```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 12:07:27 2024

@author: HP
"""
#*********************Works*********

import numpy as np
import pickle
import streamlit as st
from urllib.parse import urlparse, parse_qs
import ipaddress
import re

# Load the Decision Tree model
decision_tree_model_path = "C:/Users/HP/OneDrive/Desktop/New
folder/RandomForest_model.sav"# Update the path if needed
decision_tree_model = pickle.load(open(decision_tree_model_path, 'rb'))

# Define the feature extraction functions
def get_url_len(url):
    return len(url)

def extract_domain_length(url):
    try:
        if not url.startswith(('http://', 'https://')):
            url = 'http://' + url
        parsed_url = urlparse(url)
        domain = parsed_url.netloc
        return len(domain) if domain else 0
```

```
    except Exception:
        return 0

def has_http(url):
    return 1 if url.startswith('http://') else 0

def has_https(url):
    return 1 if url.startswith('https://') else 0

def count_dots(url):
    return url.count('.')

def count_dashes(url):
    return url.count('-')

def count_underscores(url):
    return url.count('_')

def count_ques(url):
    return url.count('?')

def count_slashes(url):
    return url.count('/')

def count_special_chars(url):
    non_alpha_num = re.findall(r'\W', url)
    return len(non_alpha_num)

def count_digits(url):
    digits = re.findall(r'\d', url)
    return len(digits)
```

```
def count_letters(url):
    letters = re.findall(r'[a-zA-Z]', url)
    return len(letters)

def has_ip_address(url):
    try:
        parsed_url = urlparse(url)
        if parsed_url.hostname:
            ip = ipaddress.ip_address(parsed_url.hostname)
            return isinstance(ip, (ipaddress.IPv4Address, ipaddress.IPv6Address))
    except ValueError:
        pass
    return 0

def count_url_parameters(url):
    parsed_url = urlparse(url)
    query = parsed_url.query
    parameters = parse_qs(query)
    return len(parameters)

def check_php_in_url(url):
    return 1 if 'php' in url.lower() else 0

def check_html_in_url(url):
    return 1 if 'html' in url.lower() else 0

def check_mal_tld(url):
    tld_list = ['.tk', '.buzz', '.xyz', '.top', '.ga', '.ml', '.info', '.cf', '.gq', '.icu']
    parsed_url = urlparse(url)
    netloc = parsed_url.netloc.lower()
```

```
    return 1 if any(netloc.endswith(tld) for tld in tld_list) else 0


def is_shortened_url(url):
    shortened_services = [
        "bit.ly", "tinyurl.com", "goo.gl", "t.co", "ow.ly", "buff.ly", "is.gd", "adf.ly"
    ]
    parsed_url = urlparse(url)
    netloc = parsed_url.netloc.lower()
    return 1 if any(service in netloc for service in shortened_services) else 0


# Function to generate features from a URL
def get_numerical_values(url):
    return {
        'url_length': get_url_len(url),
        'domain_length': extract_domain_length(url),
        'check_http': has_http(url),
        'check_https': has_https(url),
        'dot_count': count_dots(url),
        'dash_count': count_dashes(url),
        'underscore_count': count_underscores(url),
        'ques_count': count_ques(url),
        'slash_count': count_slashes(url),
        'special_chars_count': count_special_chars(url),
        'digits_count': count_digits(url),
        'letters_count': count_letters(url),
        'has_ip': has_ip_address(url),
        'param_count': count_url_parameters(url),
        'has_php': check_php_in_url(url),
        'has_html': check_html_in_url(url),
        'mal_tld': check_mal_tld(url),
        'shortened': is_shortened_url(url),
```

```
    }


# Define the Streamlit app
def main():
    st.title("Malicious URL Prediction App ")
    st.markdown("Enter a URL to predict if it's malicious or not ")

    # User input
    url_input = st.text_input("Enter URL:")

    if st.button("Predict"):
        if url_input:
            # Extract features
            features = get_numerical_values(url_input)
            feature_values = np.array(list(features.values())).reshape(1, -1)

            # Make prediction
            prediction = decision_tree_model.predict(feature_values)[0]
            class_mapping = {0: 'safe', 1: 'Unsafe', 2: 'Unsafe', 3: 'Unsafe'}
            result = class_mapping.get(prediction, "Unknown")

            # Display result
            st.success(f"The URL is predicted as: **{result}**")
        else:
            st.error("Please enter a valid URL.")

if __name__ == "__main__":
    main()
```