LABORATORY REPORT

# Application Development Lab (CS33002)

**B. Tech Program in ECSc**

*Submitted By*

**Name: -Avinash Kumar**

**Roll No:** 2230243

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

Spring 2024-2025

# TABLE OF CONTENT

| | |
|---|---|
| **Experiment Number** | 5 |
| **Experiment Title** | Web Scraper using LLMs |
| **Date of Experiment** | 10 February 2025 |
| **Date of Submission** | 17 February 2025 |

1. **OBJECTIVE** – To create a web scraper application integrated with LLMs for processing scraped data.

2. **PROCEDURE –**

   i. Use Python libraries like BeautifulSoup and Requests to scrape web data.
   ii. You can also use LlamaIndex for Web Scraping and Ollama for open ended LLMs
   iii. Integrate LLMs to process and summarize the scraped information.
   iv. Develop a Flask backend for handling scraping tasks and queries.
   v. Create an HTML/CSS frontend to initiate scraping (like the web page to scrape) and display results.
   vi. You can also take a topic and search the web for a web page and then scrape it.

3. **CODE –**

   **Backend Flask Code(app.py)**

```
from flask import Flask, request, jsonify, render_template
from scraper import scrape_website, summarize_text, chat_with_llm, select_model

app = Flask(__name__)

GROQ_API_KEY = "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"

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/scrape', methods=['POST'])
def scrape_and_summarize():
    data = request.json
    url = data.get('url')

    if not url:
        return jsonify({"error": "URL is required"}), 400


    scraped_text = scrape_website(url)

    summary = summarize_text(scraped_text)
    return jsonify({
        "scraped_text": scraped_text,
        "summary": summary
    })

@app.route('/chat', methods=['POST'])
def chat():
    data = request.json
    user_message = data.get('message')
    context = data.get('context', '')

    if not user_message:
        return jsonify({"error": "Message is required"}), 400

    response = chat_with_llm(user_message, context)

    return jsonify({
        "response": response
    })

@app.route('/select_model', methods=['POST'])
def change_model():
    data = request.json
    model_name = data.get('model')

    if not model_name:
        return jsonify({"error": "Model name is required"}), 400

    result = select_model(model_name)

    return jsonify({
        "message": result
    })

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

## Backend Code(llm_processor.py)

```
from llama_index import SimpleWebPageReader, GPTListIndex

def summarize_with_llm(text):
    """
    Summarize the given text using an LLM.
    """
    try:

        index = GPTListIndex.from_documents([SimpleWebPageReader.load_data(text)])
        query_engine = index.as_query_engine()
        summary = query_engine.query("Summarize the following text:")
        return summary.response
    except Exception as e:
        return f"An error occurred while summarizing: {str(e)}"
```

## Backend Code(scraper.py)

```
import requests
from bs4 import BeautifulSoup
from groq import Groq

GROQ_API_KEY = "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"

client = Groq(api_key=GROQ_API_KEY)

MODELS = {
    'Gemma2 💎': 'gemma2-9b-it',
    'Mixtral 🚀': 'mixtral-8x7b-32768'
}

SELECTED_MODEL = MODELS['Mixtral 🚀']

def scrape_website(url):
    """
    Scrape text content from a website.
    """
    try:

        response = requests.get(url)
        response.raise_for_status()

        soup = BeautifulSoup(response.text, 'html.parser')

        paragraphs = soup.find_all('p')
        text_content = " ".join([p.get_text() for p in paragraphs])

        return text_content
    except Exception as e:
        return f"Error scraping website: {e}"

def summarize_text(text, model=SELECTED_MODEL, max_tokens=30000):
    """
    Summarize text using the selected Groq model.
    """
    try:

        response = client.chat.completions.create(
            model=model,
            messages=[
                {"role": "system", "content": "You are a helpful assistant."},
                {"role": "user", "content": f"Summarize the following text:\n{text}"}
            ]
        )
        return response.choices[0].message.content
    except Exception as e:
        return f"Error summarizing text: {e}"

def chat_with_llm(message, context="", model=SELECTED_MODEL,max_tokens=30000):
    """
    Generate a response using the selected Groq model.
    """
    try:

        response = client.chat.completions.create(
            model=model,
            messages=[
                {"role": "system", "content": "You are a helpful assistant."},
                {"role": "user", "content": f"{context}\n\nUser: {message}\nAI:"}
            ]
        )
        return response.choices[0].message.content
    except Exception as e:
        return f"Error generating response: {e}"

def select_model(model_name):
    """
    Select the model for summarization and chatbot.
    """
    global SELECTED_MODEL
    if model_name in MODELS:
        SELECTED_MODEL = MODELS[model_name]
        return f"Model changed to {model_name}"
    else:
        return "Invalid model selected"
```

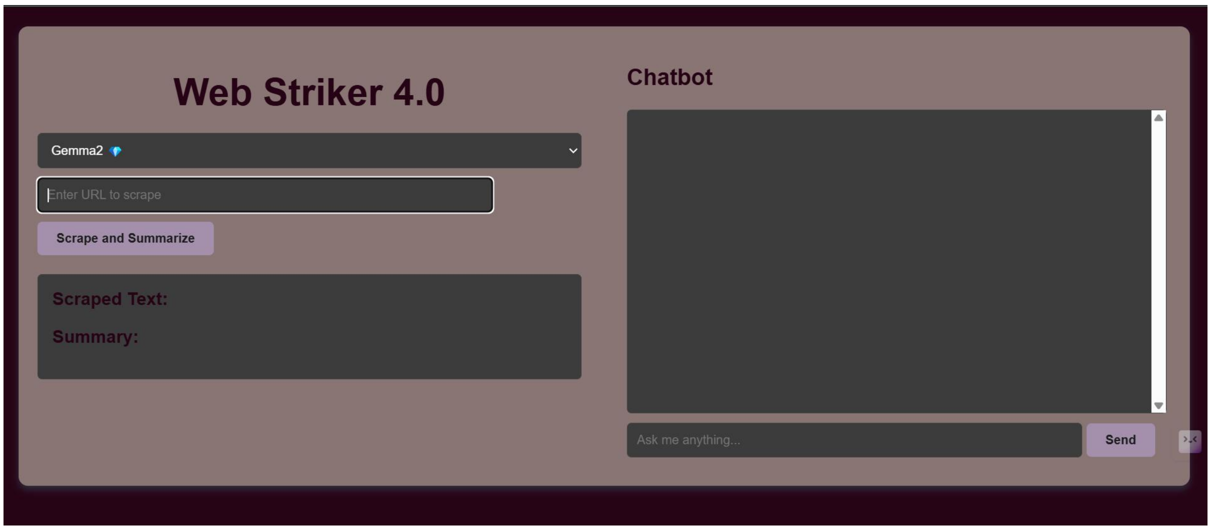
## Frontend Code(index.html)

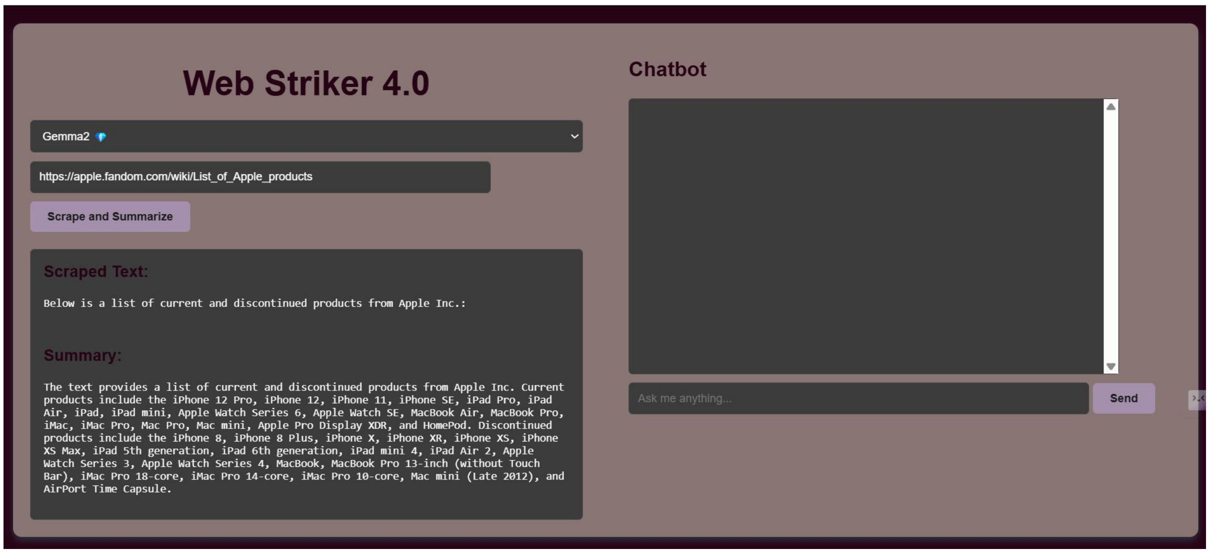
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Web Striker 4.0</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            background-color:rgb(39, 3, 22);
            color: #ffffff;
            margin: 0;
            padding: 20px;
        }
        .container {
            display: flex;
            justify-content: space-between;
            max-width: 1500px;
            margin: 0 auto;
            background-color:rgb(138, 117, 117);
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 4px 8px rgba(87, 235, 237, 0.2);
        }
        .left, .right {
            width: 48%;
        }
        h1 {
            text-align: center;
            font-size: 2.5rem;
            color:rgb(39, 3, 22);
            margin-bottom: 20px;
        }
        h2 {
            color:rgb(39, 3, 22);
            margin-top: 20px;
        }
        input[type="text"] {
            width: 80%;
            padding: 10px;
            margin-bottom: 10px;
            border: 1px solid #444;
            border-radius: 5px;
            background-color: #3c3c3c;
            color: #ffffff;
        }
        button {
            padding: 10px 20px;
            background-color:rgb(164, 144, 172);
            color: #1e1e1e;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-weight: bold;
        }
        button:hover {
            background-color: rgb(39, 3, 22);
        }
        .result {
            margin-top: 20px;
            padding: 15px;
            background-color: #3c3c3c;
            border: 1px solid #444;
            border-radius: 5px;
        }
        .result h3 {
            margin-top: 0;
            color: rgb(39, 3, 22);
        }
        pre {
            white-space: pre-wrap;
            word-wrap: break-word;
            color: #ffffff;
        }
        .chat-container {
            margin-top: 20px;
        }
        .chat-box {
            height: 300px;
            width: 550px;
            overflow-y: scroll;
            border: 1px solid #444;
            border-radius: 5px;
            padding: 10px;
            margin-bottom: 10px;
            background-color: #3c3c3c;
        }
        .chat-message {
            margin-bottom: 10px;
            padding: 10px;
            border-radius: 5px;
```

```
        }
        .chat-message.user {
            background-color: #444;
            text-align: right;
        }
        .chat-message.ai {
            background-color: #555;
            text-align: left;
        }
        .chat-message strong {
            color: #00ff88;
        }
        select {
            width: 100%;
            padding: 10px;
            margin-bottom: 10px;
            border: 1px solid #444;
            border-radius: 5px;
            background-color: #3c3c3c;
            color: #ffffff;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="left">
            <h1>Web Striker 4.0</h1>

            <!-- Model Selection Dropdown -->
            <select id="modelSelect" onchange="selectModel()">
                <option value="Gemma2    ">Gemma2    </option>
                <option value="Mixtral    ">Mixtral    </option>
            </select>

            <!-- Web Scraper Section -->
            <input type="text" id="url" placeholder="Enter URL to scrape">
            <button onclick="scrape()">Scrape and Summarize</button>

            <div class="result">
                <h3>Scraped Text:</h3>
                <pre id="scrapedText"></pre>
                <h3>Summary:</h3>
                <pre id="summary"></pre>
            </div>
        </div>

        <!-- Chatbot Section -->
        <div class="right">
            <h2>Chatbot</h2>
            <div class="chat-box" id="chatBox"></div>
            <input type="text" id="chatInput" placeholder="Ask me anything...">
            <button onclick="sendMessage()">Send</button>
        </div>
    </div>

    <script>
        async function scrape() {
            const url = document.getElementById('url').value;
            const scrapedTextDiv = document.getElementById('scrapedText');
            const summaryDiv = document.getElementById('summary');
        if (!url) {
          scrapedTextDiv.innerHTML = "Please enter a valid URL.";
          summaryDiv.innerHTML = "";
          return;
        }

        scrapedTextDiv.innerHTML = "Scraping...";
        summaryDiv.innerHTML = "Summarizing...";

        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) {
          scrapedTextDiv.innerHTML = `Error: ${data.error}`;
          summaryDiv.innerHTML = "";
        } else {
          scrapedTextDiv.innerHTML = data.scraped_text;
          summaryDiv.innerHTML = data.summary;
        }
      }

      async function sendMessage() {
        const chatInput = document.getElementById('chatInput');
        const chatBox = document.getElementById('chatBox');
        const message = chatInput.value;

        if (!message) return;

        // Add user message to chat box
        chatBox.innerHTML += `<div class="chat-message user"><strong>You:</strong>
${message}</div>`;
        chatInput.value = "";

        // Get the context (scraped text and summary)
        const scrapedText = document.getElementById('scrapedText').innerText;
        const summary = document.getElementById('summary').innerText;
        const context = `Scraped Text: ${scrapedText}\n\nSummary: ${summary}`;

        // Send message to the chatbot
        const response = await fetch('/chat', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({ message: message, context: context })
        });

        const data = await response.json();

        // Add AI response to chat box
        chatBox.innerHTML += `<div class="chat-message ai"><strong>AI:</strong>
${data.response}</div>`;
```

```
            chatBox.scrollTop = chatBox.scrollHeight;
        }

        async function selectModel() {
            const selectedModel = document.getElementById('modelSelect').value;
            const response = await fetch('/select_model', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ model: selectedModel })
            });
            const data = await response.json();
            alert(data.message);
        }
    </script>
</body>
</html>
```

## 4. OUTPUT –



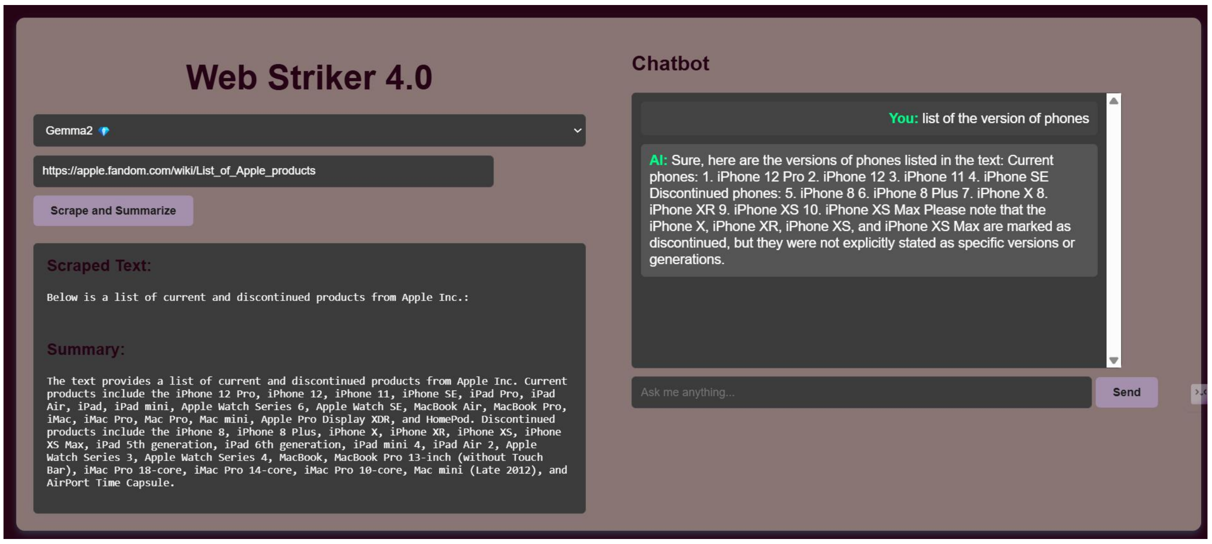


Scraped Text:

Below is a list of current and discontinued products from Apple Inc.:

Summary:

The text provides a list of current and discontinued products from Apple Inc. Current products include the iPhone 12 Pro, iPhone 12, iPhone 11, iPhone SE, iPad Pro, iPad Air, iPad, iPad mini, Apple Watch Series 6, Apple Watch SE, MacBook Air, MacBook Pro, iMac, iMac Pro, Mac Pro, Mac mini, Apple Pro Display XDR, and HomePod. Discontinued products include the iPhone 8, iPhone 8 Plus, iPhone X, iPhone XR, iPhone XS, iPhone XS Max, iPad 5th generation, iPad 6th generation, iPad mini 4, iPad Air 2, Apple Watch Series 3, Apple Watch Series 4, MacBook, MacBook Pro 13-inch (without Touch Bar), iMac Pro 18-core, iMac Pro 14-core, iMac Pro 10-core, Mac mini (Late 2012), and AirPort Time Capsule.



Scraped Text:

Below is a list of current and discontinued products from Apple Inc.:

Summary:

The text provides a list of current and discontinued products from Apple Inc. Current products include the iPhone 12 Pro, iPhone 12, iPhone 11, iPhone SE, iPad Pro, iPad Air, iPad, iPad mini, Apple Watch Series 6, Apple Watch SE, MacBook Air, MacBook Pro, iMac, iMac Pro, Mac Pro, Mac mini, Apple Pro Display XDR, and HomePod. Discontinued products include the iPhone 8, iPhone 8 Plus, iPhone X, iPhone XR, iPhone XS, iPhone XS Max, iPad 5th generation, iPad 6th generation, iPad mini 4, iPad Air 2, Apple Watch Series 3, Apple Watch Series 4, MacBook, MacBook Pro 13-inch (without Touch Bar), iMac Pro 18-core, iMac Pro 14-core, iMac Pro 10-core, Mac mini (Late 2012), and AirPort Time Capsule.

You: list of the version of phones

AI: Sure, here are the versions of phones listed in the text: Current phones: 1. iPhone 12 Pro 2. iPhone 12 3. iPhone 11 4. iPhone SE Discontinued phones: 5. iPhone 8 6. iPhone 8 Plus 7. iPhone X 8. iPhone XR 9. iPhone XS 10. iPhone XS Max Please note that the iPhone X, iPhone XR, iPhone XS, and iPhone XS Max are marked as discontinued, but they were not explicitly stated as specific versions or generations.

## 5. REMARK –

- Successfully implemented web scraping with LLMs.
- Used Python libraries like BeautifulSoup, Requests, and LlamaIndex.
- Integrated LLMs for summarization and chatbot functionality.
- Developed a Flask backend with an HTML/CSS frontend.
- Basic error handling present but can be improved.
- API key exposure should be avoided for security.
- Optimization needed for large text processing.
- Experiment successfully demonstrated LLM-powered web scraping.

Avinash kumar
Signature of the Student

Avinash Kumar

Bhargav Appasani
Signature of the Lab Coordinator

Prof. Bhargav Appasani