Structured and Hyper Text Model

Information Retrieval

Submitted by:

2021-CS-46 Muhammad Usman Asghar

Submitted to:

Dr. Syed Khaldoon Khurshid

Department of Computer Science

University of Engineering and Technology, Lahore

Pakistan

Contents

1 Structured Guided Model

This project involves the creation of a web application using Flask, combined with a web scraping component to fetch and display data about cars and bikes from PakWheels. The application dynamically generates and serves content from JSON files.

2 Flask Application Code

2.1 Function: `load_content_map`

Purpose: This function loads data from JSON files and combines them into a content map.

Code:

Listing 1: load_content_map

```
def load_content_map():
    used_cars_data = []
    bikes_data = []
    new_cars_data = []

    # Load used cars data
    with open('used_cars.json') as file:
        used_cars_data = json.load(file)

    # Load bikes data
    with open('bikes.json') as file:
        bikes_data = json.load(file)

    # Load new cars data
    with open('new_cars.json') as file:
        new_cars_data = json.load(file)

    # Combine the data into a single content map
    content_map = {
        'used_cars': used_cars_data,
        'bikes': bikes_data,
        'new_cars': new_cars_data
    }
    return content_map
```

Explanation: This function reads three JSON files (`used_cars.json`, `bikes.json`, and `new_cars.json`) and combines their data into a dictionary.

2.2 Route: /

Purpose: Serves the homepage displaying the full content map.

Code:

Listing 2: Route for index

```
@app.route('/')
def index():
    content_map = load_content_map()
    return render_template('index.html', content_map=content_map)
```

Explanation: This route uses the `load_content_map` function to fetch content and passes it to the `index.html` template for rendering.

2.3 Route: /content/<section>

Purpose: Serves content for a specific section.

Code:

Listing 3: Route for specific section

```
@app.route('/content/<section>')
def get_content(section):
    content_map = load_content_map()
    if section in content_map:
        return render_template('content.html', section=section, items=
            content_map[section])
    else:
        return jsonify({'error': 'Section not found'}), 404
```

Explanation: This route dynamically serves content for a requested section (e.g., `used_cars`, `bikes`, etc.) or returns an error if the section doesn't exist.

3 Web Scraping Code

3.1 Function: get_used_cars_structure

Purpose: Scrapes data about used cars from the given URL.

Code:

Listing 4: Scraping used cars

```
def get_used_cars_structure(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')

    cars = []
    car_elements = soup.find_all('div', class_='cards-content')

    for car in car_elements:
        title_tag = car.find('h3', class_='nomargin truncate')
        price_tag = car.find('div', class_='generic-green')
```

```
        location_tag = car.find('div', class_='generic-gray')

        if title_tag and price_tag and location_tag:
            title = title_tag.text.strip()
            price = price_tag.text.strip()
            location = location_tag.text.strip()

            cars.append({
                'title': title,
                'price': price,
                'location': location
            })

    return cars
```

Explanation: This function sends an HTTP request to the specified URL, parses the response using BeautifulSoup, and extracts data about cars.

3.2 Function: scrape_car_data

Purpose: Scrapes detailed data about new cars, including ratings and reviews.

Code:

Listing 5: Scraping new cars

```
def scrape_car_data(url):
    car_data = []
    response = requests.get(url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.text, 'html.parser')
        car_listings = soup.find_all('li', class_='col-md-3')

        for car in car_listings:
            title = car.find('h3', class_='nomargin truncate').text.strip() if car.find('h3', class_='nomargin truncate') else "N/A"
            price_element = car.find('div', class_='generic-green truncate fs14')
            price = price_element.text.strip() if price_element else "N/A"

            rating_element = car.find('span', class_='rating')
            rating_value = 0
            if rating_element:
                rating = rating_element.find_all('i')
                rating_value = sum(1 if 'fa-star' in str(r) else 0 for r in rating)

            reviews_count_element = car.find('span', class_='fs14 generic-gray ml5 dib')
            reviews_count = reviews_count_element.text.strip() if reviews_count_element else "N/A"

            car_data.append({
                'Title': title,
                'Price': price,
                'Rating': rating_value,
                'Reviews Count': reviews_count
            })

    return car_data
```

Explanation: This function extracts more detailed information from car listings, including reviews and ratings.

4 Saving Data to JSON

Code:

Listing 6: Saving data to JSON

```
with open('used_cars.json', 'w') as f:
    json.dump(used_cars_data, f, indent=4)

with open('new_cars.json', 'w') as f:
    json.dump(new_cars_data, f, indent=4)

with open('bikes.json', 'w') as f:
    json.dump(bikes_data, f, indent=4)
```

Explanation: The scraped data is saved into separate JSON files for later use in the Flask application.

5 Hyper-Text Model

This project implements a hypertext model using Flask and web scraping. The application scrapes structured content from Wikipedia, processes it into sections, and displays it with hyperlinks between related terms.

6 Flask Application Code

6.1 Route: /

Purpose: Serves the homepage with a list of headings and their content.

Code:

Listing 7: Home Route

```
@app.route('/')
def home():
    # Prepare headings for sidebar and content rendering
    headings = list(wiki_content.keys())
    return render_template('home.html', headings=headings, content=
        wiki_content)
```

Explanation: This route loads all the section headings and their respective content from the `pakistan_content.` file and renders them on the homepage.

6.2 Route: /section

Purpose: Displays the content of a specific section with clickable links to other sections.

Code:

Listing 8: Section Route

```
@app.route('/<section>')
def show_section(section):
    content = wiki_content.get(section)
    if content:
        linked_content = add_hyperlinks(" ".join(content), wiki_content.
            keys())
        return render_template('section.html', section=section, content=
            linked_content)
    return "Section not found", 404
```

Explanation: This route dynamically renders the content for a requested section, adding hyperlinks for all referenced headings within the content.

6.3 Function: add_hyperlinks

Purpose: Adds hyperlinks to terms in the content that correspond to other headings.

Code:

Listing 9: Hyperlinking Function

```
def add_hyperlinks(content, headings):
    for heading in headings:
        content = re.sub(rf"\b{heading}\b", f'<a href="#{heading}">{
            heading}</a>', content)
    return content
```

Explanation: The function scans the content for matches with the provided headings and converts these matches into clickable hyperlinks.

7 Web Scraping Code

7.1 URL: https://en.wikipedia.org/wiki/Pakistan

Purpose: Scrapes headings and paragraphs from the Wikipedia page on Pakistan.

7.2 Scraping Content

Purpose: Extracts sections and paragraphs from the Wikipedia page.

Code:

Listing 10: Scraping Content

```
content = {}
current_section = None

for element in soup.find_all(['h2', 'h3', 'p']):
    if element.name in ['h2', 'h3']:
        current_section = element.text.strip()
        content[current_section] = []
    elif element.name == 'p' and current_section:
        paragraph = ''.join(format_content(child) for child in element.
            children)
        content[current_section].append(paragraph)
```

Explanation: The code parses the Wikipedia page for headings and their corresponding paragraphs, organizing them into sections.

7.3 Function: format_content

Purpose: Formats text with hyperlinks and citations.

Code:

Listing 11: Formatting Content

```
def format_content(element):
    if element.name == 'a':
        href = element.get('href')
        if href and href.startswith('/'):
            href = base_url + href
        link_text = element.text
        return f'<a href="{href}" target="_blank">{link_text}</a>'
    elif element.name == 'sup':
        return f'<sup>{element.text}</sup>'
    return element.text
```

Explanation: The function identifies anchor tags and superscripts in the HTML, converts them into clickable links, and retains citations.

7.4 Saving Content to JSON

Purpose: Saves the structured content into a JSON file.

Code:

Listing 12: Saving JSON

```
with open('pakistan_content.json', 'w') as f:
    json.dump(content, f, indent=4)
```

Explanation: This code saves the scraped and structured content into `pakistan_content.json` for use in the Flask application.

8 Sample JSON Structure

Example:

Listing 13: Example JSON Content

```
{
    "History": [
        "Pakistan gained independence in 1947.",
        "It was created as a separate state for Muslims."
    ],
    "Geography": [
        "Pakistan is located in South Asia.",
        "It shares borders with India, Afghanistan, and Iran."
    ]
}
```

Explanation: Each section contains a list of paragraphs, with headings as keys.