Text

Description automatically generated

**Activity based**

**Project Report on**

**Computer Networks**

**Submitted to Vishwakarma University, Pune**

**Under the Initiative of**

**Contemporary Curriculum, Pedagogy, and Practice (C2P2)**

**By**

**Arun Adhikari**

**SRN No : 202100406**

**Roll No : 38**

**Div : G**

**Third Year Engineering**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2023-2024**

**Project Statement :**

Simulate HTTP web client program to download a web page using TCP sockets

**Problem Description :**

The goal of this project is to create an HTTP web client capable of downloading and displaying webpage content over a TCP connection using sockets. Traditional web browsers use such connections to fetch and render websites, but here we’re implementing a simplified client that directly sends an HTTP GET request to a specified server and displays the HTTP response headers and HTML content.

This project allows users to enter a URL, fetches the webpage's data using socket programming, and displays both the HTTP headers and content in a structured and visually appealing GUI. It simulates a basic, manually controlled browser, helping users understand how web clients work at a fundamental level.

**Project Modules:**

**GUI Module:**

* **Description**: This module creates the graphical user interface for the project using tkinter. The GUI contains fields for URL input, status updates, and display areas for webpage headers and content.
* **Key Components**:
  + URL entry field and button to initiate the webpage fetch.
  + Text areas to display HTTP headers and HTML content.
  + Status indicators to show the fetch operation's progress and outcomes.
* **Socket Communication Module**:
* **Description**: This module handles the core networking functions using Python's socket library. It connects to the specified server, sends an HTTP GET request, and retrieves the HTTP response.
* **Key Components**:
  + TCP socket creation and connection.
  + Formulating and sending an HTTP GET request.
  + Receiving and processing HTTP response data in chunks for display.
* **Data Parsing and Display Module**:
* **Description**: This module extracts, organizes, and displays the data received from the server. It separates HTTP headers from the HTML body and formats them for display in the GUI.
* **Key Components**:
  + Parsing the HTTP response to separate headers and body.
  + Handling errors gracefully and updating status indicators in the GUI.
  + Displaying headers and content in respective ScrolledText fields for better readability.

**Implementation :**

**GUI Setup**:****

* + Use tkinter to set up the main GUI window with an entry field for the URL, a "Fetch Webpage" button, and two ScrolledText fields for the headers and content.
  + Style the GUI elements with background colors, padding, and font adjustments for a professional appearance.
  + Add a status label to provide real-time feedback, such as “Fetching webpage…” or “Webpage fetched successfully.”

1. **Socket Connection and HTTP Request**:
   * Retrieve the URL from the entry field and parse it to extract the host and path.
   * Use the socket library to create a TCP connection to the specified host on port 80 (HTTP).
   * Formulate an HTTP GET request and send it to the server using the open socket.
   * Continuously receive data from the server in chunks until the full response is obtained.
2. **Response Processing and Display**:
   * Decode the response data to extract and separate HTTP headers and the HTML body.
   * Display the headers in the left ScrolledText area and the body (HTML content) in the right ScrolledText area.
   * Update the status label and show success or error messages to guide users through the fetch process.
3. **Error Handling and Edge Cases**:
   * Handle cases like invalid URLs or connection failures with appropriate error messages.
   * Ensure the GUI is responsive by disabling the "Fetch Webpage" button during processing and enabling it again once complete.
   * Gracefully handle potential encoding issues by ignoring errors when decoding the server response.

**Code:**

import socket

import tkinter as tk

from tkinter import scrolledtext, messagebox

from tkinter import ttk

# Function to connect to server and retrieve the webpage

def fetch\_webpage():

# Get the URL entered by the user

url = url\_entry.get().strip()

if not url.startswith("http://") and not url.startswith("https://"):

display\_error("URL must start with 'http://' or 'https://'.")

return

# Parse URL to get the host and path

host, path = parse\_url(url)

if not host:

display\_error("Invalid URL format.")

return

# Disable button and show loading status

fetch\_button.config(state="disabled")

status\_label.config(text="Fetching webpage...", foreground="blue")

# Create a TCP socket

try:

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

# Connect to the server

s.connect((host, 80))

# Send HTTP GET request

request = f"GET {path} HTTP/1.1\r\nHost: {host}\r\nConnection: close\r\n\r\n"

s.sendall(request.encode())

# Receive the response

response = b""

while True:

data = s.recv(4096)

if not data:

break

response += data

# Decode response and separate headers and content

response\_text = response.decode("utf-8", errors="ignore")

headers, \_, body = response\_text.partition("\r\n\r\n")

# Update status, enable button, and display fetched content

status\_label.config(text="Webpage fetched successfully!", foreground="green")

display\_content(headers, body)

except Exception as e:

display\_error(f"Error fetching webpage: {e}")

finally:

fetch\_button.config(state="normal")

# Helper function to parse URL and extract host and path

def parse\_url(url):

if url.startswith("http://"):

url = url[7:]

elif url.startswith("https://"):

url = url[8:]

else:

return None, None

if "/" in url:

host, path = url.split("/", 1)

path = "/" + path

else:

host = url

path = "/"

return host, path

# Display error message

def display\_error(message):

messagebox.showerror("Error", message)

status\_label.config(text="Failed to fetch webpage", foreground="red")

details\_text.config(state="normal")

details\_text.delete("1.0", tk.END)

details\_text.insert(tk.END, message)

details\_text.config(state="disabled")

# Display headers and content in the GUI

def display\_content(headers, body):

details\_text.config(state="normal")

details\_text.delete("1.0", tk.END)

details\_text.insert(tk.END, headers)

details\_text.config(state="disabled")

content\_text.config(state="normal")

content\_text.delete("1.0", tk.END)

content\_text.insert(tk.END, body)

content\_text.config(state="disabled")

# Set up the GUI using tkinter

root = tk.Tk()

root.title("Advanced HTTP Web Client")

root.geometry("800x600")

root.config(bg="#f4f4f4")

# Title label

title\_label = tk.Label(root, text="HTTP Web Client", font=("Helvetica", 18, "bold"), bg="#f4f4f4")

title\_label.pack(pady=10)

# Frame for URL entry

url\_frame = tk.Frame(root, bg="#f4f4f4")

url\_frame.pack(pady=10)

url\_label = tk.Label(url\_frame, text="Website URL:", font=("Helvetica", 12), bg="#f4f4f4")

url\_label.grid(row=0, column=0, padx=5)

url\_entry = tk.Entry(url\_frame, width=50, font=("Helvetica", 12))

url\_entry.grid(row=0, column=1, padx=5)

fetch\_button = tk.Button(

url\_frame, text="Fetch Webpage", font=("Helvetica", 12),

bg="#4CAF50", fg="white", command=fetch\_webpage, padx=10, pady=5

)

fetch\_button.grid(row=0, column=2, padx=5)

# Status label

status\_label = tk.Label(root, text="", font=("Helvetica", 12, "italic"), bg="#f4f4f4")

status\_label.pack()

# Frame for displaying headers and content

frame = tk.Frame(root, bg="#f4f4f4")

frame.pack(pady=10, fill="both", expand=True)

# Webpage details area

details\_label = tk.Label(frame, text="Webpage Headers:", font=("Helvetica", 12, "bold"), bg="#f4f4f4")

details\_label.grid(row=0, column=0, sticky="w")

details\_text = scrolledtext.ScrolledText(frame, wrap=tk.WORD, width=40, height=20, font=("Courier New", 10), state="disabled", bg="#f9f9f9")

details\_text.grid(row=1, column=0, padx=5, pady=5, sticky="nsew")

# Webpage content area

content\_label = tk.Label(frame, text="Webpage Content:", font=("Helvetica", 12, "bold"), bg="#f4f4f4")

content\_label.grid(row=0, column=1, sticky="w")

content\_text = scrolledtext.ScrolledText(frame, wrap=tk.WORD, width=60, height=20, font=("Courier New", 10), state="disabled", bg="#f9f9f9")

content\_text.grid(row=1, column=1, padx=5, pady=5, sticky="nsew")

# Configure resizing behavior

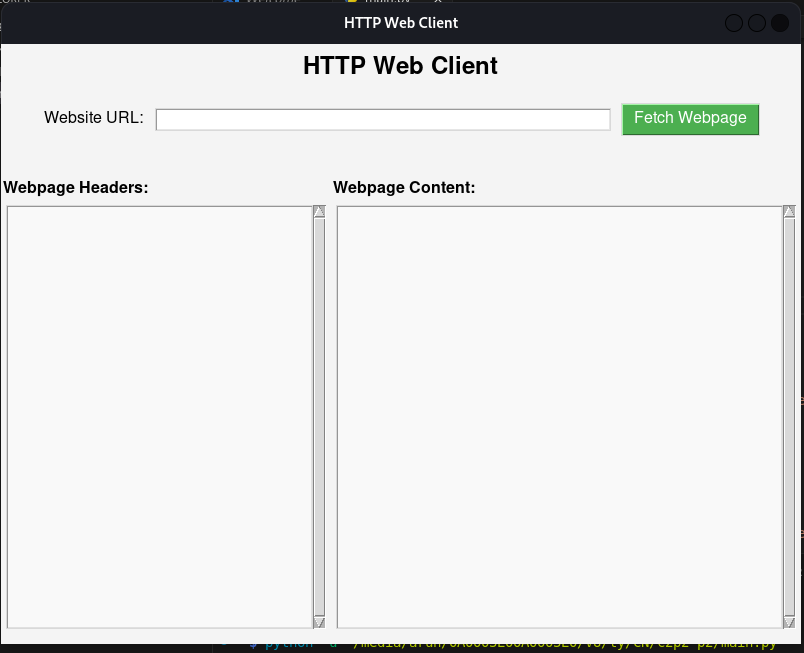
frame.grid\_columnconfigure(0, weight=1)

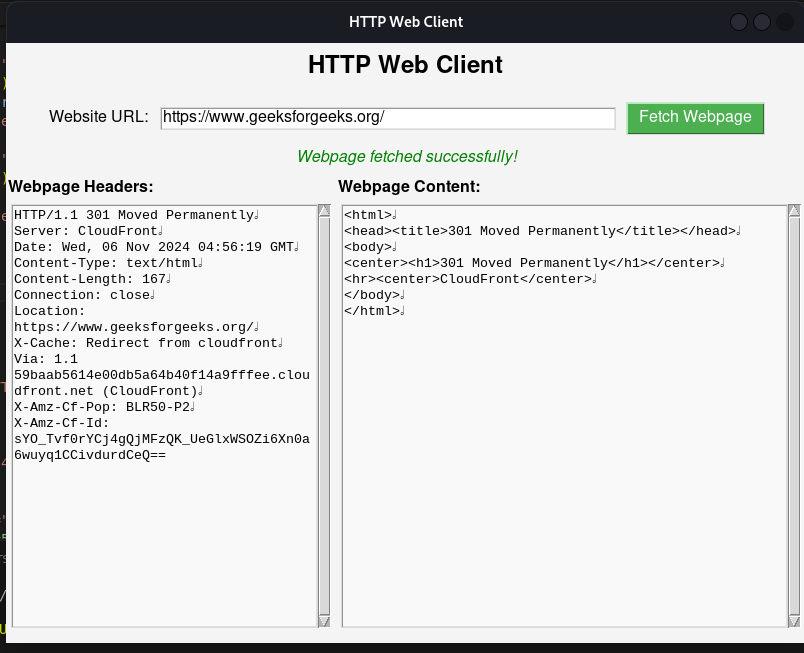
frame.grid\_columnconfigure(1, weight=2)

frame.grid\_rowconfigure(1, weight=1)

root.mainloop()

**Output:**



****

**Conclusion :**

The **Advanced HTTP Web Client** project successfully demonstrates the fundamental workings of a web client, from initiating a TCP connection to fetching and displaying webpage data. By combining socket programming with a graphical user interface, this project allows users to interact with web servers, retrieve raw HTTP data, and understand the structure of HTTP communication.

This project serves as an effective educational tool, giving hands-on experience in networking, HTTP protocol, and GUI development. Users gain insight into the mechanisms behind HTTP requests and responses, as well as how to handle network data for display purposes. Furthermore, the project highlights essential skills in error handling and parsing, both of which are crucial in real-world applications.