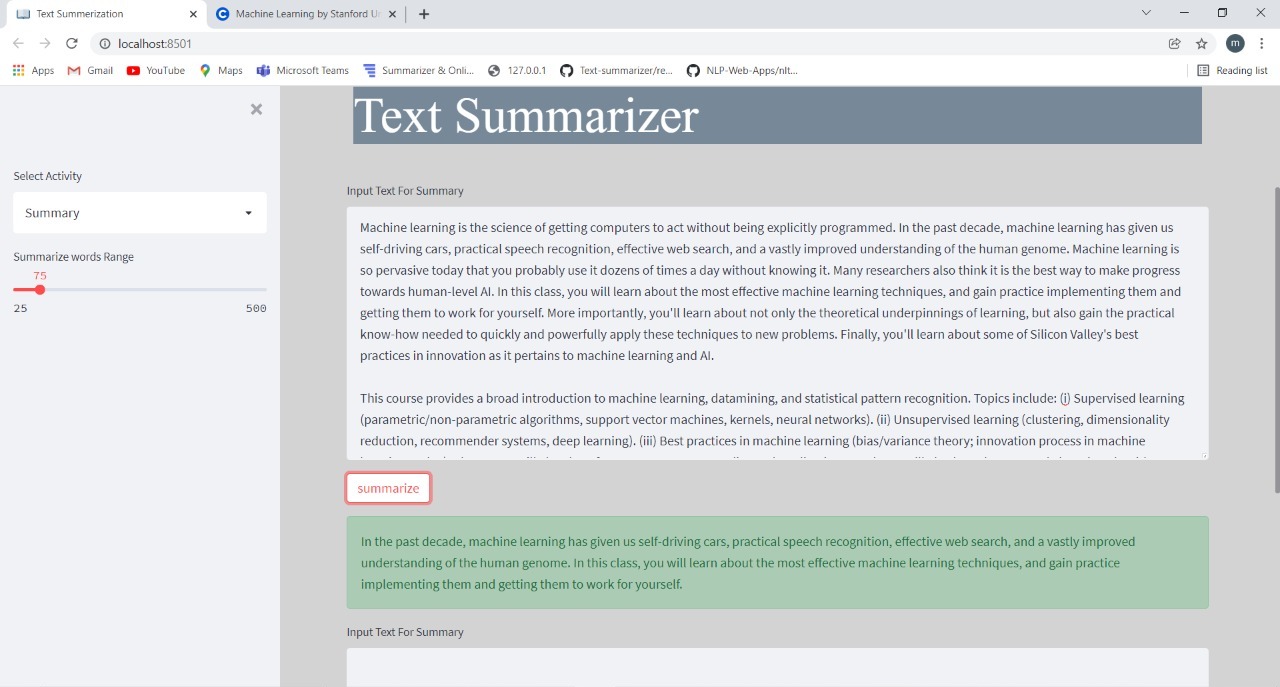
****

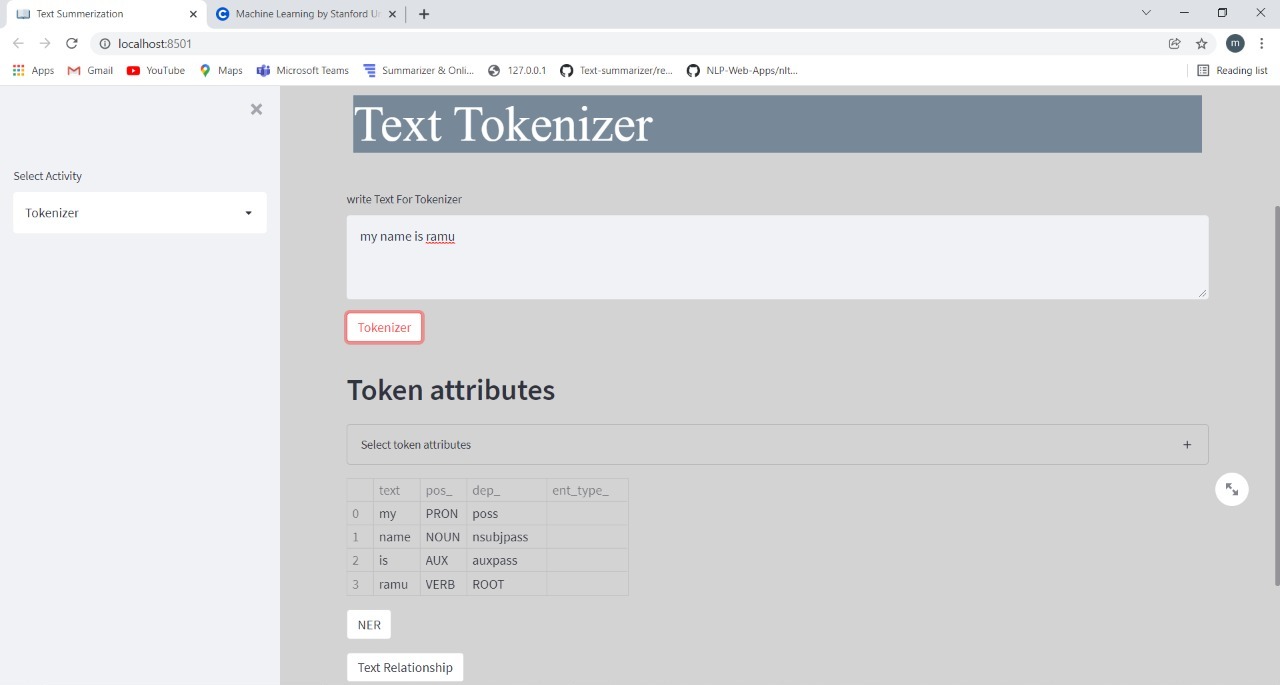
****

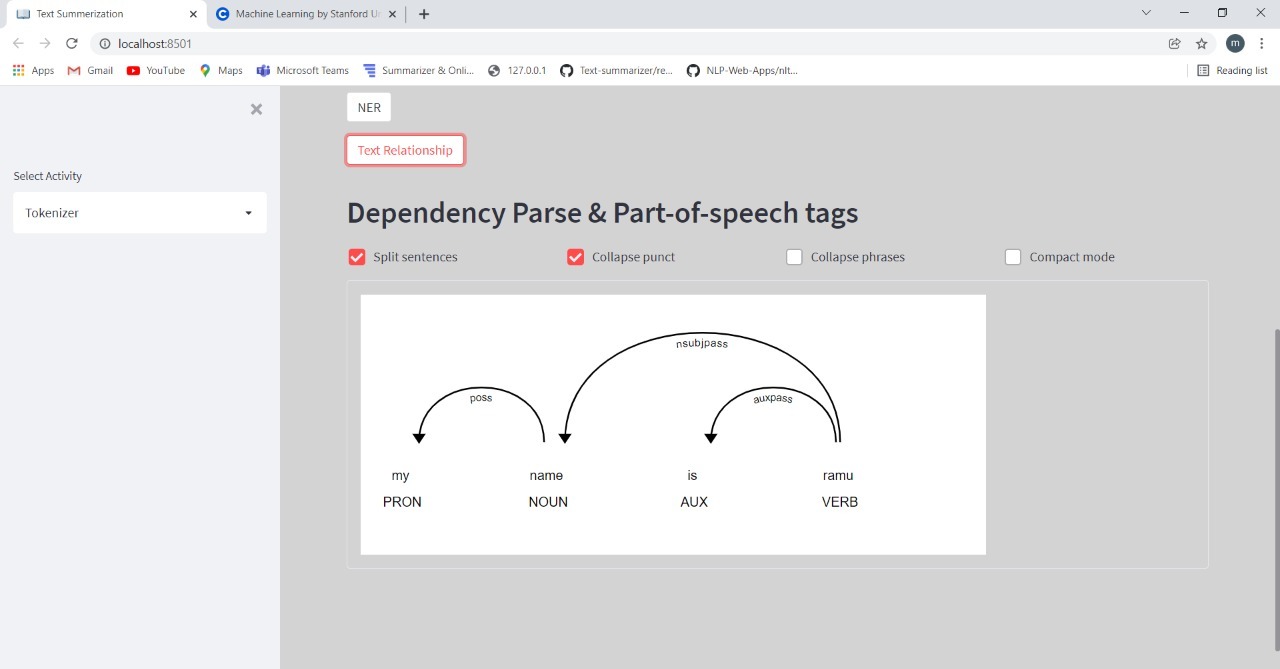
****

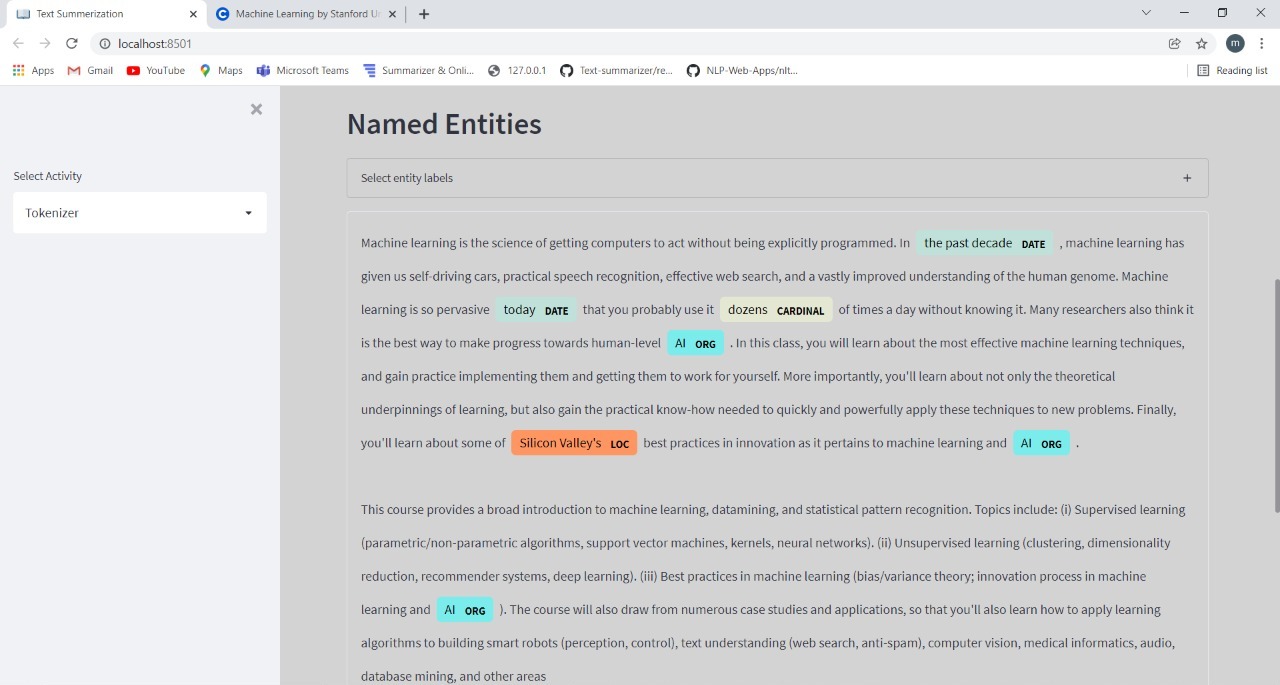
****

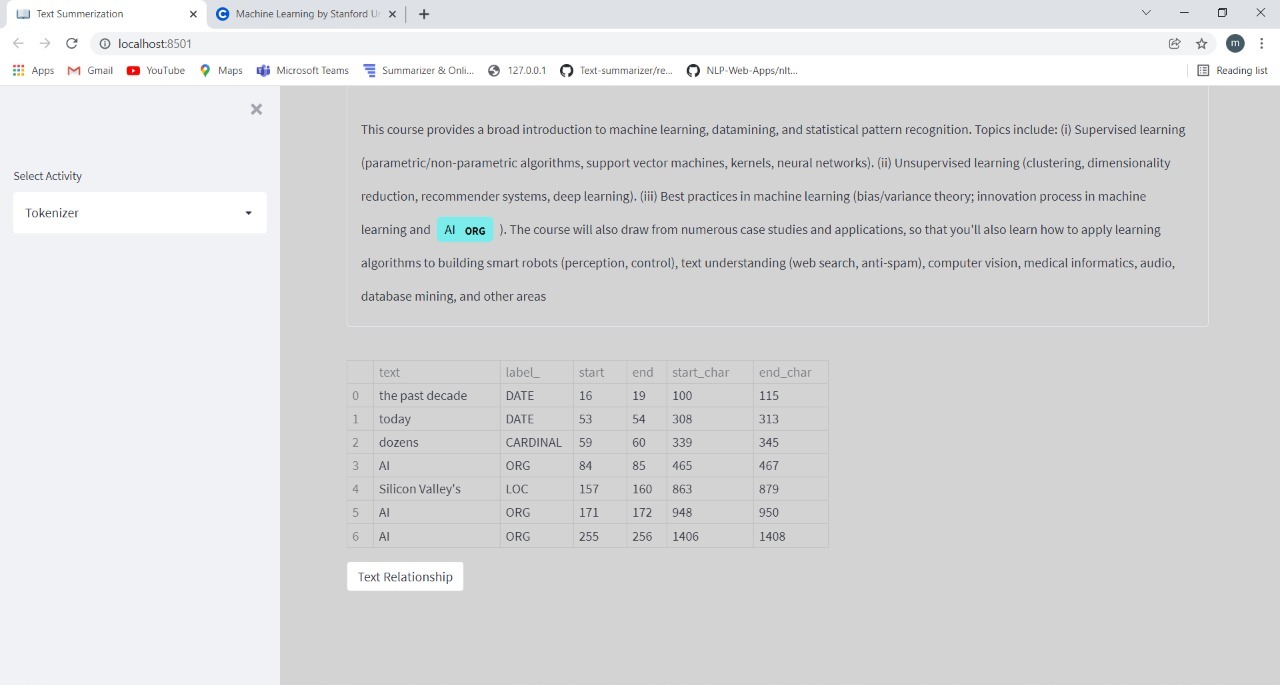
****

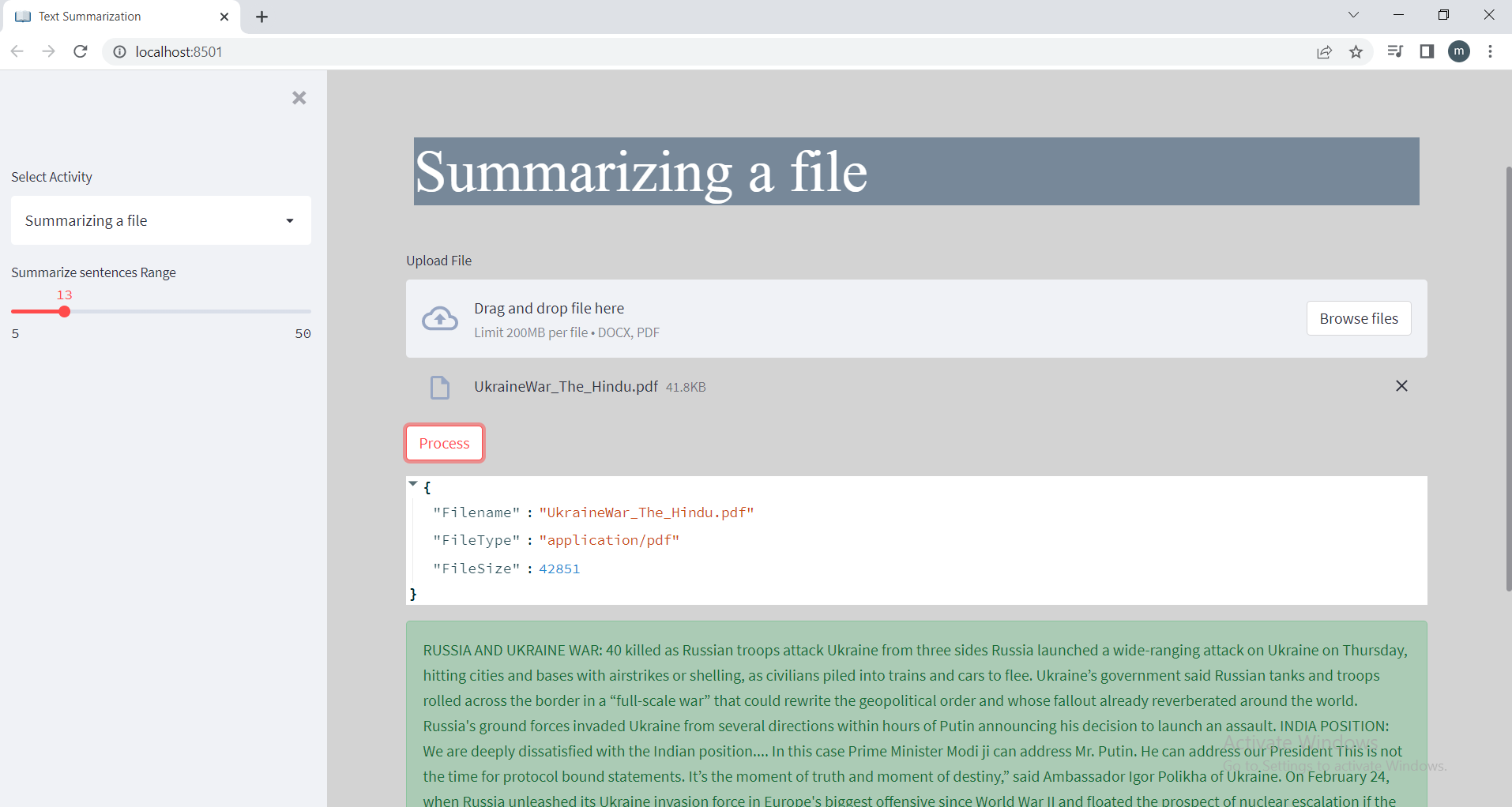
****

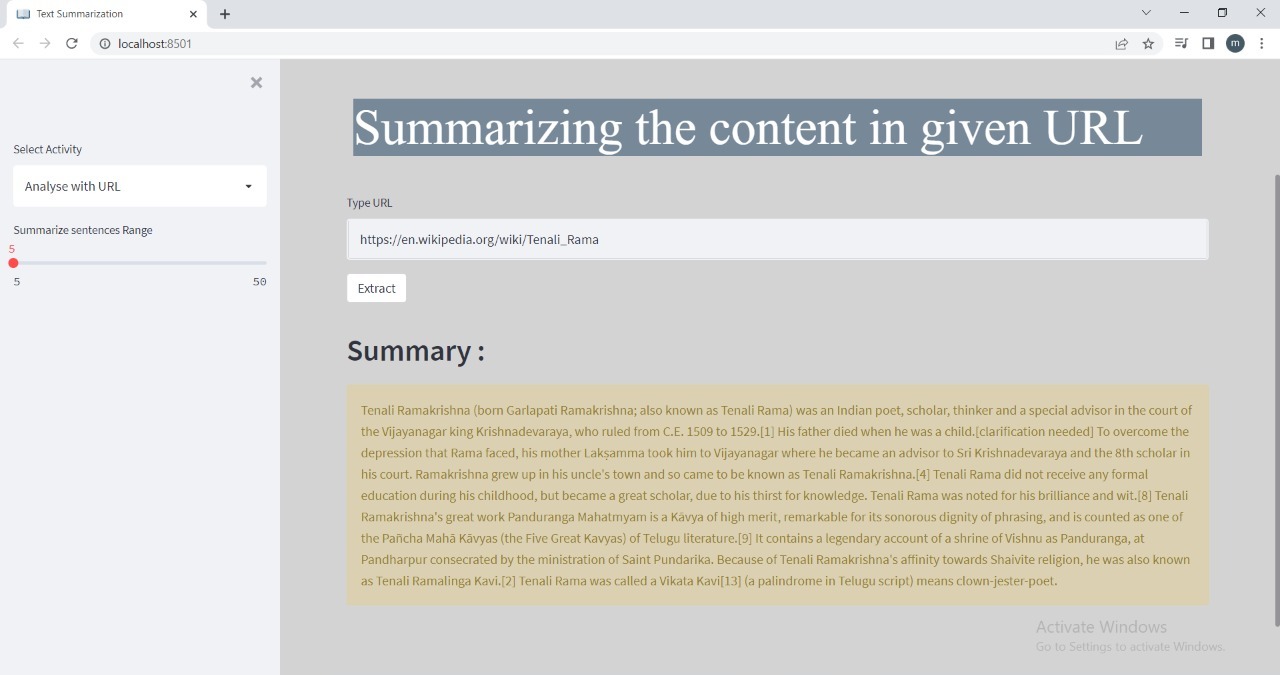
****

****

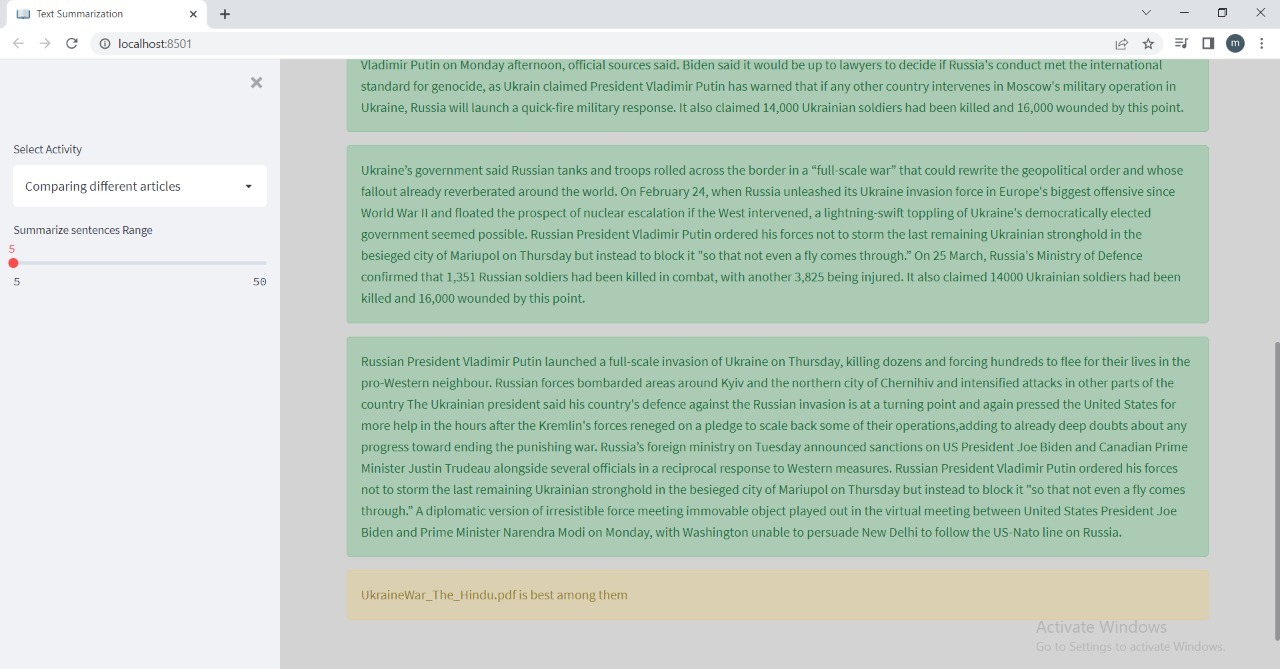
****

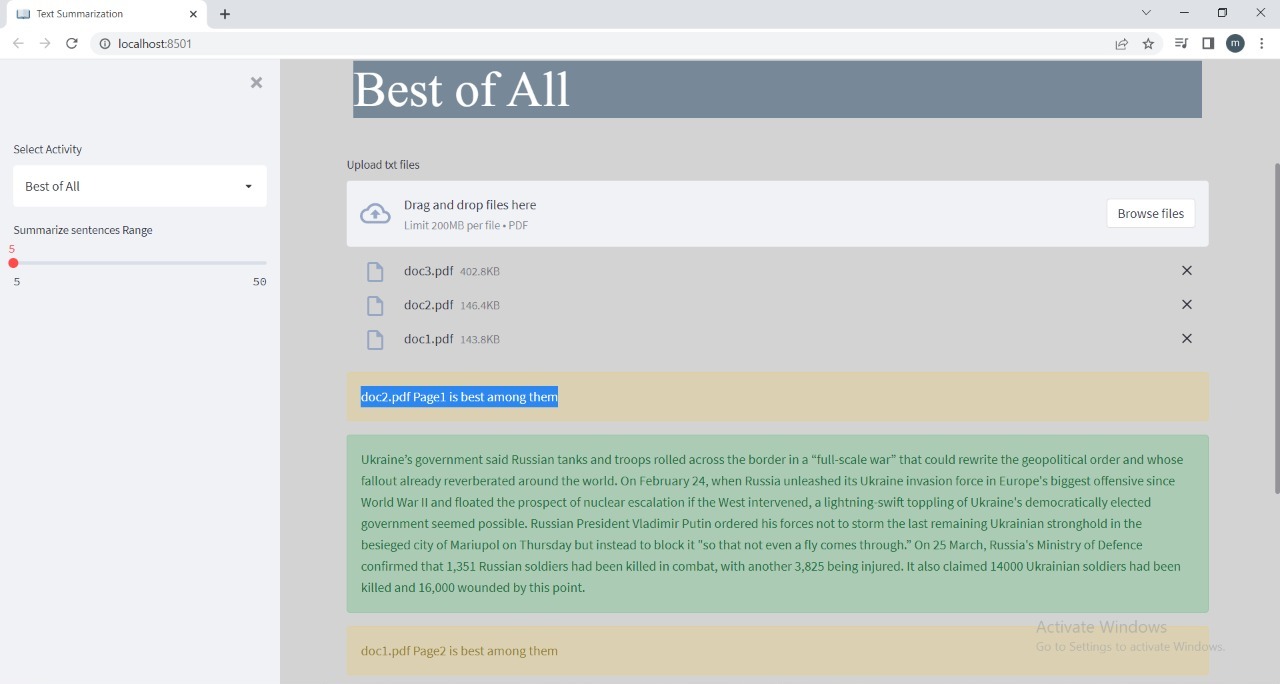
****

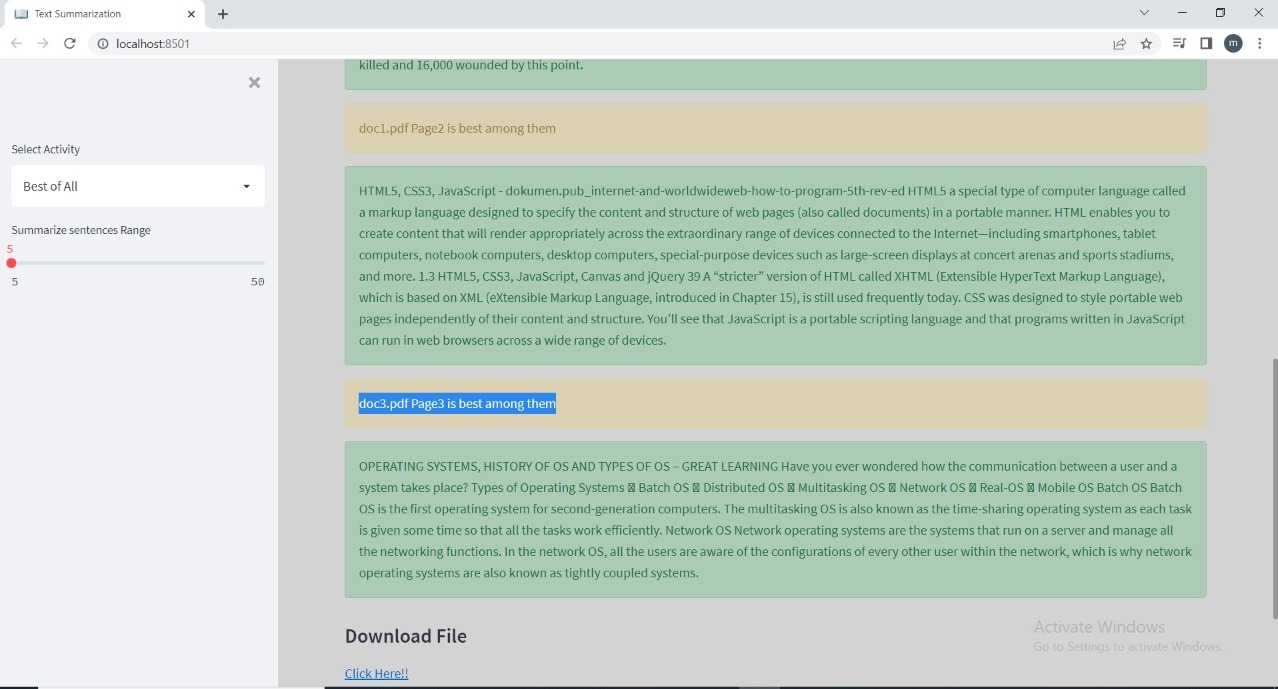


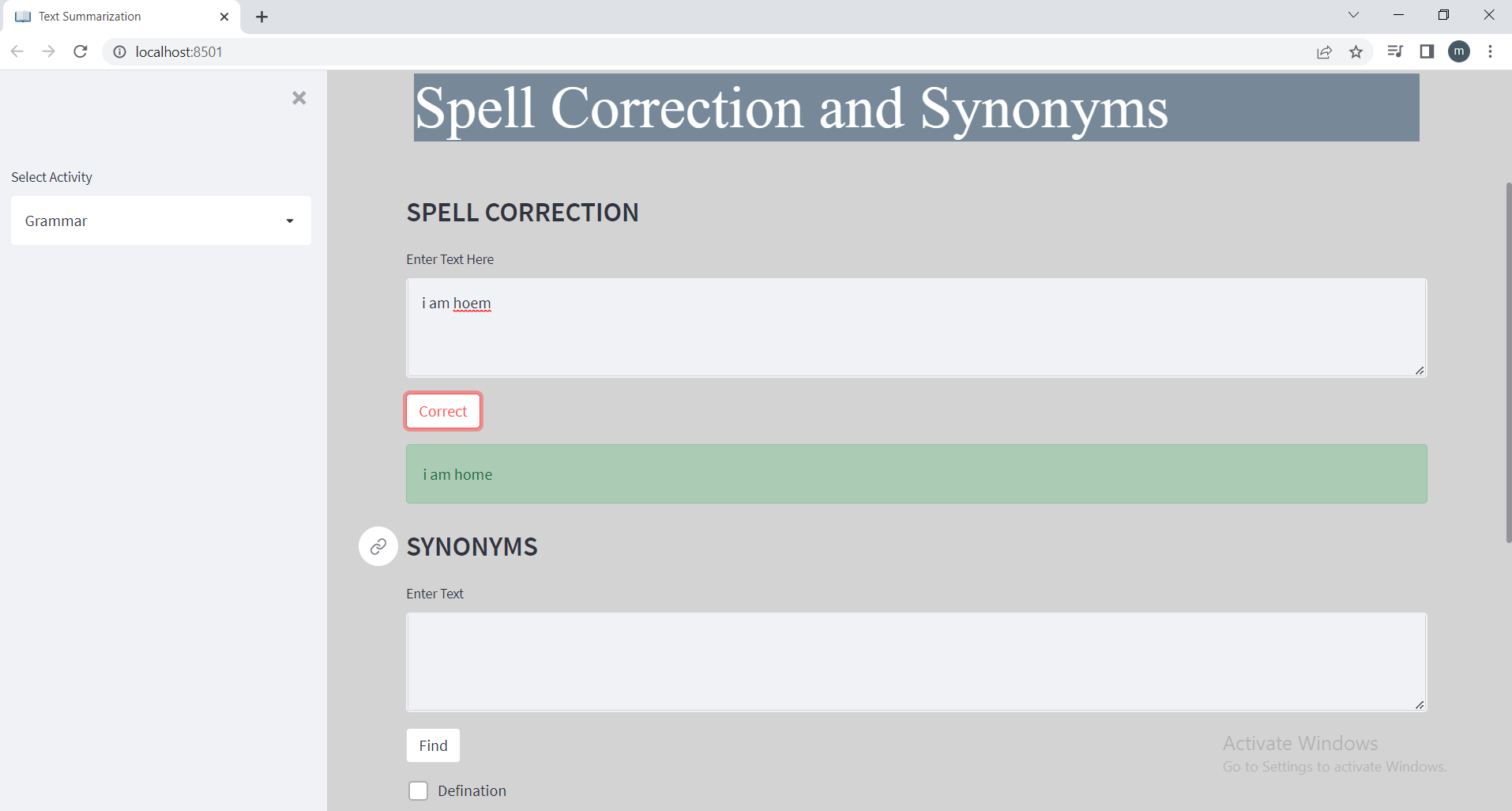


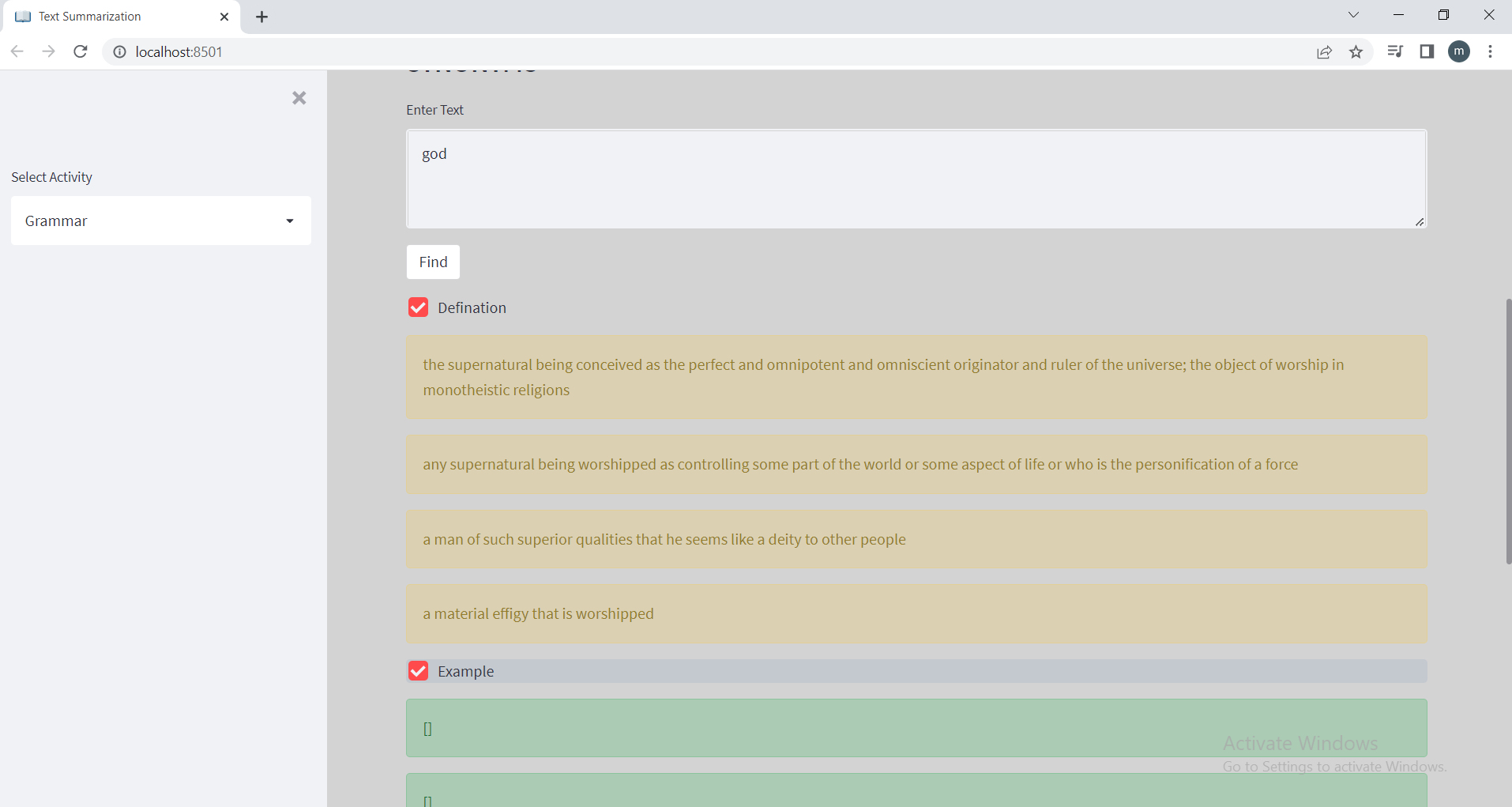












**CHAPTER 9**

**CONCLUSION & FUTURE SCOPE**

9.1 Conclusion

The project meets the objectives and requirements of the system. It successfully Summarizes the text, the text present in the given URL, the text present in the uploaded document, Tokenizes it & can recognize Named Entities, compares the different uploaded articles and predicts the best record among them, and gives synonyms and antonyms of a given text and corrects the spellings.

9.2 Future Scope

* To make an application for android users.
* Clustering the documents with the only best content from the given documents.
* Text Summarization in multiple languages.

**CHAPTER 10**

**REFERENCES**

* [**https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25**](https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25)
* [**https://www.mygreatlearning.com/blog/text-summarization-in-python/**](https://www.mygreatlearning.com/blog/text-summarization-in-python/)
* [**https://www.youtube.com/watch?v=eWKg4Id30\_k**](https://www.youtube.com/watch?v=eWKg4Id30_k)
* [**https://www.youtube.com/watch?v=ryQVfi580OI&feature=youtu.be**](https://www.youtube.com/watch?v=ryQVfi580OI&feature=youtu.be)
* [**https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/**](https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/)