LABORATORY REPORT

Application Development Lab
(CS33002)

B.Tech Program in ECSc

Submitted By

Name:-Pruthibiraj Nayak

Roll No: 2230183

Kalinga Institute of Industrial Technology
(Deemed to be University)
Bhubaneswar, India

Spring 2024-2025

Table of Content

Exp No.	Title	Date of Experiment	Date of Submission	Remarks
1.	Build a Resume using HTML/CSS	07.01.2025	13.01.2025	
2.	Machine Learning for Cat and Dog Classification	14.01.2025	20.01.2025	
3.	Regression Analysis for Stock Prediction	22.01.2025	27.01.2025	
4.	Conversational Chatbot with Any Files	04.02.2025	09.02.2025	
5.	Web Scraper using LLMs	11.02.2025	17.03.2025	
6.				
7.				
8.				
9.	Open Ended 1			
10.	Open Ended 2			

Experiment Number	5
Experiment Title	Web Scraper using LLMs
Date of Experiment	11.02.2025
Date of Submission	17.03.2025

1. Objective:-
To create a web scraper application integrated with LLMs for processing scraped data.

2. Procedure:- (Steps Followed)
1. Use Python libraries like BeautifulSoup and Requests to scrape web data.
2. You can also use LlamaIndex for Web Scraping and Ollama for open ended LLMs
3. Integrate LLMs to process and summarize the scraped information.
4. Develop a Flask backend for handling scraping tasks and queries.
5. Create an HTML/CSS frontend to initiate scraping (like the web page to scrape) and display results.
6. You can also take a topic and search the web for a web page and then scrape it.

Code:-

FLASK CODE

```
from flask import Flask, render_template, request, jsonify
import requests
from bs4 import BeautifulSoup
import ollama  # Make sure Ollama is installed and running

app = Flask(__name__)

def search_w(query):
    search_url =
f"https://en.wikipedia.org/w/index.php?search={query.replace(' ', '+')}"
    response = requests.get(search_url)
    soup = BeautifulSoup(response.text, "html.parser")
    # Check if a direct Wikipedia page exists
    if soup.find("link", {"rel": "canonical"}):
        return response.url  # Direct Wikipedia page found

    # Otherwise, get the first search result link
    result = soup.find("div", class_="mw-search-result-heading")

    if result:
        first_link = result.find("a")["href"]
        return f"https://en.wikipedia.org{first_link}"
    else:
        return "Error: No results found."
```

```python
def scrape_w(url):
    try:
        response = requests.get(url)
        response.raise_for_status()  # Check if URL is valid
        soup = BeautifulSoup(response.text, "html.parser")
        # Extract the main content of Wikipedia page
        content_div = soup.find("div", {"class": "mw-parser-output"})
        if not content_div:
            return "Error: Could not find main content."
        paragraphs = content_div.find_all("p")
        text = "\n".join([p.get_text() for p in paragraphs if
p.get_text().strip()])
        return text if text else "Error: No readable content found."
    except requests.exceptions.RequestException as e:
        return f"Error scraping website: {e}"
# ✅ Summarization using Ollama
def summarize_text(text):
    try:
        response = ollama.chat(model="mistral", messages=[{"role": "user",
"content": f"Summarize this: {text}"}])
        return response['message']['content']
    except Exception as e:
        return f"Error: {e}"
# ✅ Flask Routes
@app.route("/")
def home():
    return render_template("index.html")
@app.route("/search", methods=["POST"])
def search():
    data = request.json
    query = data.get("query")
    if not query:
        return jsonify({"error": "Invalid search query"})
    wiki_url = search_wikipedia(query)
    if "Error" in wiki_url:
        return jsonify({"error": wiki_url})
    return jsonify({"url": wiki_url})
@app.route("/scrape", methods=["POST"])
def scrape():
    data = request.json
    url = data.get("url")

    if "wikipedia.org" not in url:
        return jsonify({"error": "Invalid Wikipedia URL"})
    scraped_content = scrape_wikipedia(url)

    if "Error" in scraped_content:
        return jsonify({"error": scraped_content})
```

```
    return jsonify({"content": scraped_content})
@app.route("/summarize", methods=["POST"])
def summarize():
    data = request.json
    text = data.get("text")
    if not text or text.startswith("Error") or text.strip() == "":
        return jsonify({"error": "No valid content to summarize."})
    print(f"Received text for summarization:\n{text[:500]}")  # Log first 500
chars
    try:
        response = ollama.chat(model="mistral", messages=[{"role": "user",
"content": f"Summarize this: {text}"}])
        return jsonify({"summary": response['message']['content']})
    except Exception as e:
        return jsonify({"error": f"Ollama Error: {e}"})

if __name__ == "__main__":
    app.run(debug=True)
```

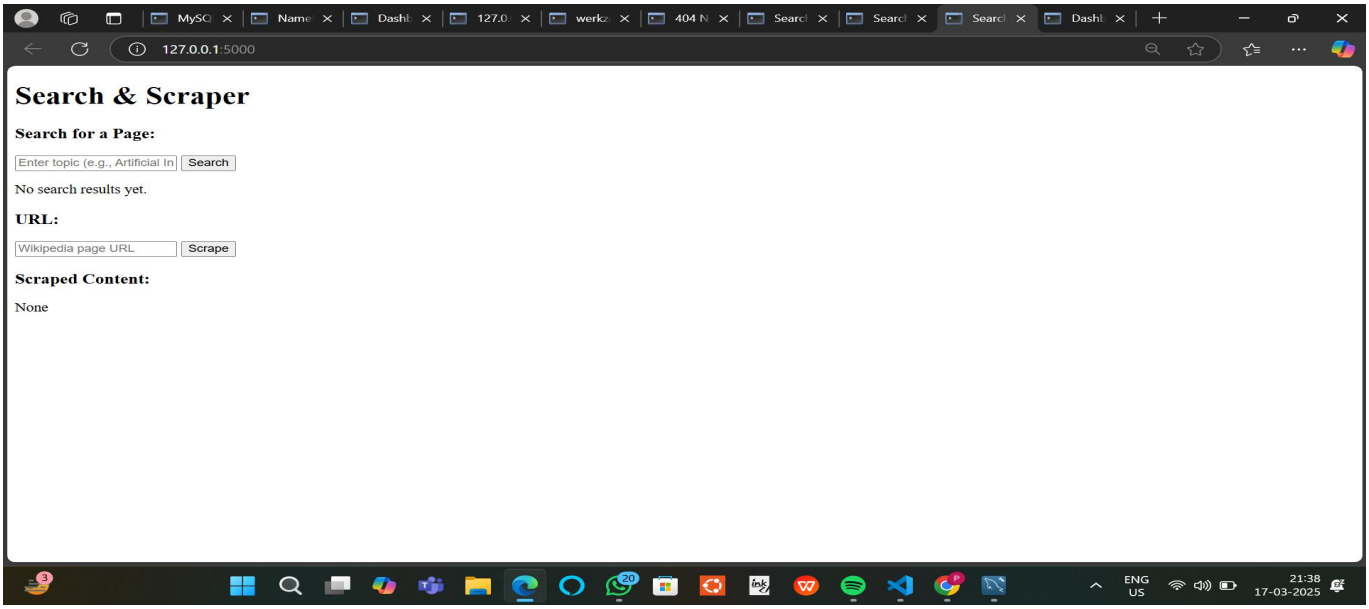
HTML CODE

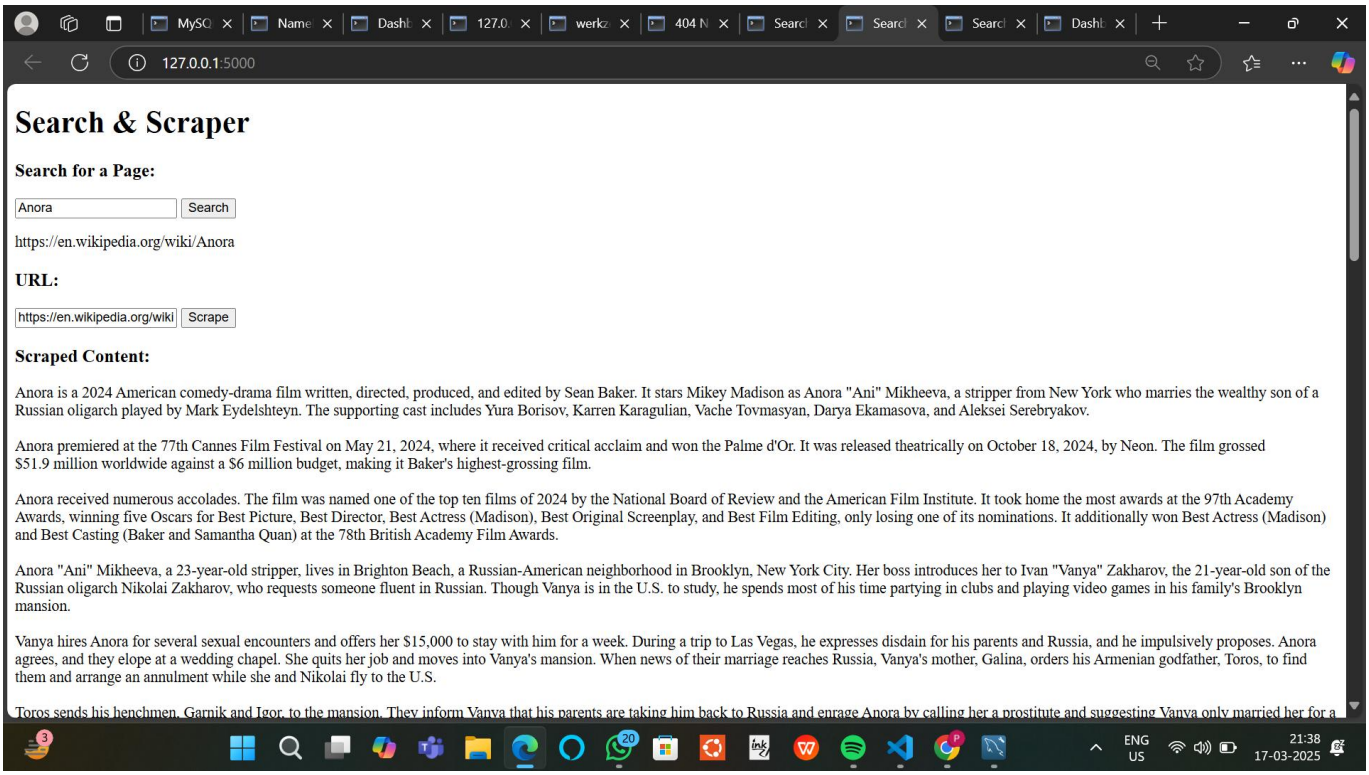
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Search & Scraper</title>
    <script>
        async function searchWiki() {
            const query = document.getElementById("searchQuery").value;
            document.getElementById("searchResult").innerText =
"Searching...";

            const response = await fetch("/search", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ query: query })
            });

            const data = await response.json();
            if (data.error) {
                document.getElementById("searchResult").innerText =
data.error;
            } else {
                document.getElementById("searchResult").innerText = data.url;
                document.getElementById("wikiUrl").value = data.url;
            }
        }
```

```
        async function scrapePage() {
            const url = document.getElementById("wikiUrl").value;
            document.getElementById("scrapedContent").innerText =
"Scraping...";

            const response = await fetch("/scrape", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ url: url })
            });

            const data = await response.json();
            if (data.error) {
                document.getElementById("scrapedContent").innerText =
data.error;
            } else {
                document.getElementById("scrapedContent").innerText =
data.content;
            }
        }

        async function summarizeContent() {
            const content =
document.getElementById("scrapedContent").innerText;
            document.getElementById("summary").innerText = "Summarizing...";
            const response = await fetch("/summarize", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ text: content })
            });
            const data = await response.json();
            if (data.error) {
                document.getElementById("summary").innerText = data.error;
            } else {
                document.getElementById("summary").innerText = data.summary;
            }
        }
    </script>
</head>
<body>
    <h1>Search & Scraper</h1>
    <h3>Search for a Page:</h3>
    <input type="text" id="searchQuery" placeholder="Enter topic (e.g.,
Artificial Intelligence)">
    <button onclick="searchWiki()">Search</button>
    <p id="searchResult">No search results yet.</p>
    <h3>URL:</h3>
    <input type="text" id="wikiUrl" placeholder="Wikipedia page URL">
```

```
    <button onclick="scrapePage()">Scrape</button>
    <h3>Scraped Content:</h3>
    <p id="scrapedContent">None</p>
    <h3>Summarize Content:</h3>
    <button onclick="summarizeContent()">Summarize</button>
    <p id="summary">None</p>
</body>
</html>
```

3. Results/Output:- Entire Screen Shot including Date & Time





4. Remarks:-

This experiment successfully developed a web scraper integrated with LLMs to search, scrape, and summarize web content. We addressed challenges like handling dynamic websites, invalid URLs, and LLM response issues. Using Flask for backend processing and JavaScript for frontend interaction, we enabled users to input a topic, find relevant pages, extract content, and generate concise summaries using Mistral via Ollama. Future improvements could include multi-page scraping, better search accuracy, and structured data extraction. This project highlights the potential of AI-driven web scraping for research, news aggregation, and automated content analysis.

Pruthibiraj Nayak (2230183)

(Name of the Student) (Name of the Coordinator)